WORKING DRAFT

X3T10 Project 990D

Revision 2
07-JUL-97

Information technology - Common Access Method - 3 (CAM-3)

This is a draft proposed American National Standard of Accredited Standards Committee X3. As such this is not a completed standard. The X3T10 Technical Committee may modify this document as a result of comments received during public review and its approval as a standard.

ASC X3T10 Technical Editor: William D. Dallas
Digital Equipment Corporation
110 Spit Brook Road
Nashua NH 03060
USA

Telephone: 603-881-2508
Facsimile: 603-881-2257
Email: dallas@wasted.zk3.dec.com

Reference number ISO/IEC **** : 199x
ANSI X3.232 - 199x

POINTS OF CONTACT

X3T10 Chair	**X3T10 Vice-Chair**
John B. Lohmeyer	Lawrence J. Lamers
Symbios Logic	Adaptec
1635 Aeroplaza Drive	691 South Milpitas Blvd.
Colorado Springs, CO 80916	Milpitas, CA 95035
Tel: (719) 573-3362	Tel: (408) 957-7817
Fax: (719) 597-3037	Fax: (408) 957-7193
Email: john.lohmeyer@symbios.com	Email: ljlamers@aol.com

X3 Secretariat
Lynn Barra
Administrator Standards Processing
X3 Secretariat
1250 Eye Street, NW Suite 200
Washington, DC 20005

Tel: (202 626-5738
Fax: (202) 638-4922

SCSI Reflector
Internet address for subscription to the SCSI reflector: scsi-request@symbios.com
Internet address for distribution via SCSI reflector: scsi@symbios.com

SCSI Bulletin Board
Tel: (719) 574-0424

Document Distribution
Global Engineering
15 Inverness Way East
Englewood, CO 80112-5704

Tel: (303) 792-2181 or (800) 854-7179
Fax: (303) 792-2192

ABSTRACT

To be supplied

PATENT STATEMENT

The developers of this standard have requested that holder's of patents that may be required for the implementation of the standard , disclose such patents to the publisher. However neither the developers nor the publisher have undertaken a patent search in order to identify which patents may apply to this standard.

No position is taken with respect to the validity of any claim or any patent rights that may have been disclosed. Details of submitted statements may be obtained from the publisher concerning any statement of patents and willingness to grant a license under these rights on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license.

Document Revision Status

Revision 2

- Converted document from Word Perfect to MS Word
- Defined CCB formats for CAM-3
- Added dynamic topology methodology (SSA, FC etc.)
- Added Async events

Planned changes for Revision 3

- Place all structure definitions into tables
- Place all data values and bit definitions into tables
- Change where applicable for SAM terminology
- Add peripheral driver model
- Add Target model
- Update definitions
- Add notes to explain reasons why certain functions/service are defined (e.g. prevents race conditions on SMP systems.
- Add diagrams to show data structure relationships in EDT
- Review document for terminology mismatches and change where applicable
- Review sentence structure and change where applicable
- Review for design flaws and change where applicable
- Added methodology to obtain interconnect status (up, down, initializing)
- Update undefined cross references in document - (e.g. refer to Clause XX to refer to Clause 9.1.2)

Table Of Contents

Tables

Figures

Notes

Foreword

(Editors Mark - Create forward)

Introduction

The industry provides a diverse range of peripherals for attachment to a wide range of computing equipment. Some system manufacturers have developed approaches for their attachment which are widely followed, increasing the applications available for the attachment of peripheral devices. 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 peripheral to an host adapter.

In an effort to broaden the application base for 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 continuing standard.

Information processing systems -- Common Access Method - 3

1. Scope

This standard defines the Common Access Method - 3 (CAM-3) for the control of devices.

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

The CAM-3 provides a structured method for supporting peripherals with the software (e.g., peripheral driver) and hardware (e.g., host adapter) associated with any computer.

This standard addresses the following inter-connects
- SCSI

(Editors Mark - ATA? Network devices/protocols)

2. Normative references

ANSI X3.131-1994, *Small Computer Systems Interface - 2*

3. Definitions and Conventions

3.1 **Definitions**

For the purposes of this standard the following definitions apply

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

3.1.2 CCB (CAM-3 control block)
The data structure provided by peripheral drivers to the XPT to control execution of a function by the SIM.

3.1.3 Connection_ID
An address specifier that contains the Port_ID and the protocol specific physical address specifiers. The protocol specific address identifiers may be either the protocol specific address specifiers or a SIM/HA representation of the protocol specific address identifiers (e.g., a SSA HA may translate a HA representation specific address identifiers to hop counts).

3.1.4 Device
A physical piece of equipment or mechanism designed to serve a special purpose or perform a special function (e.g., a SCSI disk device). A device is an addressable entity that performs a function.

3.1.5 Device Query
A mechanism by which the XPT determines the device configuration of a specific SIM/HA as specified by the Port_ID (e.g., the addresses of devices seen for a Port_ID by a SIM/HA).

3.1.6 Immediate CCB
Provides valid completion status when the call to xpt_action () returns (e.g., path inquiry).

3.1.7 Queued CCB

Provides status when the completion callback routine is called, or the CAM-3 Status field in the CCB changes from valid completion Request In Progress to another valid CAM-3 Status.

3.1.8 CDB (command descriptor block)

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

3.1.9 DMA (direct memory access)

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

3.1.10 freeze

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

3.1.11 HA (host adapter)

The hardware and microcode which provides the interface between system memory and any number of protocol inter-connects (e.g., SCSI parallel bus(es) host adapter or SCSI FCP serial bus(es)).

3.1.12 null

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

3.1.13 optional

This term describes features which are not required by this standard. However, if any feature defined by this 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.

3.1.14 PD (Peripheral Driver)

A software module designed to control device models under the CAM-3 framework.

3.1.15 OSD (Operating System Dependant)

This term describes a capability, method of operation, or feature that depends on the specific operating system on which CAM-3 is implemented.

3.1.16 Path

This term describes the Port_ID of the XPT or a Port_ID of a SIM/HA combined with a physical address a specific device.

3.1.17 Port_ID

A XPT assigned value for an inter-connect, which may have device(s). The Port_ID is a component of a Connection_ID.

3.1.18 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.

3.1.19 Scan

A CAM-3 CCB function that directs a Port_ID to determine its device configuration.

3.1.20 SIM (system interface module)

A software module designed to accept the CAM-3 control blocks routed through the XPT in order to execute commands and perform other functions.

3.1.21 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 among vendors.

3.1.22 XTP (transport)

A layer of software which peripheral drivers use to originate the execution of CAM-3 functions.

3.2 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. 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).

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

4. 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 standard.
- correctly interoperate with any conforming System 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 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 functions as required by this 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 functions.

If an operating system is not specified in this standard, then that operating system shall conform to Clause XXX in this standard. (See also annex XXX .)

Claims of conformance to this 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 Adapter (HA) engines are supported.

5. General description

The application environment for CAM-3 is any computer communicating with a device through a protocol chip on a motherboard or a host adapter for the defined inter-connects in this standard.

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

The purpose of the CAM-3 is to define a standard for the support of Host Adapters (HA) and the like by peripheral driver software.

Software in the operating system dispatches I/O requests to the peripherals in a number of different ways depending on the software architecture. The operating system dependencies (OSD) are defined in clause XXX for certain named software and hardware platforms.
(Editors Mark - Should the above be a clause or annexes - I think it all the OSs should be in annexes)

5.1 Environment

A model of the CAM-3 usage environment is illustrated in figure 1. Multiple applications are shown accessing a variety of devices. Several 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 XXX defines this dependency for certain named software and hardware platforms.

Requests for I/O are made through the CAM-3 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. The XPT services are incorporated into a single logical module which integrates both XPT and SIM functionality. The XPT services/functionality may be provided by the operating system, or can be achieved through associating multiple separately loaded software modules.

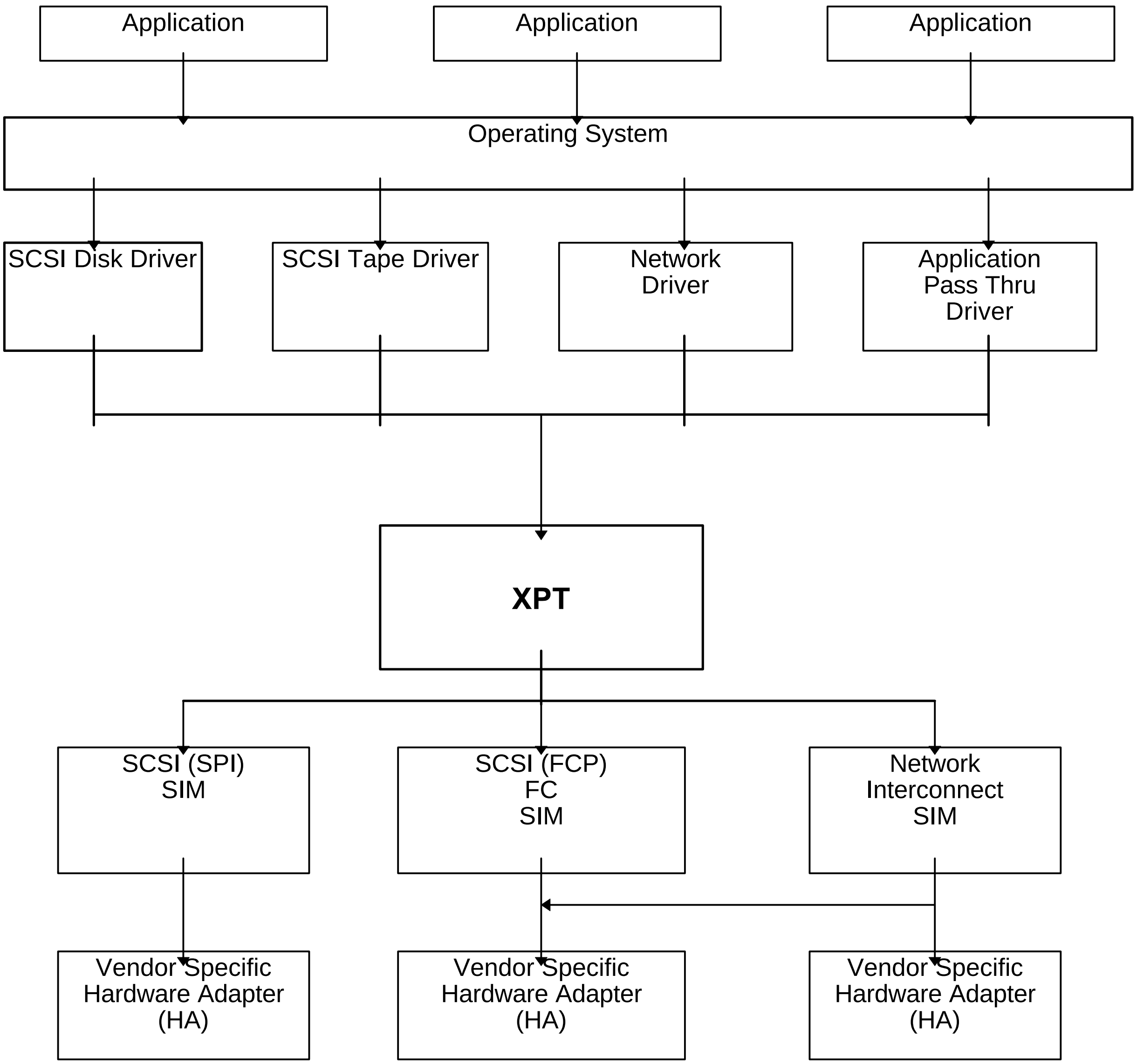

Figure 1 Cam-3 environment model

5.2 Peripheral driver functions

Peripheral drivers provide the following functionality:

1. Interpreting of application or system level requests;
2. Mapping of application level requests to XPT/SIM control blocks;
3. Requesting of resources to initiate a CAM-3 request:
    a) CAM-3 control blocks and supporting blocks that may be needed,
    b) Buffer requirements;
4. Handling of exception conditions not managed transparently by the architecture (e.g., a SCSI check condition status, unexpected bus free, resets, etc.);
5. Logging of exception conditions for maintenance analysis programs;

6. Format utility or services required by format utilities;
7. Establishing parameters for HA operation;
8. Set up routing of I/O requests to the correct path;
9. Initialization and configuration functions of a device not handled by a utility at installation and formatting time;
10. Establishing a time-out value for a task and passing this value in the CCB.

5.3 XPT functions

XPT services provide the following functionality to process CCBs:

1. Routing of the CCB to the proper SIM;
2. OSD management of CCB resources;
3. Maintenance of the Equipment Device Table(s). (This consists of owning the table and servicing requests to read and write the table.);
4. Providing properly formatted control blocks and priming the fields needed to accomplish a request;
5. Routing of asynchronous events back to peripheral drivers;
6. Mapping of operating system serves in a generic fashion as specified;
7. Provide the mandatory infrastructure services;

5.4 SIM functions

1. Perform all interface functions to the HA;
2. Manage or delegate (as appropriate) all the HA protocol steps;
3. Distinguish abnormal behavior and perform error recovery, as required;
4. Manage data transfer path hardware, including DMA circuitry and address mapping, and establish DMA resource requests (if necessary);
5. Queuing of multiple operations for different logical devices as well as the same logical device;
6. Freeze and unfreeze the queue as necessary to accomplish queue recovery;
7. Post the completed operation to the initiating device driver;
8. Manage protocol specific transactions;
9. implement a timer mechanism, using values provided by the peripheral driver;

6. Background

CAM-3 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 HA and may present different device specific protocol interfaces.

6.1 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 peripherals, so separate drivers may be needed for each protocol class of installed device (e.g., SCSI disk, SCSI tape and ATA disk). These same protocol drivers need to be able to share the HA hardware that supports that protocol. These drivers also have to work with a broad range of HA hardware, from highly intelligent coprocessors to the most primitive, including a chip on a motherboard. A standard programming interface layer is essential to insulate peripheral drivers and utilities from the HA hardware implementation, and to allow multiple drivers to share a single hardware interface.

6.2 CAM-3 (Common Access Method - 3)

This standard describes the general definition of the CAM-3 (Common Access Method). CAM-3 functionality has been separated into a few major elements.

- XPT (Transport)
  The XPT (transport) defines a software architecture for peripheral drivers and programs to submit I/O requests to the HA specific SIM module(s). Routing of requests to the correct HA and posting the results of a request back to the driver are responsibilities of the transport.

- SIM (System Interface Module)
  The SIM (System Interface Module) manages HA resources and provides a hardware independent interface for applications and drivers. The SIM is responsible to process and execute inter-connect specific requests, and manage the interface to the HA 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-3 Control Block)
  The CAM-3 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.

6.3 OSD (Operating System Dependencies)

The system environment in which the CAM-3 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 CAM-3 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-3 modules implemented by different manufacturers.

OSD issues are discussed in Clause XXX.

6.4 Architectural considerations

Programming effort has been minimized by making the interfaces as similar as possible across OS platforms, and customizing the SIM for each HA to maximize performance under each OS. HAs 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 for devices at initialization, the XPT and the SIM(s)/HA(s) ascertains all installed devices for the supported inter-connects and constructs Equipment Data Table(s) for the supported protocols. XPT services are used by drivers and programs to access these tables.

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

XPT routing is a mechanism to support concurrent multiple co-resident SIMs so that different HAs 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 device into the OS through the XPT, independent of the installed HA hardware.

This is a list of requirements that CAM-3 is designed to meet:

1. The ability to communicate (e.g., send commands) with any logical device on a supported inter-connect;
2. No restrictions on the size or format of transferred data;
3. Allow all the capabilities of high end host adapters to be fully utilized and accommodating HAs which do most of the protocol processing on board;
4. Allow the calling peripheral driver or program to interpret event information returned by logical devices (e.g., sense data returned by SCSI devices);
5. Fully re-entrant code;
6. Support of multiple HAs;
7. Peripheral drivers may ascertain which optional features are available;
8. Define a single XPT-based device scanning algorithm (so that each peripheral driver need not use host and inter-connect bandwidth to perform this function;
9. Provide a mechanism to abort I/O requests (at request of peripheral driver);
10. Ability to issue multiple I/O requests from one or more peripheral drivers to a single device;
11. Providing peripheral drivers with a mechanism for allocating a event data area and for specifying the number of event bytes to be automatically returned (e.g., the number of sense bytes returned as a response to a SCSI CHECK CONDITION status).

7. CAM-3 Data Type and Structure Size Definitions

To allow easier transportability (not binary compatibility) of CAM-3 peripheral drivers, and SIM/HA's between different machine platforms and operating systems. This international standard defines data types and storage classes. The XPT supplier shall adhere to the data definitions defined by this international standard.

The supplier of the XPT shall also the supply all structure definitions mandated by this international standard. The structure definitions shall use the defined data types and CAM boundary rules as specified by CAM-3. The term “CAM_boundary” is a defined term and refers to the address alignment of a data type within a structure and the structure itself. (refer to the Definitions Clause X.X and Structure Member CAM_boundary rules Clause X.X for further details).

CAM_boundaries (address boundaries) of the CAM-3 defined structures and its members is specified by CAM-3. It shall be the responsibility of the XPT supplier to preserve those CAM_boundaries as specified by the CAM_boundary rules. The CAM_boundary rules allows CCB structures and members to be aligned to a known offset for a 16, 32, 64, etc. bit processors regardless of the platform or O.S. defined pointer size.

7.1 Data and structure declarations

The XPT vendor shall provide a single file called cam_definitions.h that shall provide the mechanisms for peripheral drivers or SIMs to obtain data declarations. This file shall contain the declarations or point to other data declaration files that define the data structures, data types and bit definitions defined by this international standard supported by the XPT.

7.2 Data Type Sizes

CAM-3 defines the storage sizes and whether the storage class is signed or unsigned for the structures it defines. The supplier of the XPT shall define the storage class as follows:

- CAM_U8:

An unsigned a 8 bit quantity

- CAM_S8:
  A signed a 8 bit quantity
- CAM_U16:
  An unsigned a 16 bit quantity.
- CAM_S16:
  A signed a 16 bit quantity.
- CAM_U32:
  An unsigned a 32 bit quantity.
- CAM_S32:
  A signed a 32 bit quantity.
- CAM_VM_OFFSET:
  A virtual address pointer within the operating system.
- CAM_PM_OFFSET:
  A physical address pointer within the host.
- CAM_VOID_OFFSET:
  A virtual address pointer within the operating system or a physical address pointer within the host.
- CAM_IO_HANDLE:
  A virtual address pointer or physical address pointer within an I/O buses address space (e.g., a register address of a HA on a PCI bus). The obtaining this data type and values is operating system dependent.
- CAM_VOID:
  A void or nothing.

Pointer sizes within CAM-3 structures shall be a power of 2 of 32 bits and shall be the size that the O.S. defines for its pointer size rounded up to a power of 2 of 32 bits. The following is an example of pointer size rules:

OS Pointer Size	Cam-3 Structure Storage Size
1 to 32 bits	32 bits
33 to 64 bits	64 bits
65 to 128 bits	128 bits

Table 1 Operating System Pointer Storage Sizes

7.3 Structure Member CAM Boundary Rules

The defined CAM-3 data types and pointers declared within a defined CAM-3 structure shall be naturally aligned to their addressed boundary.

The XPT supplier shall ensure the CAM_boundary by padding the structure so that the member aligns with its address boundary. If the next defined member data type does not naturally align then the XPT supplier shall ensure this CAM_boundary by padding the structure so that the member naturally aligns with it address boundary.

All structures and structures within structures shall be aligned to the naturally aligned pointer boundary.

All arrays within a structure shall be aligned to the naturally aligned pointer boundary.

8. The XPT Model

Due to the planned inclusion of device protocols other than SCSI into CAM and evolution of certain protocols (SCSI) the XPT's Equipment Data Table has changed to support the SCSI protocols and other device protocols. The following describes the methods, mechanisms and some of the requirements that the XPT vendor shall employ for compliance to this standard. Further information and requirements are specified in the protocol specific Clauses of this standard.

A CAM-3 XPT shall provide all the functionality as defined in CAM-1. This requirement allows peripheral drivers and SIMs that conform to the CAM-1 specification to operate with a CAM-3 XTP. The requirement shall be withdrawn in the subsequent version of CAM. It is recommended that both peripheral drivers and SIMs migrate to CAM-3 compliance.

8.1 The Equipment Data Tables (EDTs)

The Equipment Data Tables are a collection of data elements that describe logical devices for the device protocols. There shall be one Equipment Data Table for each protocol supported by the XPT. The XPT reports the list of device protocols supported in the data returned in the PATH INQUIRY function directed to the XPT.

The EDT for each protocol shall not be visible (e.g., the organization and structure is vendor unique) to the peripheral drivers or the SIMs but information contained within an EDT shall be presented through XPT services. The construction and organization of the EDT(s) is vendor unique but the required information contained within an EDT and functionality shall be maintained.

Each EDT contains Logical Identifiers (LIDs) that are assigned to devices for that protocol when seen (e.g., responds protocol specific command) by XPT. Each protocol specific EDT shall be capable of storing a 32 bit (CAM_U32) LID and each protocol specific EDT may contain LIDs from 0h to FFFFFFFFh. An example of this is the XPT supports 2 defined protocols (SCSI and ATA), one EDT contains LIDs 0h to 100h that describe logical devices for its protocol and the other EDT contains LIDs 0 to 47h.

The Logical Identifier for a device shall be associated with Connection_ID(s) (physical address(es)) so that the Connection_ID(s) can be obtained by peripheral drivers for routing of CAM-3 CCBs to a SIM/HA. A logical device may be uniquely identified either through a protocol specific response (e.g., SCSI-3 World Wide Identifier), the nature of the protocol or other information obtained. An EDT shall be required to store an unique identifier if the protocol specific Clause specifies the obtaining of an unique id and the EDT may optionally store an unique identifier if the XPT vendor provides mechanisms for a protocol or version of the protocol that does not provide unique identifiers. If a logical device can be uniquely identified for a protocol then there shall only be one LID assigned for that logical device for that host but there may be multiple Connection_IDs for the device. An example of this is a SCSI-3 device on a shared SCSI bus (dual initiator HA1 and HA2) for a single host. The device can be seen from both HAs and can be uniquely identified by the SCSI-3 World Wide Identifier and its Logical Unit Number there would be one LID assigned to the device but two physical addresses associated to the LID (e.g., HA1s Port_ID/Target Id/Logical Unit and HA2s Port_ID/Target Id/Logical Unit).

The association of LID with Connection_IDs shall be unique within an EDT. This shall mean that a LID shall not have associated to it more than one Connection_ID having the same values.

The information contained within the EDT(s) may either be persistent across boots of a host or may be constructed at boot time with information when a assigned Port_ID is has a Topology Discover process initiated on it. Once a LID is assigned to a logical device for a protocol, the XPT shall not delete that LID from the EDT unless directed through mechanisms which are vendor unique (e.g., human intervention directs removal). This shall mean the following:

- If the EDT(s) LID and specified persistent information is persistent across boots of a host, the XPT shall not remove the LID assignment and persistent information unless directed to.

- If the EDT(s) information is not persistent across boots then once a LID has been assigned for this boot, the XPT shall not remove the LID assignment and persistent information unless directed to.

Clause 8.2 specifies the rules for the EDT data information persistence and modification for a LID.

The following information elements shall be associated with a LID in an EDT:
- Unique logical device identifiers:
  - Protocol specific logical device identifier, if the protocol supports an unique identifier for a logical device (e.g., SCSI). If the supported version of protocol for the logical device does not provide an unique identifier this information element shall be NULL.
  - Vendor unique logical device identifier, if the XPT vendor supports a mechanism to uniquely identify a logical device for a protocol in absence of a protocol specific way of uniquely identifying a device. An example of this is a SCSI-2 device can be uniquely identified by the Inquiry response data of the vendor id and product id of the device plus serial number of the device and the Logical Unit Number (LUN). The storage and encapsulation of this information is vendor unique.
- Device type identifier:
  - This shall be the protocol specific device type (e.g., SCSI Inquiry response data of peripheral device type). See protocol specific Clauses for further information.
- Connection_ID Information:
  - This shall be the assigned Port_ID for the logical device and the protocol specific address specifiers (e.g., SCSI target and Logical Unit identifiers). See protocol specific Clauses for further information.

8.2 EDT Information Data Persistence and Modification Rules

This Clause specifies the rules that shall be maintained for information contained within an EDT. There may be other requirements specified by the protocol specific Clauses, refer to those Clauses for further information.

There are two distinct cases of data persistence for a XPT vendor, these are across host boots information persistence and at boot time information persistence. The XPT vendor shall support shall only support one case of data persistence.

The information passed to build or update the EDTs is contained within an XPT structure, refer to Clause XXX for information on this data type.

8.2.1 EDT Information Data Persistence Across Boots

At the boot of a host the XPT shall determine in a vendor unique manner if persistent information storage can be obtained for the protocol specific EDTs. An example of persistent information storage is that the information pertaining to a protocol specific EDT may reside on the boot device. The XTP vendor would read the information in off the boot device and construct EDTs based on the protocol. This obtaining of any persistent storage information for the EDTs shall be done before any SIM registers with the XPT.

The XPT vendor shall provide the mechanisms to retain/obtain the following information in a vendor unique manner across boot of the host.
- LID assignment, if a LID has been assigned to a logical device;
- Any unique identifier associated to an assign LID;
- Protocol specific device type identifier (e.g., SCSI disk);
- If a unique identifier is not associated to a LID, the Connection_ID associated to the LID;

The XPT vendor may also retain/obtain the Connection_ID(s) for uniquely identified devices and the associated LID(s) or the XPT vendor may create the Connection_IDs when the device is seen at

Port_ID(s).

At boot, the XPT shall issue a CCB Scan function after the successful registration of a Port_ID through the xpt_bus_register3() service. Upon the successful completion of the CCB Scan function for that Port_ID, the XPT shall issue CCB Device Query functions to determine the Port_ID's configuration (e.g., the physical addresses of devices). For each device obtained by the CCB Device Query functions the XPT shall issue an protocol specific CCB Execute I/O function to identify the device (see protocol specific Clauses for further information).

If the device can be uniquely identified (e.g., SCSI-3 WWID or vendor unique method) the XPT shall do the following:

– Search the unique identifiers in the EDT representing this protocol;
  - If no match is found (e.g., no other unique id matches this unique id);
    - ⇒ Assign a LID value for this device and associate the Connection_ID to the assigned LID;
    - ⇒ Associate the unique id to the assigned LID;
    - ⇒ Set protocol specific device type value for the assigned LID;
    - ⇒ Set the Device Seen Flag representing this Connection_ID;
  - If a match is found (e.g., an unique id matches this unique id);
    - ⇒ Ensure that no other Connection_ID associated to this LID has the same value as this Connection_IDs Port_ID;
    - ⇒ If a match is found on Port_IDs;
      - ◊ Update the EDT Connection_ID having the match with new values for this Connection_ID;
    - ⇒ Compare protocol specific device type value for the LID to the device type value contained within the passed XPT_TRANSLATION structure;
      - ◊ If values do not match, update device type associated to this LID;

If the device can not be uniquely identified the XPT shall do the following:

– Search the Connection_IDs in the EDT representing this protocol;
  - If no match is found (e.g., no other Connection_ID matches this Connection_ID);
    - ⇒ Assign a LID value for this device and associate the Connection_ID to the assigned LID;
    - ⇒ Set an indication that there is no unique id associated to this LID;
    - ⇒ Set protocol specific device type value for the assigned LID;
    - ⇒ Set the Device Seen Flag representing this Connection_ID;
  - If a match is found (e.g., an Connection_ID matches this Connection_ID);
    - ⇒ Compare protocol specific device type value for the LID to the device type value contained within the passed XPT_TRANSLATION structure;
      - ◊ If values do not match, update device type associated to this LID;

Note 1
A device that can not be uniquely identified will only have one Connection_ID associated to a LID

After the boot process (e.g., at run time) upon the successful completion of the CCB Scan function for a Port_ID, the XPT shall issue CCB Device Query functions to determine the Port_ID's configuration (e.g., the physical addresses of devices). For each device obtained by the CCB Device Query functions the XPT shall issue an protocol specific CCB Execute I/O function to identify the device (see protocol specific Clauses for further information).

If the device can be uniquely identified (e.g., SCSI-3 WWID or vendor unique method) the XPT shall do the following:

– Search the unique identifiers in the EDT representing this protocol;

- If no match is found (e.g., no other unique id matches this unique id);
  - ⇒ Assign a LID value for this device and associate the Connection_ID to the assigned LID;
  - ⇒ Associate the unique id to the assigned LID;
  - ⇒ Set protocol specific device type value for the assigned LID;
  - ⇒ Set the Device Seen Flag representing this Connection_ID;
  - ⇒ Perform as specified in Clause XX an Asynchronous Event Callback for new logical devices found during scan;
- If a match is found (e.g., an unique id matches this unique id);
  - ⇒ Ensure that no other Connection_ID associated to this LID has the same values as this Connection_ID;
    - ◊ Compare the address specifiers;
      - – If not equal;
        - Update the EDT Connection_ID having the match with new values for this Connection_ID;
        - Perform as specified in Clause XX an Asynchronous Event Callback for address change of a logical device found during scan;
    - ◊ Compare protocol specific device type value for the LID to the device type value contained within the passed XPT_TRANSLATION structure;
      - – If values do not match;
        - Update device type associated to this LID;
      - – Perform as specified in Clause XX an Asynchronous Event Callback for device type change of a logical device found during scan;

If the device can not be uniquely identified the XPT shall do the following:

- – Search the Connection_IDs in the EDT representing this protocol;
  - If no match is found (e.g., no other Connection_ID matches this Connection_ID);
    - ⇒ Assign a LID value for this device and associate the Connection_ID to the assigned LID;
    - ⇒ Set an indication that there is no unique id associated to this LID;
    - ⇒ Set protocol specific device type value for the assigned LID;
    - ⇒ Set the Device Seen Flag representing this Connection_ID;
    - ⇒ Perform as specified in Clause XX an Asynchronous Event Callback for new logical devices found during scan;
  - If a match is found (e.g., an Connection_ID matches this Connection_ID);
    - ⇒ Compare protocol specific device type value for the LID to the device type value contained within the passed XPT_TRANSLATION structure;
      - ◊ If values do not match;
        - – Update device type associated to this LID;
        - – Perform as specified in Clause XX an Asynchronous Event Callback for device type change of a logical device found during scan;

8.2.2 EDT Information Data Boot Time Information Persistence

At the boot of a host the XPT shall either construct the EDTs for all the protocols supported or construct a EDT for a protocol when a SIM registers for that protocol.

At boot, the XPT shall issue a CCB Scan function after the successful registration of a Port_ID through the xpt_bus_register3() service. Upon the successful completion of the CCB Scan function for that Port_ID, the XPT shall issue CCB Device Query functions to determine the Port_ID's configuration (e.g., the physical addresses of devices). For each device obtained by the CCB Device Query functions the XPT shall issue an protocol specific CCB Execute I/O function to identify the device (see protocol specific Clauses for further information).

If the device can be uniquely identified (e.g., SCSI-3 WWID or vendor unique method) the XPT shall do the following:

- Search the unique identifiers in the EDT representing this protocol;
  - If no match is found (e.g., no other unique id matches this unique id);
    - ⇒ Assign a LID value for this device and associate the Connection_ID to the assigned LID;
    - ⇒ Associate the unique id to the assigned LID;
    - ⇒ Set protocol specific device type value for the assigned LID;
    - ⇒ Set the Device Seen Flag representing this Connection_ID;

  - If a match is found (e.g., an unique id matches this unique id);
    - ⇒ Ensure that no other Connection_ID associated to this LID has the same values as this Connection_ID;
      - ◊ If a match is found;
        - Update the EDT Connection_ID having the match with new values for this Connection_ID;
      - ◊ Compare protocol specific device type value for the LID to the device type value contained within the passed XPT_TRANSLATION structure;
        - If values do not match, update device type associated to this LID;

If the device can not be uniquely identified the XPT shall do the following:

- Search the Connection_IDs in the EDT representing this protocol;
  - If no match is found (e.g., no other Connection_ID matches this Connection_ID);
    - ⇒ Assign a LID value for this device and associate the Connection_ID to the assigned LID;
    - ⇒ Set an indication that there is no unique id associated to this LID;
    - ⇒ Set protocol specific device type value for the assigned LID;
    - ⇒ Set the Device Seen Flag representing this Connection_ID;
  - If a match is found (e.g., an Connection_ID matches this Connection_ID);
    - ⇒ Compare protocol specific device type value for the LID to the device type value contained within the passed XPT_TRANSLATION structure;
      - ◊ If values do not match, update device type associated to this LID;

Note 2

A device that can not be uniquely identified will only have one Connection_ID associated to a LID.

After the boot process (e.g., at run time) upon the successful completion of the CCB Scan function for a Port_ID, the XPT shall issue CCB Device Query functions to determine the Port_ID's configuration (e.g., the physical addresses of devices). For each device obtained by the CCB Device Query functions the XPT shall issue an protocol specific CCB Execute I/O function to identify the device (see protocol specific Clauses for further information).

If the device can be uniquely identified (e.g., SCSI-3 WWID or vendor unique method) the XPT shall do the following:

- Search the unique identifiers in the EDT representing this protocol;
  - If no match is found (e.g., no other unique id matches this unique id);
    - ⇒ Assign a LID value for this device and associate the Connection_ID to the assigned LID;
    - ⇒ Associate the unique id to the assigned LID;
    - ⇒ Set protocol specific device type value for the assigned LID;
    - ⇒ Set the Device Seen Flag representing this Connection_ID;
    - ⇒ Perform as specified in Clause XX an Asynchronous Event Callback for new logical devices found during scan;

- If a match is found (e.g., an unique id matches this unique id);
  - ⇒ Ensure that no other Connection_ID associated to this LID has the same value as this Connection_IDs Port_ID;
    - ◊ If a match is found on Port_IDs;
      - Compare the address specifiers;
        - If not equal;
          - ⇒ Update the EDT Connection_ID having the match with new values for this Connection_ID;
          - ⇒ Perform as specified in Clause XX an Asynchronous Event Callback for address change of a logical device found during scan;
    - ◊ Compare protocol specific device type value for the LID to the device type value contained within the passed XPT_TRANSLATION structure;
      - If values do not match;
        - Update device type associated to this LID;
        - Perform as specified in Clause XX an Asynchronous Event Callback for device type change of a logical device found during scan;

If the device can not be uniquely identified the XPT shall do the following:

- Search the Connection_IDs in the EDT representing this protocol;
  - If no match is found (e.g., no other Connection_ID matches this Connection_ID);
    - ⇒ Assign a LID value for this device and associate the Connection_ID to the assigned LID;
    - ⇒ Set an indication that there is no unique id associated to this LID;
    - ⇒ Set protocol specific device type value for the assigned LID;
    - ⇒ Set the Device Seen Flag representing this Connection_ID;
    - ⇒ Perform as specified in Clause XX an Asynchronous Event Callback for new logical devices found during scan;
  - If a match is found (e.g., an Connection_ID matches this Connection_ID);
    - ⇒ Compare protocol specific device type value for the LID to the device type value contained within the passed XPT_TRANSLATION structure;
      - ◊ If values do not match;
        - Update device type associated to this LID;
        - Perform as specified in Clause XX an Asynchronous Event Callback for device type change of a logical device found during scan;

9. XPT Transport Functionality

This Clause describes the set of services (functions) the XPT shall provide and may optionally provide to the peripheral drivers and the SIM/HAs. The supplier of the XPT (usually the O.S. provider) shall provide the XPT services listed as mandatory and may optionally provide other XPT services listed as optional.

The implementation of an XPT service is in a vendor unique manner but shall conform to the following:

- syntax of the XPT service call
- return value of the XPT service call
- behavior specified for the XPT service

The supplier of the XPT may implement any service as function call or as a macro.

The intention of the mandatory XPT services is to provide a common set of functionality for both the peripheral drivers and the SIMs/HAs that allows for easier transportability across operating systems and platforms.

9.1 CAM-3 Locking

The XTP shall supply in a vendor unique manner locking constructs to provide the peripheral drivers and SIMs synchronization mechanisms (e.g., prevents concurrent updating of a data structure by multiple threads of execution). The XPT shall provide five types locking levels in a descending hierarchy order. The lock structure contents and semantics shall be opaque to the peripheral drivers and SIMs. The locks may be an interrupt priority level type lock (e.g., non symmetrical multi-processor platform, where the processor priority level is raised allow synchronization) or a lock data structure (e.g., symmetrical multi-processor platform).

The lock structure definitions shall be defined by the XPT supplier and the XPT locks shall be defined as follows:

- XPT_LOCK_LV1
- XPT_LOCK_LV2
- XPT_LOCK_LV3
- XPT_LOCK_LV4
- XPT_LOCK_LV5

Peripheral drivers and SIMs shall provide the storage for any defined XPT lock they use. An example of this a SIM writer has determined that they need two XPT locking levels. One for a general overall structure lock and one for a register access lock. The SIM writer may declare the example data structure as follows:

```
typedef struct mysim_data
{
        CAM_U32            My_flags;              /* Generic state flags */
        CAM_U32            My_port_number;        /* My assigned port number */
        CAM_VM_OFFSET         My_que_ptr;         /* Pointer to my queue structures */
        CAM_U8             My_num_targets;        /* Number of targets supported */
        CAM_U8             My_num_luns;           /* Number of Logical Units supported */
        XPT_LOCK_LV1          My_sim_lck;         /* My data structure lock */
        XPT_LOCK_LV2          My_reg_lck;         /* register access lock */
} MYSIM_DATA;
```

The XPT shall provide the following services:

- xpt_lock_init()
- xpt_lock()
- xpt_unlock()

Refer to Clause XX for the syntax and descriptions of these services.

9.2 CAM Locking Rules

The locking services that the XPT shall provide have specific rules which the peripheral drivers and SIMs shall conform to. These rules allow for a consistent bounded locking model which if implemented properly by the peripheral driver and SIM writers prevent lock contention deadlock. The following are the rules that the XPT, CAM peripheral drivers and SIMs shall adhere to:

- No lock(s) shall be held when calling a XPT service or operating system service unless an explicit statement specifies that a lock shall be held. This shall mean that no functional module (e.g., XPT, CAM peripheral driver or SIM) will hold any lock when calling another service/routine outside their functional module boundary.

- If a lock is held by thread of execution no attempt shall be made to take the same physical lock.

- No locks shall be held by a thread of execution when calling xpt_thread_block() service.

- The hierarchy of XPT lock levels is specified and the peripheral drivers and SIMs shall adhere to the following:
    - The lock hierarchy level is descending from XPT_LOCK_LV1 to XPT_LOCK_LV5.
    - No attempt shall be made to acquire a higher level lock if lower level lock is held by the same thread of execution (e.g., XPT_LOCK_LV3 is held by a thread of execution, it is not allowed to acquire a XPT_LOCK_LV2 lock, but it is permissible to attempt to acquire the XPT_LOCK_LV5 lock).

- The order of releasing of acquired XPT locks is specified. The order of releasing acquired locks is in a LIFO order (e.g., last lock acquired is the first released). An example of this is the following:
    - Acquire level 2 lock;
    - Acquire level 4 lock;
    - Acquire level 5 lock;
    - Release level 5 lock;
    - Release level 4 lock;
    - Release level 2 lock;

9.3 XPT CAM-3 Mandatory Services

The CAM-3 XPT shall provide the mandatory services detailed in this International standard and may provide the optional services.

9.3.1 XPT Translation Services

The XPT translation services provides the services into the Equipment Data Tables to obtain assigned LIDs, addressing information, and unique identifiers. Valid pieces of information may be translated into its other components (e.g., protocol and Connection_ID to a Logical Identifier).

9.3.1.1 Structures Used with XTP Translation Services

9.3.1.1.1 The Translation Structure

The supplier of the XPT shall define the translation structure as follows:

```
typedef struct translation
    {
    CAM_U32        protocol_type;           /* The protocol number - SCSI, NETWORK, ATA, etc. */
    CAM_U32        logical_id;              /* The assigned Logical ID of the device. */
    struct connection_id connection_id;     /* Structure contains port_id and address specifiers. */
    CAM_U32        num_connect_ids;         /* Number of paths to the device (e.g., number of paths the
                                               device was seen on). */
    CAM_U32        dev_type;                /* Protocol specific device type (e.g., SCSI-3 Inquiry byte 0
                                                bits 0-5) */
    CAM_U32        id_lenght;               /* The length of the unique identifier */
    CAM_U32        pd_reg_num;              /* Peripheral drivers registration number */
    CAM_VOID_OFFSET   pd_specific;          /* A peripheral drivers vendor unique specific pointer */
    }TRANSLATION;
```

9.3.1.1.2 Member Descriptions of the Translation Structure

- protocol_type;
  This member is the protocol number of the described device (e.g., SCSI, NETWORK). This member shall always be set by the caller of the service with a valid protocol number.
- logical_id;
  This member is the XPT assigned logical identifier of the device for the protocol specified.
- connection_id:
  This member is a structure containing the address specifiers for a device. See Clause 9.3.1.1.3 for further information on the members of this data structure. The members of the connection_id data structure is as follows:
  - port_id;
    This member is the XPT assigned port number of the described device. This member is set when translating a specific device's protocol address to a Logical_ID. The member may be set by the caller of a XPT translation service.
  - addr_spec1;
    This member is an array of CAM_U32s that shall contain the first component the of a protocol specific address for a specific device. The format of the array is as follows:
    - ⇒ addr_spec1[0] shall contain the least significant portion of a protocol specific address (e.g. the least significant portion of a SCSI-3 target address or lower 32 bits).
    - ⇒ addr_spec1[1] shall contain the most significant portion of a protocol specific address (e.g. the most significant portion of a SCSI-3 target address or upper 32 bits)

    If the protocol uses up to sixty four (64) bits to address a device on a specific Port_ID then addr_spec1[0] and addr_spec1[1] shall represent the devices physical address for the specified Port_ID. See protocol specific addressing for further information. The member may be set by the caller of a XPT translation service.
  - addr_spec2;
    If a protocol uses two (2) specific and distinct components to address a device then addr_spec2 member array shall contain the second address component (e.g., SCSI Logical Unit address). This member is an array of CAM_U32s that shall contain the second half the of a protocol specific address for a specific device. The format of the array is as follows:
    - ⇒ addr_spec2[0] shall contain the least significant portion of a protocol specific address (e.g. the least significant portion of a SCSI-3 Logical Unit address or lower 32 bits).
    - ⇒ addr_spec2[1] shall contain the most significant portion of a protocol specific address (e.g. the most significant portion of a SCSI-3 Logical Unit address or upper 32 bits).

    If the protocol up to sixty four (64) bits for the second address component to address a device on a specific Port_ID then addr_spec2[0] and addr_spec2[1] shall represent the devices second physical address component for the specified Port_ID. See protocol specific addressing for further information. The member may be set by the caller of a XPT translation service.
- num_connect_ids;
  This member shall be the number of paths to the device. It is representative of the number of physical ports the device has been seen on. If the device has not been seen the field shall be set to 0 by the XPT translation service. The member shall be set by the XPT translation service.
- dev_type;
  This member shall be the protocol specific device type of the device (e.g., SCSI-3 Inquiry byte 0 bits 0 - 5). The member shall be set by the XPT translation service.
- id_lenght;
  This member shall be the storage size of the unique identifier for the represented in CAM_U8s types. This member shall include the NULL terminated value if the XTP stores the unique identifiers in ASCII strings. The member shall be set by the XPT translation service.
- pd_reg_num;
  The member shall be the peripheral driver's registration number if set to a value other then zero. The caller of a XTP translation service shall set the member to zero or to its acquired peripheral driver registration number.

- pd_specific;

  This member shall by the XPT translation service to either a NULL or a void pointer under the following conditions. Refer to the specific XPT translation service for further information.
  - If the pd_reg_num member is zero then pd_specific shall be set to NULL.
  - If the pd_reg_num member is non zero and a peripheral driver specific structure has been allocated for the specified device and driver. The XPT translation service shall set pd_specific to the pointer to the allocated peripheral driver specific structure for the described peripheral driver (pd_reg_num).
  - If the pd_reg_num member is non zero and a peripheral driver specific structure has not been allocated for the specified device and driver. The XPT translation service shall set pd_specific to NULL.

9.3.1.1.3 The Connections Structure

The connections structure describes a protocol specific Connection_ID(s) (path) to a device. The may be multiple Connection_IDs to a specific device. The actual members meanings are protocol specific and are described in the protocol specific section of this International standard.

The xpt_get_connect() service may return properly set connection structure(s) relative to the described device. Refer to xpt_get_connections() for further information.

The supplier of the XPT shall define the XTP connections as follows:

```
typedef struct connection_id
        {
        CAM_U32       conn_flags;          /* The last known state of this connection id */
        CAM_U32       port_id;             /* A registered SIM/HA port number */
        CAM_U32       addr_spec1[2];       /*Array of 4 CAM_U32s to contain the first half of a
                                             Protocol specific address (i.e., SCSI-3 target
                                             identifier) */
        CAM_U32       addr_spec2[2];       /* Array of 4 CAM_U32s to contain the second half of a
                                             Protocol specific address (i.e., SCSI-3 LUN identifier)
                                             */
        }CONNECTION_ID;
```

```
typedef struct connections
        {
        CAM_U32           protocol_type;     /* The protocol number - SCSI, NETWORK, etc.*/
        CAM_U32           logical_id;        /* The assigned Logical ID of the device. *?
        CAM_U32           num_alloc_c_id;    /* Number of allocated Connect_IDs (caller) */
        CAM_U32           num_ret_c_id;      /* Number of valid (returned) Connection_IDs */
        CONNECTION_ID  *c_id;                /* Pointer to array of CONNECT_ID structures */
        } CONNECTIONS;
```

9.3.1.1.4 Member Descriptions of the Connections Structure

The CONNECTION_ID structure members are set by the xpt_get_connections() service. See xpt_get_connections() for further information.

The CONNECTION_ID structure members:

- conn_flags;

  This member represents the last known state of this connection identifier. The member is a defined set of flags that represents the state of the device relative to a Port_ID. The conn_flags member shall have the following meanings and be defined as follows:
  - The state of the connection is valid when all the flags in the conn_flags member are set to zero.

The port_id, addr_spec1 and addr_spec2 members are valid based upon the last Topology Discovery process done for the identified Port_ID. This shall mean that the CONNECTION_ID can be used with a high degree of confidence to send commands to the identified device.

```
#define CONN_VALID          0x00000000
```

- Connection invalid port_id member valid flag specifies that at one time a valid connection id existed for the device at the identified Port_ID (port_id member). The last Topology Discovery process done for the identified Port_ID determined that the device did not respond (exists). The flag shall denote that the port_id member has a valid Port_ID.

```
#define CONN_INVAL_PID      0x00000001
```

– port_id;
This member is the XPT assigned port number of the described device.

– addr_spec1;
This member is an array of CAM_U32s that shall contain the first component the of a protocol specific address for a specific device. The format of the array is as follows:

- addr_spec1[0] shall contain the least significant portion of a protocol specific address (e.g. the least significant portion of a SCSI-3 target address or lower 32 bits).
- addr_spec1[1] shall contain the most significant portion of a protocol specific address (e.g. the most significant portion of a SCSI-3 target address or upper 32 bits).

If the protocol uses up to sixty four (64) bits to address a device on a specific Port_ID then addr_spec1[0] and addr_spec1[1] shall represent the devices physical address for the specified Port_ID. See protocol specific addressing for further information. The member may be set by the caller of a XPT translation service.

– addr_spec2;
If a protocol uses two (2) specific and distinct components to address a device then addr_spec2 member array shall contain the second address component (e.g., SCSI Logical Unit address). This member is an array of CAM_U32s that shall contain the second half the of a protocol specific address for a specific device. The format of the array is as follows:

- addr_spec2[0] shall contain the least significant portion of a protocol specific address (e.g. the least significant portion of a SCSI-3 Logical Unit address or lower 32 bits).
- addr_spec2[1] shall contain the most significant portion of a protocol specific address (e.g. the most significant portion of a SCSI-3 Logical Unit address or upper 32 bits).

If the protocol uses up to sixty four (64) bits for the second address component to address a device on a specific Port_ID then addr_spec2[0] and addr_spec2[1] shall represent the devices second physical address component for the specified Port_ID. See protocol specific addressing for further information. The member may be set by the caller of a XPT translation service.

The CONNECTIONS structure members:

– protocol_type;
This member is the protocol number of the described device (e.g., SCSI, NETWORK). This member shall always be set by the caller of the service with a valid protocol number.

– logical_id;
This member is the XPT assigned logical identifier of the device for the protocol specified.

– num_alloc_c_id;
This member represents number of allocated CONNECTION_ID structures in an array that has been allocated. The array of CONNECTION_ID structures is represented as a pointer in the c_id member.

– num_ret_c_id;
This member represents the number of valid CONNECTION_ID structures in the array that have been set. This member is only valid upon the successful return of a XPT translation service (xpt_get_connections()).

– c_id;

This member points to an array of CONNECTION_ID structures that the caller has allocated.

9.3.1.2 xpt_get_logical_id(TRANSLATION *trans)

Data type returned CAM_U32;

The xpt_get_logcal_id() service translates a Connection_ID (physical address) for a device into a Logical_ID based upon the specified protocol type. The service shall set the pointed to members of the TRANSLATION structure, as specified below, if translation is successful. The service shall return a zero if the translation is successful.

The service shall return an error indication if for any reason the service can not translate to a valid Logical_ID.. The error conditions and return values are specified below.

Note 3

This service is useful when a peripheral driver has the physical address of a device (protocol_type, Port_ID, and the physical address specifiers) and wishes to obtain the assigned Logical_ID of the device. The caller may also use the service to determine if a device ,as specified, has been seen by the XPT (Logical_ID assigned).

Arguments:

- trans;
  Shall be a pointer a properly formatted TRNASLATION structure as specific below.

The caller of the xpt_get_logical() service shall set the following members of the TRANSLATION structure as specified:

- protocol_type;
  Shall contain a valid protocol type (e.g., SCSI).
- Connection_ID;
  - port_id;
    Shall contain a valid XPT assigned Port_ID.
  - addr_spec1;
    May contain a valid value based upon the specified protocol_type specified. This member is the first component of a device’s physical address (e.g., SCSI target specifier). See protocol specific addressing for further information
  - addr_spec2;
    May contain a valid value based upon the specified protocol_type specified. This member is the second component of a device’s physical address (e.g., SCSI Logical Unit specifier). See protocol specific addressing for further information.
- pd_reg_num;
  The member shall be the peripheral driver’s registration number if set to a value other then zero. The caller of a XTP translation service shall set the member to zero or to its acquired peripheral driver registration number.

Upon the services successful translation for the specified device. The service shall set the following pointed members of the TRANSLATION structure as specified.

- logical_id;
  Shall contain a valid XPT assigned logical identifier for the device as specified by the protocol type and Connection Identifier.
- num_connect_ids;
  This member shall be the number of paths to the device. It is representative of the number of physical ports the device has been seen on. If the device has not been seen the field shall be set to 0 by the service.
- dev_type;
  This member shall be set by the service and shall be the protocol specific device type of the device (e.g., SCSI-3 Inquiry byte 0 bits 0 - 5).
- id_length;

This member shall be set to either zero (0) or the storage size of the unique identifier for the represented in CAM_U8s types. This representative number shall include the NULL terminated value if the XTP stores the unique identifiers in ASCII strings. The value of zero (0) shall be set when the XPT can not uniquely identify the device.

- pd_specific;
  This member shall by the XPT translation service to either a NULL or a void pointer under the following conditions
  - If the pd_reg_num member is zero then pd_specific shall be set to NULL.
  - If the pd_reg_num member is non zero and a peripheral driver specific structure has been allocated for the specified device and driver. The XPT translation service shall set pd_specific to the pointer to the allocated peripheral driver specific structure for the described peripheral driver (pd_reg_num).
  - If the pd_reg_num member is non zero and a peripheral driver specific structure has not been allocated for the specified device and driver. The XPT translation service shall set pd_specific to NULL.

The service shall return an error indication (a non-zero value) when the service can not translate the passed parameters into a Logical_ID. When the service can not translate, the service shall not set any member of the pointed to TRANSLATION structure. The following are the conditions in which the service may return an error indication.

- The specified protocol type is not a valid or supported protocol type.
- The specified Connection_ID does not have a device associated to a logical identifier.

Values returned;

- A zero (0) value shall indicate that the service successfully translated the protocol type and Connect_ID to a Logical_ID.
- A one (1) value shall indicate that the specified protocol type is not a valid or supported protocol type.
- A two (2) shall indicate the specified Connection_ID does not have a device associated to a logical identifier.

9.3.1.3 xpt_get_phys_attrib(TRANSLATION *trans)

Data type returned CAM_U32;

The xpt_get_phys_attrib() (xpt get physical attributes) service translates a Logical_ID (logical identifier) for a device into the physical attributes for the device based upon the specified protocol type. The service shall set the pointed to members of the TRANSLATION structure, as specified below, if translation is successful. The service shall return a zero if the translation is successful.

The service shall return an error indication if for any reason the service can not translate the Logical_ID to a valid physical attributes. The error conditions and return values are specified below.

Arguments:

- trans;
  Shall be a pointer a properly formatted TRNASLATION structure as specific below.

  The caller of the xpt_get_logical() service shall set the following members of the TRANSLATION structure as specified:
  - protocol_type;
    Shall contain a valid protocol type (e.g., SCSI).
  - logical_id;
    Shall contain a Logical_ID for the specified protocol.
  - pd_reg_num;

The member shall be the peripheral driver's registration number if set to a value other then zero. The caller of a XTP translation service shall set the member to zero or to its acquired peripheral driver registration number.

Upon the services successful translation for the specified device. The service shall set the following pointed members of the TRANSLATION structure as specified.

If the XPT has only one Connection_ID for the specified Logical_ID. The XPT shall the num_connect_id (number of connection identifiers) member to a one and shall set Connection_ID as follows:

- Connection_ID;
  - port_id;
    Shall contain a valid XPT assigned Port_ID.
  - addr_spec1;
    May contain a valid value based upon the specified protocol_type specified. This member is the first component of a device's physical address (e.g., SCSI target specifier). See protocol specific addressing for further information
  - addr_spec2;
    May contain a valid value based upon the specified protocol_type specified. This member is the second component of a device's physical address (e.g., SCSI Logical Unit specifier). See protocol specific addressing for further information.

If the XPT has more then one Connection_ID for the specified Logical_ID. The XPT shall set the num_connect_id (number of connection identifiers) member to that number of Connection_IDs and shall not set Connection_ID members. The caller after a successful return from the service may call the xpt_get_connections() service to obtain the Connection_IDs for the device.

- num_connect_id;
  This member shall be the number of paths to the device. It is representative of the number of physical ports the device has been seen on. If the device has not been seen the field shall be set to 0 by the service.
- dev_type;
  This member shall be set by the service and shall be the protocol specific device type of the device (e.g., SCSI-3 Inquiry byte 0 bits 0 - 5).
- id_length;
  This member shall be set to either zero (0) or the storage size of the unique identifier for the represented in CAM_U8s types. This representative number shall include the NULL terminated value if the XTP stores the unique identifiers in ASCII strings. The value of zero (0) shall be set when the XPT can not uniquely identify the device.
- pd_specific;
  This member shall by the XPT translation service to either a NULL or a void pointer under the following conditions
  - If the pd_reg_num member is zero then pd_specific shall be set to NULL.
  - If the pd_reg_num member is non zero and a peripheral driver specific structure has been allocated for the specified device and driver. The XPT translation service shall set pd_specific to the pointer to the allocated peripheral driver specific structure for the described peripheral driver (pd_reg_num).
  - If the pd_reg_num member is non zero and a peripheral driver specific structure has not been allocated for the specified device and driver. The XPT translation service shall set pd_specific to NULL.

The service shall return an error indication (a non-zero value) when the service can not translate the passed parameters into the devices physical attributes. When the service can not translate, the service shall not set any member of the pointed to TRANSLATION structure. The following are the conditions in which the service may return an error indication.

- The specified protocol type is not a valid or supported protocol type.

- The specified Logical_ID does not have a device associated to it (e.g. Logical_ID not assigned).

Values returned;
- A zero (0) value shall indicate that the service successfully translated the protocol type and Logical_ID into the devices physical attributes..
- A one (1) value shall indicate that the specified protocol type is not a valid or supported protocol type.
- A two (2) shall indicate the specified Logical_ID does not have a device associated to it.

9.3.1.4 xpt_get_connections(CONNECTION *connect)

Data type returned CAM_U32;

The xpt_get_connections() service sets into the pointed members of the CONNECTION structure the physical connections to a specified device. This service is used to obtain a list of Connection_ID(s).

Arguments;
- connect;
  Shall be a pointer to a properly formatted CONNECTION structure as specific below:
  - protocol_type;
    This member is the protocol number of the described device (e.g., SCSI, NETWORK). This member shall always be set by the caller of the service with a valid protocol number.
  - logical_id;
    This member is the valid XPT assigned logical identifier of the device for the protocol specified.
  - num_alloc_c_id;
    The caller shall set this member to number of allocated CONNECTION_ID structures in an array that has been allocated. The array of CONNECTION_ID structures is represented as a pointer in the c_id member.
  - c_id;
    The caller shall set this member to the beginning address of the array of CONNECTION_ID structures that the caller has allocated.

Upon the services successful translation for the specified device. The service shall set the following pointed members of the CONNECTION structure as specified.
- num_ret_c_id;
  This member represents the number of valid CONNECTION_ID structures that have been set into the array of CONNECTION_ID structures by the service.

For each valid Connect_ID that the XPT has stored for the specified device. The service shall set into the each CONNECTION_ID structure into the pointed to array the following:
- port_id;
  This member is the XPT assigned port number of the described device.

- addr_spec1;
  Shall contain a valid value based upon the specified protocol_type specified. This member is the first component of a device's physical address (e.g., SCSI target specifier). See protocol specific addressing for further information
- addr_spec2;
  May contain a valid value based upon the specified protocol_type specified. This member is the second component of a device's physical address (e.g., SCSI Logical Unit specifier). See protocol specific addressing for further information.

The service shall set each CONNECTION structure in linear order. The first CONNECTION structure set shall be the first structure in the array.

Values returned;

- A zero (0) value shall indicate that the service successfully translated the protocol type and Logical_ID into the devices Connection_IDs
- A one (1) value shall indicate that the specified protocol type is not a valid or supported protocol type.
- A two (2) shall indicate the specified Logical_ID does not have a device associated to it.

9.3.2 XPT Lock Services

9.3.2.1 xpt_lock_init(CAM_VOID_OFFSET lock, CAM_U8 lock_level)

Data type returned none

The xpt_lock_init() service shall provide the means for the caller to initialize a declared CAM-3 lock structure. There shall be only one call to the xpt_lock_init() service per declared CAM-3 lock structure.

The service shall initialize the lock for subsequent use as defined by this international standard and as needed by the operating system type.

Arguments:

- lock;
  Shall be a pointer to an allocated XPT lock structure. The pointer shall point to a XPT lock structure having one of the following types:
  - XPT_LOCK_LV1
  - XPT_LOCK_LV2
  - XPT_LOCK_LV3
  - XPT_LOCK_LV4
  - XPT_LOCK_LV5

- lock_level;
  Shall be one of the following defined values the corresponds to the XPT lock type;
  - #define XPT_LV1 0x01
  - #define XPT_LV2 0x02
  - #define XPT_LV3 0x03
  - #define XPT_LV4 0x04
  - #define XPT_LV5 0x05

  If the lock argument is a pointer to a XPT_LOCK_LV1 then the lock_level argument shall be XPT_LV1.

  If the lock argument is a pointer to a XPT_LOCK_LV2 then the lock_level argument shall be XPT_LV2.

  If the lock argument is a pointer to a XPT_LOCK_LV3 then the lock_level argument shall be XPT_LV3.

  If the lock argument is a pointer to a XPT_LOCK_LV4 then the lock_level argument shall be XPT_LV4.

  If the lock argument is a pointer to a XPT_LOCK_LV5 then the lock_level argument shall be XPT_LV5.

Values returned:
- None

9.3.2.2 xpt_lock(CAM_VOID_OFFSET lock)

Data type returned none

The xpt_lock() service shall provide the means for the caller to perform a synchronization lock for access to data structures.

There shall be no calls to this service if the pointed to XPT lock structure has not been initialized by the xpt_lock_init() service.

Arguments:
- lock;
  Shall be a pointer to an allocated XPT lock structure. The pointer shall point to a XPT lock structure having one of the following types:
  - XPT_LOCK_LV1
  - XPT_LOCK_LV2
  - XPT_LOCK_LV3
  - XPT_LOCK_LV4
  - XPT_LOCK_LV5

Values returned:
- None

9.3.2.3 xpt_unlock(CAM_VOID_OFFSET lock)

Data type returned none

The xpt_unlock() service shall provide the means for the caller to remove a synchronization lock for access to data structures.

Arguments:
- lock;
  Shall be a pointer to an allocated XPT lock structure. The pointer shall point to a XPT lock structure having one of the following types:
  - XPT_LOCK_LV1
  - XPT_LOCK_LV2
  - XPT_LOCK_LV3
  - XPT_LOCK_LV4
  - XPT_LOCK_LV5

Values returned:
- None

9.3.3 XPT Generic Services

9.3.3.1 xpt_isr()

Data type returned CAM_U8;

The xpt_isr() service shall provide a means for components of the CAM-3 subsystem to determine if the CAM-3 component is in interrupt service context.

Values returned:

- 0X00:
  Indicates that the caller is not in interrupt context.
- 0X01:
  Indicates that the caller is in interrupt context.

9.3.3.2 xpt_alloc_pd_specific(TRANSLATION *trans, CAM_VOID (*spec_init()), CAM_U32 size)

Data type returned CAM_U32;

The xpt_alloc_pd_specific() (allocate peripheral driver structure associated to the LID) service translates the protocol_type ,a Logical_ID (logical identifier) and the peripheral driver registration number to allocate peripheral driver specific structure. The service shall allocate the request size in bytes and associate it to the peripheral driver registration number, the protocol type (e.g., SCSI) and the logical identifier (Logical_ID) for a device.

The service shall provide the following:

- Ensure that there is a device associated to the passed Logical_ID. If no association return the specified error indication.

- Ensure that no other xpt_alloc_pd_specific() or xpt_get_pd_specific() thread of execution will execute for described device and peripheral driver registration number.

- Ensure that there is no other association for the peripheral driver registration number, the protocol type (e.g., SCSI) and the logical identifier (Logical_ID) for a device. If an association is found the service shall set into shall set pd_specific member the pointer to the allocated peripheral driver specific structure for the described peripheral driver (pd_reg_num) and return a success indication.

- Allocate the required memory as specified by the size argument. If memory can not be obtained return the specified error indication.

- Associate the allocated memory to the described device and peripheral driver registration number.

- Set into set pd_specific member the pointer to the allocated peripheral driver specific structure

- Call the peripheral driver's initialization routine passing the pointer to the passed TRANSLATION structure (trans)..

The peripheral drivers initialization routine shall be designed as follows:

- Having 1 argument passed with the type of TRANSLATION *.
- Having no return value (CAM_VOID).
- Shall not acquire (take or hold) any locks.

The peripheral drivers initialization should cast the pd_specific member to its own structure declaration. The peripheral driver should also initialize all defined locks, acquire other memory requirements and initialize all queue headers.

Arguments:

- trans;
  Shall be a pointer a properly formatted TRANSLATION structure as specific below.

  The caller of the xpt_alloc_pd_specific() service shall set the following members of the TRANSLATION structure as specified:
  - protocol_type;
    Shall contain a valid protocol type (e.g., SCSI).
  - logical_id;
    Shall contain a Logical_ID for the specified protocol.
  - pd_reg_num;
    The member shall be the peripheral driver's registration number.

- (*spec_init());
  The peripheral drivers device specific structure initialization routine.

- size;
  The size in bytes of the peripheral driver's specific structure to be allocated (e.g. sizeof(struct mydriver_structure).

Values returned;
- A zero (0) value shall indicate that the service successfully allocated and associated the referenced peripheral driver structure. The pointer to that allocated structure is in the TRANSLATION structure in the
- A one (1) value shall indicate that the specified protocol type is not a valid or supported protocol type.
- A two (2) shall indicate the specified Logical_ID does not have a device associated to it.
- A three (3) shall indicated memory could not be allocated.

9.3.3.3 xpt_get_pd_specific(TRANSLATION *trans)

Data type returned CAM_U32;

The xpt_get_pd_specific() (get a peripheral driver structure associated the LID) service translates the protocol_type ,a Logical_ID (logical identifier) and the peripheral driver registration number to obtain the already allocated peripheral driver specific structure. The association shall be between the peripheral driver registration number, the protocol type (e.g., SCSI) and the logical identifier (Logical_ID) for a device.

The service shall look up the logical device (logical_id) for the protocol type specified and find the peripheral drivers structure as referenced by the peripheral drivers registration number (pd_reg_num).

If the XTP does not support the protocol, the logical identifier is not found or there is no peripheral driver structure stored for this driver. The service shall return an error indication as specified.

Arguments:
- trans;
  Shall be a pointer a properly formatted TRNASLATION structure as specific below.

  The caller of the xpt_get_pd_specific() service shall set the following members of the TRANSLATION structure as specified:
  - protocol_type;
    Shall contain a valid protocol type (e.g., SCSI).

  - logical_id;
    Shall contain a Logical_ID for the specified protocol.
  - pd_reg_num;
    The member shall be the peripheral driver's registration number.

Upon the services successful look up for the specified peripheral driver (e.g., found the peripheral driver structure). The service shall set the following pointed members of the TRANSLATION structure:

- pd_specific;
  The XPT translation service shall set pd_specific member with the pointer to the allocated peripheral driver specific structure for the described peripheral driver (pd_reg_num).

Values returned;

- A zero (0) value shall indicate that the service successfully found the referenced peripheral driver structure.
- A one (1) value shall indicate that the specified protocol type is not a valid or supported protocol type.
- A two (2) shall indicate the specified Logical_ID does not have a device associated to it.
- A three (3) shall indicated that the referenced peripheral driver structure could not be found. A xpt_alloc_pd_specific() is suggested.

9.3.3.4 xpt_dealloc_pd_specific(TRANSLATION *trans)

Data type returned CAM_U32;

The xpt_dealloc_pd_specific() (deallocate peripheral driver structure associated to the LID) service translates the protocol_type ,a Logical_ID (logical identifier) and the peripheral driver registration number to deallocate peripheral driver specific structure. The service shall deallocate the memory associated to the peripheral driver registration number, the protocol type (e.g., SCSI) and the logical identifier (Logical_ID) for a device.

The service shall provide the following:

- Ensure that there is a device associated to the passed Logical_ID. If no association return the specified error indication.

- Ensure that no other xpt_alloc_pd_specific() or xpt_get_pd_specific() thread of execution will execute for described device and peripheral driver registration number.

- Disassociate the allocated memory to the described device and peripheral driver registration number.

- Set into set pd_specific member a NULL.

- Free the memory.

Arguments:

- trans;
  Shall be a pointer a properly formatted TRNASLATION structure as specific below.

  The caller of the xpt_get_logical() service shall set the following members of the TRANSLATION structure as specified:
  - protocol_type;
    Shall contain a valid protocol type (e.g., SCSI).
  - logical_id;
    Shall contain a Logical_ID for the specified protocol.

- pd_reg_num;
  The member shall be the peripheral driver's registration number.

Values returned;

- A zero (0) value shall indicate that the service successfully deallocated and disassociated the referenced peripheral driver structure
- A one (1) value shall indicate that the specified protocol type is not a valid or supported protocol type.
- A two (2) shall indicate the specified Logical_ID does not have a device associated to it.

9.3.3.5 xpt_mem_alloc(CAM_U32 size, CAM_U32 flags)

Data type returned CAM_VM_OFFSET *;

The xpt_mem_alloc() service shall provide a means for the peripheral drivers, SIMs and the XPT to allocate memory for buffers and data structures. The xpt_mem_alloc() service shall allocate at least the specified size in CAM_U8s (e.g., bytes) of memory, if memory can be allocated at this time. The caller of the service shall not call the service with the XPT_WAITOK flag set to a one when in interrupt service context. Interrupt context shall be verified and the XPT_WAITOK flag set according before requesting the xpt_mem_alloc() service.

Arguments:

- CAM_U32 size:
  Specifies the size of memory in CAM_U8s (e.g., bytes) to allocate.

- CAM_U32 flags:
  XPT_WAITOK:
  The flag set to a one specifies that if the requested size can not be allocated at this time the service may block (e.g., suspend the execution of the caller) waiting for memory resources, if the service can not provide the memory resource.

  The flag set to a zero (0) specifies that if the requested size can not be allocated at this time the service shall not block.

  Note 4

  The service may not provide synchronization of requests for callers that are blocked (e.g., when resources become available the first caller that was blocked does not necessarily get the resource). If synchronization is needed by the caller of this service it is up to the caller to provide the mechanisms needed.

  XPT_BZERO:
  The flag set to a one specifies that if the requested memory has been allocated for the caller, before return to the caller, the service shall set each CAM_U8 (e.g., byte) of the allocated memory to a zero (0).

Values returned:

- Non NULL value:
  Shall indicate that the request was successful and the value is a pointer to the memory allocated.
- NULL value:
  Shall indicate that the request failed.

Example:

```
struct xyz *xyz_ptr;

    xyz_ptr = (struct xyz *)xpt_mem_alloc(  sizeof(struct xyz), (XPT_WAITOK | XPT_BZERO));
```

9.3.3.6 xpt_mem_free((CAM_VM_OFFSET)addr)

Data type returned CAM_VOID

The xpt_mem_free() service shall provide a means for peripheral drivers, SIMs and the XPT to free (e.g., return to the O.S.) memory buffers allocated by the xpt_mem_alloc() service.

Arguments:

– CAM_VM_OFFSET addr:
  Shall specify the memory pointer that points to the allocated memory to be freed (e.g. returned to the O.S.). The argument addr * shall contain a pointer to memory that was previously allocated in a call to the xpt_mem_alloc() service. There shall be only one call to the xpt_mem_free() service for each memory buffer allocated by the xpt_mem_alloc() service.

Example:

```
struct xyz *xyz_ptr;

    xyz_ptr = (struct xyz *)xpt_mem_alloc( sizeof(struct xyz), (XPT_WAITOK | XPT_BZERO));

    if ( xyz_ptr != (struct xyz *)NULL){
        xpt_mem_free( (CAM_VM_OFFSET )xyz_ptr);
    }
```

9.3.3.7 xpt_ccb_alloc3(CAM_U32 flags)

Data type returned CCB_HEADER3 *;

The xpt_ccb_alloc3() service shall provide the means for the peripheral drivers, SIMs and the XPT to allocate CAM-3 CCBs for use. It shall be the responsibility of the XPT to ensure that the pointer of the allocated CAM-3 CCB returned from the xpt_ccb_alloc3() service call shall point to a memory buffer large enough to contain any of the possible XPT/SIM function request CAM-3 CCBs. The XPT shall also ensure that the SIM and peripheral drivers working set pointers are initialized to memory buffers large enough to contain any registered peripheral driver or SIM and the working set buffers are zeroed. The xpt_ccb_alloc3() service shall return a NULL pointer if memory resources are not immediately available.

The returned CAM-3 CCB shall be properly initialized for use as a I/O request CCB. The allocated CAM-3 CCB may be used (i.e., sent to the XPT), multiple times. If the CAM-3 CCB is sent the XPT/SIM more then one time after initial allocation from the xpt_ccb_alloc3() service, the requester shall set the following fields to the following values:
(Editors Mark - fill in fields.)

Once the CCB is no longer needed for the XPT/SIM function request, the CAM-3 CCB shall be returned using the xpt_ccb_free3() service.

Arguments:

– CAM_U32 flags:
  XPT_WAITOK:
  The flag set to a one specifies that if the CAM-3 CCB can not be allocated at this time the service may block (e.g., suspend the execution of the caller) waiting for resources, if the service can not provide the CCB. The caller of the service shall not call the service with the XPT_WAITOK flag set to a one when in interrupt service context.

  The flag set to a zero (0) specifies that if the CAM-3 CCB can not be allocated at this time the service shall not block. The xpt_ccb_alloc3() service shall return a NULL pointer if a CAM-3 CCB is not available for allocation.

Note 5

The service may not provide synchronization of requests for callers that are blocked (e.g., when resources become available the first caller that was blocked does not necessarily get the resource first). If synchronization is needed by the caller of this service it is up to the caller to provide the mechanisms needed.

Values returned:

- Non NULL *:
  Indicates that the service has allocated a CAM-3 CCB for the caller and the value is the pointer to the CCB_HEADER3.
- NULL *:
  Indicates that the service could not allocate a CAM-3 CCB at this time.

9.3.3.8 xpt_ccb_free3(CCB_HEADER3 *ccb_header3)

Data type returned CAM_VOID.

The xpt_ccb_free3() service shall provide the means for the peripheral drivers, SIMs and the XPT to free CAM-3 CCBs after use. The xpt_ccb_free3() service takes a pointer to the CAM-3 CCB_HEADER3 that the caller has finished with so it can be returned to the CAM subsystem.

Arguments:

- CCB_HEADER3 *ccb_header3 :
  Pointer to the CAM-3 CCB_HEADER3 to be freed to the CAM subsystem.

Values returned:

- None;

9.3.3.9 xpt_action3(CCB_HEADER3 *ccb_header3)

Data type returned CAM_U32;

All CAM-3 CCB requests to the XPT or a SIM/HA are placed through the xpt_action3() service call. The CAM Status information for callback on completion CCBs shall be obtained at the callback point via the CAM status fields. The CAM Status information for non callback on completion - non immediate CCBs shall be obtained by polling the CAM status field for a non Request in Progress status. The CAM Status information for immediate CCBs shall be obtained on return from the service call by examining the CAM Status field.

Arguments:

- ccb_header3:
  Pointer to a CCB_HEADER3 that shall be transported/passed to a SIM/HA or the XPT.

Values returned:

- For Immediate CAM-3 CCBs:
  Any Valid CAM Status
- For Queued CAM-3 CCBs
  Request In Progress - Indicates that the CCB has been accepted. Any other valid CAM Status - Indicates that the CCB has not been accepted

The ultimate status of any CAM-3 CCB shall be obtained from the CAM status field of the CCB.

9.3.3.10 xpt_callback(CCB_HEADER3 *ccb)

Data type returned CAM_VOID.

The xpt_callback() service shall not be used by peripheral drivers. The service shall only be used by the XPT and SIMs.

The XPT or SIMs shall not call this service with a pointer to a CAM-3 CCB that does not specify Callback on completion.

The XPT or SIM shall set all appropriate queued CCB fields marked as specified by the CCB function and shall provide autoevent information (e.g., autosense data for SCSI inter-connects) as specified in this standard before calling the xpt_callback() service. See Clause XX (autoevent) and the appropriate CCB tables in Clauses XX, XX, XX, and XX for further information.

The xpt_callback() service routine may queue the CCB for later callback or callback the peripheral immediately. The queued CCB includes a pointer to the peripheral driver's callback routine (in the Callback on Completion field).

The xpt_callback() service shall pass the address of the CAM-3 CCB when calling a peripheral driver's callback routine for a completed CAM-3 CCB.

Arguments:

- ccb;
  Pointer to a CCB_HEADER3.

Values returned:

- None

9.3.3.11 xpt_virt_to_phys(CAM_VM_OFFSET addr, CAM_MAP *cam_map)

Data type returned CAM_PM_OFFSET;

The xpt_virt_to_phys() service shall convert the passed virtual address (addr) argument to its associated physical address and return that address. The cam_map argument is O.S. dependent pointer (see O.S. dependent clauses for further details). The caller shall pass a NULL cam_map value if the CAM_MAP * is not available to the caller (e.g., a kernel virtual I/O transfer may not have an associated map). The caller of the xpt_virt_to_phys() service shall ensure that if a non NULL cam_map value is passed, that the CAM_MAP * is associated with the passed virtual address (addr).

Arguments:

- addr :
  Pointer of the virtual address that shall be converted to a physical address.
- cam_map :
  Pointer to the O.S. dependent map structure if available. Refer to the O.S. specific clauses for more information.

Values returned:

- All values:
  The value is the associated physical address.

Example:

```
routine ( CCB_SCSIIO *ccb)
{
    CAM_PM_OFFSET p_addr;
    p_addr = xpt_virt_to_phys((CAM_VM_OFFSET *)ccb->cam_data_ptr, ccb->cam_req_map);
}
```

9.3.3.12 xpt_page_size(CAM_VM_OFFSET addr, CAM_MAP *map)

Data type returned CAM_U32;

The xpt_page_size() service shall return the virtual page size that is associated with the passed virtual address (addr). The cam_map argument is an O.S. dependent argument (see O.S. dependent clauses for further details). The caller shall pass a NULL cam_map value if the CAM_MAP * is not available to the caller (e.g., usually a kernel virtual I/O transfer does have an associated map). The caller of the xpt_page_size() service shall ensure that if a non NULL CAM_MAP *value is passed, that CAM_MAP * is associated with the passed virtual address (addr).

Arguments:
– addr:
Pointer of the virtual address that shall be converted to a physical address.
– cam_map:
Pointer to the O.S. dependent map structure if available. Refer to the O.S. specific clauses for more information.

Values returned:
– All values:
The value is the page size for the passed virtual address.

9.3.3.13 xpt_pdrv_reg(CAM_S8 *name, CAM_U32 working_set_size)

Data type returned CAM_U32

The xpt_pdrv_reg() service shall provide the means for peripheral drivers and SIM/Has to register with the XPT. The xpt_pdrv_reg() service shall return a unique number (the driver registration number) for a peripheral driver that calls this service. The argument name shall point to a NULL terminated string that represents the peripheral drivers name (e.g., “cam_xyz_disk_driver”). The argument working_set_size shall represent the number of bytes that this peripheral driver needs for its working set size.

The service shall ensure that the drivers name has not been already assigned a peripheral driver registration number. This should be accomplished by comparing the peripheral drivers name to stored peripheral driver names which have assigned driver registration number. If a match is found the returned value of this service shall be the already assigned value.

The service shall ensure that the all CCBs allocated after the return of this service call shall have the largest number of bytes for a peripheral working set.

Note 6

An example of this is that 3 peripheral drivers have registered with the XPT. Peripheral driver 1 has requested 30 bytes, peripheral driver 2 has requested 60 bytes, and peripheral driver 3 has requested 65 bytes. The XPT will supply CCBs with peripheral driver working set of at least 65 bytes after peripheral driver 3 has registered.

A peripheral driver shall ensure that it registers with the XPT only once at peripheral driver initialization. The assigned peripheral driver registration number shall be used by the peripheral driver when requesting an advisory lock on a logical device. The peripheral driver shall call the xpt_pdrv_unreg() service when it is unloaded.

Arguments:
– name:
Shall point to a NULL terminated string that represents the peripheral drivers name

- working_set_size:
  Number of bytes this peripheral driver needs for a peripheral driver working set size.

Values returned:
- A positive value:
  The value shall be the peripheral driver registration number for this peripheral driver.
- A zero (0)
  Shall indicate that the service has encountered an error and the peripheral driver is not registered.

9.3.3.14 xpt_pdrv_unreg(CAM_U32 pdrv_reg_num)

Data type returned CAM_VOID;

The xpt_pdrv_unreg() service shall provide the means for peripheral drivers to de-register with the XPT. The xpt_pdrv_unreg() service shall break all associations made when this peripheral driver registration number was assigned. This peripheral driver registration number shall be available for re-assignment upon return of this service.

Arguments:
- pdrv_reg_num:
  The peripheral driver registration number to disassociate with a peripheral driver (e.g., number available for re-assignment).

Values Returned:
- None

9.3.3.15 xpt_unit_lock_exclus(TRANS *trans, CAM_U32 pdrv_reg_num)

Data type returned CAM_U8;

The xpt_unit_lock_exclus() service shall provide an exclusive lock service for the peripheral drivers, to exclusively lock an identified logical device. The argument trans passed by the caller shall be a pointer to a TRANSLATION structure and the TRANSLATION structure shall have the following members set.
- protocol type
- logical_id

The argument pdrv_reg_num shall be the peripherals drivers XPT registration number for the caller requesting the lock.

Arguments:
- trans
  Pointer to a valid TRANSLATION structure.
- pdrv_reg_num:
  The callers peripheral drivers registration number obtained from the xpt_pdrv_reg() service.

The service shall behave as follows:
- Maintain a indication that an exclusive lock has been granted for the identified logical device.

- Ensure that when an exclusive lock is requested that there a no exclusive locks currently against the identified Logical unit that another registered peripheral driver holds.

- Ensure that the identified logical device is represented in the protocol specific Equipment Data Table as a valid logical device.

- The service shall store the peripheral driver registration for a granted lock against a logical device when an exclusive lock is granted.

The peripheral drivers should ensure that they only call the service once for a granted lock (e.g., when they are ready to operate with the identified logical device). The peripheral driver shall not operate with any Logical unit which it has not be granted an lock for.

Values returned:
Zero (0x0):

- The requested lock has been granted.

- A positive value:
  Bit 0 = 1:
  A exclusive lock has been requested and an exclusive lock already exists on the Logical Unit which another register peripheral driver holds.

  Bit 1 = 1:
  The identified logical device for the identified protocol does not exist in the Protocol Translation Table. A Scan logical device function is suggested.

9.3.3.16 xpt_unit_unlock_exclus(TRANSLATION *trans, CAM_U32 pdrv_reg_num)

Data type returned CAM_U8;

The xpt_unit_unlock() service shall provide an unlock service for the peripheral drivers to unlock a logical device. The argument trans passed by the caller shall be a pointer to a TRANSLATION structure and the TRANSLATION structure shall have the following members set.

- protocol type
- logical_id

Arguments:

- trans
  Pointer to a valid TRANSLATION structure.
- pdrv_reg_num:
  The callers peripheral drivers registration number obtained from the xpt_pdrv_reg() service.

The service shall behave as follows:

- Ensure that the lock being requested to be released is currently active against the logical device and is currently held by the calling peripheral driver.
  - Compare callers pdrv_reg_num argument to the stored value for the lock.

  - If the compare succeeds (e.g., pdrv_reg_num argument equals stored value), the service shall release the lock and all indications that a lock is held for the logical device and shall indicate success to the caller.

  - If the compare fails, the service shall not release the lock and shall indicate failure to the caller.

The peripheral drivers shall ensure that they only call the service once for the granted lock (e.g., when they are ready to no longer operate with the identified logical device).

Values returned:
- Zero (0x0):
  The requested lock has been release.
- A positive value (release of lock denied)
  Bit 0 = 1:
    A lock release has been requested and no lock exists on the identified device.

  Bit 1 = 1:
    A lock release has been requested and lock exists on the identified device but the registered peripheral driver does not hold the lock.

9.3.3.17 xpt_bcopy(CAM_VM_OFFSET src, CAM_VM_OFFSET dest, CAM_U32 len)

Data type returned CAM_VOID;

The xpt_bcopy() service shall copy n number of bytes from the src pointer to dest pointer. If either the src or dest argument is in user address space, the process that represents the user address space pointer shall be the currently mapped process.

Arguments:
- src:
  Specifies a pointer to a buffer (array of bytes). The pointer may reside in kernel address space or user address space.

- dest:
  Specifies a pointer to a buffer of at least n bytes. The pointer may reside in kernel address space or user address space.

- len:
  Specifies the number of bytes to be copied.

Values Returned:
- None

9.3.3.18 xpt_bzero(CAM_VM_OFFSET src, CAM_U32 len)

Date type returned CAM_VOID;

The xpt_bzero() service shall zero n bytes of memory beginning at the address specified by src.

Arguments:
- src:
  Specifies a pointer to a buffer of at least len bytes.

- len:
  Specifies the number of bytes to be zeroed.

Values returned:
- None

9.3.3.19 xpt_copy_to_phys(CAM_VM_OFFSET virt_src, CAM_PM_OFFSET phys_dest, CAM_U32 len)

Data type returned CAM_VOID;

The xpt_copy_to_phys() service shall copy the specified amount of virtually addressed memory to physically addressed memory. The addresses shall reside only in system memory space and not in the memory space of the I/O buses.

Arguments:
- virt_src:
  Specifies the virtual address of the data to be copied.

- phys_dest:
  Specifies the physical address to copy the data to.

- len:
  Specifies the number of bytes to copy.

Values returned:
- None

9.3.4 XPT Queue Services

The XPT defined queues shall be constructed of doubly linked lists. Each element is linked into the queue through a queue header. All queue headers shall be of the generic form XPT_QUEHEAD. A given element may have multiple queue headers. This allows each element to be simultaneously linked onto multiple queues.

The callers of the XPT Queues Service may call these services with acquired XPT lock levels. Any caller of the XPT queue services that manipulates queues should have acquired an XPT lock level before inserting or removing an element from a queue to prevent queue corruption.

The XPT_QUEHEAD structure shall be defined as follows:

```
typedef struct xpt_quehead
{
    struct xpt_quehead  *flink;          /* Forward pointer */
    struct xpt_quehead  *blink;          /* Backward pointer */
} XPT_QUEHEAD;
```

9.3.4.1 xpt_que_init(XPT_QUEHEAD *quehead)

Data type returned CAM_VOID;

The xpt_que_init() service shall initialize the specified XPT_QUEHEAD structure (e.g., argument quehead) so that both structure members shall point to the XPT_QUEHEAD structure.

Peripheral drivers and SIMs should call this service prior to calling the other XPT Queue Services to initialize the members of the XPT_QUEHEAD data structure.

Arguments:
- *quehead:
  Shall point to a XPT_QUEHEAD structure.

Values returned:
- None

9.3.4.2 xpt_insque(XPT_QUEHEAD *data_element, XPT_QUEHEAD *element_position)

Data type returned CAM_VOID;

The xpt_insque() service shall add the data element that the data_element argument specifies to the queue. The xpt_insque() service shall insert the data_element in the next position after the argument element_position in the queue.

Arguments:
- *data_element:
  Shall point to a XPT_QUEHEAD structure and shall be the data element inserted into the queue.

- *element_position:
  Shall point to a XPT_QUEHEAD structure and shall be the position of where the caller wishes to place the data_element.

Values returned:
- None

9.3.4.3 xpt_remque(XPT_QUEHEAD *data_element)

Data type returned CAM_VOID;

The xpt_remque() service shall remove the data element that the data_element argument specifies from the queue.

Arguments:
- *data_element:
  Shall point to a XPT_QUEHEAD structure and shall be the data element removed from the queue.

Values returned:
- None

9.3.4.4 xpt_insque_head(XPT_QUEHEAD *data_element, XPT_QUEHEAD *quehead)

Data type returned CAM_VOID;

The xpt_insque_head() service shall add the data element that the data_element argument specifies to the head of the queue.

Arguments:
- *data_element:
  Shall point to a XPT_QUEHEAD structure and shall be the data element inserted into the queue.

- *quehead:
  Shall point to the head of the queue as defined by the XPT_QUEHEAD structure.

Values returned:
- None

9.3.4.5 xpt_remque_head(XPT_QUEHEAD *quehead)

Data type returned XPT_QUEHEAD *;

The xpt_remque_head() service shall remove a data element from the head of the queue that the quehead argument specifies. The return of the service shall either be a pointer to a XPT_QUEHEAD structure within a data element removed from the queue or a XPT_QUEHEAD NULL pointer if no data elements are on the queue.

Arguments:
- quehead:
  Shall point to the head of the queue as defined by the XPT_QUEHEAD structure.

Values returned:
- A pointer to a XPT_QUEHEAD structure indicating an element has been removed.

- NULL pointer to a XPT_QUEHEAD structure indicating that the queue is empty.

9.3.4.6 xpt_insque_tail(XPT_QUEHEAD *data_element, XPT_QUEHEAD *quehead)

Data type returned CAM_VOID;

The xpt_insque_tail() service shall add the data element that the data_element argument specifies to the tail of the queue.

Arguments:
- *data_element:
  Shall point to a XPT_QUEHEAD structure and shall be the data element inserted into the queue.

- *quehead:
  Shall point to the head of the queue as defined by the XPT_QUEHEAD structure.

Values returned:
- None

9.3.4.7 xpt_remque_tail(XPT_QUEHEAD *quehead)

Data type returned XPT_QUEHEAD *;

The xpt_remque_tail() service shall remove a data element from the tail of the queue that the quehead argument specifies. The return of the service shall either be a pointer to a XPT_QUEHEAD structure within a data element removed from the queue or a XPT_QUEHEAD NULL pointer if no data elements are on the queue.

Arguments:
- quehead:
  Shall point to the head of the queue as defined by the XPT_QUEHEAD structure.

Values returned:
- A pointer to a XPT_QUEHEAD structure indicating an element has been removed.

- NULL pointer to a XPT_QUEHEAD structure indicating that the queue is empty.

9.3.5 XPT Synchronization services

The XPT synchronization services provide mechanisms that allow threads of execution to synchronize on events that shall or may happen at a time period in the future. The xpt_sleep() service and xpt_wakeup() services block and then wake up a threads of execution. An example of this is that a peripheral driver may call these services to wait for the completion of a CCB. The peripheral driver after sending a CCB to the XPT for transport to a SIM/HA may call xpt_sleep() with the address of the CCB. The callback on completion service that may be specified in a CCB may call the xpt_wakeup() service with the address of the CCB when called. It shall be the responsibility of the wakened process to check if the condition for which it was sleeping (blocked) has occurred.

9.3.5.1 xpt_sleep(CAM_VM_OFFSET channel)

Data type returned CAM_VOID;

The xpt_sleep() service shall put the calling thread of execution (process thread) to sleep (blocked) on the address specified by the channel argument. This address should be unique to prevent unexpected wake/sleep cycles, which can occur if a number of different threads of execution (process threads) are sleeping (blocked) on the same address.

The service shall not be called when in interrupt context.

Arguments:
- channel:
  Shall specify an address associated with the calling thread of execution to be put to sleep.

Values returned:
- None

9.3.5.2 xpt_wakeup(CAM_VM_OFFSET channel)

Data type returned CAM_VOID;

The xpt_wakeup() service shall schedule all threads of execution to run (e.g., placed into a runnable state) that are sleeping (blocked) on the address specified by the channel argument. the calling thread of execution (process thread) to sleep (blocked) on the address specified by the channel argument. It is possible that there are no threads of execution sleeping on the channel at the time the wakeup is issued. This situation can occur for a variety of reasons and shall not represent an error condition.

Arguments:
- channel:
  Shall specify the address on which the wakeup is to be issued.

Values returned;
- None

9.4 CAM-3 XPT Optional Services

The following services may be provided by the supplier of the XPT.

9.4.1 XPT DMA Services

The XPT DMA services shall provide the mechanisms that support direct memory access (DMA) of data transfers for HAs that are capable of DMA transfers. The XPT DMA services operate on a generic data type of XPT_DMA_HANDLE. The data type provides an operating system independent means of conveying bus addresses and byte counts to HAs for DMA transfers.

The XPT_DMA_HANDLE data type and its members shall only be used as reference (read) by a SIM. This shall mean that no SIM shall modify its contents or members directly, any modifications of the XPT_DMA_HANDLE shall be accomplished through calls to the services in Clause 9.4.1 . Once a XPT_DMA_HANDLE is allocated through a call to the xpt_dma_map_alloc(), it may be passed to other routines within the SIM and to other XPT DMA Services. Once a XPT_DMA_HANDLE is deallocated through a call to the xpt_dma_map_dealloc() service it shall not be referenced again until reallocated.

The XPT_DMA_HANDLE may be embedded in another structure that contains other data a needed by the OS or the XPT to keep track of resources and information on OSD data. This data shall be transparent to the callers of the XPT DMA services. An example of this is the following:

```
typedef struct osd_dma_context
{
    XPT_DMA_HANDLE     *osd_sgp;   /* Pointer to allocated resources */
    OSD_MAP       *osd_map;        /* Pointer to map struct that the *sgp resources came from */
    OSD_BUS       *osd_bus;        /* Pointer to an OSD struct that contains information on
                                      the bus (e.g., PCI bus) */
} OSD_DMA_CONTEXT;
```

9.4.1.1 The XPT_DMA_HANDLE Structure

```
typedef struct xpt_dma_handle
{
    XPT_DMA_SGLIST*xpt_sgp;          /* Pointer to an array of XPT_DMA_SGLIST that contains the
                                        bus address and byte counts */
        CAM_U32       xpt_val_ents;  /* Number of entries in the list (XPT_DMA_SGLIST */
        CAM_U32       xpt_num_ents;  /* Number of loaded entries in the list */
} XPT_DMA_HANDLE;
```

Member descriptions for the XPT_DMA_HANDLE structure:

- *xpt_sgp:
  This member shall point to an array of structures having a data type of XPT_DMA_SGLIST. The member shall be set by the xpt_dma_map_alloc() service if the call is successful.

- xpt_val_ents:
  This member shall contain the number of XPT_DMA_SGLIST data types in the array that the xpt_sgp member points to. The member shall be set by the xpt_dma_map_alloc() service if the call is successful.
- xpt_num_ents:
  This member shall contain the number of loaded XPT_DMA_SGLIST data types in the array that the xpt_sgp member points to. The member shall be set by the xpt_dma_map_load() and the xpt_dma_map_unload() services if the call(s) are successful.

9.4.1.2 The XPT_DMA_SGLIST Structure.

```
typedef struct xpt_dma_sglist {
    CAM_VOID_OFFSET   bus_addr;   /* Base address to start this DMA segment */
    CAM_U32        byte_count;    /* Byte count for this segments DMA */
}XPT_DMA_SGLIST;
```

9.4.1.3 xpt_dma_map_alloc(CAM_U32 byte_count, CAM_VM_OFFSET OSD, XPT_DMA_HANDLE *xpt_dma_handle, flags)

Data type returned CAM_U32;

The xpt_dma_map_alloc() service shall allocate the resources (mapping registers, I/O channels, and other hardware and software resources) for DMA data transfers. The size of the DMA data transfer is specified in the byte_count argument.

The xpt_dma_map_alloc() service returns to the dma_handle_p argument a handle to DMA resources associated with the mapping of an in-memory I/O buffer. SIM writers can view the DMA handle as the tag to the allocated system resources needed to perform a DMA operation.

The xpt_dma_map_alloc() service shall allocate only the necessary resources for a SIM/HA to perform a maximum transfer of size byte_count. The maximum transfer size is the size of the returned byte count if the returned byte count is not equal to byte_count. All SIM/HAs shall be prepared for a returned byte count that is less than byte_count. The reason for this is that system resources can have physical limits that may never satisfy an allocation request of size byte_count. To actually initialize and set up the resources, the SIM/HA shall make a call to the xpt_dma_map_load() service.

The xpt_dma_map_alloc() service shall not put the caller to sleep (block) and returns the value zero (0) if:

– The SIM writer sets the flags argument to XPT_DMA_SLEEP and the specified byte_count exceeds all available system resources (e.g. the system cannot provide the resources for a data transfer of size byte_count).

If the returned byte count does not equal byte_count, the device driver can perform one of the following tasks:

– A SIM/HA can partition the DMA data transfer into a byte_count that is less than or equal to the returned byte count and then perform a sequence of DMA data transfer operations until the transfer has completed.
– If the SIM/HA needs more resources associated with the specified byte_count than xpt_dma_map_alloc() can allocate, the SIM calls xpt_dma_map_dealloc() to release and deallocate these resources. The SIM then recalls xpt_dma_map_alloc() (possibly with the DMA_SLEEP flag set) until the necessary resources are available.

Arguments:

– byte_count:
  Specifies the maximum size (in bytes) of the data to be transferred during the DMA transfer operation. The XPT uses this size to determine the resources (mapping registers, I/O channels, and other software resources) to allocate.

– OSD:
  This is a pointer to an Operating System specific structure and is defined by the supplier of the XPT or the O.S.. The service uses the OSD pointer to obtain the O.S. dependent bus-specific interfaces and data structures that it needs to allocate the necessary mapping resources.

– *xpt_dma_handle:
  Specifies a pointer to a XPT_DMA_HANDLE to DMA resources associated with the mapping of an in-memory I/O buffer onto an I/O bus. The XPT_DMA_HANDLE provides the information to access bus address/byte count pairs. A bus address/byte count pair is represented by the ba and bc members of an xpt_sg_entry structure pointer. SIM writers can view the XPT_DMA_HANDLE as the tag to the allocated system resources needed to perform a DMA operation.

The SIM//HA passes an argument of type XPT_DMA_HANDLE * the xpt_dma_map_alloc() service returns to this variable the address of the DMA handle. The SIM uses this address in a call to xpt_dma_map_load.

- flags:
  XPT_DMA_SLEEP:
  Specifies that the caller requests that it be blocked if the resources requested are not currently available.

  XPT_DMA_CONTIG:
  The XPT_DMA_CONTIG flag is a request for contiguous memory space on an I/O bus for a virtually mapped buffer in system memory space that may be physically discontiguous. The call to the xpt_dma_map_alloc() or xpt_dma_map_load() services with the DMA_CONTIG flag shall not fail if a contiguous I/O address space cannot be used to map the memory buffer (e.g. if more than one dma_map_load interface).

  The XPT_DMA_CONTIG flag is useful for I/O devices whose DMA typically crosses one or more system pages, since system hardware scatter-gather resources can be set up and used to do scatter-gather mapping of a virtually contiguous, physically discontiguous I/O buffer during the calls to xpt_dma_map_alloc() or xpt_dma_map_load(). This DMA mapping makes a physically discontiguous memory buffer appear physically contiguous to an HA.

  Even if an HAs DMA engine has scatter-gather resources or support, DMA is typically faster if the system scatter-gather resources are used. This is due to the system's lower overhead to set up scatter-gather resources relative to an HA reading and processing multiple scatter-gather data structures.

  XPT_DMA_ALL:
  Specifies that xpt_dma_map_alloc() shall return a non zero value only if the system can satisfy a transfer size of byte_count. If the system cannot support a transfer of size byte_count (even if all DMA resources were made available), xpt_dma_map_alloc() shall not allocate any portion of the resources associated with the specified byte_count and returns a byte count of zero (0). The behavior of no allocation of resources unless xpt_dma_map_alloc() can allocate the resources needed to do an uninterruptible transfer of the requested size, avoids extra calls to xpt_dma_map_dealloc().

Values returned:

- A positive value:
  Upon successful completion, xpt_dma_map_alloc() returns a byte count (in bytes) that indicates the DMA transfer size it can map. A positive value returned not equal to the byte count request shall indicate that the resources have been obtain but not all that is needed.

- A zero value (0):
  Indicates a failure to obtain the needed resources as specified (e.g., flags or no resources).

9.4.1.4 xpt_dma_map_dealloc(XPT_DMA_HANDLE *xpt_dma_handle)

Data type returned CAM_U32;

The xpt_dma_map_dealloc() service shall release and deallocates the resources for DMA data transfers that were previously allocated in a call to xpt_dma_map_alloc() or xpt_dma_map_load() service.

Arguments:

- *xpt_dma_handle:
  Specifies a XPT_DMA_HANDLE pointer to DMA resources associated with the mapping of an in-memory I/O buffer onto a controller's I/O bus. The XPT_DMA_HANDLE provides the information to access bus address/byte count pairs. A bus address/byte count pair is represented by the ba and bc members of an sg_entry structure pointer. SIM/HA writers can view the XPT_DMA_HANDLE pointer as the tag to the allocated system resources needed to perform a DMA operation.

Values returned:
- A positive value:
  Upon successful completion, xpt_dma_map_dealloc() shall return a positive value.
- A zero (0) value:
  An error has occurred when releasing the resources.

9.4.1.5 xpt_dma_map_load(CAM_U32 byte_count, CAM_VM_OFFSET virtual_addr, CAM_MAP *cam_map, XPT_DMA_HANDLE *xpt_dma_handle, CAM_VM_OFFSET OSD)

Data type returned CAM_U32;

The xpt_dma_map_load service shall load and set the system resources necessary to perform a DMA transfer of size byte_count to the virtual address specified in the virt_addr argument. This virtual address must be valid in the context of the OSD CAM_MAP pointer. The OSD CAM_MAP pointer may be a NULL pointer if it is not associated with a user process request (e.g., the DMA request is associated with a kernel virtual buffer).

The SIM shall call the xpt_dma_map_alloc() service prior to calling the xpt_dma_map_load() service. The xpt_dma_map_load() service shall use the xpt_dma_handle argument (allocated in xpt_dma_map_alloc) to load and set the appropriate DMA mapping resources.

Arguments:
- byte_count:
  Specifies the maximum size (in bytes) of the data to be transferred during the DMA transfer operation. The XPT uses this size to determine the resources (mapping registers, I/O channels, and other software resources) to load and set.

- virtual_addr:
  Shall specify the virtual address where the DMA transfer occurs. The service uses this address with the OSD CAM_MAP pointer to obtain the physical addresses of the system memory pages to load into DMA mapping resources.

- cam_map:
  Specifies a OSD specific pointer to map structures associated with the valid context for the virtual address specified in virtual_addr. If the cam_map * is a null, the address is a kernel space address.

- *xpt_dma_handle:
  Specifies a pointer to a XPT_DMA_HANDLE to DMA resources associated with the mapping of an in-memory I/O buffer onto an I/O bus which was allocated in the call to the xpt_dma_map_alloc() service. The XPT_DMA_HANDLE provides the information to access bus address/byte count pairs. A bus address/byte count pair is represented by the ba and bc members of an xpt_sg_entry structure pointer. SIM writers can view the XPT_DMA_HANDLE as the tag to the allocated system resources needed to perform a DMA operation.

  The SIM//HA passes an argument of type XPT_DMA_HANDLE * the xpt_dma_map_alloc() service returns to this variable the address of the DMA handle. The SIM uses this address in a call to xpt_dma_map_load.

- OSD:
  This is a pointer to an Operating System specific structure and is defined by the supplier of the XPT or the O.S. The service uses the OSD pointer to obtain the O.S. dependent bus-specific interfaces and data structures that it needs to allocate the necessary mapping resources.

Values returned:

- A positive value:
  Upon successful completion, xpt_dma_map_load() returns a byte count that shall indicate the DMA transfer size. A positive value returned not equal to the byte count request shall indicate that the resources have been obtain but not all that is needed.
- A zero (0) value:
  Indicates a failure to obtain the needed resources as specified (e.g., flags or no resources).

9.4.1.6 xpt_dma_map_unload(XPT_DMA_HANDLE *dma_handle)

Data type returned CAM_U32;

The xpt_dma_map_unload() service shall unload (invalidates) the resources that were loaded and set up in a previous call to xpt_dma_map_load(). A call to xpt_dma_map_unload() shall not release or deallocate the resources that were allocated in a previous call to xpt_dma_map_alloc() service.

Arguments:

- *xpt_dma_handle:
  Specifies a pointer to a XPT_DMA_HANDLE to DMA resources associated with the mapping of an in-memory I/O buffer onto an I/O bus which was allocated in the call to the xpt_dma_map_alloc() service.

Values returned:

- A positive value:
  Upon successful completion, xpt_dma_map_unload shall return a positive value.
- A zero (0) value:
  An error has occurred when releasing the resources.

9.4.2 XPT SIM Services

The following services may be used by SIMs to allow movement of data and control information to Has.

9.4.2.1 xpt_io_copyin(CAM_IOHANDLE srcaddr, CAM_VM_OFFSET destaddr, CAM_U32 count)

Data type returned CAM_VOID;

The xpt_io_copyin() service shall copy data as specified by the count argument from bus address space (e.g., PCI bus) as specified by the srcaddr argument to kernel system memory as specified by the destaddr argument. The CAM_IOHANDLE srcaddr shall identify the location in bus address space where the copy of data shall originate and shall be physically contiguous as specified by the argument count. The destaddr argument shall not represent user process address space.

The xpt_io_copyin() service shall assume no alignment of data associated with srcaddr and destaddr.

Arguments:

- srcaddr:
  Shall specify a CAM_IOHANDLE which is operating system dependant (e.g., an I/O handle that you can use to reference a HA register or HA memory located in bus address space).

- destaddr:
  Shall specify the kernel virtual address where xpt_io_copyin() service copies the data to in-system memory.

- count
  Shall specify the number of CAM_U8s (bytes) to be copied.

Values returned
- None

9.4.2.2 xpt_io_copyout(CAM_VM_OFFSET srcaddr, CAM_IOHANDLE destaddr, CAM_U32 count)

Data type returned CAM_VOID;

The xpt_io_copyout() service shall copy data as specified by the count argument from kernel system memory as specified by the srcaddr argument to bus address space (e.g., PCI bus) specified by the destaddr argument. The CAM_IOHANDLE destaddr shall identify the location in bus address space where the copy of data shall be placed and shall be physically contiguous as specified by the argument count. The srcaddr argument shall not represent user process address space.

The xpt_io_copyout() service shall assume no alignment of data associated with srcaddr and destaddr.

Arguments:
- destaddr:
  Shall specify the kernel virtual address where xpt_io_copyin() service copies the data to in-system memory.

- srcaddr:
  Shall specify a CAM_IOHANDLE which is operating system dependant (e.g., an I/O handle that you can use to reference a HA register or HA memory located in bus address space.

- count:
  Shall specify the number of CAM_U8s (bytes) to be copied.

Values returned:
- None

9.4.2.3 xpt_readbus_d8(CAM_IOHANDLE hba_addr)

Data type returned CAM_U8;

The xpt_readbus_d8() service shall read a CAM_U8 (byte) from a HA device register located in the bus I/O address space as specified by argument hba_addr.

Arguments:
- hba_addr:
  Shall specify a CAM_IOHANDLE which is operating system dependant (e.g., an I/O handle that you can use to reference a HA register or HA memory located in bus address space).

Values returned:
– A CAM_U8 read from the HA.

9.4.2.4 xpt_readbus_d16(CAM_IOHANDLE hba_addr)

Data type returned CAM_U16;

The xpt_readbus_d16() service shall read a CAM_U16 (short) from a HA device register located in the bus I/O address space as specified by argument hba_addr. The hba_addr argument shall point to a naturally aligned address boundary of a CAM_U16 data type (see Clause 7.2 for further information).

Arguments:
– hba_addr:
  Shall specify a CAM_IOHANDLE which is operating system dependant (e.g., an I/O handle that you can use to reference a HA register or HA memory located in bus address space).

Values returned:
– A CAM_U16 read from the HA.

9.4.2.5 xpt_readbus_d32(CAM_IOHANDLE hba_addr)

Data type returned CAM_U32;

The xpt_readbus_d32() service shall read a CAM_U32 from a HA device register located in the bus I/O address space as specified by argument hba_addr. The hba_addr argument shall point to a naturally aligned address boundary of a CAM_U32 data type(see Clause 7.2 for further information).

Arguments:
– hba_addr:
  Shall specify a CAM_IOHANDLE which is operating system dependant (e.g., an I/O handle that you can use to reference a HA register or HA memory located in bus address space).

Values returned
– A CAM_U32 read from the HA.

9.4.2.6 xpt_readbus_d64()

Editors Mark - Do we want to define a xpt_readbus_d64().

9.4.2.7 xpt_writebus_d8(CAM_IOHANDLE hba_addr, CAM_U8 data)

Data type returned CAM_VOID;;

The xpt_writebus_d8() service shall write a CAM_U8 (byte) as specified by the data argument to a HA device register located in the bus I/O address space as specified by argument hba_addr.

Arguments:
– hba_addr:
  Shall specify a CAM_IOHANDLE which is operating system dependant (e.g., an I/O handle that you can use to reference a HA register or HA memory located in bus address space).
– data:
  Shall contain the data to be written.

Values returned:

- None

9.4.2.8 xpt_writebus_d16(CAM_IOHANDLE hba_addr, CAM_U16 data)

Data type returned CAM_VOID;

The xpt_write bus_d16() service shall write a CAM_U16 (short) as specified by the data argument to a HA device register located in the bus I/O address space as specified by argument hba_addr. The hba_addr argument shall point to a naturally aligned address boundary of a CAM_U16 data type (see Clause 7.2 for further information).

Arguments:
- hba_addr:
  Shall specify a CAM_IOHANDLE which is operating system dependant (e.g., an I/O handle that you can use to reference a HA register or HA memory located in bus address space).
- data:
  Shall contain the data to be written.

Values returned:
- None

9.4.2.9 xpt_writebus_d32(CAM_IOHANDLE hba_addr, CAM_U32 data)

Data type returned CAM_VOID;

The xpt_write bus_d32() service shall write a CAM_U32 as specified by the data argument to a HA device register located in the bus I/O address space as specified by argument hba_addr. The hba_addr argument shall point to a naturally aligned address boundary of a CAM_U32 data type (see Clause 7.2 for further information).

Arguments:
- hba_addr:
  Shall specify a CAM_IOHANDLE which is operating system dependant (e.g., an I/O handle that you can use to reference a HA register or HA memory located in bus address space).
- data:
  Shall contain the data to be written.

Values returned:
- None

9.4.2.10 xpt_writebus_d64(CAM_IOHANDLE hba_addr, CAM_U64 data)

Editors Mark - Do we want to define a xpt_writebus_d64().

10. Principles of operation

10.1 Accessing the XPT

The OS peripheral drivers access the XPT through software calls through a number of entry points. The method for determining whether the XPT exists differs between operating systems.

The XPT is responsible for routing a CCB to the correct SIM indicated by the Port ID field.

The XPT is involved in the reverse process of advising the peripheral driver of the completion of a queued request through the xpt_callback() service (see Clause XXX for further information). The xpt_callback() service permits SIM(s) a single point service for the return of queued CCB(s) 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.

10.2 Initialization

The XPT and the SIM shall be responsible for determining the inter-connect configuration at initialization for each SIM.

(Editors Mark - update device config process)

10.3 CCB completion

CCB completion is either immediate or queued.

10.3.1 Completion of immediate CCB

All CCBs are complete when the xpt_action3() call returns except:

For SCSI Inter-connects:
- Execute SCSI I/O
- Execute Target I/O
- Accept Target I/O
- Continue Target I/O
- Immediate Notify
- Execute Engine Request

10.3.2 Completion of queued CCBs

The XPT/SIM shall set all appropriate queued CCB fields marked as IN or IN/OUT and shall provide autoevent information (e.g., autosense data for SCSI inter-connects) as specified in this standard before CCB completion notification is done. See Clause XX (autoevent) and the appropriate CCB tables in Clauses XX, XX, XX, and XX 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-3 Flags field in the CCB:
- CCB Callback on completion.
  The SIM calls the xpt_callback() service routine with the address of the CCB as the single argument, when execution of the queued CCB completes.

  The xpt_callback() service routine may queue the CCB for later callback or callback the peripheral immediately. 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.

- Disable Callback of Completion:
  If the CCB CAM-3 Flags has the Disable Callback of Completion set the XPT/SIM shall notify the peripheral driver of a completed CCB by changing the CAM-3 Status field of the CCB from Request in Progress to another valid CAM-3 Status. The peripheral driver shall be responsible for monitoring the CCB CAM-3 Status field for completion.

10.4 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 CCB request can be queued in any of the following places:

- in the SIM
- in the target/Logical Unit
- in the peripheral driver

The SIM shall keep a logically distinct and separate queue of all the CCB requests from the various peripheral drivers that access a logical device.

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

10.4.1 The logical device and the peripheral driver

Based on needs outside the scope of CAM-3, 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 device. When the logical device completes an operation, the peripheral driver is advised by the SIM and the XPT via a callback or by checking CAM-3 Status for completion (see Clause XX for additional information).

10.4.2 SIM queuing

The SIM shall maintain a request queue for each logical device present. The request queue is logically shared by all peripheral drivers allowed to operate with the logical device.

(Editors Mark - Move to SCSI specific section)
The request queue may support tagged commands. Queue priority shall be supported.

10.4.3 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-3 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 protocol specific Clause 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.

10.4.3.1 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 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.

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/HA 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 interconnect in a usable state.

When a SIM's logical device 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 device except to retrieve autoevent (e.g., SCSI autosense) information. Peripheral drivers can still send CCBs to the SIM for the logical device, or any other logical device. Any new CCBs received by the SIM shall be placed in the queue according to the rules specified in Clause 10.4.3

Note 7

Since the SIM is the controls the hosts access to the inter-connect, the SIM's internal queue goes into a frozen state so that the pending event information in the logical device is not discarded (e.g., SCSI sense information is not lost). The SIM holds it's internal logical device queue in the frozen state until RELEASE SIM QUEUE CCB(s) are received that decrement the SIM queue frozen count to zero.

Note 8

The SIM queue frozen count can be greater then 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-3 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 Autoevent bit in the CAM-3 Flags field does not affect how the SIM handles freezing the SIM's internal queue (i.e., the SCSI REQUEST SENSE command issued by the SIM to recover the sense data does not release the SIM queue). See Clause XX for further information on autoevent.

10.5 Asynchronous event callback

There are certain out of band events that occur not in the context of any CCB function for devices. These events are called Asynchronous Events. These events vary due to the different types of device classes (e.g., SCSI, NETWORK, ATA) and are detailed in the protocol specific clauses of this document.

When event is detected by a SIM/HA or XPT (e.g., SCSI bus reset) 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. The peripheral driver shall return from the callback.

The peripheral driver should register for an asynchronous event callbacks for each logical device with which it is working and once with the XPT. A SIM may be required to register with the XPT based upon the protocols it supports (refer to the protocol specific Clauses for further information).

In order for a peripheral driver to receive asynchronous event callbacks for a protocol specific logical device, it shall issue a SET ASYNCHRONOUS CALLBACK CCB addressed to the protocol specific logical device 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 device or the XPT per peripheral driver. 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 protocol specific logical device or the XPT by issuing the SET ASYNCHRONOUS CALLBACK CCB to a logical device 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 events callbacks for a logical device or the XPT by issuing the SET ASYNCHRONOUS CALLBACK CCB to the logical device 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 be responsible for ensuring that requests to the xpt_async3() routine are processed in serial fashion.

Refer to the protocol specific Clauses for further information.

10.6 Autoevent

Autoevent causes event data to be retrieved automatically if an Autoevent condition is detected by a SIM/HA for a logical device (e.g., if a CHECK CONDITION or COMMAND TERMINATED status is reported in the SCSI Status field of the CCB). Unless the Disable Autoevent CAM-3 Flag is a 1, the SIM shall perform an Autoevent function as described in this Clause and the applicable protocol specific Clause.

The SIM shall perform operations required by the applicable protocol specific Clause to obtain Autoevent data for the event (e.g., form a SCSI REQUEST SENSE command and send it to the same Logical Unit). The location and amount of event data are specified in the Event Info Buffer Pointer and Event Info Buffer Length fields respectively of the I/O Request CCB. If the length field is 0 or the Event Info Buffer Pointer field is null, the SIM/HA shall not perform operations required by the applicable protocol specific Clause to obtain Autoevent data for the event.

After completing the Autoevent sequence without failure the CAM-3 Status shall contain the status of the original command and a protocol specific status that caused the event.

Refer to the specific protocol specific Clauses for further information.

10.7 SIM Loading at Boot and Run Time

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 "labels" or "entry points", which are to used as references by the other loadable modules. The XPT manages the port number assignment of loading of SIMs at both boot and run time.

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 HAs (Port IDs) that are present in the system and logical devices that have been identified as being on those HAs.

When a SIM is loaded, the SIM and XPT shall work together to have the SIM supported HAs registered as addressable Port IDs. The SIM shall call the XPT once for each supported protocol per inter-connect in order to obtain the Port ID for that protocol on the inter-connect (e.g., a FCP inter-connect that supports both the SCSI protocols and network protocols, the SIM would call xpt_bus_register3() twice, once for the SCSI protocol and once for the network protocols).

```
CAM_32 xpt_bus_register3(CAM-3_SIM_ENTRY *)
```

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

A SIM shall call the xpt_bus_register3() service for each interconnect and protocol on an HA passing a pointer to a CAM-3_SIM_ENTRY structure.

When the xpt_bus_register3 function is called, the XPT shall update EDT and then call the sim_init(CAM-3_SIM_ENTRY *) function pointed to by the CAM-3_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 perform a Topology Discovery Process for the interconnect as specified by the applicable protocol Clause in order to update its internal tables containing device identification information.

The SIM shall call the XPT once to de-register the inter-connect protocol for a given Port ID:

```
CAM_U32 xpt_bus_deregister3(Port_ID)
```

The argument is the Port 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_deregister3 function is called, the XPT shall update its internal tables to reflect that the Port_ID (HA) is not present.

Peripheral drivers can request to be informed when a Port ID is registered or de-registered via the async callback feature (see Clauses 10.5 and the applicable protocol specific Clauses).

10.7.1 The CAM-3 SIM_ENTRY3 Structure

The XPT vendor shall define the CAM-3 SIM_ENTRY3 structure. The CAM-3 SIM_ENTRY3 structure shall be used by the SIMs to define the entry points for the SIMs to the XPT and to obtain a Port_ID from the XPT.

```
typedef struct sim_entry3
    {
    CAM_U32             (*sim_init)();          /* Pointer to the SIM init routine */
    CAM_U32             (*sim_action)();        /* Pointer to the SIM CCB go routine */
    CAM_VOID_OFFSET     simha_handle;           /* Pointer to a SIM/HA handle (VU) */
    CAM_U32             max_addr_spec1[2];      /* Maximum value for physical address (target id) */
    CAM_U32             max_addr_spec2[2];      /* Maximum value for physical address (LUN id) */
    CAM_U32             sim_flags;              /* SIM features supported flags */
    CAM_U32              sim_ws_size;           /* The size of this SIM's working set needs */
    CAM_U32             port_id;                /* Port Identifier */
    CAM_U32             protocol_type;          /* Protocol Type/Number (SCSI, NETWORK) */
    CAM_U32             xpt_reg_num;            /* The XPT's registration number. */
    CAM_VOID_OFFSET     vu_entry;               /* Vendor unique pointer defined by OS */
    CAM_8               sim_name[32];           /* Array of 32 bytes for a NULL terminated string */
    CAM_8               vendor_keystring[32];   /* Array of 32 bytes for a NULL terminated string */
    CAM_U32             vendor_key1;            /* SIM vendor or OSD key */
    CAM_U32             vendor_key2;            /* SIM vendor or OSD key */
    CAM_U32             vendor_key3;            /* SIM vendor or OSD key */
    CAM_U32             vendor_key4;            /* SIM vendor or OSD key */
    CAM_U32             vendor_key5;            /* SIM vendor or OSD key */
    CAM_U32             vendor_key6;            /* SIM vendor or OSD key */
    } SIM_ENTRY3;
```

10.7.2 Member Descriptions for the CAM-3 SIM_ENTRY3 Structure

– (*sim_init)():
This member shall be set in by the SIM with the address of this SIM's initialization routine. The SIM's initialization routine shall take one argument which is the pointer of the passed SIM_ENTRY3 structure.

– (*sim_action)():
This member shall be set by the SIM with the address of this SIM's CCB action routine. The SIM's CCB action routine shall accept CCBs from the XPT for execution.

The arguments to the sim_action routine shall be the pointer of a passed CCB pointer and the value of the SIM_ENTRY3 structure member simha_handle. The pointer to the CCB shall be the first argument and the value of the simha_handle shall be the second argument (e.g., (*sim_action)(*CCB, simha_handle)).

– simha_handle:
This member may be set by the SIM before the call to xpt_bus_register3(), after the call to the SIM's initialization routine, or not at all. The setting of this member is vendor unique and may be used for any purpose. The XTP shall not modify this member. The XTP shall pass this argument to any call to the (*sim_action) routine for this specific registered Port_ID.

– max_addr_spec1;
An array of 4 CAM_U32s that shall contain the maximum address specifier that the SIM/HA supports. The format of the array is as follows:

- max_addr_spec1[0] shall contain the least significant portion of a protocol specific address (e.g. the least significant portion of a SCSI-3 target address or lower 32 bits).
- max_addr_spec1[1] shall contain the most significant portion of a protocol specific address (e.g. the most significant portion of a SCSI-3 target address or upper 32 bits).

If the protocol uses up to sixty four (64) bits to address a device on a specific Port_ID then max_addr_spec1[0] and max_addr_spec1[1] shall represent the maximum physical address for the specified Port_ID. See protocol specific addressing for further information.

– max_addr_spec2;
If a protocol uses two (2) specific and distinct components to address a device then max_addr_spec2 member array shall contain the second address component (e.g., SCSI Logical Unit address). This member is an array of CAM_U32s that shall contain the second half the of a protocol specific address for a specific device. The format of the array is as follows:

- max_addr_spec2[0] shall contain the least significant portion of a protocol specific address (e.g. the least significant portion of a SCSI-3 Logical Unit address or lower 32 bits).

- max_addr_spec2[1] shall contain the most significant portion of a protocol specific address (e.g. the most significant portion of a SCSI-3 Logical Unit address or upper 32 bits).

If the protocol uses up to sixty four (64) bits for the second address component to address a device on a specific Port_ID then max_addr_spec2[0] and max_addr_spec2[1] shall represent the devices second maximum physical address component for the specified Port_ID. See protocol specific addressing for further information.

– sim_flags;
This member shall be set by the SIM before the call to xpt_bus_register3(). The member is a bit setting field of the features that this SIM supports. A bit set to a one shall indicate that the feature is supported. The XPT vendor shall define the following flags for the sim_flags member:

- SIM supports CAM-1 CCBs. The flag denotes that the SIM can accept CAM-1 type CCBs for the registered Port_ID. This flag is mutually exclusive with the SIM_CAM3_CCBS flag.
  #define SIM_CAM1_CCBS 0x00000001

- SIM supports CAM-3 CCBs. The flag denotes that the SIM can accept CAM-3 type CCBs for the registered Port_ID. This flag is mutually exclusive with the SIM_CAM1_CCBS flag.
  #define SIM_CAM3_CCBS 0x00000002

- SIM supports Automatic Device Address Presence. This flag denotes that the SIM supports a methodology that allows the XPT to obtain the devices addresses for the Port ID without having to issue a command to every possible device address that the SIM supports for this interconnect. Refer to the protocol specific clauses for further details. A SIM shall set this flag if it supports this type of address resolution methodology.
  #define SIM_AUTO_DEV_PRESENCE 0x40000000

- SIM supports Dynamic Device Addresses. This flag denotes that the SIM supports a methodology that allows a device to change its physical address specifiers through interconnect events (e.g., SCAM, FC). A SIM shall set this flag if it supports this type of address resolution methodology.
  #define SIM_DYNAMIC_DEV_ADDRS 0x80000000

– sim_ws_size;
This member shall be set by the SIM before the call to xpt_bus_register3(). The member shall represent in bytes the SIM's required SIM working set size for CAM-3 CCBs.

The XPT shall ensure that the all CCBs allocated (xpt_ccb_alloc3()) shall have the largest number of bytes for a SIM working set. The XTP shall accomplish this requirement before the call the to the SIMs sim_init() routine.

– port_id:
This member shall be set by the SIM with the SIMs preferred Port_ID number or the XPT's Port_ID number (0xFF). If the SIM sets the member to its preferred Port_ID number there is no guarantee that the XPT will allocated the SIM's preferred Port_ID for the calling SIM. The Port_ID of the XPT's (0xFF) indicates to the XPT that the SIM has no preferred Port_ID (e.g., sysgen parameter).

The allocation of Port_ID by the XPT, shall be based on the rules specified in Clause XXX. The XPT shall set this member to the assigned/allocated Port_ID for this SIM or to the Port_ID of the XPT's (0xFF) to indicate a failure. The member shall be set by the XPT, before the call to the SIM's initialization routine.

– cam_protocol:
This member shall be set by the SIM with the defined CAM-3 protocol number that this inter-connect will be supporting for the assigned Port_ID. This member shall be used by the XPT to assign or

reassign the same Port_ID for this SIM. See Clause XXX for further details.

- xpt_reg_num;
  This member shall be set by the XPT with its peripheral driver registration number before a successful return. The members value shall be stored by SIMs to be used for verification for Discovery and Bind CCB functions refer to Clauses XXXX and XXXX for further information.

- vu_entry;
  The member is a OS specified pointer. Based upon the OS definition for this member, the SIM shall set this value before the call to xpt_async3.

- sim_name:
  This member may be set by the SIM with a NULL terminated character string that shall represent the SIMs name.

- vendor_keystring[32];
  This member may be set by the SIM with a NULL terminated character string that may represent a unique identifier for the HA (e.g., World Wide Identifier). This member if not NULL shall be used by the XPT to assign or reassign the same Port_ID for this SIM. See Clause XXX for further details.

- vendor_key1 - vendor_key6:
  These members may be set by the SIM so that the XPT may uniquely identify the SIM, inter-connect, and protocol when assigning/re-assigning Port_IDs. The values of the vendor keys are not defined by this standard but may be defined by the OS. An example of the usage is as follows:

– vendor_key1	Host bus type (e.g., EISA, PCI, or Direct Attach);
– vendor_key2	Slot Number of Host bus (e.g., Slot number of PCI Bus);
– vendor_key3	Bridge Number (e.g., Number of PCI bridges before this PCI bus);
– vendor_key4	Hardware address component of HA, least significant portion;
– vendor_key5	Hardware address component of HA, most significant portion;
– vendor_key6	HA Board Identifier (e.g., serial number)

11. The CAM-3 SCSI Protocol

This clause defines the model and control to operate a SCSI-2 or SCSI-3 device.

The value assigned for the SCSI Protocol number shall be 0x01 and shall be defined as the following:

```
#define SCSI_PROTOCOL            0x00000001            /* The SCSI protocol number */
```

A SCSI peripheral driver or SIM shall use or set this value where required as specified by this International standard (e.g. a member of a structure having the name protocol_type).

11.1 XPT SCSI Device Topology Discovery Process

The XPT's Topology Discovery Process is responsible for the probing of the SCSI target identification address space to locate SCSI Logical Units on CAM-3 SCSI SIMs/HAs and to perform address authentication on those Logical Units.

Address authentication is defined as :

- Validating that associated Connection_ID(s) (Port_ID, target identifier and, Logical Unit identifier) to the unique identifier and assigned Logical_ID for a SCSI Logical Unit continues to describes the same entity (SCSI Logical Unit) after an event occurs which may have disrupted the association of the Connection_ID(s) to the Logical Unit.

- The Logical Unit shall be validated using unique device identifiers (preferably world-wide unique) obtained from the device. The algorithms for generating the unique identifiers are XPT vendor-specific. However, it is highly recommended that SCSI Logical Units support the Inquiry Command - Vital Products Data Page 0x83 (Device ID) to supply this information to the host.

There are two types of Topology Discovery:
- Scan Port ID: All targets identifiers on a Port_ID are scanned and Logical Units associated with a target identifier
- Scan One Target identifier: A single target identifier on a Port_ID is scanned.

In all cases, when scanning a target identifier (Scan One Target identifier), all Logical Units on that target identifier shall be authenticated.

A topology event is defined as any event on the SCSI interconnect that may change a SCSI device's address. Topology events are SCSI interconnect specific.

Examples of such an events are LIPs on FC interconnect, Bus Resets on parallel SCSI (to cover SCAM devices). These are physical events during which devices may appear/disappear in the Port_ID's topology or where existing devices may change addressing. Topology events may invoke either a Scan Port ID or Scan One Target identifier Discovery Process. Refer to Clause 11.2

Topology Discovery is invoked at the following points.
- After a SIM/HA has performed a successful return from xpt_bus_register3() function. The XPT shall discover the device topology for the registered Port_ID. The XPT shall resolve the entire address space for the Port_ID (Scan Port ID).

- After a SIM detects a topology event.

- Via CAM-3 SCSI CCB functions from a Peripheral Driver to scan a Port_ID. These may be Scan Port ID or Scan One Target identifier requests.

- By vendor unique manual invocation via requests from user space utilities. These may be Scan Port ID or Scan One Target identifier requests.

11.1.1 SIM Discovery Process Information Methodology

A SIM shall provide the XPT with the methodology it supports when it registers with the XTP. A CAM-3 SIM shall provide the required functionality to support the Topology Discover Process. Refer to 10.7 for further on the information required for xpt_bus_register3(). There are indicators (flags) and information that shall be provided by the SIM to the XPT. The required indicators/information placed in the SIM_ENTRY3 data structure for the XPT Topology Discovery process shall be as follows:
- sim_flags:
  - SIM supports CAM-3 CCBs. The flag denotes that the SIM can accept CAM-3 type CCBs for the registered Port_ID
  - An indication of whether the SIM can detect that the topology may have changed. A SIM that can detect that topology may have changed shall when registering with the XTP (xpt_bus_register3()) set the SIM_DYNAMIC_DEV_ADDRS flag. Two examples of this are a SCAM compliant SCSI parallel bus or a SCSI FCP Fibre Channel interconnect.

  - A indication that a SIM supports a feature that is capable of probing for all target identifiers on the interconnect and can report those found to the XPT (SIM_AUTO_DEV_PRESENCE in sim_flags member). A SIM indicating that it cannot support this feature indicates to the XPT, that the XPT shall probe each and every target identifier that may be present on the Port ID.

    The XPT shall probe the Logical Unit address space for every target identifier that is present on the Port ID.

- max_addr_spec1:
  This member shall represent the maximum value for a SCSI target identifier supported the SIM/HA. Targets are addressed by the XPT by specifying a value in the range of 0 to max_addr_spec1. Target identifiers may, but are not required to map to the physical address of the target on the interconnect. Target identifiers may, but are not required to be persistent with the physical target device on the interconnect.

  This member is primarily used to manage the range of targets that are probed by the XPT when probing a SIM that does not support SIM_AUTO_DEV_PRESENCE.

  - max_addr_spec1[0] shall contain the least significant portion of the maximum target address identifier value.

  - max_addr_spec1[1] shall contain the most significant portion of the maximum target address identifier value.

- max_addr_spec2:
  This member shall represent the maximum value for a SCSI Logical Unit identifier supported by the SIM/HA. Logical Units are addressed by the XPT by specifying a value in the range of 0 to max_addr_spec2. Logical Unit identifiers may, but are not required to map to the physical address of the Logical Unit on the interconnect. Logical Unit identifiers may, but are not required to, be persistent with the physical target device on the interconnect.

  - max_addr_spec1[0] shall contain the least significant portion of the maximum Logical Unit address identifier value.

  - max_addr_spec1[1] shall contain the most significant portion of the maximum Logical Unit address identifier value.

Note 9

A SIM may impose a topology limitation that must be dealt with by the integrator if the range of Logical Unit identifiers is not large enough to cover the addressable Logical Unit space indicated by a target device (e.g., SIM supports a fixed 8 Logical Units per target, while the target can support 128 Logical Units, which is independent of the physical Logical Unit values used to address each Logical Unit).

11.1.2 Discovery Process XPT Model

The XPT starts the Topology Discovery Process by issuing Discovery Start CCB. Before the start of any Topology Discovery Process for a registered Port_ID, the XPT shall notify all registered Set Asynchronous Callback CCBs that apply of a AC_DISCOV_START asynchronous event. Refer to Clause 11.7.9 for further information.

If the XPT is notified that a new Discovery Process is needed for a Port_ID through an xpt_async3 call that is currently under going the Topology Discovery process. The current discovery process shall be terminated before a new Topology Discovery process is started. Refer to Clause 11.2 for further information on notification of the need for a Discovery Process. The XPT shall terminate the Topology Discovery Process in the following manner:

- Wait for the completion of any CCB it has sent to the Port_ID it is terminating the Topology Discovery process on.
- Release all Binds that it has obtained.
- Issue a DISCOVERY END CCB function to close the Topology Discovery process.

When the Topology Discovery process has terminated the XPT shall start a new instance of the Topology Discovery process.

If a Topology Discovery process of Scan Port ID is in existence for a Port_ID and the XPT receives a CCB function of Scan Bus (XPT_SCAN_BUS) or Scan Target (XPT_SCAN_TARGET) for that Port_ID, the XTP shall fail the CCB request. The CAM Status shall be Discovery in progress (CAM_DISCOVERY_INPROG).

If a Topology Discovery process of Scan One Target identifier is in existence for a Port_ID and the XPT receives a CCB function of Scan Bus (XPT_SCAN_BUS) or Scan Target (XPT_SCAN_TARGET) for that Port_ID, the XTP shall grant the CCB request. The XPT shall terminate the Topology Discovery Process and shall start a new instance of the Topology Discovery process.

If a Topology Discovery process of Scan One Target identifier is in existence for a Port_ID and the XPT receives a CCB function of Scan Target (XPT_SCAN_TARGET) for that Port_ID for the same target identifier, the XTP shall fail the CCB request. The CAM Status shall be Discovery in progress (CAM_DISCOVERY_INPROG).

If a Topology Discovery process of Scan One Target identifier is in existence for a Port_ID and the XPT receives a CCB function of Scan Target (XPT_SCAN_TARGET) for that Port_ID for a different target identifier, the XTP shall grant the CCB request. The XPT shall queue or defer the new request until the current Topology Discovery process of Scan One Target identifier is completed.

11.1.2.1 Discovery Process Scan Port ID

The XPT shall send a Discovery Start CCB to the SIM/HA and await completion. The CCB function code shall indicate this is a Scan Port ID discovery type.

If the SIM for this assigned Port ID indicated that it supports Automatic Device Address Presence (SIM_AUTO_DEV_PRESENCE). The SIM/HA shall obtain a list of target identifiers present on the identified Port ID before setting CCB completion status and before the CCB function return.

Upon the successful completion of the Discovery Start CCB for the SIM/HA that supports Automatic Device Address Presence, the XPT shall:

– Send a DISCOVERY ADDR CCB to the SIM/HA.
– Upon successful completion of the DISCOVERY ADDR CCB the XPT shall:
    - Obtain the target identifier from the returned DISCOVERY ADDR CCB. Refer to Clause XXX for further information.
    - Issue a successful Bind CCB function to Logical Unit identifier zero (0) for the obtained target identifier.
    - Obtain Inquiry data for the Logical Unit and update the EDT as specified in Clause 8.
    - Issue a SCSI REPORT LUNs command.
    - Issue a successful Bind Release CCB function to Logical Unit identifier zero (0) for the obtained target identifier.
    - If the device supports the SCSI REPORT LUNs command and the command is successful. For each Logical Unit reported in the response data from the SCSI REPORT LUNs command.
        ⇒ Issue a successful Bind CCB function to the Logical Unit identifier for the obtained target identifier.
        ⇒ Obtain Inquiry data for the Logical Unit and update the EDT as specified in Clause 8.
        ⇒ Issue a successful Bind Release CCB function to the Logical Unit identifier for the obtained target identifier.
    - If the device does not support the SCSI REPORT LUNs command. For each Logical Unit identifier starting at Logical Unit identifier zero (0) and proceeding up to and including the

max_addr_spec2 value.

- ⇒ Issue a successful Bind CCB function to the Logical Unit identifier for the obtained target identifier.
- ⇒ Obtain Inquiry data for the Logical Unit and update the EDT as specified in Clause 8.
- ⇒ Issue a successful Bind Release CCB function to the Logical Unit identifier for the obtained target identifier.

- The above steps shall be repeated until a sent DISCOVERY ADDR CCB to the SIM/HA returns a CAM Status of CAM_REQ_CMP_ERR (Request Completed with Error) indicating that there are no more target identifiers to be obtained.

Upon the successful completion of the Discovery Start CCB for the SIM/HA that does not support Automatic Device Address Presence. The XTP shall for each target identifier starting at target identifier zero (0) and proceeding up to and including the max_addr_spec1 value:

- Issue a Bind CCB function to Logical Unit identifier zero (0) for this target identifier.
- If the Bind CCB function is successful.
  - Try to obtain Inquiry data for the Logical Unit and update the EDT as specified in Clause 8.
  - If Inquiry data is successfully obtained:
    - ⇒ Issue a SCSI REPORT LUNs command.
    - ⇒ Issue a successful Bind Release CCB function to the Logical Unit identifier for the target identifier.
    - ⇒ If the device supports the SCSI REPORT LUNs command and the command is successful. For each Logical Unit reported in the response data from the SCSI REPORT LUNs command.
      - ◊ Issue a successful Bind CCB function to the Logical Unit identifier for the target identifier.
      - ◊ Obtain Inquiry data for the Logical Unit and update the EDT as specified in Clause 8.
      - ◊ Issue a successful Bind Release CCB function to Logical Unit identifier for the target identifier.
    - ⇒ If the device does not support the SCSI REPORT LUNs command. For each Logical Unit identifier starting at Logical Unit identifier zero (0) and proceeding up to and including the max_addr_spec2 value.
      - ◊ Issue a successful Bind CCB function to the Logical Unit identifier for the target identifier.
      - ◊ Obtain Inquiry data for the Logical Unit and update the EDT as specified in Clause 8.
      - ◊ Issue a successful Bind Release CCB function to the Logical Unit identifier for the target identifier.
  - If Inquiry data is not successfully obtained due to a CAM Status of Target Selection Timeout (CAM_SEL_TIMEOUT).
    - ⇒ Issue a successful Bind Release CCB function to the Logical Unit identifier for the target identifier.
    - ⇒ Proceed to next incremental target identifier value and repeat the steps.
- If the Bind CCB function was not successful due to a CAM Status of Target Selection Timeout (CAM_SEL_TIMEOUT).
  - Proceed to next incremental target identifier value and repeat the steps.

When there are no more target identifiers to be probed, the XTP shall issue a DISCOVERY END CCB function to close the Topology Discovery process. After completion of the DISCOVERY END CCB function, the XPT shall notify all registered Set Asynchronous Callback CCBs that apply of a AC_DISCOV_END asynchronous event. Refer to Clause 11.7.9 for further information..

11.1.2.2 Discovery Process Scan One Target Identifier

The XPT shall send a Discovery Start CCB to the SIM/HA and await completion. The CCB function code shall indicate this is a Scan One Target identifier discovery type.

Upon the successful completion of the Discovery Start CCB for the SIM/HA, the XPT shall:

- Obtain the target identifier from the Scan Target CCB that started the Topology Discovery process. Refer to Clause XXX for further information.
- Issue a Bind CCB function to Logical Unit identifier zero (0) for this target identifier.
- If the Bind CCB function is successful.
  - Try to obtain Inquiry data for the Logical Unit and update the EDT as specified in Clause 8.
  - If Inquiry data is successfully obtained:
    - ⇒ Issue a SCSI REPORT LUNs command.
    - ⇒ Issue a successful Bind Release CCB function to the Logical Unit identifier for the target identifier.
    - ⇒ If the device supports the SCSI REPORT LUNs command and the command is successful. For each Logical Unit reported in the response data from the SCSI REPORT LUNs command.
      - ◊ Issue a successful Bind CCB function to the Logical Unit identifier for the target identifier.
      - ◊ Obtain Inquiry data for the Logical Unit and update the EDT as specified in Clause 8.
      - ◊ Issue a successful Bind Release CCB function to Logical Unit identifier for the target identifier.
    - ⇒ If the device does not support the SCSI REPORT LUNs command. For each Logical Unit identifier starting at Logical Unit identifier zero (0) and proceeding up to and including the max_addr_spec2 value.
      - ◊ Issue a successful Bind CCB function to the Logical Unit identifier for the target identifier.
      - ◊ Obtain Inquiry data for the Logical Unit and update the EDT as specified in Clause 8.
      - ◊ Issue a successful Bind Release CCB function to the Logical Unit identifier for the target identifier.
  - If Inquiry data is not successfully obtained due to a CAM Status of Target Selection Timeout (CAM_SEL_TIMEOUT).
    - ⇒ Issue a successful Bind Release CCB function to the Logical Unit identifier for the target identifier.
    - ⇒ End the Topology Discovery Process
- If the Bind CCB function was not successful due to a CAM Status of Target Selection Timeout (CAM_SEL_TIMEOUT).
  - End the Topology Discovery Process

When the probe is finished, the XTP shall issue a DISCOVERY END CCB function to close the Topology Discovery process. After completion of the DISCOVERY END CCB function, the XPT shall notify all registered Set Asynchronous Callback CCBs that apply of a AC_DISCOV_END asynchronous event. Refer to Clause 11.7.9 for further information.

11.1.3 XPT Releasing of Binds during Topology Discovery

There are certain conditions and events that shall cause the XTP to release Binds (if any). This Clause describes what conditions and events that shall cause the XPT to release Binds. The method for releasing of Binds is SCSI Asynchronous Events callbacks. The XPT shall perform the registered Port ID A Asynchronous Events callbacks before the Logical ID A Asynchronous Events callbacks. Refer to Clause 11.2 for further information.

When a Scan Port ID Topology Discovery process is started by the XTP it shall in vendor unique manner:

- Find every Connection_ID for this Port_ID in the EDT and indicate that it has not been seen for this Topology Discovery process.
- For each target identifier that is present (e.g., responds) and all the Logical Unit identifiers for that target identifier:
  - Perform an Address authentication for the Logical Unit. Based upon the unique identifier for the Logical Unit the XPT shall perform the following in a vendor unique manner:
    - ⇒ Determine if the Logical Unit has been assigned a Logical ID (seen before).
    - ⇒ If the Logical Unit has a Logical ID assigned, the XPT shall ensure the following:
      - ◊ For the Logical ID determine if the Connection_ID (Port ID, target identifier, and Logical Unit identifier) is associated to the Logical ID.
      - ◊ If the Connection_ID is associated to the Logical ID, mark the Connection_ID as valid.
      - ◊ If the Connection_ID is not associated to the Logical ID.
        - ∗ Determine if the Connection_ID is associated to another Logical ID. If the Connection_ID is associated to another Logical ID. The XPT shall release the Bind as specified in Clause 11.2 for the Logical ID that this Connection_ID was associated to. The XPT shall then associate the Connection_ID to Logical ID and mark as valid.
    - ⇒ If the Logical Unit does not have a Logical ID assigned, the XPT shall ensure the following:
      - ◊ Assigned a unique Logical ID to the represented Logical Unit and associate the unique identifier to the Logical ID.
      - ◊ Associate the Connection_ID to the Logical ID and mark as valid.
      - ◊ Perform Asynchronous Events callbacks as specified in Clause 11.2 (New logical devices(s) found during scan).
- Once each target identifier that is present (e.g., responds) and all the Logical Unit identifiers for that target identifier has had Address authentication performed, the XPT shall:
  - Find every Connection_ID for this Port_ID in the EDT that still has the indication that it has not been seen for this Topology Discovery process.
  - Disassociate the Connection_ID from the Logical ID.
  - Release the Bind as specified in Clause 11.2 for the Logical ID that this Connection_ID was associated to.

When a Scan One Target identifier Topology Discovery process is started by the XTP it shall in vendor unique manner:

- Find every Connection_ID for this Port_ID and target identifier in the EDT and indicate that it has not been seen for this Topology Discovery process.
- For all the Logical Unit identifiers for that target identifier that is present (e.g., responds):
  - Perform an Address authentication for the Logical Unit. Based upon the unique identifier for the Logical Unit the XPT shall perform the following in a vendor unique manner:
    - ⇒ Determine if the Logical Unit has been assigned a Logical ID (seen before).
    - ⇒ If the Logical Unit has a Logical ID assigned, the XPT shall ensure the following:
      - ◊ For the Logical ID determine if the Connection_ID (Port ID, target identifier, and Logical Unit identifier) is associated to the Logical ID.
      - ◊ If the Connection_ID is associated to the Logical ID, mark the Connection_ID as valid.
      - ◊ If the Connection_ID is not associated to the Logical ID.
        - ∗ Determine if the Connection_ID is associated to another Logical ID. If the Connection_ID is associated to another Logical ID. The XPT shall release the Bind as specified in Clause 11.2 for the Logical ID that this Connection_ID was associated to. The XPT shall then associate the Connection_ID to Logical ID and mark as valid.
    - ⇒ If the Logical Unit does not have a Logical ID assigned, the XPT shall ensure the following:
      - ◊ Assigned a unique Logical ID to the represented Logical Unit and associate the unique identifier to the Logical ID.

  - ◊ Associate the Connection_ID to the Logical ID and mark as valid.
  - ◊ Perform Asynchronous Events callbacks as specified in Clause 11.2 (New logical devices(s) found during scan).
- – Once each Logical Unit that is present (e.g., responds) has had Address authentication performed, the XPT shall:
  - Find every Connection_ID for this Port_ID in the EDT that still has the indication that it has not been seen for this Topology Discovery process.
  - Disassociate the Connection_ID from the Logical ID.
  - Release the Bind as specified in Clause 11.2 for the Logical ID that this Connection_ID was associated to.

11.1.4 SIM Model for Topology Discovery Process

The SIM shall maintain a generation number per Logical Unit. The generation number is updated each time a Logical Unit Binding is abnormally terminated. Abnormal terminations (Bind lost) may be generated by events that the SIM detects. The generation number shall be paired with the Bind Handle to ensure the validity of CCB functions issued to a Logical Unit Binding. Refer to Clauses 11.1.2, 11.1.2.1, 11.1.2.2, 11.1.3, 11.2 and 11.7.3.1.

If a SIM detects a topology event indicating a Scan Port ID is needed, the SIM shall mark the entire SCSI Port_ID with a vendor unique indication that a Topology Discovery process is needed for the Port_ID (e.g., DISCOV_NEED_PORT). The SIM shall call xpt_async3 with the required information as specified in Clause 11.2.

If a SIM detects a topology event indicating a Scan One Target identifier is needed, the SIM shall mark the target identifier with a vendor unique indication that a Topology Discovery process is needed for the target identifier. The SIM shall call xpt_async3 with the required information as specified in Clause 11.2.

A SIM shall in a vendor unique manner indicated that a Topology Discovery process is in existence for a Port_ID or target identifier when a Discovery Start CCB is received with a function of either a Scan Port ID or a Scan One Target identifier (i.e., DISCOVERY_ACTIVE_PORT_ID or DISCOVERY_ACTIVE_TARGET_ID). Once the SIM has marked the either the Port_ID or target identifier with the vendor unique indicator of a Topology Discovery process is in existence or needed for a Port_ID or target identifier.

The SIM shall not initiate and new SCSI tasks or SCSI task management functions for any CCBs queued within the SIM for either that Port_ID or target identifier based upon whether the Topology Discovery process is either a Scan Port_ID or Scan One Target identifier respectively. The SIM may complete SCSI tasks or SCSI task management functions that were in “existence” before the Topology Discovery process. The term “existence” shall not mean that the SCSI task or SCSI task Management function is within the task set for a SCSI device server. The term “existence” shall mean that the SIM has delivered the SCSI task or SCSI task management function to an HA for delivery to a SCSI device server.

When a Topology Discovery process is in existence the SIM shall do the following for CCBs received:

- – Accept, at all times, CCBs properly formatted CCBs from the XPT. The SIM shall determine if the Bind CCB functions originated from the XPT by comparing the CCB’s cam_pdrv_reg member to the value storage when the SIM successfully registered with the XPT. CCBs having other functions codes shall be determined if they have originated from the XPT by comparing the cam_sim_generation and cam_sim_bhandle members values to the values returned in the BIND CCB that originated from the XPT. Refer to Clause 10.7 for further information.

- – If the Topology Discovery process is a Scan Port_ID, the SIM shall reject CCBs not from the XPT addressed to that Port_ID with a CAM Status of CAM_DISCOVERY_INPROG.

- – If the Topology Discovery process is a Scan One Target identifier, the SIM shall reject CCBs not originating from the XPT addressed to that Port_ID and target identifier with a CAM Status of CAM_DISCOVERY_INPROG.

Upon the reception of a Scan Port_ID DISCOVERY_START CCB function from the XPT, the SIM shall do the following:

– If there is already a Topology Discovery progress in existence for that Port_ID, the CCB shall be rejected with a CAM_DISCOVERY_INPROG status.

– Clear its Port_ID vendor unique discovery needed indicator, set its vendor unique discovery active indicator, and begin the Topology Discovery process.

– If the SIM does not support AUTO_DISCOVERY. The SIM shall immediately complete the Scan Port_ID DISCOVERY_START CCB.

– If the SIM does support AUTO_DISCOVERY. The SIM shall prepare a list of target identifiers that shall be returned to the XPT via the DISCOVERY ADDR CCB function requests. Once the list is, prepared, the SIM shall complete the DISCOVERY_START CCB.

Note 10

The target identifiers returned to the XPT may be logical values that the SIM later translates into the physical device addresses used on the interconnect. There is no requirement that a logical target identifier to physical address mapping remain persistent (across boots or otherwise). However, as CAM-3 is basing it's notion of device addressing or pathing on the target identifiers values exported by the SIM, it is recommended that the SIM make the target identifiers to physical address mapping persistent between topology changes. Otherwise, the peripheral drivers will experience more address changes, which may cause erratic device behavior (e.g.,, it may not be possible to reissue commands after a path change to sequential access devices without side effects).

Note 11

Discovery of the target identifiers on the interconnect is not implicitly tied to the reception of the DISCOVERY_START CCB. The SIM may actually perform this discovery upon the receipt of the external event which caused it to notify the XPT.

If the SIM is on an interconnect which guarantees notification of target device insertion/removal/address change and if there was no intervening topology change and if the SIM maintains an internal list of target addresses, the SIM may simply re-export it's list of target addresses to the XPT.

However, if the SIM cannot guarantee that it's list of target addresses exactly matches that of the physical interconnect, the SIM will have to re-probe/regenerate the list of addresses.

Upon the reception of a Scan One Target identifier DISCOVERY_START CCB function from the XPT, the SIM shall do the following:

– If there is already a Topology Discovery progress in existence for that Port_ID, the CCB shall be rejected with a CAM_DISCOVERY_INPROG status.

– Clear its Port_ID vendor unique discovery needed indicator, set its vendor unique discovery active indicator, and begin the Topology Discovery process.

– If the SIM does not support AUTO_DISCOVERY. The SIM shall immediately complete the Scan Port_ID DISCOVERY_START CCB.

– If the SIM does support AUTO_DISCOVERY. The SIM shall prepare a list of target identifiers that shall be returned to the XPT via the DISCOVERY ADDR CCB function requests. Once the list is, prepared, the SIM shall complete the DISCOVERY_START CCB.

Upon reception of BIND CCB functions originating from the XTP, the SIM shall do the following:

- If there is already an outstanding Binding with the XPT for the identified Logical Unit. The CCB shall be rejected with a CAM status of CAM_REQ_INVAL (Request Invalid).

- If there is no outstanding Binding with the XPT for the identified Logical Unit. The SIM shall allow the XPT to BIND to the Logical Unit.

- If there is a Binding present on the Logical Unit that was in existence before the Topology Discovery process began. The SIM shall preserve that Binding and perform a Binding for the XPT. This shall be the only case where there are two (2) separate and distinct Binding to a Logical Unit is allowed. The SIM keep these two (2) Bindings as separate and distinct entities.

  The SIM shall preserve the state (e.g. SIM queue frozen) of the Logical Unit's CCB queue for the original Binding (e.g., the Binding from the peripheral driver). The preserving of the state of the original Binder allows operations to continue (if Bind not automatically released) when the Topology Discovery process ends.

The SIM shall handle CCBs of non Bind type functions as follows:

- If the CCB is from the XPT send SCSI task and task management functions to Logical Units as directed.

- If the CCB did not originate from the XPT, the SIM shall reject the CCB with a CAM Status of CAM_DISCOVERY_INPROG.

Upon reception of a XPT BIND RELEASE CCB function, the SIM shall do the following::

- If the CCB is not from the XPT, the SIM shall reject the CCB with a CAM Status of CAM_DISCOVERY_INPROG.

- If the CCB is from the XPT , the SIM shall release the XPT's binding. The SIM shall not release any other Bind if another one is active for the Logical Unit.

Upon the reception of the DISCOVERY_END CCB :

- The SIM shall clear any vendor unique Topology Discovery in-progress indicators and complete the CCB.

11.1.5 Peripheral Driver Model for Topology Discovery Process

The peripheral drivers shall be asynchronously notified of Topology Discovery processes related to devices that it has active bindings on. The peripheral drivers shall register for Discovery and Binding Asynchronous Events. Ultimately, it is the responsibility of the PD to register for the appropriate Asynchronous Events at the proper granularity.

The peripheral drivers shall also be notified of discovery processes by the return status of CCBs (CAM_DISCOVERY_INPROG). that it may have issued just before the Asynchronous Event notification

Upon being notified of a related Topology Discovery process, the PD shall cease issuing CCBs to any Connection_ID associated with the Topology Discovery process. The PD shall not resume operations to any Connection_ID until it has received the Asynchronous Event indicating that the Topology Discovery process is complete.

Note 12

The peripheral drivers are responsible for re-establishing any command order it requires before resuming CCBs to the SIM. This includes the reissuing of the commands rejected with the DISCOVERY_IN_PROGRESS status.

During the discovery process the peripheral driver may be notified of a automatic Bind release through an Asynchronous Event (e.g., device changed its address).

At all times, the peripheral driver is free to manipulate/access the device via other Connection_IDs not related to the Topology Discovery Process. The peripheral driver shall be responsible for maintaining device state, data coherency, command ordering, and any other behavior required for proper operation of the device.

11.2 SCSI Asynchronous Events Callbacks

In an event such as a SCSI bus reset, XPT detected Auto Bind Release or an asynchronous event notification (AEN) the XPT/SIM has to be able to make a callback to the peripheral driver(s) and SIMs.

Callback routines have the same privileges and restrictions as hardware interrupt service routines. The peripheral driver or SIM shall return from the callback.

The Asynchronous Events for SCSI Logical_IDs registration are the following:
- Sent Bus Device Reset to target.
- SCSI AEN.
- Unsolicited reselection.
- SCSI Bus Reset.
- Discovery Process Started
- Discovery Process Ended
- Auto Bind Release.

The Asynchronous Events for SCSI Port_IDs registration are the following:
- Discovery Process Needed for Port_ID
- Discovery Process Needed for Single Target
- Auto Bind Release

The Asynchronous Events for the XPT registration are the following:
- New logical device(s) found during a scan.
- Port ID de-registered.
- Port ID registered.

The supplier of the XPT shall define the Asynchronous Events opcodes/flags and shall be defined follows:

– #define AC_PORT_ID_REGISTRATION	0x80000000	/* Indicates Port_ID Registration */
– #define AC_AUTO_BIND_RELEASE	0x1000	/* Automatic release of Bind */
– #define AC_DISCOV_NEED_TARGET	0x800	/* Discovery needed for target */
– #define AC_DISCOV_NEED_PORT	0x400	/*Discovery needed for Port_ID */
– #define AC_DISCOV_START	0x200	/* Discovery Process has started */
– #define AC_DISCOV_END	0x100	/* Discovery Process has ended */
– #define AC_FOUND_DEVICES	0x80	/* New device 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_SCSI_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 */

The Asynchronous Events opcodes are also the bit flag definitions for the Set Asynchronous Callback CCB.

At peripheral driver initialization time the peripheral driver should register with the XPT for XPT registered events. When wishes to operate with a device the peripheral driver should register with the XPT for Logical_IDs registered events. It is not recommended that peripheral drivers register for asynchronous Port_ID events. Those events are mainly interesting to SIMs.

There are some instances where a SIM may not have the knowledge to determine when a device's address has changed or the device has changed type. SIM/HAs that support Auto Device Address Resolution (e.g., FCP, SPI SCAM), the SIM shall register with the XPT for a Port_ID registration with the Auto Bind Release flag set. This shall apply for each Port_ID assigned a SCSI SIM/HA that supports Auto Device Address Resolution.

In order for a peripheral driver or SIM to receive asynchronous event callbacks, it shall issue a CAM-3 SET ASYNCHRONOUS CALLBACK CCB addressed to the protocol specific Logical_ID or Port_ID with the Asynchronous Event Enables fields set to a 1 for those events the peripheral driver or SIM wishes to be notified of through an asynchronous callback. For the Port_ID asynchronous event callbacks, the peripheral driver or SIM shall issue a SET ASYNCHRONOUS CALLBACK CCB addressed to the Port_ID (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. For the XPT asynchronous event callbacks, the peripheral driver or SIM 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_ID, Port_ID or the XPT per peripheral driver or SIM. The use of wildcards shall not be supported for the SET ASYNCHRONOUS CALLBACK CCB.

The peripheral driver or SIM can change its asynchronous events callbacks for a particular logical device, Port_ID or the XPT by issuing the SET ASYNCHRONOUS CALLBACK CCB to a logical device 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 or SIM can de-register for asynchronous events callbacks for a logical device, Port_ID, or the XPT by issuing the SET ASYNCHRONOUS CALLBACK CCB to the logical device, Port_ID 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 or SIM wishes to change its asynchronous event callback routine, it shall do so by de-registering and then shall follow the registration procedure.

11.2.1 xpt_async3 (callable only by SIMs)

A CAM-3 SIM upon detection of a supported event shall do the following once for each detected event:

- Classify the event:
  ascertain the opcode as specified.

- Format the associated data within an internal (to the SIM) buffer, (e.g., the SCSI sense data received from an AEN).

- Perform the XPT reverse routing required by the event. The CAM-3 SIM shall call the xpt_async3 callback entry point in the XPT:

CAM_U32 xpt_async3(CAM_U32 protocol_type, CAM_U32 opcode, CAM_32 port_id, &addr_spec1[0] ,&addr_spec2[0], CAM_VOID *buffer_ptr, CAM_U32 data_cnt)

The arguments to xpt_async3() are as follows:

- protocol_type;
  This shall be set to the SCSI_PROTOCOL number for all

- opcode
  This shall be a valid opcode as defined by this International standard..

- addr_spec1[2];
  The address of an array of 2 CAM_U32s to contain the SCSI target specifier. The addr_spec1[0] member shall contain the lower 32 bits (least significant portion) of the SCSI target specifier. . The addr_spec1[1] member shall contain the upper 32 bits (most significant portion) of the SCSI target specifier.

- addr_spec2[2];
  The address of an array of 2 CAM_U32s to contain the SCSI Logical Unit specifier. The addr_spec1[0] member shall contain the lower 32 bits (least significant portion) of the SCSI Logical Unit specifier. . The addr_spec1[1] member shall contain the upper 32 bits (most significant portion) of the SCSI Logical Unit specifier.

- buffer_ptr;
  Is a pointer to a buffer that contains information relevant to the event. A null buffer pointer value are valid for the events that do not require a valid buffer.

- data_cnt;
  The number of bytes buffer pointed to by the buffer_ptr.

The XPT shall not modify any argument passed by the caller of xpt_async3(). The XPT shall store a copy of the arguments if needed.

The CAM-3 XPT shall provide the CAM-1 xpt_async() routine for those SIMs that have not migrated to CAM-3 compliance. The XPT shall convert the arguments passed by the caller of the xpt_async() routine for use in the asynchronous callbacks to the peripheral drivers and SIMs. The XPT shall not modify any argument passed by the caller of xpt_async(). The XPT shall store a copy of the arguments if needed.

The XPT shall be responsible for ensuring that requests to the xpt_async3() or xpt_async() routines are processed in serial fashion.

When a CAM-3 SCSI SIM calls the xpt_async3 routine, the protocol_type argument shall be set to SCSI_PROTOCOL and shall supply the required arguments as specified by Table 2. The use of the term Valid in Table 2 shall mean that the argument shall be passed with valid information as it relates to the event. The use of N/A in Table 2 shall mean that no valid information is required to be set in the argument.

opcode	port_id	addr_spec1	addr_spec2	buffer_ptr	data_cnt
AC_BUS_RESET	Valid	N/A	N/A	N/A	N/A
AC_UNSOL_RESEL	Valid	Valid	Valid	N/A	N/A
AC_SCSI_AEN	Valid	Valid	Valid	Valid	Minimum of 22
AC_SENT_BDR	Valid	Valid	N/A	N/A	N/A

AC_DISCOV_NEED_PORT	Valid	N/A	N/A	N/A	N/A
AC_DISCOV_NEED_TARGET	Valid	Valid	N/A	N/A	N/A
AC_AUTO_BIND_RELEASE	Valid	Valid	N/A	N/A	N/A

Table 2 Valid arguments requirements for calls to xpt_async3()

Using the protocol_type, port_id, addr_spec1, addr_spec2 and event opcode information available directly from the SIM, the XPT scans its internal tables looking for "matches" with the registered asynchronous callback peripheral drivers and SIMs. When a match is found, either exactly or with the information supplied as it relates to a Logical_ID, 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.

11.2.2 XPT asynchronous callbacks to peripheral drivers and SIMs

The XTP when notified of an asynchronous event through the xpt_async3 routine it shall call the registered peripheral drivers and SIMs based upon the rules specified in this Clause. The XPT shall use the registered CAM-3 SET ASYNCHRONOUS CALLBACK CCB (*cam_async_func)() member to callback the peripheral driver or SIM.

Peripheral drivers that have not migrated to CAM-3 compliance shall be able to register for asynchronous callbacks using the CAM-1 method. The XPT shall provide the same level of functionality for registered CAM-1 asynchronous callbacks as specified by the CAM-1 specification.

The arguments to the peripheral driver's and SIMs asynchronous callback routine are defined below:

> CAM_VOID (*cam_async_func)(CAM_U32 protocol_type, CAM_U32 opcode, CAM_U32 logical_id, CAM_U32 port_id, &addr_spec1[0] ,&addr_spec2[0], CAM_VOID_OFFSET buffer_ptr, CAM_U32 data_cnt)

The arguments to (*cam_async_func)() are as follows:

- protocol_type;
  This shall be set to the SCSI_PROTOCOL number for all

- opcode
  This shall be a valid opcode as defined by this International standard.

- logical_id;
  This shall be the XPT assigned logical identifier if the registered CAM-3 SET ASYNCHRONOUS CALLBACK CCB is for a Logical_ID;

- addr_spec1[2];
  The address of an array of 2 CAM_U32s to contain the SCSI target specifier. The addr_spec1[0] member shall contain the lower 32 bits (least significant portion) of the SCSI target specifier. . The addr_spec1[1] member shall contain the upper 32 bits (most significant portion) of the SCSI target specifier.

- addr_spec2[2];
  The address of an array of 2 CAM_U32s to contain the SCSI Logical Unit specifier. The addr_spec1[0] member shall contain the lower 32 bits (least significant portion) of the SCSI Logical Unit specifier. . The addr_spec1[1] member shall contain the upper 32 bits (most significant portion) of the SCSI Logical Unit specifier.

- buffer_ptr;

The buffer_ptr value shall be the value of the registered SET ASYNCHRONOUS CALLBACK CCB pdrv_buf member.

- data_cnt;
  The data_cnt value shall be what the XPT has to transfer from the SIM's buffer for the xpt_async() call up to the limit of the buffer as described by the registered SET ASYNCHRONOUS CALLBACK CCB pdrv_buf_len.

When the XPT calls the registered peripheral driver's or SIM's (*cam_async_func)() routine, the protocol_type argument shall be set to SCSI_PROTOCOL and shall supply the required arguments as specified by Table 3. The XPT shall also set the AC_PORT_ID_REGISTRATION flags into the opcode argument if the registered SET ASYNCHRONOUS CALLBACK CCB is registered for a Port_ID. An example of this the a AC_AUTO_BIND_RELEASE event is being delivered to a SIM registered for that Port_ID. The opcode passed to its (*cam_async_func)() would be (AC_AUTO_BIND_RELEASE or'ed with AC_PORT_ID_REGISTRATION).

The use of the term Valid in Table 1 shall mean that the argument shall be passed with valid information as it relates to the event. The use of N/A in Table 3 shall mean that no valid information is required to be set in the argument.

The buffer_ptr and data_cnt arguments are dependent upon whether the originator of SET ASYNCHRONOUS CALLBACK CCB has supplied a buffer and length. Table 3 assumes the originator of the SET ASYNCHRONOUS CALLBACK CCB with a buffer large enough to contain all data requirements.

Opcode	logical_id	port_id	addr_spec1	addr_spec2	buffer_ptr	data_cnt
AC_BUS_RESET	Valid	Valid	N/A	N/A	N/A	N/A
AC_UNSOL_RESEL	Valid	Valid	Valid	Valid	N/A	N/A
AC_SCSI_AEN	Valid	Valid	Valid	Valid	Valid	Minimum of 22
AC_SENT_BDR	Valid	Valid	Valid	N/A	N/A	N/A
AC_SIM_REGISTER	N/A	Valid (0xff)	N/A	N/A	Valid	Valid (4 bytes)
AC_SIM_DEREGISTER	N/A	Valid (0xff)	N/A	N/A	Valid	Valid (4 bytes)
AC_FOUND_DEVICES	Valid	Valid	N/A	N/A	N/A	N/A
AC_DISCOV_NEED_PORT	Valid	Valid	N/A	N/A	N/A	N/A
AC_DISCOV_NEED_TARGET	Valid	Valid	Valid	N/A	N/A	N/A
AC_AUTO_BIND_RELEASE	Valid	Valid	Valid	Valid	N/A	N/A

Table 3 Valid arguments requirements for calls to (*cam_async_func)()

The data requirements for Asynchronous Events for the XPT Port ID registered and the Port ID de-registered data requirements shall be a minimum of 4 bytes. This CAM_U32 shall contain the Port ID needed to access the SIM. Since Port ID registered and Port ID de-registered Asynchronous Callback Requests are directed to the XPT, the Port ID argument is the XPT's Port ID (0xFF).

The new devices found opcode shall be returned whenever the XPT enters a new logical device into equipment data table/database (EDT) (e.g., a printer powered on after system initialization was completed or a new device has been detected by the SIM).

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

The following is a description of the opcodes/flags used in the xpt_async3() routine (opcodes) and the

Set Asynchronous Callback CCB (flags). Also described are the responsibilities of the peripheral drivers, SIMs and the XPT:

- AC_PORT_ID_REGISTRATION:
  Indicates when set to a one that the Set Asynchronous Callback CCB applies to a Port_ID and when set to a zero applies to the Logical_ID.
  - Peripheral drivers
    Shall be responsible for setting this flag to a zero or a one whether it is are registering for a Logical_ID or a Port_ID Asynchronous Event callback in the Set Asynchronous Callback CCB. It is not recommended that peripheral driver register for Port_ID callbacks.

  - SIMs:
    Shall set this flags when registering for a Port_ID Asynchronous Event callback in the Set Asynchronous Callback CCB. A SIM shall not register for a Logical_ID callback.

  - XTP
    The XTP shall be responsible for determining the state of this flag in the Set Asynchronous Callback CCB. Upon the determination of the state of this flag, the XPT shall register the Set Asynchronous Callback CCB with either the specified Logical_ID or Port_ID contained within the CCB_HEADER3.

- AC_AUTO_BIND_RELEASE
  Indicates when set to a one in the Set Asynchronous Callback CCB that the originator of the CCB a callback when an Automatic Release of a Bind may have occurred.
  - Peripheral drivers:
    Shall register for this Asynchronous Event when operating with logical device (Logical_ID) that has a SIM that supports Auto Device Resolution as reported by the Path Inquiry CCB function. The registration shall be to a Logical_ID.

    Peripheral drivers may be notified of an Automatic Release of a Bind even though the peripheral driver has not performed a Bind to a specific Path.

    For example, a SCSI sequential-access device (tape) on a Fibre Channel FCP inter-connect may exist on multiple Paths (multi-initiator) as viewed by a host. The peripheral driver only operates with the device on one Path at a time. The peripheral driver may have performed a single Bind to the Path which it will send CCBs to. An event may occur on the inter-connect which causes the devices address to change as seen by all initiators. This change would cause a Discovery Process to occur for all Port_IDs that detected a possible address change for device(s). The XPT would detect the address change on all the Port_IDs the device is connected to and perform an Asynchronous Event callback for each change on a Port_ID as it relates to a Logical_ID. Since the peripheral driver registered for a Asynchronous Event callback on the related Logical_ID the peripheral driver would receive multiple Asynchronous Event callbacks.

    Peripheral drivers shall be able to determine if the their Bind to a specific Connection_ID has been released by examination of the arguments passed in their registered asynchronous callback routine. Refer to Table 3 for information on information which is valid for this asynchronous callback.

    Upon the notification of this event through a Asynchronous Event, peripherals driver shall determine if a Discovery Process for the Logical_ID (e.g., a Discovery Process Start event with no corresponding Discovery Process End). If a Discovery Process is still in existence for the Logical_ID, it shall wait until a Discovery Process End event for the Logical_ID.

    Once the Discovery Process has ended (if one was in existence), Peripheral drivers shall determine the new Connection_ID(s) for the specified Logical_ID and perform Bind functions for any Connection_ID it wishes to send CCBs to.

- SIMs:
  A SIM shall call xpt_async3() with this opcode when it detects that a Binding has been lost to a device (e.g. FC logout). The SIM shall not post this opcode when the XTP has directed the SIM to release a Binding. The SIM shall for the target identifier contained within addr_spec1 that lost the Bind, return all CCBs for all Logical Units for that target identifier with a CAM Status of CAM_NO_BIND. See Table 2 for argument requirements.

  When the XPT notifies a SIM of a AC_AUTO_BIND_RELEASE asynchronous event. The SIM shall return all CCBs for that target identifier contained within addr_spec1 and Logical Unit identifier contained within addr_spec2 with a CAM Status of CAM_NO_BIND. See Table 3 for information on valid arguments.

  Once the Bind has been released the SIM shall not allow operations on that Connection_ID until a new Bind has been established. See Clause XXX for further information.

- XPT:
  When the XPT detects that a device has changed address, gone non-existent or has changed its device type during a Discovery Process. The XPT shall callback the originators of all registered Set Asynchronous Callback CCBs that apply to either the Port_ID or Logical_IDs. The XPT may detect this event during a Discovery Process. See Clause XXX for further details.

  When the XPT is notified that a SIM has automatically released a Bind through the xpt_async3() routine. The XPT shall callback the originators of all registered Set Asynchronous Callback CCBs that applies to the associated Logical_IDs. The associated Logical_IDs shall be any Logical_ID that has the same Port_ID and target identifier contained within addr_spec1 for the event.

– AC_DISCOV_NEED_TARGET
The Set Asynchronous Callback CCB does not have a corresponding Asynchronous Event flag this is a SIM Asynchronous Event opcode only.

- Peripherals drivers:
  There is no corresponding Asynchronous Event flag for Set Asynchronous Callback CCB.

- SIMs:
  A SIM shall call xpt_async3() with this opcode when it detects that a Discovery Process is needed on the specified target identifier contained within addr_spec1.

- XPT:
  The XPT shall transform the xpt_asyn3() opcode of AC_DISCOV_NEED_TARGET into a AC_DISCOV_START. The XPT shall callback the originators of all registered Set Asynchronous Callback CCBs that apply to either the Port_ID or Logical_IDs. The XPT shall ensure that all callbacks are accomplished for any Logical_ID that have the same addr_spec1 for the Port_ID. The XTP shall perform callbacks to the peripheral drivers and SIMs shall have the opcode AC_DISCOV_START. See Clause XXX for further details.

– AC_DISCOV_NEED_PORT
The Set Asynchronous Callback CCB does not have a corresponding Asynchronous Event flag this is a SIM Asynchronous Event opcode only.

- Peripherals drivers:
  There is no corresponding Asynchronous Event flag for Set Asynchronous Callback CCB.

- SIMs:

A SIM shall call xpt_async3 with this opcode when it detects that a Discovery Process is needed on the assigned Port_ID.

- XPT:
  The XPT shall transform the xpt_asyn3() opcode of AC_DISCOV_NEED_PORT into a AC_DISCOV_START. The XPT shall callback the originators of all registered Set Asynchronous Callback CCBs that apply to either the Port_ID or Logical_IDs. ensure that all callbacks are accomplished for any Logical_ID that has the same Port_ID. The XTP shall perform callbacks to the peripheral drivers and SIMs shall have the opcode AC_DISCOV_START. See Clause XXX for further details.

– AC_DISCOV_START
Indicates that a XPT Discovery Process is pending or has begun on a SIM that supports Auto Device Resolution (e.g. SCAM, FCP). When set to a one in the Set Asynchronous Callback CCB that the originator of the CCB wishes a callback when a XPT Discovery Process is pending or has begun on a Port_ID or Logical_ID.

- Peripheral drivers:
  Shall suspend operations for the identified device until the peripheral driver is notified that the Discovery Process has ended (AC_DISCOV_END).

- SIMs:
  SIMs should not register for this asynchronous event. The Discovery Process for SIMs is handled by the Discovery CCBs. Refer to Clause XXX for further details.

- XTP:
  The XPT shall callback the originators of all registered Set Asynchronous Callback CCBs that apply to either the Port_ID or Logical_IDs. The XTP shall accomplish the callbacks for registered Set Asynchronous Callback CCBs before it issues a Discovery Start CCB function to the identified Port_ID.

– AC_DISCOV_END
Indicates that a XPT Discovery Process has ended on a SIM that supports Auto Device Resolution (e.g. SCAM, FCP). When set to a one in the Set Asynchronous Callback CCB that the originator of the CCB wishes a callback when a XPT Discovery Process has ended on a Port_ID or Logical_ID.

- Peripheral drivers:
  Peripheral drivers shall be able to determine if the their Bind to a specific Connection_ID has been released by examination of the arguments passed in their registered asynchronous callback routine. Refer to Table 3 for information on information which is valid for this asynchronous callback.

  Upon the notification of this event through a Asynchronous Event, peripherals driver shall determine if a Discovery Process for the Logical_ID (e.g., a Discovery Process Start event with no corresponding Discovery Process End). If a Discovery Process is still in existence for the Logical_ID, it shall wait until a Discovery Process End event for the Logical_ID.

  Once the Discovery Process has ended, Peripheral drivers shall determine whether an AC_AUTO_BIND_RELEASE event has been posted for the Logical_ID. The peripheral driver shall re-Bind to the Connection_ID if it wishes to continue to send CCBs to that Connection_ID.

  Peripheral drivers shall determine the now current Connection_ID(s) to a the device for the specified Logical_ID and perform Bind functions for any new Connection_ID it wishes to send CCBs to.

Peripheral drivers at this point in time may continue to send CCBs to the identified device.

- SIMs:
  SIMs should not register for this asynchronous event. The Discovery Process for SIMs is handled by the Discovery CCBs. Refer to Clause XXX for further details.

- XTP:
  The XPT shall callback the originators of all registered Set Asynchronous Callback CCBs that apply to either the Port_ID or Logical_IDs. The XTP shall accomplish the callbacks for registered Set Asynchronous Callback CCBs after it issues a DISCOVERY END CCB function to the identified Port_ID.

– AC_FOUND_DEVICES
Indicates when set to a one in the Set Asynchronous Callback CCB that the originator of the CCB wishes a callback when new devices are found by the XPT.

- Peripheral drivers:
  Peripheral drivers shall copy the May obtain information about the identified device and operate with the device as specified by this document.

- SIMs:
  SIMs should not register for this asynchronous event.

- XPT:
  The XPT shall when a new Logical_ID is assigned for uniquely identified device, callback the originators of all registered Set Asynchronous Callback CCBs that have registered with the XPT (0xFF).

– AC_SIM_DEREGISTER
Indicates when set to a one in the Set Asynchronous Callback CCB that the originator of the CCB wishes a callback when a SIM de-registers for a Port_ID (e.g. SIM is unloading from the operating system).

- Peripheral drivers:
  Shall copy the CAM_U32 from the buffer pointed to by the buffer_ptr if a buffer_ptr with a minimum length of four bytes was supplied in the Set Asynchronous Callback CCBs. The CAM_U32 contains the Port_ID which de-registered.

  When notified of asynchronous event should not send any CCBs to a Connection_ID that contains the Port_ID that de-registered.

- SIMs:
  SIMs shall not de-register any Port_ID that has any Binds active.

- XPT:
  Shall delete any Connection_ID from the EDT that contains the de-register Port_ID.

  The XPT shall after the deletion of the Connection_ID(s), callback the originators of all registered Set Asynchronous Callback CCBs that have registered with the XPT (0xFF) for this event. For each asynchronous callback (*cam_async_func)()) the XPT shall:
  ⇒ Copy the de-registered Port_ID into buffer provided (if any) in the Set Asynchronous Callback CCB.
  ⇒ Set the port_id argument in the call to the (*cam_async_func)()) to 0xFF, which is the XPT's Port_ID.

- AC_SIM_REGISTER
  Indicates when set to a one in the Set Asynchronous Callback CCB that the originator of the CCB wishes a callback when a SIM registers for a Port_ID (e.g. SIM has loaded into the operating system).

  - Peripheral drivers:
    Shall copy the CAM_U32 from the buffer pointed to by the buffer_ptr if a buffer_ptr with a minimum length of four bytes was supplied in the Set Asynchronous Callback CCBs. The CAM_U32 contains the Port_ID which registered.

    When notified of this asynchronous event peripheral drivers should acquire any new Connection_IDs for a Logical_ID for which it is operating with.

  - SIMs:
    There are no requirements or suggestions for this event. The SIM has just loaded.

  - XPT:
    The XPT shall copy the assigned Port_ID into buffer provided (if any) in the Set Asynchronous Callback CCBs

    The XPT shall callback the originators of all registered Set Asynchronous Callback CCBs that have registered with the XPT (0xFF) for this event. The asynchronous callback shall occur after a successful xpt_bus_register3 function and a topology discover process. See Clauses 10.7.1 and XXX for further information.

    For each asynchronous callback (*cam_async_func)()) the XPT shall:
    ⇒ Copy the registered Port_ID into buffer provided (if any) in the Set Asynchronous Callback CCB.
    ⇒ Set the port_id argument in the call to the (*cam_async_func)()) to 0xFF, which is the XPT's Port_ID.

- AC_SENT_BDR
  Indicates when set to a one in the Set Asynchronous Callback CCB that the originator of the CCB wishes a callback when a SIM/Ha sends a BUS DEVICE RESET message to a target.

  - Peripheral drivers:
    Peripheral drivers shall recognize that a SIM has issued a BUS DEVICE RESET message for the specified Logical_ID. The Port_ID and target identifier contained within addr_spec1 is further qualification of the event.

    Peripheral drivers should provide error recovery mechanisms to properly operate with the device after this event is posted.

  - SIMs:
    Should not register for this asynchronous event.

    When a SIM sends BUS DEVICE RESET message to a target identifier on a Port_ID the SIM shall in the following order:
    ⇒ For that Port_ID return all CCBs for all Logical Units that have the same target identifier that the BUS DEVICE RESET message was sent to.
    ⇒ call xpt_async3() with the AC_SENT_BDR opcode when it sends a BUS DEVICE RESET message to a target identifier. See Table 2 for argument requirements.

  - XPT:

The XPT shall callback the originators of all registered Set Asynchronous Callback CCBs that apply to the Logical_IDs that have the same Port_ID and target identifier contained within addr_spec1.

- AC_SCSI_AEN
  Indicates when set to a one in the Set Asynchronous Callback CCB that the originator of the CCB wishes a callback when a SIM/HA receives SCSI AEN from a Logical Unit.

  Peripheral drivers:
  Shall copy the valid AEN data from the buffer pointed to by the buffer_ptr. The number of valid bytes in the buffer is specified by the data_cnt argument. This is contingent upon if a buffer_ptr with a length other then zero bytes was supplied in the Set Asynchronous Callback CCB.
  - SIMs:
    Should not register for this asynchronous event

  - XTP
    The XPT shall copy into a XPT allocated buffer the AEN information provide in the SIM's buffer_ptr argument up to the data_cnt argument.

    The XPT shall callback the originators of all registered Set Asynchronous Callback CCBs that have registered with the XPT (0xFF) for this event. The asynchronous callback shall occur after a successful xpt_bus_register3 function and a topology discover process. See Clauses 10.7.1 and XXX for further information.

    For each asynchronous callback (*cam_async_func)()) the XPT shall:
    ⇒ Copy from the XPT's allocated buffer into the buffer provided (if any) in the Set Asynchronous Callback CCB. The number of bytes copied into Set Asynchronous Callback CCB's pdrv_buf_ptr shall be limited by the pdrv_buf_len member.

- AC_UNSOL_RESEL
  Indicates when set to a one in the Set Asynchronous Callback CCB that the originator of the CCB wishes a callback when a SIM/HA receives an unsolicited re-selection from a Logical Unit.

  - Peripheral drivers:
    Should report an error when this event occurs. The SIM or the device has lost context.

  - SIMs
    Should not register for this event.

  - XPT:
    The XPT shall callback the originators of all registered Set Asynchronous Callback CCBs that apply.

- AC_BUS_RESET
  Indicates when set to a one in the Set Asynchronous Callback CCB that the originator of the CCB wishes a callback when a SIM/Ha issues a BUS RESET to a Port_ID.

  - Peripheral drivers:
    Peripheral drivers shall recognize that a SIM has issued a BUS DEVICE RESET message for the specified Logical_ID. The port_id argument is further qualification of the event.

    Peripheral drivers should provide error recovery mechanisms to properly operate with the device after this event is posted.

- SIMs:
  Should not register for this asynchronous event.

  When a SIM sends BUS DEVICE RESET message to a target identifier on a Port_ID the SIM shall in the following order:
  - ⇒ For that Port_ID return all CCBs with the Cam Status of CAM_SCSI_BUS_RESET for all Logical Units..
  - ⇒ call xpt_async3() with the AC_BUS_RESET opcode when it detects a BUS RESET. See Table 2 for argument requirements.

- XPT:
  The XPT shall callback the originators of all registered Set Asynchronous Callback CCBs that apply to the Logical_IDs that have the same Port_ID.

11.3 CAM-3 Control Blocks

The CCBs used by peripheral drivers and applications to request functions of the XPT and a SIM have a common header, as defined by the CCB_HEADER3 structure.

The sequence of the members in the data structures shall be as defined by this International standard. The memory address offsets of the members of the structure shall be as defined by the CAM_boundary rules.

The definition of the members in the data structures shall not vary among operating systems and hardware platforms.. Several fields in the CCB definitions 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 Clause XX for further information on CCB completion). A peripheral driver, the XPT, or a SIM shall not free a CCB more then 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.

11.4 SCSI Messaging Functionality

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. Some SCSI protocols do not support all the SCSI messages. A peripheral driver shall determine which messages a SIM/HA supports by the Path Inquiry function. Table 4 summarizes the message support.

Message	Supported By
ACA	Transparently supported by SIM
ABORT	Discretely supported by function codes
ABORT TAG	Discretely supported by function codes
BUS DEVICE RESET	Discretely supported by function codes
CLEAR ACA	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
RELEASE RECOVERY	Not supported
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
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

* Issuing this message influenced by peripheral driver via CAM Flags

Table 4 Support of SCSI Messages

11.5 CAM-3 SCSI CCB Table Definitions and Value Definitions

The CAM-3 SCSI CCBs are defined in Tables. Each Table for a CCB has four columns with a header at the top of the table. The vendor of the XTP shall define the data structure as defined for DATA TYPE Table header and the MEMBER NAME Table header columns. This shall mean that the that the data type and CCB data structures member names shall be defined as reflected in the CCB Tables. The vendor of the XTP shall also define the members in the order shown.

Each CCB Table has the CAM-3 CCB_HEADER3 shown. The CCBs defined contains the CCB_HEADER3 data structure as an embedded data structure and shall be referenced as such. A "C" language example for setting the logical_id member of the CCB_HEADER3 for a Bind CCB is as follows:

```
struct ccb_bind3 *bind_ccb;

bind_ccb = (struct ccb_bind3 *) xpt_ccb_alloc3( (CAM_U32)XPT_WAITOK);

bind_ccb->ccb_header3.logical_id = 0x100;
```

The ORIG/RECP Table header reflects the originator of the CCB, the recipient of the CCB and/or the XPT for the required setting of the member. The column with an ORIG denotes that the originator of the CCB sets the member. The column with an RECP denotes that the recipient of the CCB sets the member. The column with XPT denotes that the XPT sets the CCB member before the return from xpt_ccb_alloc3(). The XPT may also be a recipient of a CCB (e.g., CCBs addressed to the XPT). An example of this is that a peripheral driver is usually the originator of the CCB and it is usually sent to a SIM (recipient). If the Table field is blank then the field is not used. If the field is marked with a *** the responsibility for setting the member is explicitly described per CCB function.

The DESCRIPTIVE TEXT Table heading is a descriptive text meaning of the member.

The vendor of the XPT shall also define the CAM-3 CCB data structure name as reflected by the Table label (e.g., Table X ccb_header3, where the label is ccb_header3).

Data value constants are also defined in Tables. Each Table for data value constants has three columns with a header at the top of the table. The vendor of the XTP shall define the data constant name as defined for the NAME Table header. The vendor of the XPT shall define the data constant value as defined by the HEX VALUE Table header.

Following the tables are descriptive text that further clarifies the use and meanings. “C” language examples are also provided.

11.6 CCB_HEADER3 Structure

The supplier of the XPT shall define the CCB_HEADER3 structure shall be as defined in Table 5

DATA TYPE	MEMBER NAME	ORIG/RECP	DESCRIPTIVE TEXT
data structure ccb_header3 pointer	my_addr;	XPT	The address of this CCB
CAM_U16	cam_ccb_len;	XTP	Length of the entire CCB
CAM_U8	cam_func_code;	ORIG	XPT function code contains CAM-3 CCB Identifier
CAM_U8	cam_status;	RECP	Returned CAM subsystem status
CAM_U32	logical_id;	***	The assigned Logical ID of the device.
CAM_U32	cam_flags;	ORIG	Flags for operation of the subsystem
CAM_U32	cam3_func_code;	ORIG	The actual function code for a CAM-3 CCB
CAM_U32	cam_protocol;	ORIG	The protocol type SCSI, NETWORK, etc.
CAM_U32	port_id;	ORIG	A registered SIM/HA port number
CAM_U32	addr_spec1[2];	ORIG	Array of 2 CAM_U32s to contain the target specifier
CAM_U32	addr_spec2[2];	ORIG	Array of 2 CAM_U32s to contain the LUN specifier
CAM_U32	reserved1;		Reserved for future expansion
CAM_U32	cam_sim_generation;	***	Generation number returned by BIND CCB
CAM_VOID_OFFSET	cam_sim_bhandlle	***	SIM Bind Handle returned by the BIND CCB
CAM_VOID_OFFSET	cam_xpt_ptr;	XPT	Pointer to the XPT working space */
CAM_VOID_OFFSET	cam_pdrv_ptr;	XTP	Pointer to the XPT assigned peripheral driver working space
CAM_VOID_OFFSET	cam_sim_ptr;	XPT	Pointer to the XPT assigned SIM working space
CAM_U32	cam_pdrv_len;	XPT	The length in bytes of the peripheral driver working space
CAM_U32	cam_sim_len;	XPT	The length in bytes of the SIM working space

Table 5 ccb_header3

“C” language example for ccb_header3.

```
typedef struct ccb_header3
    {
    struct ccb_header3 *my_addr;            /* The address of this CCB */
    CAM_U16 cam_ccb_len;                    /* Length of the entire CCB */
    CAM_U8 cam_func_code;                   /* XPT function code contains CAM-3 CCB Identifier */
    CAM_U8 cam_status;                      /* Returned CAM subsystem status */
    CAM_U32 logical_id;                     /* The assigned Logical ID of the device. */
    CAM_U32 cam_flags;                      /* Flags for operation of the subsystem */
    CAM_U32 cam3_func_code;                 /* The actual function code for a CAM-3 CCB */
    CAM_U32 cam_protocol;                   /* The protocol type SCSI, NETWORK, etc. */
    CAM_U32 port_id;                        /* A registered SIM/HA port number */
    CAM_U32 addr_spec1[2];                  /*Array of 2 CAM_U32s to contain the target specifier */
    CAM_U32 addr_spec2[2];                  /* Array of 2 CAM_U32s to contain the LUN specifier */
    CAM_U32 reserved1;                      /* Reserved for future expansion */
    CAM_U32 cam_sim_generation;             /* Generation number returned by BIND CCB */
    CAM_VOID_OFFSET cam_sim_bhandle;        /* SIM Bind Handle returned by the BIND CCB */
    CAM_VOID_OFFSET cam_xpt_ptr;            /* Pointer to the XPT working space */
    CAM_VOID_OFFSET cam_pdrv_ptr;           /* Pointer to the XPT assigned peripheral driver working
                                               space */
    CAM_VOID_OFFSET cam_sim_ptr;            /* Pointer to the XPT assigned SIM working space */
    CAM_U32 cam_pdrv_len;                   /* The length in bytes of the peripheral driver working
                                                 space */
    CAM_U32 cam_sim_len;                    /* The length in bytes of the SIM working space */
    } CCB_HEADER3;
```

11.6.1 Member Descriptions of the CCB_HEADER3 Structure

– my_addr;
Pointer containing the physical or virtual address of this CCB. The address type (virtual or physical) is depend on the operating system.

– cam_ccb_len;
This field contains the length in bytes of the CCB, including this field and the address of this CCB in the total.

– cam_func_code;
This member shall contain the CAM-3 function code of XPT_CAM3_CCB. This function code indicates that the CCB passed is a CAM-3 CCB. Refer to Table 7 for the XPT function codes;

– 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 Statuses are defined in Table 6.

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).

The CAM status codes shall be defined and shall be defined as shown in Table 6:

NAME	HEX VALUE	DESCRIPTIVE TEXT
CAM_REQ_INPROG	00h	Request in Progress
CAM_REQ_CMP	01h	Request Completed without Error
CAM_REQ_ABORTED	02h	Request Aborted by Host
CAM_UA_ABORT	03h	Unable to Abort Request
CAM_REQ_CMP_ERR	04h	Request Completed with Error
CAM_BUSY	05h	CAM Busy
CAM_REQ_INVALID	06h	Invalid Request
CAM_PATH_INVALID	07h	Invalid Path ID
CAM_DEV_NOT_THERE	08h	SCSI Device Not Installed
CAM_UA_TERMIO	09h	Unable to Terminate I/O Process
CAM_SEL_TIMEOUT	0Ah	Target Selection Timeout
CAM_CMD_TIMEOUT	0Bh	Command Timeout
	0Ch	Reserved
CAM_MSG_REJECT_REC	0Dh	Message Reject Received
CAM_SCSI_BUS_RESET	0Eh	SCSI Bus Reset Sent/Received
CAM_UNCOR_PARITY	0Fh	Uncorrectable Parity Error Detected
CAM_AUTOSENSE_FAIL	10h	Autosense Request Sense Cmd Failed
CAM_NO_HA	11h	No HA Detected
CAM_DATA_RUN_ERR	12h	Data Overrun
CAM_UNEXP_BUSFREE	13h	Unexpected Bus Free
CAM_SEQUENCE_FAIL	14h	Target Bus Phase Sequence Failure
CAM_CCB_LEN_ERR	15h	CCB Length Inadequate
CAM_PROVIDE_FAIL	16h	Cannot Provide Requested Capability
CAM_BDR_SENT	17h	Bus Device Reset Sent
CAM_REQ_TERMIO	18h	Terminate I/O Process
CAM_HA_ERR	19h	Unrecoverable Host Bus Adapter Error
	1Ah	Reserved

CAM_NO_BIND	1Bh	Bind Lost or Needed
CAM_DISCOVERY_INPROG	1Ch	Discovery Process Needed or In Progress
	1Dh - 32h	Reserved
CAM_IDE	33h	Initiator Detected Error Received
CAM_RESRC_UNAVAIL	34h	Resource Unavailable
CAM_UNACKED_EVENT	35h	Unacknowledged Event by Host
CAM_MESSAGE_RECV	36h	Message Received
CAM_INVALID_CDB	37h	Invalid CDB
CAM_LUN_INVALID	38h	Invalid LUN
CAM_TID_INVALID	39h	Invalid Target ID
CAM_FUNC_NOTAVAIL	3Ah	Function Not Implemented
CAM_NO_NEXUS	3Bh	Nexus Not Established
CAM_IID_INVALID	3Ch	Invalid Initiator ID
CAM_CDB_RECVD	3Dh	SCSI CDB Received
CAM_LUN_ALLREADY_ENAB	3Eh	LUN Already Enabled
CAM_SCSI_BUSY	3Fh	SCSI Bus Busy
CAM_SIM_QFRZN	40H	Value or’ed with CAM Status to indicate that SIM queue is frozen
CAM_AUTOSNS_VALID	80h	Value or’ed with CAM Status to indicate that autosense is valid

Table 6 Cam Statuses

Cam Status descriptions:

- 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 SIM/HA.
- 03h; Unable to Abort Request: the SIM/HA 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/HA 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. Prior to reporting this status the SIM/HA shall ensure the command is no longer active in the target.
- 0Dh; Message Reject Received: The SIM/HA received a SCSI MESSAGE REJECT message.
- 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 uncorrected SCSI bus parity error was detected.
- 10h Autosense Request Sense Command Failed: The SIM/HA attempted to obtain sense data and failed.
- 11h; No HA Detected: HA 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 Logical Unit failed to operate in compliance with ANSI X3.131-1994.
- 15h; CCB Length Inadequate: more private data area is required in the CCB (see Clause 9.2.3 for further information).

- 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 message was sent to the target.
- 18h; Terminate I/O Process: this CCB was terminated because a Terminate I/O Process function was specified for this CCB and the CCB was not an I/O process within the Logical Unit.
- 19h; Unrecoverable Host Bus Adapter Error: this CCB was terminated because of a hardware error detected by the HA. The error does not indicate a SCSI bus problem but an error within the HA or host.
- 1Bh; Bind Lost or Needed: this CCB was terminated due to a Binding that has been lost or that the originator of the CCB has not performed a Bind function.
- 1Ch; Discovery Needed or In Progress: CCB was terminated due to a XPT Discovery function is needed or in progress.
- 33h; Initiator Detected Error: indicates the SIM/HA has received an INITIATOR DETECTED ERROR message.
- 34h; Resource Unavailable: indicates that the SIM/HA has run out of resources for processing connections (Host Target Mode only).
- 35h; Unacknowledged Event: indicates that the Host Target Mode peripheral driver has not acknowledged an event.
- 36h; Message Received: indicates that a message has been received by the SIM/HA that requires attention.
- 37h; Invalid CDB: indicates that the SIM/HA has detected an error condition on reception of a CDB.
- 38h; Invalid Logical Unit: indicates that the Logical Unit specified is outside the supported range of the SIM/HA.
- 39h; Invalid Target ID indicates that the Target ID does not match that used by the HA 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 Logical Unit with any initiator.
- 3Ch; Invalid Initiator ID: the initiator ID specified is outside the valid range that is supported.

Note 13

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 Logical Unit Already Enabled: the Logical Unit identified in Enable LUN CCB is already enabled.
- 3Fh SCSI Bus Busy: the SIM failed to win arbitration for the SCSI bus during several different bus free phases.

“C” language example of CAM Statuses data value constants definitions:

```
#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_TERMIO           0x09   /* Unable 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_SCSI_BUS_RESET       0x0E    /* SCSI bus reset sent/received */
#define CAM_UNCOR_PARITY         0x0F    /* Uncorrectable parity err occurred */
#define CAM_AUTOSENSE_FAIL       0x10    /* Autosense: Request sense cmd fail */
#define CAM_NO_HA                0x11    /* No HA 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 inadequate */
#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_HA_ERR               0x19    /* Unrecoverable host bus adapter err */
#define CAM_NO_BIND              0x1B    /* Binding has been lost or not obtained */
#define CAM_DISCOVERY_INPROG     0x1C    /* Discovery Process needed or in progress */
#define CAM_IDE                  0x33    /* Initiator Detected Error Received */
#define CAM_RESRC_UNAVAIL        0x34    /* Resource unavailable */
#define CAM_UNACKED_EVENT        0x35    /* Unacknowledged event by host */
#define CAM_MESSAGE_RECV         0x36    /* Msg received in Host Target Mode */
#define CAM_INVALID_CDB          0x37    /* Invalid CDB recvd in HT Mode */
#define CAM_LUN_INVALID          0x38    /* Logical Unit 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            0x3D    /* The SCSI CDB has been received */
#define CAM_LUN_ALLREADY_ENAB    0x3E    /* Logical Unit already enabled */
#define CAM_SCSI_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 target */
```

- logical_id;

  This member is the XPT assigned logical identifier of the device for the protocol specified. Most functions do not require this member to be set by the originator. For those functions it is recommended that the member be set to a valid value for tracking purposes and debug.

- cam_flags;

  The CAM flags member qualifies the function to be executed, and vary by function code. See the specified function code CCB for the defined CAM flags.

- cam3_func_code;

  This member contains the CAM-3 function code for the CAM-3 CCB.

  The function codes used to identify the SCSI service being requested is listed in Table 7. The function codes shall be defined and shall be defined as shown in Table 7

NAME	HEX VALUE	DESCRIPTIVE TEXT
	00-0F	**Common functions**
XPT_NOOP	00h	NOP
XPT_SCSI_IO	01h	Execute SCSI I/O
XPT_GDEV_TYPE	02h	Get Device Type

XPT_PATH_INQ	03h	Path Inquiry
XPT_REL_SIMQ	04h	Release SIM Queue
XPT_SASYNC_CB	05h	Set Asynchronous Callback
XPT_SDEV_TYPE	06h	Set Device Type
XPT_SCAN_BUS	07h	Scan SCSI Bus
	08h - 0Fh	**Reserved**
	10h - 1Fh	**SCSI control functions**
XPT_ABORT	10h	Abort SCSI Command
XPT_RESET_BUS	11h	Reset SCSI Bus
XPT_RESET_DEV	12h	Reset SCSI Device
XPT_TERM_IO	13h	Terminate I/O Process
XPT_SCAN_LUN	14h	Scan Logical Unit
XPT_CAM3_CCB	15h	CAM-3 CCB Indicator
XPT_BIND	16h	Bind to a Connection_ID
XPT_BIND_QUERY	17h	Bind Query
XPT_BIND_REL	18h	Bind Release
XPT_DISCOV_START_PORT_ID	19h	Discovery Start Port_ID
XPT_DISCOV_START_TARGET_ID	1Ah	Discovery Start Target Identifier
XPT_DISCOV_ADDR	1Bh	Discovery Get Address
XPT_DISCOV_END	1Ch	Discovery End
	1Dh-1Fh	reserved
XPT_ENG_INQ	20h	Engine Inquiry
XPT_ENG_EXEC	21h	Execute Engine
	22h - 2Fh	reserved
	30h - 3Fh	Target Mode
XPT_EN_LUN	30h	Enable LUN
XPT_TARGET_IO	31h	Execute Target I/O
XPT_ACCEPT_TARG	32h	Accept Target I/O
XPT_CONT_TARG	33h	Continue Target I/O
XPT_IMMED_NOTIFY	34h	Immediate Notify
XPT_NOTIFY_ACK	35h	Notify Acknowledge
	36h - 3Fh	reserved
	Reserved Function Codes	
	40h - 7Fh	reserved
	Vendor Unique Function Codes	Defined by Vendor
	80h - FFh	Vendor Unique
	100h - FFFFFFFFh	Reserved

Table 7 CAM-3 SCSI Function Codes for CCBs

“C” language example of CAM-3 function codes data value constants definitions:

```
#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 SIM queue that is frozen */
#define XPT_SASYNC_CB                        0x05      /* Set Async callback parameters */
#define XPT_SDEV_TYPE                        0x06      /* Set the device type information */
#define XPT_SCAN_BUS                         0x07      /* Scan SCSI Bus */
```

```
/* 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 device, BDR */
#define XPT_TERM_IO                      0x13      /* Terminate the I/O process */
#define XPT_SCAN_LUN                     0x14      /* Scan Logical Unit */
#define XPT_CAM3_CCB                     0x15      /* The CAM-3 CCB indicator */
#define XPT_BIND                         0x16      /*Bind to a Connection */
#define XPT_BIND_QUERY                   0x17      /* Query if Bind exist or options */
#define XPT_BIND_REL                     0x18      /* Release your Bind */
#define XPT_ DISCOV_START_PORT_ID        0x19      /* Discovery Start Port_ID *.
#define XPT_DISCOV_START_TARGET_ID       0x1A      /* Discovery Start Target Identifier */
#define XPT_DISCOV_ADDR                  0x1B      /* Discovery Get Address */
#define XPT_DISCOV_END                   0x1C      /* Discovery End */

/* HA engine commands, 0x20 - 0x2F */
#define XPT_ENG_INQ                      0x20      /* HA engine inquiry */
#define XPT_ENG_EXEC                     0x21      /* HA execute engine request */

/* 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_ACCEPT_TARG       0x32                 /* Accept Host Target Mode CDB */
#define XPT_CONT_TARG         0x33                 /* Cont. Host Target I/O Connection */
#define XPT_IMMED_NOTIFY      0x34                 /* Notify Host Target driver of event */
#define XPT_NOTIFY_ACK        0x35                 /* Acknowledgment of event */
```

If a function code which is not supported is issued to the XPT or a SIM, the XPT or SIM shall complete the request and post CAM Status of Invalid Request.

- cam_protocol;
  The valid CAM defined protocol type (i.e., SCSI, NETWORK).

- port_id;
  This member is the XPT assigned port number of the described device (e.g., SCSI bus number).

- addr_spec1[2];
  Array of 2 CAM_U32s to contain the SCSI target specifier. The addr_spec1[0] member shall contain the lower 32 bits (least significant portion) of the SCSI target specifier. . The addr_spec1[1] member shall contain the upper 32 bits (most significant portion) of the SCSI target specifier.

- addr_spec2[2];
  Array of 2 CAM_U32s to contain the SCSI Logical Unit specifier. The addr_spec1[0] member shall contain the lower 32 bits (least significant portion) of the SCSI Logical Unit specifier. . The addr_spec1[1] member shall contain the upper 32 bits (most significant portion) of the SCSI Logical Unit specifier.

- cam_sim_generation;
  This member reflects the SIM generation number that is associated to a Bind. The value is returned upon a successful Bind function. See Clause XXX for further information on binding functions.

- cam_sim_bhandle;
  This member reflects the SIM bind handle that is return when a binding operation is performed by the peripheral driver. See Clause XXX for further information on binding functions.

- cam_xpt_ptr;
  Pointer to the XPT's working space. The vendor of the XPT may use this space for any purpose it deems appropriate.

- cam_pdrv_ptr;
  Pointer to the XPT assigned peripheral driver working space. See Clause XXX for further information on how a peripheral driver requests this space to be allocated that meets the peripheral driver's needs.

- cam_sim_ptr;
  Pointer to the XPT assigned SIM working space. See Clause XXX for further information on how a SIM requests this space to be allocated that meets the SIM's needs.

- cam_pdrv_len;
  The length in bytes of the peripheral driver working space as pointed to by the cam_pdrv_ptr member.

- cam_sim_len;
  The length in bytes of the SIM working space as pointed to by the cam_sim_ptr member.

11.7 SCSI CAM-3 Specific CCB Function Formats

11.7.1 CAM-3 NOP CCB

A peripheral driver can execute this function at any time. The XPT shall call the indicated SIM's sim_action() routine passing this CCB if the Path ID is valid. The SIM shall return immediately.

The supplier of the XPT shall define the CCB3 NOP CCB structure as shown in Table 8:

DATA TYPE	MEMBER NAME	ORIG/RECP	DESCRIPTIVE TEXT
data structure ccb_header3 pointer	my_addr;	XPT	The address of this CCB
CAM_U16	cam_ccb_len;	XTP	Length of the entire CCB
CAM_U8	cam_func_code;	ORIG	XPT function code contains CAM-3 CCB Identifier
CAM_U8	cam_status;	RECP	Returned CAM subsystem status
CAM_U32	logical_id;		The assigned Logical ID of the device.
CAM_U32	cam_flags;		Flags for operation of the subsystem
CAM_U32	cam3_func_code;	ORIG	The actual function code for a CAM-3 CCB
CAM_U32	cam_protocol;	ORIG	The protocol type SCSI
CAM_U32	port_id;	ORIG	A registered SIM/HA port number
CAM_U32	addr_spec1[2];		Array of 2 CAM_U32s to contain the target specifier
CAM_U32	addr_spec2[2];		Array of 2 CAM_U32s to contain the LUN specifier
CAM_U32	reserved1;		Reserved for future expansion
CAM_U32	cam_sim_generation;		Generation number returned by BIND CCB
CAM_VOID_OFFSET	cam_sim_bhandlle		SIM Bind Handle returned by the BIND CCB
CAM_VOID_OFFSET	cam_xpt_ptr;	XPT	Pointer to the XPT working space */
CAM_VOID_OFFSET	cam_pdrv_ptr;	XTP	Pointer to the XPT assigned peripheral driver working space
CAM_VOID_OFFSET	cam_sim_ptr;	XPT	Pointer to the XPT assigned SIM working space
CAM_U32	cam_pdrv_len;	XPT	The length in bytes of the peripheral driver working space
CAM_U32	cam_sim_len;	XPT	The length in bytes of the SIM working space

Table 8 ccb_noop3

“C” language example for ccb_noop3:

```
typedef struct ccb_noop3
     {
     CCB_HEADER3 ccb_header3;              /* CCB_HEADER3 information fields */
     } CCB_NOOP3;
```

11.7.1.1 Member Descriptions for NOP

– Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.

- cam_func_code;
  This member shall contain the XTP_CAM-3-CCB function code.

- cam3_func_code;
  This member shall contain the XPT_NOOP function code;

- port_id;
  This member shall contain a port number (e.g., SCSI bus number).

- cam_flags;
  There are no CAM flags defined for this function.

11.7.1.2 Returns for Nop

This function shall return a CAM Status of:

– CAM Status of Request Completed without Error.
– CAM Status of Invalid Path ID indicates that the specified Path ID is not installed.

11.7.2 Discovery CCB Functions

This Clause describes the CAM-3 Discovery CCB functions to obtain topology information for SCSI devices. The behavior of the XTP and SIMs are described in detail in Clause 11.1.

11.7.2.1 CAM-3 Discovery Start CCB - Scan Port ID function

The Discovery Start CCB - Scan Port ID function starts a Topology Discovery Process for the specified Port_ID.

The supplier of the XPT shall define the CCB3 Discovery Start CCB - Scan Port ID function structure as shown in Table 9

DATA TYPE	MEMBER NAME	ORIG/RECP	DESCRIPTIVE TEXT
data structure ccb_header3 pointer	my_addr;	XPT	The address of this CCB
CAM_U16	cam_ccb_len;	XTP	Length of the entire CCB
CAM_U8	cam_func_code;	ORIG	XPT function code contains CAM-3 CCB Identifier
CAM_U8	cam_status;	RECP	Returned CAM subsystem status
CAM_U32	logical_id;		The assigned Logical ID of the device.
CAM_U32	cam_flags;		Flags for operation of the subsystem
CAM_U32	cam3_func_code;	ORIG	The actual function code for a CAM-3 CCB
CAM_U32	cam_protocol;	ORIG	The protocol type SCSI
CAM_U32	port_id;	ORIG	A registered SIM/HA port number
CAM_U32	addr_spec1[2];		Array of 2 CAM_U32s to contain the target specifier

CAM_U32	addr_spec2[2];		Array of 2 CAM_U32s to contain the LUN specifier
CAM_U32	reserved1;		Reserved for future expansion
CAM_U32	cam_sim_generation;		Generation number returned by BIND CCB
CAM_VOID_OFFSET	cam_sim_bhandlle		SIM Bind Handle returned by the BIND CCB
CAM_VOID_OFFSET	cam_xpt_ptr;	XPT	Pointer to the XPT working space */
CAM_VOID_OFFSET	cam_pdrv_ptr;	XTP	Pointer to the XPT assigned peripheral driver working space
CAM_VOID_OFFSET	cam_sim_ptr;	XPT	Pointer to the XPT assigned SIM working space
CAM_U32	cam_pdrv_len;	XPT	The length in bytes of the peripheral driver working space
CAM_U32	cam_sim_len;	XPT	The length in bytes of the SIM working space

Table 9 ccb_discov_port_id3

“C” language example for ccb_discov_port_id3:

```
typedef struct ccb_disc_port_id3
     {
     CCB_HEADER3 ccb_header3;              /* CCB_HEADER3 information fields */
     }CCB_DISCOV_PORT_ID3;
```

11.7.2.1.1 Member Descriptions for Discovery Start CCB - Port ID function

– Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.

- cam_func_code;
  This member shall contain the XTP_CAM-3-CCB function code.

- cam3_func_code;
  This member shall contain the XPT_DISCOV_START_PORT_ID function code;

- port_id;
  This member shall contain a port number (e.g., SCSI bus number).

- cam_flags;
  There are no CAM flags defined for this function.

11.7.2.1.2 Returns for Discovery Start CCB - Port ID function

This function shall return a CAM Status of:

- Request Completed without Error.
- Invalid Path ID indicates that the specified Path ID is not installed.
- Discovery in progress indicates that a Topology Discovery Process is already in existence for the specified Port_ID.

11.7.2.2 CAM-3 Discovery Start CCB - Scan Target ID function

The Discovery Start CCB - Scan Target ID function starts a Topology Discovery Process for the specified Port_ID and target identifier.

The supplier of the XPT shall define the CCB3 Discovery Start CCB - Scan Target ID function structure as shown in Table 10

DATA TYPE	MEMBER NAME	ORIG/RECP	DESCRIPTIVE TEXT

data structure ccb_header3 pointer	my_addr;	XPT	The address of this CCB
CAM_U16	cam_ccb_len;	XTP	Length of the entire CCB
CAM_U8	cam_func_code;	ORIG	XPT function code contains CAM-3 CCB Identifier
CAM_U8	cam_status;	RECP	Returned CAM subsystem status
CAM_U32	logical_id;		The assigned Logical ID of the device.
CAM_U32	cam_flags;		Flags for operation of the subsystem
CAM_U32	cam3_func_code;	ORIG	The actual function code for a CAM-3 CCB
CAM_U32	cam_protocol;	ORIG	The protocol type SCSI
CAM_U32	port_id;	ORIG	A registered SIM/HA port number
CAM_U32	addr_spec1[2];	ORIG	Array of 2 CAM_U32s to contain the target specifier
CAM_U32	addr_spec2[2];		Array of 2 CAM_U32s to contain the LUN specifier
CAM_U32	reserved1;		Reserved for future expansion
CAM_U32	cam_sim_generation;		Generation number returned by BIND CCB
CAM_VOID_OFFSET	cam_sim_bhandlle		SIM Bind Handle returned by the BIND CCB
CAM_VOID_OFFSET	cam_xpt_ptr;	XPT	Pointer to the XPT working space */
CAM_VOID_OFFSET	cam_pdrv_ptr;	XTP	Pointer to the XPT assigned peripheral driver working space
CAM_VOID_OFFSET	cam_sim_ptr;	XPT	Pointer to the XPT assigned SIM working space
CAM_U32	cam_pdrv_len;	XPT	The length in bytes of the peripheral driver working space
CAM_U32	cam_sim_len;	XPT	The length in bytes of the SIM working space

Table 10 ccb_discov_target_id3

“C” language example for ccb_discov_target_id3:

```
typedef struct ccb_discov_target_id3
    {
    CCB_HEADER3 ccb_header3;            /* CCB_HEADER3 information fields */
    }CCB_DISCOV_TARGET_ID3;
```

11.7.2.2.1 Member Descriptions for Discovery Start CCB - Target ID function

– Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.

- cam_func_code;
  This member shall contain the XTP_CAM-3-CCB function code.

- cam3_func_code;
  This member shall contain the XPT_DISCOV_START_TARGET_ID function code;

- port_id;
  This member shall contain a port number (e.g., SCSI bus number).

- addr_spec1;
  This member shall contain the SCSI target specifier.

- cam_flags;
  There are no CAM flags defined for this function.

11.7.2.2.2 Returns for Discovery Start CCB - Target ID function

This function shall return a CAM Status of:

– Request Completed without Error.

- Invalid Path ID indicates that the specified Path ID is not installed.
- Discovery in progress indicates that a Topology Discovery Process is already in existence for the specified Port_ID or target identifier.

11.7.2.3 CAM-3 Discovery Address CCB

The Discovery ADDRESS CCB function queries the identified SIM/HA for the next available target identifier that has not been returned by previous Discovery Address CCB functions. The SIM/HA returns the next available target identifier in the ccb_header3 addr_spec1 member.

The supplier of the XPT shall define the Discovery Address CCB structure as shown in Table 11

DATA TYPE	MEMBER NAME	ORIG/RECP	DESCRIPTIVE TEXT
data structure ccb_header3 pointer	my_addr;	XPT	The address of this CCB
CAM_U16	cam_ccb_len;	XTP	Length of the entire CCB
CAM_U8	cam_func_code;	ORIG	XPT function code contains CAM-3 CCB Identifier
CAM_U8	cam_status;	RECP	Returned CAM subsystem status
CAM_U32	logical_id;		The assigned Logical ID of the device.
CAM_U32	cam_flags;		Flags for operation of the subsystem
CAM_U32	cam3_func_code;	ORIG	The actual function code for a CAM-3 CCB
CAM_U32	cam_protocol;	ORIG	The protocol type SCSI
CAM_U32	port_id;	ORIG	A registered SIM/HA port number
CAM_U32	addr_spec1[2];	RECP	Array of 2 CAM_U32s to contain the target specifier
CAM_U32	addr_spec2[2];		Array of 2 CAM_U32s to contain the LUN specifier
CAM_U32	reserved1;		Reserved for future expansion
CAM_U32	cam_sim_generation;		Generation number returned by BIND CCB
CAM_VOID_OFFSET	cam_sim_bhandlle		SIM Bind Handle returned by the BIND CCB
CAM_VOID_OFFSET	cam_xpt_ptr;	XPT	Pointer to the XPT working space */
CAM_VOID_OFFSET	cam_pdrv_ptr;	XTP	Pointer to the XPT assigned peripheral driver working space
CAM_VOID_OFFSET	cam_sim_ptr;	XPT	Pointer to the XPT assigned SIM working space
CAM_U32	cam_pdrv_len;	XPT	The length in bytes of the peripheral driver working space
CAM_U32	cam_sim_len;	XPT	The length in bytes of the SIM working space

Table 11 ccb_discov_addr3

"C" language example for ccb_discov_addr3:

```
typedef struct ccb_discov_addr3
    {
    CCB_HEADER3 ccb_header3;            /* CCB_HEADER3 information fields */
    }CCB_DISCOV_ADDR3;
```

11.7.2.3.1 Member Descriptions for Discovery Address CCB function

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
  - cam_func_code;
    This member shall contain the XTP_CAM-3-CCB function code.
  - cam3_func_code;
    This member shall contain the XPT_DISCOV_ADDR function code;
  - port_id;

This member shall contain a port number (e.g., SCSI bus number).

- addr_spec1;
  This member shall be set by the SIM/HA before the successful completion of this CCB. The value shall be the next available target identifier not returned for the Topology Discover process.

- cam_flags;
  There are no CAM flags defined for this function.

11.7.2.3.2 Returns for Discovery Address CCB function

This function shall return a CAM Status of:

- Request Completed without Error.
- Invalid Path ID indicates that the specified Path ID is not installed.
- Invalid Request indicates that the SIM does not support this CCB function
- Request Completed with error indicates that there are no more target identifiers available for this Topology Discovery process (all have been obtained).

11.7.2.4 CAM-3 Discovery End CCB

The Discovery End CCB function ends the Topology Discovery process for the identified SIM/HA.

The supplier of the XPT shall define the Discovery End CCB structure shown in

DATA TYPE	MEMBER NAME	ORIG/RECP	DESCRIPTIVE TEXT
data structure ccb_header3 pointer	my_addr;	XPT	The address of this CCB
CAM_U16	cam_ccb_len;	XTP	Length of the entire CCB
CAM_U8	cam_func_code;	ORIG	XPT function code contains CAM-3 CCB Identifier
CAM_U8	cam_status;	RECP	Returned CAM subsystem status
CAM_U32	logical_id;		The assigned Logical ID of the device.
CAM_U32	cam_flags;		Flags for operation of the subsystem
CAM_U32	cam3_func_code;	ORIG	The actual function code for a CAM-3 CCB
CAM_U32	cam_protocol;	ORIG	The protocol type SCSI
CAM_U32	port_id;	ORIG	A registered SIM/HA port number
CAM_U32	addr_spec1[2];	RECP	Array of 2 CAM_U32s to contain the target specifier
CAM_U32	addr_spec2[2];		Array of 2 CAM_U32s to contain the LUN specifier
CAM_U32	reserved1;		Reserved for future expansion
CAM_U32	cam_sim_generation;		Generation number returned by BIND CCB
CAM_VOID_OFFSET	cam_sim_bhandlle		SIM Bind Handle returned by the BIND CCB
CAM_VOID_OFFSET	cam_xpt_ptr;	XPT	Pointer to the XPT working space */
CAM_VOID_OFFSET	cam_pdrv_ptr;	XTP	Pointer to the XPT assigned peripheral driver working space
CAM_VOID_OFFSET	cam_sim_ptr;	XPT	Pointer to the XPT assigned SIM working space
CAM_U32	cam_pdrv_len;	XPT	The length in bytes of the peripheral driver working space
CAM_U32	cam_sim_len;	XPT	The length in bytes of the SIM working space

Table 12 ccb_discov_end3

“C” language example for ccb_discov_end3:

```
typedef struct ccb_discov_end3
    {
    CCB_HEADER3 ccb_header3;              /* CCB_HEADER3 information fields */
```

}CCB_DISCOV_END3;

11.7.2.4.1 Member Descriptions for Discovery End CCB function

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
  - cam_func_code;
    This member shall contain the XPT_CAM-3-CCB function code.

  - cam3_func_code;
    This member shall contain the XPT_DISCOV_END function code;

  - port_id;
    This member shall contain a port number (e.g., SCSI bus number).

  - cam_flags;
    There are no CAM flags defined for this function.

11.7.2.4.2 Returns for Discovery Address CCB function

This function shall return a CAM Status of:
- Request Completed without Error.
- Invalid Path ID indicates that the specified Path ID is not installed.

11.7.3 Binding CCB Functions

This Clause describes the CAM-3 Binding functions and behavior for peripheral drivers and SIMs.

11.7.3.1 CAM-3 Bind CCB

The Bind function binds the originator of this CCB to a logical representation of a Logical Unit within a SIM. An originator of CCBs (XPT or peripheral driver) to a Logical Unit shall not send any CCB that requires a SIM Bind Handle without first performing a Bind function. This shall mean that the documented description for a particular CCB function shall state whether a SIM Bind Handle is required as follows;
- cam_sim_bhandle;
  This member shall reflect the SIM bind handle that is return for the current bind operation.

The Bind function allows a SIM to ready itself for operations with a specified device. The Bind function also provides an interlock between a peripheral driver and a SIM for those events that can cause a devices interconnect address specifiers to change.

For the SIMs that support a SCSI protocol that logs into a device (e.g., FCP). The SIM shall cause a protocol specific login to the device. If the protocol specific login fails the SIM shall fail the Bind as specified.

The originator of CCBs (peripheral drivers) shall perform a valid Bind function for each Connection_ID that it wishes to send CCBs to.

There shall be only one Bind on a device for a specific Port_ID and a Connection_ID (e.g., interconnects address specifiers) , unless the second Bind is from the XTP. If there currently is a Bind on a Connection_ID the SIM shall compare the second Bind requests cam_pdrv_reg member to that of the XPT's peripheral drivers registration number (Refer to Clause XX for further information). If the compare is equal to the XPT's peripheral drivers registration number then the second Bind shall be allowed.

The XPT shall share the current Bind parameters if a Bind is currently present, to allow the XPT the operations needed to perform a Discovery process (e.g. send INQUIRY commands to the specified device). If for any reason the device reports that the connection has been lost (e.g., Device reports a log out) the SIM shall release the original Bind. The SIM shall also perform an Asynchronous Event notification as specified for the original Bind and Bind to the for the XPT's request.

Note 14

The above case occurs during Discovery Process and the XPT is discovering topology.

There may be more then one Bind for the uniquely identified device (Logical_ID) in effect for different Port_IDs or the same Port_ID. For example a uniquely identified device (Logical_ID) can be seen from the same host on multiple SIM/Has (multi-initiator). The peripheral driver may Bind to the device through each identified Port_ID.

The SIM shall term a successful Bind function as the following:
- There are no current Binds against the Connection_ID;
- The Bind operational attributes requested can be fulfilled;

The SIM, upon a successful Bind function for the specified Connection_ID, shall do the following:
- Set the cam_sim_generation member so that it reflects the SIM generation number that is associated to a Bind. The value shall be unique and shall be different for every successful Bind.
- set into the cam_sim_bhandle its SIM Bind Handle for the Connection_ID. The SIM's cam_sim_bhandle can be a pointer or a number. It is recommended that the SIM Bind Handle be a pointer to a SIM specific data structure representing the described device (Connection_ID).

The SIM shall store the following information when a successful Bind is accomplished:
- cam_pdrv_reg member value;
- logical_id member value;

If the Bind function is not successful due to a requested Bind operational attribute that can not be furnished. The SIM shall clear the not supported Bind operational attribute flag(s) and return the specified error indication.

If the interconnect protocol supports the concept of command delivery time reporting (e.g. FC). The SIM shall place into the cam_delivery_time member the current time, in seconds, to deliver a command to the device. The SIM shall always round up the time to the nearest second (e.g., 125 milliseconds is round up to 1 second). SIMs that can not determine the delivery time of a command due to non-existence mechanisms in the protocol shall set this value to 0 (e.g., SPI).

The SIM shall post an Asynchronous Event if the SIM detects a delivery time change as reported by the protocol. This shall mean that if the delivery time changes so that it is different then the what was reported in the current Bind, the SIM shall post an Asynchronous Event. See SCSI Asynchronous Events for further detail.

A Bind can be released automatically by a SIM when notified by the XPT, through an Asynchronous Event Callback of a change to the device. The changes to a device that can cause a automatic release of a Bind are as follows:
- Device has a new Connection_ID;
- Device is no longer in existence;
- Device has changed type;

The shall release a Bind and perform an Asynchronous Event when:
- SIM detects that the device has performed explicit protocol specific events that denotes a lost of communication to the device (e.g., a FCP logout).

When a Bind is automatically broken by the SIM or by notification by the XPT to release a Bind (if one exists) the SIM shall:

- Return all CCBs for that Connection_ID with a CAM Status of Bind Broken.. Refer to Clauses XXX and XXX for further information.

The originator of the Bind (peripheral drivers) shall be prepared to deal with the automatic release of a Bind.

The supplier of the XPT shall define the CAM-3 BIND CCB as follows:

```
typedef struct ccb_bind3
    {
    CCB_HEADER3 ccb_header3;            /* CCB_HEADER3 information fields */
    CAM_U32 cam_bind_ops;           /* Operational attributes */
    CAM_U32 cam_pdrv_reg;           /* The requesters driver registration number */
    CAM_U32 cam_delivery_time;      /* Command delivery time if available */
    } CCB_BIND3;
```

•

11.7.3.1.1 Member Descriptions for Bind

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
    - cam_func_code;
      This member shall contain the XTP_CAM-3-CCB function code.
    - cam3_func_code;
      This member shall contain the XPT_BIND function code;
    - logical_id;
      This member shall reflect the correct logical identifier for the Connection_ID.
    - port_id;
      This member shall contain a port number (e.g., SCSI bus number).
    - addr_spec1;
      This member shall contain the SCSI target specifier.
    - addr_spec2;
      This member shall contain the SCSI Logical Unit specifier.
    - cam_flags;
      There are no CAM flags defined for this function.
- cam_bind_ops;
  There are no currently defined operational attributes and this member shall be set to zero.
- cam_pdrv_reg;
  This member is the requesters peripheral driver registration number. This member shall be set by the requesters acquired peripheral driver registration number.

11.7.3.1.2 Returns for Bind

- Request Completed without Error: the Bind function was successful.

- Request Completed with Error: indicates that a Bind is currently in existence for the Connection_ID.
- Invalid Request: the request has been rejected because one or more of the operational attributes can not be fulfilled.
- Invalid Path ID indicates that the specified Path ID is not installed.

11.7.3.2 CAM-3 Bind Release

This function shall cause the release of a Bind for the specified Connection_ID. The SIM shall verify that the owner of the Bind is the same as the requester releasing the Bind. The SIM shall compare the cam_pdrv_reg member, the cam_sim_bhandle member, and the cam_sim_generation member to the original stored values to for the current Bind. If the values are not equal to the to what has been stored the SIM shall not release the Bind.

The supplier of the XPT shall define the CAM-3 Bind Release CCB structure as follows:

```
typedef struct ccb_bind_release3
    {
    CCB_HEADER3 ccb_header3;            /* CCB_HEADER3 information fields */
    CAM_U32 cam_pdrv_reg;            /* The requesters driver registration number */
    } CCB_BIND_RELEASE3;
```

11.7.3.2.1 Member Descriptions for Bind Release

- Required information for the CCB members that shall be set by the originator of the CCB.
    - cam_func_code;
      This member shall contain the XTP_CAM-3-CCB function code.
    - cam3_func_code;
      This member shall contain the XPT_BIND_REL function code;
    - port_id;
      This member shall contain a port number (e.g., SCSI bus number).
    - addr_spec1;
      This member shall contain the SCSI target specifier.
    - addr_spec2;
      This member shall contain the SCSI Logical Unit specifier.
    - cam_sim_generation;
      This member shall reflect the SIM bind generation number that was returned for the current bind operation.
    - cam_sim_bhandle;
      This member shall reflect the SIM bind handle that was returned for the current bind operation.
    - cam_flags;
      There are no CAM flags defined for this function.
    - cam_pdrv_reg;
      This member is the requesters peripheral driver registration number. This member shall be set by the requesters acquired peripheral driver registration number.

11.7.3.2.2 Returns for Bind Release

- Request Completed without Error: the Bind Release function was successful.
- Request Completed with Error: indicates that a Bind is no current in existence for the Connection_ID.
- Invalid Request: the request has been rejected because the CCB originator is not current owner of Bind.
- Invalid Path ID indicates that the specified Path ID is not installed.

11.7.3.3 CAM-3 Bind Query CCB

This function shall cause the SIM to set into the cam_bind_ops member the functionality it supports and to report whether there is a current Bind to the Connection_ID. If there is a current Bind for the Connection_ID the SIM shall place into the cam_sim_bhandle member an arbitrary positive value (recommended value is a 1).

If the interconnect protocol supports the concept of command delivery time reporting (e.g. FC). The SIM shall place into the cam_delivery_time member the current time, in seconds, to deliver a command to the device. The SIM shall always round up the time to the nearest second (e.g., 125 milliseconds is round up to 1 second). SIMs that can not determine the delivery time of a command due to non-existence mechanisms in the protocol shall set this value to 0 (e.g., SPI).

The supplier of the XPT shall define the CAM-3 Bind Query CCB structure as follows:

```
typedef struct ccb_bind_query3
    {
    CCB_HEADER3 ccb_header3;            /* CCB_HEADER3 information fields */
    CAM_U32 cam_bind_ops;           /* Operational attributes */
    CAM_U32 cam_pdrv_reg;           /* The requesters driver registration number */
    } CCB_BIND_QUERY3;
```

11.7.3.3.1 Member Descriptions for Bind Query

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
    - cam_func_code;
      This member shall contain the XTP_CAM-3-CCB function code.
    - cam3_func_code;
      This member shall contain the XPT_BIND_REL function code;
    - port_id;
      This member shall contain a port number (e.g., SCSI bus number).
    - addr_spec1;
      This member shall contain the SCSI target specifier.
    - addr_spec2;
      This member shall contain the SCSI Logical Unit specifier.
    - cam_flags;
      There are no CAM flags defined for this function.

- cam_bind_ops;
  There are no currently defined operational attributes and this member shall be set to zero for any SIM that complies with this version of CAM-3..

11.7.3.3.2 Returns for Bind Query

- Request Completed without Error: the Bind Release function was successful.
- Invalid Path ID indicates that the specified Path ID is not installed.

11.7.4 CAM-3 Get Device Type

For a given Logical Unit this function returns the Peripheral Device Type of the INQUIRY response data in t he cam_pd_type, and optionally the first 36 bytes of inquiry data in the buffer supplied.

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

The supplier of the XPT shall define the Get Device Type CCB structure as follows:

```
typedef struct ccb_getdev3
    {
    CCB_HEADER3 ccb_header3;            /* Header information fields */
    CAM_S8 *cam_inq_data;               /* Pointer to the inquiry data space */
    CAM_U8 cam_pd_type;             /* Peripheral device type */
    } CCB_GETDEV3;
```

11.7.4.1 Member Descriptions for Get Device Type

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
  - cam_func_code;
    This member shall contain the XTP_CAM-3-CCB function code.
  - cam3_func_code;
    This member shall contain the XPT_GDEV_TYPE function code;
  - cam_protocol;
    The shall contain the SCSI_PROTOCOL number.
  - logical_id;
    This member is the XPT assigned logical identifier of the device for the protocol specified.
  - cam_flags;
    There are no CAM flags defined for this function.
- cam_pd_type;
  The Peripheral Device Type of the Logical Unit field is the Peripheral Device Type from the INQUIRY response data. The XPT/SIM shall generate this data by taking byte 0 of the INQUIRY response data and setting bits 7-5 to zero.
- cam_inq_data;
  If the Inquiry Data Pointer field contains a value other then null, it shall point to a buffer of at least 36

bytes. When the Inquiry Data Pointer field is not null the XPT/SIM shall copy the first 36 of INQUIRY response data from its internal tables to the identified buffer, if the Logical Unit responded with INQUIRY response data to the INQUIRY command as defined by SCSI-2.

11.7.4.2 Returns for Get Device Type

This function shall return a CAM Status other than Request in Progress. The CAM Status shall be one of the following:

- Request Completed without Error indicates that the specified device is installed and the peripheral device type field is valid.
- SCSI Device Not Installed indicates that the described peripheral device (CAM protocol type or logical_id) was not found in the EDT.

11.7.5 CAM-3 Path Inquiry

This function shall return information on the installed addressed HA/SCSI bus(es) hardware, or the XPT. To obtain further information on a specific HA(s)/SCSI bus(es) attached, this function can be issued for each assigned Path ID.

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.

If the CCB is addressed to the XTP, the XPT shall set into the protocol_type member all Protocol types it supports. If the CCB is directed to a SIM, the SIM shall set into protocol_type member the Protocol number it supports for the specified port_id member.

The supplier of the XPT shall define the Path Inquiry CCB structure and all flag bit defines. The Path Inquiry CCB structure shall be defined follows:

```
typedef struct ccb_pathinq3
    {
    CCB_HEADER3 ccb_header3;              /* Header information fields */
    CAM_U8 cam_version_num[32];           /* ASCII NULL terminated string Version number */
    CAM_U8 cam_interconnect[32];          /* ASCII NULL terminated string Interconnect protocol
                                             type (e.g., SIP, FCP) */
    CAM_U32 cam_ha_inquiry;               /* SIM/HA support of TASK control etc. */
    CAM_U32 cam_target_sprt;              /* Flags for target mode support */
    CAM_U32 cam_ha_misc;                  /* Misc HA feature flags */
    CAM_U32 cam_ha_eng_cnt;               /* HA engine count */
    CAM_U32 cam_max_targ_addr[2];         /* Maximum SCSI target address */
    CAM_U32 cam_max_lun_addr[2];          /* Maximum SCSI Logical Unit address */
    CAM_U8 cam_vuha_flags[ 16 ];          /* Vendor unique capabilities */
    CAM_U32 cam_async_flags;              /* Event cap. for Async Callback */
    CAM_U32 cam_hpath_id;                 /* Highest path ID in the subsystem */
    CAM_U32 cam_initiator_id[2];          /* ID of the HA on the SCSI bus */
    CAM_U32 cam_prsvd0;                   /* Reserved field, for alignment */
    CAM_U8 cam_sim_vid[32];               /* Vendor ID of the SIM */
    CAM_U8 cam_ha_vid[ 32 ];              /* Vendor ID of the HA */
    CAM_U8 *cam_osd_usage;                /* Ptr for the OSD specific area */
    } CCB_PATHINQ3;
```

Note 15

For CCB's other than PATH INQUIRY CCB with a Path ID of the XPT the CCB is returned with a CAM Status of Invalid Path ID.

11.7.5.1 Member Descriptions for Path Inquiry

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
  - cam_func_code;
    This member shall contain the XTP_CAM-3-CCB function code.

  - cam3_func_code;
    This member shall contain the XPT_PATH_INQ function code;

  - port_id;
    This member shall contain a port number (e.g., SCSI bus number) or the port_id number of the XPT (0xFF).

  - cam_flags;
    There are no CAM flags defined for this function.

- cam_version_num;
  The Version Number member shall identify the revision number of CAM-3 the SIM/HA or the XPT conforms to. PATH INQUIRY CCBs addressed to a valid SIM/HA or the XPT shall respectively place in the Version Number member the ASCII NULL terminated string CAM-3 revision number it conforms to. When the ANSI CAM-3 document is accepted as a standard the revision string shall be “CAM-3 V1” for vendors that comply with the standard. Compliance to a working draft of this document shall represented by a string of “CAM-3 WD R[x]” where [x] is the revision number. The current CAM-3 revision number is 2 and shall be represented by the ASCII NULL terminated string of “CAM-3 WD R2”.

- cam_interconnect
  The Interconnect Protocol type shall identify the SIM/HA’s interconnect protocol.

  For SIMs that comply with one of the ANSI SCSI-3 protocol documents. The addressed Port_ID SIM shall place into the member a NULL. terminated string that shall be the ANSI document name with its version number, if any, for the interconnect protocol.

  For example a Fibre Channel SIM/HA that complies with FCP the interconnect string shall be “FCP”. For an SSA SIM/HA that complies with SSA-2 the interconnect string shall be “SSA2”.

  For SIMs that comply with the SCSI-2 standard for the SCSI parallel bus but not the ANSI SCSI-3 SIP standard the SIM shall place into the member a NULL terminated string of “SCSI-2 SIP”.

- cam_ha_inquiry;
  The SCSI Capabilities member may be duplicate of byte 7 field in inquiry data. Based upon the varying SCSI transport protocol (e.g., SIP, FCP, SBP) some of the defined capabilities do not apply. The SIM shall set the capabilities flags it supports as defined by the following:

  - SIM supports Auto Device Resolution. This flag denotes that the SIM supports a methodology that allows a SCSI device to change its physical address specifiers through interconnect events (e.g., SCAM, FC). A SIM shall set this flag if it supports this type of address resolution methodology.
    #define PI_AUTO_DEV_RESOLVE 0x80000000

  - SIM does support BUS RESET TASK
    PI_BUSRESET_ABLE 0x200

- SIM does support TERMIATE IO TASK
  #define PI_TERMINATE_IO_ABLE 0x100

- SIM supports the Modify Data Pointers Message
  #define PI_MDP_ABLE 0x80

- SIM supports 32 bit wide SCSI data transfers
  #define PI_WIDE_32 0x40

- SIM supports 16 bit wide SCSI data transfers
  #define PI_WIDE_16 0x20

- SIM supports synchronous data transfers
  #define PI_SDTR_ABLE 0x10

- SIM supports linked SCSI commands
  #define PI_LINKED_CDB 0x08

- SIM supports tagged queuing of command
  #define PI_TAG_ABLE 0x02

- SIM supports soft reset functionality
  #define PI_SOFT_RST 0x01

- Reserved flag bit defines of 0x00

– cam_target_sprt;
The Target Mode supported member reports the functionality support for Target Mode The SIM shall set the capabilities flags it supports as defined by the following:

- Reserved flag bit defines of 0x100 to 0xFFFFFF00

- SIM supports Host Target Mode
  #define PIT_HOST_MODE 0x80

- SIM supports Phase Cognizant Mode
  #define PIT_PHASE_COG_MODE_ 0x40

- SIM supports target mode disconnects
  #define PIT_TMODE_DISCON 0x20

- SIM supports Terminate I/O Process Messages
  #define PIT_TERMINATE_IO 0x10

- SIM supports Group 6 Commands
  #define PIT_Group_6 0x10

- SIM supports Group 5 Commands
  #define PIT_Group_5 0x20

- Reserved flag bit define of 0x01 and 0x00

- cam_ha_misc;
  The miscellaneous supported member reports miscellaneous functionality support by the XPT. The XPT shall set the capabilities flags it supports as defined by the following:
  - Reserved flag bit defines of 0x10, 0x08, 0x04, 0x02, 0x01 and 0x100 to 0xFFFFFF00
  - SCSI Bus Scan Direction; If the XPT scans Low to High (e.g. target 0x0 to target 0xF) the defined bit shall be cleared. If the XPT scans high to low (e.g. target 0xF to target 0x0) the defined bit shall be set.
    #define PIM_SCANHILO 0x80
  - Removable devices not included in scan, indicates that the XPT does not keep inquiry data on those devices. If the XPT does not keep inquiry data for removable device the indicated bit shall be set.
    #define PIM_NOREMOVE 0x40
  - Inquiry Data not Kept by XPT set indicates that the XPT does not store inquiry data.
    #define PIM_NOINQUIRY 0x20

- cam_ha_eng_cnt;
  Engine count signifies the number of engines available at this specified Port_ID for an HA engine.

- cam_max_targ_addr;
  The maximum addressable target address is an array of 2 CAM_U32s. This member represents the highest target address the SIM/HA is capable of. The cam_max_target_addr[0] member shall contain the lower 32 bits (least significant portion) of the maximum SCSI target specifier. The cam_max_target_addr[1] member shall contain the upper 32 bits (most significant portion) of the maximum SCSI target specifier.

- cam_max_lun_addr;
  The maximum addressable Logical Unit address is an array of 2 CAM_U32s. This member represents the highest Logical Unit address the SIM/HA is capable of. The cam_max_lun_addr[0] member shall contain the lower 32 bits (least significant portion) of the SCSI Logical Unit specifier. . The cam_max_lun_addr[1] member shall contain the upper 32 bits (most significant portion) of the SCSI Logical Unit specifier.

- cam_vuha_area
  Vendor Unique storage area of 16 bytes.

- cam_sim_priv;
  In some environments, the Size of Private Data Area field 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 HA capabilities peculiar to a particular HA implementation.

- cam_async_flags;
  The Asynchronous Event Capabilities field indicate what reasons can cause the XPT/SIM to generate an asynchronous event callback. The XTP or SIM shall set the capabilities flags it supports as defined by the following:
  - Bit values 0x1000000 to 0xFF000000 (bits 24 - 31) are vendor unique.
  - Bit values 0x100 to 0x0FFFF00 (bits 8 - 23) are reserved

- New Devices Found During Rescan, CCB addressed to XTP (XTP sets).
  - ◊ #define AC_FOUND_DEVICES 0x80

- SIM Module De-registered, CCB addressed to XTP (XTP sets).
  - ◊ #define AC_SIM_DEREGISTER 0x40

- SIM Module Registered, CCB addressed to XTP (XTP sets).
  - ◊ #define AC_SIM_REGISTER 0x20

- Sent Bus Device Reset to Target, CCB addressed to SIM (SIM sets).
  - ◊ #define AC_SENT_BDR 0x10

- SCSI AEN, CCB addressed to SIM (SIM sets).
  - ◊ #define AC_SCSI_AEN 0x08

- 0X04 Reserved

- Unsolicited Reselection, CCB addressed to SIM (SIM sets).
  - ◊ #define AC_UNSOL_RESEL 0x02

- Unsolicited SCSI Bus Reset, CCB addressed to SIM (SIM sets).
  - ◊ #define AC_BUS_RESET 0x01

– cam_hpath_id;
If the Path ID field of the CCB has a value of FFh (the XPT Path ID), then the only fields that shall be valid upon return to the caller are the Highest Path ID Assigned member and the Version Number member. The highest Path ID assigned field shall not be valid if the Path ID field in the CCB contains a value other than FFh.

If no Path IDs exist (i.e., no SCSI buses are registered), then the highest Path ID Assigned field shall be FFh, the ID of the XPT.

– cam_initiator_id;
The CAM initiator identifier is an array of 2 CAM_U32s. This member represents the target address (initiator id) of this HA as identifier by the port_id member. The cam_initiator_id[0] member shall contain the lower 32 bits (least significant portion) of the SCSI target specifier (initiator id). The cam_initiator_id[1] member shall contain the upper 32 bits (most significant portion) of the SCSI target specifier (initiator id).

– cam_sim_vid[16];
The vendor ID of SIM supplier member shall contain a NULL terminated sting of the name of the SIM vendor.

– cam_ha_vid[16];
The vendor ID of HA supplier member shall contain a NULL terminated sting of the name of the HA vendor.

– cam_osd_usage;
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 in this standard.

11.7.5.2 Returns for Path Inquiry

This function shall return a CAM Status other than Request in Progress. The CAM Status shall be one of the following:

- Request Completed without Error indicates that the other returned fields are valid.
- No HA Detected indicates that the HA is no longer responding to the SIM.
- Invalid Path ID indicates that the specified Path ID is not installed.

11.7.6 CAM-3 Release SIM Queue

This function is provided so that the peripheral driver can decrement a SIM queue frozen count or ascertain the SIM queue frozen count without modification for the addressed Logical Unit. Determining the current SIM queue frozen count without modification is accomplished by issuing this function with the CAM Flag of SIM Queue Freeze bit set to a one. With the CAM Flag of SIM Queue Freeze bit set to zero, the SIM/HA shall decrement the SIM queue frozen count by one and shall release the SIM queue for the addressed Logical Unit when the count reaches zero. The SIM/HA shall return the current SIM queue freeze count for the Logical Unit in the SIM Queue Frozen Count Field for either value of the CAM Flag of SIM Queue Freeze bit. See Clause XXX for further clarification on the SIM queue frozen state.

The supplier of the XPT shall define the Release SIM Queue CCB structure as follows:

```
typedef struct ccb_relsim3
    {
    CCB_HEADER3 ccb_header3;            /* Header information fields */
    CAM_U32 cam_frzn_cnt;           /* Current freeze count */
    } CCB_RELSIM3;
```

11.7.6.1 Member Descriptions for Release SIM Queue

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
    - cam_func_code;
      This member shall contain the XTP_CAM-3-CCB function code.
    - cam3_func_code;
      This member shall contain the XPT_NOOP function code;
    - port_id;
      This member shall contain a port number (e.g., SCSI bus number).
    - addr_spec1;
      This member shall contain the SCSI target specifier.
    - addr_spec2;
      This member shall contain the SCSI Logical Unit specifier.
    - cam_sim_generation;
      This member shall reflect the SIM bind generation number that was returned for the current bind operation.
    - cam_sim_bhandle;
      This member shall reflect the SIM bind handle that is return for the current bind operation.
    - cam_flags;
      If the defined CAM Flag of CAM_SIM_QFREEZE bit is set to a one, the SIM/HA shall place the current SIM queue frozen count without any modifications into the cam_frzn_cnt member.

If the defined CAM Flag of CAM_SIM_QFREEZE bit is set to a zero, the SIM/HA shall decrement its by one the current SIM queue frozen count and shall place that value if positive or a zero if negative into the cam_frzn_cnt member.

– cam_frzn_cnt;
  The SIM Queue Frozen Count field shall be zero or a positive value and shall be ascertained by the setting of the defined CAM Flag of CAM_SIM_QFREEZE.

11.7.6.2 Returns for Release SIM Queue

This function shall return a CAM Status other than Request in Progress. The CAM Status shall be one of the following:

- Request Completed without Error.
- Invalid Path ID indicates that the Path ID is invalid.
- No Bind indicates that either no Bind has been established or Bind has been lost.

11.7.7 CAM-3 Scan SCSI Bus

This function shall cause the XPT/SIM to update its internal tables on the installed devices on the identified Path ID. The target and Logical Unit fields shall be ignored. The XPT/SIM shall scan each Logical Unit address on the SCSI bus and update its tables with the inquiry data provided by each Logical Unit that responds.

This function shall not be issued to a Port_ID that supports Auto Device Resolution as indicated in the Path Inquiry function.

The information on attached SCSI devices is gathered at initialization by the XPT. This function is provided to force an update of the EDT contents. Using this function at any other time is not recommended because its execution can take a long time. Also execution of this function severely degrades SCSI Bus performance. Furthermore the thread that calls this function can do no further work until this function completes. Any new devices detected during the scan shall generate asynchronous callbacks to peripheral drivers registered for new devices found.

The supplier of the XPT shall define the Scan Bus CCB as follows:

```
typedef struct ccb_scanbus3
    {
    CCB_HEADER3 ccb_header3;            /* Header information fields */
    }CCB_SCANBUS3;
```

11.7.7.1 Member Descriptions for Scan Bus

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
  - cam_func_code;
    This member shall contain the XTP_CAM-3-CCB function code.
  - cam3_func_code;
    This member shall contain the XPT_SCAN_BUS function code;
  - port_id;
    This member shall contain a port number (e.g., SCSI bus number).

- cam_flags;
  There are no CAM flags for this function.

11.7.7.2 Returns for Scan Bus

This function shall return a CAM Status other than Request in Progress. The CAM Status shall be one of the following:

- Request Completed without Error indicates that the devices have been scanned and the table updated.
- Invalid Path ID indicates that the Path ID is invalid.
- Invalid Request indicates that the SIM that supports Auto Device Resolution and this CCB should not be issued.

11.7.8 CAM-3 Scan Logical Unit

This function shall cause the XPT/SIM to update its internal tables on the installed device on the identified Path ID, Target ID, and Logical Unit. The XPT/SIM shall scan the Logical Unit addressed on the SCSI bus and update its tables with the inquiry data provided by the Logical Unit that responds.

This function is provided to force an update of the table contents for a Logical Unit not present or configured when the Scan SCSI Bus function was invoked. Execution of this function can severely degrade SCSI Bus performance if the scanned Logical Unit does not respond to selections. Furthermore the thread that calls this function can do no further work until this function completes. A new Logical Unit detected during the scan shall generate asynchronous callbacks to peripheral drivers registered for new devices found.

The supplier of the XPT shall define the Scan Logical Unit CCB structure as follows:

```
typedef struct ccb_scanlun3
    {
    CCB_HEADER3 ccb_header3;            /* Header information fields */
    } CCB_SCANLUN3;
```

11.7.8.1 Member Descriptions for Scan Logical Unit

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
  - cam_func_code;
    This member shall contain the XTP_CAM-3-CCB function code.
  - cam3_func_code;
    This member shall contain the XPT_SCAN_LUN function code;
  - port_id;
    This member shall contain a port number (e.g., SCSI bus number).
  - addr_spec1;
    This member shall contain the SCSI target specifier.
  - addr_spec2;
    This member shall contain the SCSI Logical Unit specifier.

- cam_flags;
  There are no CAM flags for this function

11.7.8.2 Returns for Scan Logical Unit

This function shall return a CAM Status other than Request in Progress. The CAM Status shall be one of the following:

- Request Completed without Error indicates that the devices have been scanned and the table updated.
- Invalid Path ID indicates that the Path ID is invalid.
- SCSI device not installed there indications that the Logical Unit responded with XXX
- Target selection timeout indicated that the SCSI target did not respond.

11.7.9 CAM-3 Set Asynchronous Callback

Editors Mark this needs to be updated for New method (e.g. Logical unit) Currently not valid.

Asynchronous event callbacks are described in Clause XXX.

This function is provided so that a peripheral driver or SIM can register a callback routine for the selected logical_id or port_id. The function shall register a routine for receipt of callbacks for selected asynchronous events that occur on the selected logical_id or port_id. 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 supplier of the XPT shall define the Scan Logical Unit CCB structure as follows:

```
typedef struct ccb_setasync3
    {
    CCB_HEADER3 ccb_header3;            /* Header information fields */
    U32 cam_async_flags;            /* Event enables for Callback resp */
    CAM_VOID (*cam_async_func)();            /* Async Callback function address */
    CAM_U8 *pdrv_buf;                /* Buffer set aside by the Per. drv */
    CAM_U8 pdrv_buf_len;            /* The size of the buffer */
    } CCB_SETASYNC3;
```

11.7.9.1 Member Descriptions for Set Asynchronous Callback

```
#define AC_FOUND_DEVICES 0x80    /* During a rescan new device 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_SCSI_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 */
```

11.7.9.2 Returns for Set Asynchronous Callback

This function shall return a CAM Status other than Request in Progress. The CAM Status shall be one of the following:

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

- Invalid Path ID indicates that the Path ID is invalid.

11.7.10 CAM-3 Set Device Type - To be DELETED by committee

In response to this function, the XPT/SIM shall add the Target ID, Logical Unit, and peripheral type to the table of attached peripherals built during CAM initialization.

The XPT/SIM does not check the validity of the information supplied by the peripheral driver.

Note 16

Insertion of device type information may corrupt the table, and the results would be unpredictable.

The supplier of the XPT shall define the Set Device Type CCB structure as follows:

```
typedef struct ccb_setdev3
    {
    CCB_HEADER3 ccb_header3;            /* Header information fields */
    CAM_U8 cam_dev_type;                /* Valid for the device type field in EDT */
    } CCB_SETDEV3;
```

11.7.10.1 Member Descriptions for Set Device Type

- cam_dev_type;
  Peripheral Device Type of Logical Unit.

11.7.10.2 Returns for Set Device Type

This function shall return a CAM Status other than Request in Progress. The CAM Status shall be one of the following:

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

11.7.11 CAM 3 Abort SCSI Command

This function requests that a SCSI command 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 expects to abort. The SIM/HA shall issue an ABORT message or an ABORT TAG message on the SCSI bus based on the following conditions:

- ABORT message:
  The CCB identified in the CCB to be Aborted Pointer field is not an established I_T_L_Q nexus I/O process and is an established I_T_L nexus I/O process (untagged command currently active).

- ABORT TAG message:
  The CCB identified in the CCB to be Aborted Pointer field is an established I_T_L_Q nexus I/O process.

If a contingent allegiance condition is in effect for the specified Execute I/O Request CCB (e.g., a SCSI status has been returned from the Logical Unit of CHECK CONDITION or COMMAND TERMINATED), then the EXECUTE SCSI I/O REQUEST CCB shall complete normally as specified by this standard. If autosense is specified for the EXECUTE SCSI I/O REQUEST CCB then the SIM/HA shall retrieve the autosense data.

This request does not necessarily result in an ABORT message or an ABORT TAG message being issued over the SCSI Bus if the CCB identified is not an established I/O process.

The specified Execute I/O Request CCB may be in one of the following states which ascertains the ultimate results of the Abort SCSI Command function.

- Not contained within the SIM/HA queues for the addressed Logical Unit.

  The SIM/HA shall not be responsible for any further processing for the specified CCB.

- Contained within the SIM/HA queues for the addressed Logical Unit, but is not an I/O process within the Logical Unit.

  The SIM/HA shall set the CAM Status field in the specified CCB to Request Aborted by Host and return the CCB by the mechanisms specified within the CCB.

- Contained within the SIM/HA queues for the addressed Logical Unit, and is an I/O process within the Logical Unit.

  The SIM/HA detects a SCSI bus phase other than BUS FREE in response to the initiators ABORT or ABORT TAG message. The SIM/HA shall set the CAM Status field for the specified CCB to Unable to Abort Request and shall complete processing for the specified CCB as specified in this standard.

  The SIM/HA detects a SCSI bus phase of BUS FREE in response to the initiators ABORT or ABORT TAG message. The SIM/HA shall set the CAM Status field for the specified CCB to Request Aborted by Host and shall complete processing for the specified CCB as specified in this standard.

The supplier of the XPT shall define the Abort SCSI Command CCB structure as follows:

```
typedef struct ccb_abort
    {
    CCB_HEADER3 ccb_header3;                /* Header information fields */
    CCB_HEADER3 *cam_abort_ch;          /* Pointer to the CCB to abort */
    } CCB_ABORT;
```

11.7.11.1 Member Descriptions for Abort SCSI Command

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
  - cam_func_code;
    This member shall contain the XTP_CAM-3-CCB function code.
  - cam3_func_code;
    This member shall contain the XPT_ABORT function code;
  - port_id;
    This member shall contain a port number (e.g., SCSI bus number).
  - addr_spec1;
    This member shall contain the SCSI target specifier.
  - addr_spec2;
    This member shall contain the SCSI Logical Unit specifier.
  - cam_sim_generation;
    This member shall reflect the SIM bind generation number that was returned for the current bind operation.

  - cam_sim_bhandle;
    This member shall reflect the SIM bind handle that is return for the current bind operation.

  - cam_flags;
    There are no CAM flags for this function. The state of the queue for this device (e.g., frozen or not frozen) shall be controlled by the CAM flags of the CCB that is being aborted.

- cam_abort_ch;
  CCB to be aborted pointer

11.7.11.2 Returns for Abort SCSI Command

This function shall return a CAM Status other than Request in Progress. The CAM Status shall be one of the following:

- Request Completed without Error indicates that the Path ID is valid.
- Invalid Path ID indicates that the Path ID is invalid.
- No Bind indicates that either no Bind has been established or Bind has been lost.

11.7.12 CAM-3 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 Clause XXX).

The supplier of the XPT shall define the Reset SCSI Bus CCB structure as follows:

```
typedef struct ccb_resetbus3
    {
    CCB_HEADER3 ccb_header3;              /* Header information fields */
    } CCB_RESETBUS3;
```

11.7.12.1 Member Descriptions for Reset SCSI Bus

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
  - cam_func_code;
    This member shall contain the XTP_CAM-3-CCB function code.

  - cam3_func_code;
    This member shall contain the XPT_RESET_BUS function code;

  - port_id;
    This member shall contain a port number (e.g., SCSI bus number).

  - cam_flags;
    There are no CAM flags for this function.

11.7.12.2 Returns for Reset SCSI Bus

This function shall return a CAM Status other than Request in Progress. The CAM Status shall be one of the following:

- Request Completed without Error indicates that the Path ID is valid.
- Invalid Path ID indicates that the Path ID is invalid.

The actual failure or success of the Reset SCSI Bus function is indicated by the asynchronous callback information.

11.7.13 CAM-3 Reset SCSI Device

This function is used to reset the specified SCSI target. This function should not be used in normal operation. This request shall always result in a BUS DEVICE RESET message being issued over the SCSI Bus (see Clause XXX).

The supplier of the XPT shall define the Reset SCSI Device CCB structure as follows:

```
typedef struct ccb_resetdev3
    {
    CCB_HEADER3 ccb_header3;            /* Header information fields */
    } CCB_RESETDEV3;
```

11.7.13.1 Member Descriptions for Reset SCSI Device

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
    - cam_func_code;
      This member shall contain the XTP_CAM-3-CCB function code.
    - cam3_func_code;
      This member shall contain the XPT_RESET_DEV function code;
    - port_id;
      This member shall contain a port number (e.g., SCSI bus number).
    - addr_spec1;
      This member shall contain the SCSI target specifier.
    - addr_spec2;
      This member shall contain the SCSI Logical Unit specifier.
    - cam_sim_bhandle;
      This member shall reflect the SIM bind handle that is return for the current bind operation.
    - cam_flags;
      There are no CAM flags for this function.

11.7.13.2 Returns for Reset SCSI Device

This function shall return a CAM Status other than Request in Progress. The CAM Status shall be one of the following:

- Request Completed without Error indicates that the Path ID is valid.
- Invalid Path ID indicates that the Path ID is invalid.
- No Bind indicates that either no Bind has been established or Bind has been lost.

The actual failure or success of the Reset SCSI Device function is indicated by the asynchronous callback information.

11.7.14 CAM-3 Terminate I/O Process

This function requests that a SCSI 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 expects to terminate. The SIM/HA shall issue an TERMINATE I/O PROCESS message on the SCSI bus based on one of the following conditions:

- The CCB identified in the CCB to be Terminated Pointer field is an established I_T_L nexus I/O process (untagged command currently active).

- The CCB identified in the CCB to be Terminated Pointer field is an established I_T_L_Q nexus I/O process.

If a contingent allegiance condition is in effect for the specified Execute I/O Request CCB (e.g., a SCSI status has been returned from the Logical Unit of CHECK CONDITION or COMMAND TERMINATED), then the EXECUTE SCSI I/O REQUEST CCB shall complete normally as specified by this standard. If autosense is specified for the EXECUTE SCSI I/O REQUEST CCB then the SIM/HA shall retrieve the autosense data.

This request does not necessarily result in an TERMINATE I/O PROCESS message being issued over the SCSI Bus if the CCB identified is not an established I/O process.

The supplier of the XPT shall define the Terminate I/O Process CCB structure as follows:

```
typedef struct ccb_termio3
    {
    CCB_HEADER3 ccb_header3;                    /* Header information fields */
    CCB_HEADER3 *cam_termio_ch;         /* Pointer to the CCB to terminate */
    } CCB_TERMIO33;
```

11.7.14.1 Member Descriptions for Terminate I/O Process

- Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.
    - cam_func_code;
      This member shall contain the XTP_CAM-3-CCB function code.
    - cam3_func_code;
      This member shall contain the XPT_TERM_IO function code;
    - port_id;
      This member shall contain a port number (e.g., SCSI bus number).
    - addr_spec1;
      This member shall contain the SCSI target specifier.
    - addr_spec2;
      This member shall contain the SCSI Logical Unit specifier.
    - cam_sim_generation;
      This member shall reflect the SIM bind generation number that was returned for the current bind operation.
    - cam_sim_bhandle;
      This member shall reflect the SIM bind handle that is return for the current bind operation.

- cam_flags;
  There are no CAM flags for this function. The state of the queue for this device (e.g., frozen or not frozen) shall be controlled by the CAM flags of the CCB that is being aborted.

– cam_termio_ch;
  CCB to be Terminated Pointer

11.7.14.2 Returns for Terminate I/O Process

This function shall return a CAM Status other than Request in Progress. The CAM Status shall be one of the following:

- Request Completed without Error indicates that the Path ID is valid.
- Invalid Path ID indicates that the Path ID is invalid.
- No Bind indicates that either no Bind has been established or Bind has been lost.

The specified Execute I/O Request CCB may be in one of the following states which ascertains the ultimate results of the Terminate I/O Process function.

- Not contained within the SIM/HA queues for the addressed Logical Unit.

  The SIM/HA shall not be responsible for any further processing for the specified CCB.

- Contained within the SIM/HA queues for the addressed Logical Unit, but is not an I/O process within the Logical Unit.

  The SIM/HA shall set the CAM Status field in the specified CCB to Terminate I/O Process and return the CCB by the mechanisms specified within the CCB.

- Contained within the SIM/HA queues for the addressed Logical Unit, and is an I/O process within the Logical Unit.

  The SIM/HA receives a MESSAGE REJECT message in response to the initiators TERMINATE I/O PROCESS message. The SIM/HA shall set the CAM Status field for the specified CCB to Unable to TERMINATE I/O Process when the I/O Process is completed as specified in ANSI X3.131-1994 and shall complete processing for the specified CCB as specified in this standard.

  The SIM/HA receives a SCSI-2 status byte other than COMMAND TERMINATED for the specified CCB. The SIM/HA shall set the CAM Status field for the specified CCB to Unable to TERMINATE I/O Process when the I/O Process is completed as specified in ANSI X3.131-1994 and shall complete processing for the specified CCB as specified in this standard.

  The SIM/HA receives a SCSI-2 status byte of COMMAND TERMINATED for the specified CCB. The SIM/HA shall set the CAM Status field to Request Completed with Error when the I/O Process is completed as specified in ANSI X3.131-1994 and shall complete processing for the specified CCB as specified in this standard (e.g., Autosense mechanisms).

11.8 CAM-3 Control Blocks to Request I/O

Peripheral drivers should make all of their SCSI I/O requests using this CCB, which is designed to take advantage of all features of SCSI that can be provided by virtually any HA/SIM combination. The CCB is common in format and structure for the following function codes:

– Execute SCSI I/O (see Clause XXX and Table XXX for further information)
– Execute Target I/O (see Clause XXX and Table XXX for further information)
– Accept Target I/O (see Clause XXX and Table XXX for further information)
– Continue Target I/O (see Clause XXX and Table XXX for further information)
– Execute Engine Request (see Clause XXX and Table XXX for further information)

11.8.1 CAM-3 Execute SCSI I/O Request

This function typically returns a CAM Status of Request in Progress, indicating that the request was queued successfully. Request completion can be ascertained by polling for a CAM status other than Request in Progress or through use of the Callback on Completion field. Polling for completion of a CCB is not recommended.

The SCSI Command Descriptor Block can be either a contiguous array of 12 bytes or can be a pointer to a contiguous array of bytes. The supplier of the XPT shall define the SCSI Command Descriptor Block for the Execute SCSI I/O Request CCB structure as follows:

```
typedef union cdb_un
    {
    CAM_U8 *cam_cdb_ptr;                /* Pointer to the CDB bytes to send */
    CAM_U8 cam_cdb_bytes[ 16 ];         /* Array of bytes for the CDB to send */
    } CDB_UN;
```

The supplier of the XPT shall define the Execute SCSI I/O Request CCB structure as follows:

```
typedef struct ccb_scsiio3
    {
    CCB_HEADER3 ccb_header3;            /* Header information fields */
    CCB_HEADER3 *cam_next_ccb;      /* Ptr to the next CCB for action */
    CAM_VOID_OFFSET *cam_req_map;   /* Ptr for mapping info on the Req. */
    CAM_VOID (*cam_cbfcnp)();       /* Callback on completion function */
    CAM_U8 *cam_data_ptr;           /* Pointer to the data buf/SG list */
    CAM_U32 cam_dxfer_len;          /* Data xfer length */
    CAM_U8 cam_cdb_len;             /* Number of bytes for the CDB */
    CAM_U8 cam_reserved1;           /* Reserved for alignment */
    CAM_U16 cam_sglist_cnt;         /* Num. of scatter gather list entries */
    CAM_U32 cam_vu_field;           /* Vendor Unique field/
    CAM_U8 cam_scsi_status;         /* Returned SCSI device status */
    CAM_U8 cam_reserved2;           /*  Reserved for alignment */
    CAM_U16 cam_sense_resid;        /* Autosense resid length: 2's comp */
    CAM_I32 cam_resid;              /* Transfer residual length: 2's comp */
    CAM_U32 cam_timeout;            /* Timeout value */
    CDB_UN3 cam_cdb_io;             /* Union for CDB bytes/pointer */
    CAM_U8 *cam_msg_ptr;            /* Pointer to the message buffer */
    CAM_U16 cam_msgb_len;           /* Num. of bytes in the message buf */
    CAM_U16 cam_vu_flags;           /* Vendor unique flags */
    CAM_U8 cam_tag_action;          /* What to do for tag queuing */
    CAM_U8 cam_reserved3[3];        /* Reserved for alignment */
    CAM_U32 cam_tag_id;             /* Tag ID */
    CAM_U32 cam_initiator_id[2];    /* Initiator ID target operations */
    CAM_U16 cam_sense_len;          /* Number of bytes to request for Autosense */
    CAM_U8 cam_reserved4;           /* Reserved for alignment */
    CAM_U32 cam_sim_sense[16];            /* Working area for a SIM for retrieving sense data */
    CAM_U8 cam_sense_buf[256];            /* Sense data buffer */

    } CCB_SCSIIO3;
```

11.8.1.1 Member Descriptions for Execute SCSI I/O Request

– Required information for the CCB_HEADER3 members that shall be set by the originator of the CCB.

- cam_func_code;
  This member shall contain the XTP_CAM-3-CCB function code.

- cam3_func_code;
  This member shall contain the XPT_SCSI_IO function code;

- port_id;
  This member shall contain a port number (e.g., SCSI bus number).

- addr_spec1;
  This member shall contain the SCSI target specifier.

- addr_spec2;
  This member shall contain the SCSI Logical Unit specifier.

- cam_sim_generation;
  This member shall reflect the SIM bind generation number that was returned for the current bind operation.

- cam_sim_bhandle;
  This member shall reflect the SIM bind handle that is return for the current bind operation.

- cam_flags;
  This member contains bit settings as described to indicate special handling of the requested function. The setting of required behavior specifics as specified by the CAM flags should be done with a logical or function. An example of this is a peripheral wishes to indicate that the data transfer direction is into the initiator and that scatter/gather list is valid. The peripheral driver may accomplish this by the following:

  ```
  ccb->ccb_header3.cam_flags |= (CAM_DIR_IN | CAM_SCATTER_VALID);
  ```

  The flags shall be defined as follows:

  - CAM Direction flags shall specify the direction of the data transfer in relation to the SCSI initiator. These encoded bits identify the direction of data movement during data transfer.
    - ◊ Reserved
      ```
      #define CAM_DIR_RESV          0x00000000
      ```
    - ◊ Data direction in (read from Logical unit);
      ```
      #define CAM_DIR_IN            0x00000040
      ```
    - ◊ Data direction out (write to logical unit);
      ```
      #define CAM_DIR_OUT           0x00000080
      ```
    - ◊ Data direction none (no data transfer);
      ```
      #define CAM_DIR_NONE          0x000000C0
      ```
  - The Disable Autosense feature flag has been deleted. Autosense is now a mandatory feature in CAM-3.
  - Scatter/gather list is valid 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 this case the Data Buffer Pointer refers to a list of addresses and lengths in bytes at each address to which the data is to be transferred.
    ```
    #define CAM_SCATTER_VALID     0x00000010
    ```

    The format of the SG List shall be defined as follows:
    ```
    typedef struct sg_elem
        {
        CAM_U8 *cam_sg_address;        /* Scatter/Gather address */
        U32 cam_sg_count;            /* Scatter/Gather byte count */
        } SG_ELEM;
    ```
  - 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 is polling for a CAM Status other than Request in Progress status, which indicates completion of the request.
    ```
    #define CAM_DIS_CALLBACK      0x00000008
    ```
  - Linked CDB - When set to 1, this CDB is a SCSI 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 Clause XXX for further information.
    ```
    #define CAM_CDB_LINKED        0x00000004
    ```
  - Tag Queue actions are enabled when set to a 1. The SCSI CDB contained or pointed to within

the SCSI I/O REQUEST CCB shall have Tag Queuing attributes. See Clause XXX for further information.
#define CAM_QUEUE_ENABLE 0x00000002

- CDB member is a pointer - When set to a 1, the first four bytes of the CDB field shall contain a pointer to the location of the CDB
  #define CAM_CDB_POINTER 0x00000001

- Disable SCSI bus disconnects when set to a 1. The disconnect capability of SCSI is disabled. The default of 0 sets bit 6=1 in the SCSI IDENTIFY message.
  #define CAM_DIS_DISCONNECT 0x00008000

- Initiate Synchronous Transfers when set to a 1, indicates the SIM shall negotiate for the best transfer parameters it is capable of with the target, and wherever possible execute the negotiated transfer parameters (synchronous, fast, wide transfers). The peripheral driver shall not set this bit and the bit CAM_DIS_SYNC. These bits are mutually exclusive.
  #define CAM_INITIATE_SYNC 0x00004000

- Disable Synchronous Transfers when set to a 1, indicates the SIM shall negotiate for the least transfer parameters (asynchronous, narrow transfers) if the SIM previously negotiated synchronous. If unable to negotiate synchronous (best transfer parameters) or negotiation has not yet been attempted, the SIM shall not initiate negotiation. The peripheral driver shall not set this bit and the bit CAM_INITIATE_SYNC. These bits are mutually exclusive.
  #define CAM_DIS_SYNC 0x00002000

- SIM Queue Priority when set to a 1, the SIM shall place this CCB ahead of all CCB operations with normal priority sent to the Logical Unit and at the tail of the Logical Unit's (FIFO order) priority internal queue. When set to a 0 the SIM shall place the CCB at the tail of all CCB operations with normal priority sent to the Logical Unit.
  #define CAM_SIM_QHEAD 0x00001000

- SIM Queue Freeze - When set to a 1, the SIM shall place its internal Logical Unit 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. Refer to Table XXX for further information. The peripheral driver shall not set this bit and the bit CAM_SIM_QFRZDIS. These bits are mutually exclusive.
-
  #define CAM_SIM_QFREEZE 0x00000800

- SIM Queue Freeze Disable - When set to a 1, the SIM queue freeze mechanism shall be disabled (i.e., the SIM queue shall not be frozen for the Logical Unit addressed in this CCB in the event of a CAM Status other than Request Complete without Error). Refer to Table XXX for further information. The peripheral driver shall not set this bit and the bit CAM_SIM_QFRZDIS. These bits are mutually exclusive.
  #define CAM_SIM_QFRZDIS 0x00000400

CAM_SIM_QFREEZE	CAM_SIM_QFRZDIS	ACTION
0	0	SIM Queue Frozen if CAM Status not Request Complete w/o Error
0	1	SIM Queue not Frozen for all CAM Statuses
1	0	SIM Queue Frozen when CCB completes
1	1	Invalid setting of CAM Flags

Table 13 SIM Queue Actions

- Engine Synchronize - This bit is used in conjunction with the Direction in or out settings to flush

any residual bits before terminating engine processing (see Clause XXX for further information).
#define CAM_ENG_SYNC 0x00000200

- Reserved bit value 0x00000100

- Scatter/Gather Host/Engine is used to accommodate buffering associated with an HA Engine. The flag when set to a 1 is used to specify that the normal data buffer pointer is actually a physical address in the buffer space of the engine. When set to a 0 the data buffer is in host memory. The format of the address (physical/virtual) is controlled by the CAM_DATA_PHYS bit flag.
  #define CAM_ENG_SGLIST 0x00800000

- The following CAM flags bit definitions describe the memory type (virtual/physical) of certain described members of the Execute SCSI I/O Request CCB. The Pointer fields are set up to have one characteristic. If a bit is set to 1 it shall indicate the pointer field described contains a Physical Address. If set to 0 it shall indicate the pointer contains a Virtual Address. If the SIM needs an address in a different form to that provided, it shall be converted by the SIM (using OSD facilities) and stored in Private Data
  - ◊ CDB pointer is physical when set to a 1 shall indicate that the pointer in the cam_cdb_io member is physical.
    #define CAM_CDB_PHYS 0x00400000
  - ◊ Data Buffer/Scatter Gather pointer is physical when set to a 1 shall indicate that the pointer in the cam_data_ptr member is physical.
    #define CAM_DATA_PHYS 0x00200000
  - ◊ Message buffer pointer is physical when set to a 1 shall indicate that the pointer in the cam_sense_ptr member is physical.
    #define CAM_SNS_BUF_PHYS 0x00100000
  - ◊ Message buffer pointer is physical when set to a 1 shall indicate that the pointer in the cam_msg_ptr member is physical.
    #define CAM_MSG_BUF_PHYS 0x00080000
  - ◊ Next CCB pointer is physical when set to a 1 shall indicate that the pointer in the cam_next_ccb member is physical.
    #define CAM_NXT_CCB_PHYS 0x00040000
  - ◊ Callback on Completion pointer is physical when set to a 1 shall indicate that the pointer in the cam_cbfcnp member is physical.
    #define CAM_CALLBCK_PHYS 0x00020000
  - ◊ Scatter/Gather list pointers within the array of SG_ELEM structure when set to a 1 shall indicate that the pointer in the cam_sg_address member is physical.
    #define CAM_SG_LIST_PHYS 0x00010000

- The following CAM flags bit definitions describe both Phase-Cognizant mode and Host Target mode. The actual meaning of the flags shall be dependent upon whether Phase-Cognizant mode or Host Target mode is specified as dictated by the CAM_TGT_PHASE_MODE flag. See below for correct setting of the flag.

◊ The following flags shall apply to Phase-Cognizant mode. The Phase-Cognizant mode only flags are only active on ENABLE LUN or EXECUTE TARGET I/O CCBs. See Clause XXX for complete details.

- ⇒ 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.

    - ♦ Data buffer valid when set to a 1 shall indicate that the data as specified in cam_data_ptr member is valid for use.
      #define CAM_DATAB_VALID 0x80000000

    - ♦ Status valid when set to a 1 shall indicate that the data as specified in cam_scsi_status member is valid for use.
      #define CAM_STATUS_VALID 0x40000000

    - ♦ Message Buffer valid when set to a 1 shall indicate that the data as specified in cam_msg_ptr member is valid for use.
      #define CAM_MSGB_VALID 0x20000000

- ⇒ Phase-Cognizant mode when set to a 1 shall indicate that the peripheral driver indicates that it wants Phase-Cognizant functionality. If target operations are supported, when set to 1, the SIM shall operate in Phase-Cognizant Mode, otherwise it shall operate in Host Target Mode.
  #define CAM_TGT_PHASE_MODE 0x08000000

- ⇒ Target CCB Available when set to 1, this bit indicates that the SIM can use this CCB to process this request. A value of 0 indicates that this CCB is not available to the SIM.
  #define CAM_TGT_CCB_AVAIL 0x04000000

- ⇒ Autodisconnect when set to 1, this bit disables autodisconnect. The default of 0 causes the SIM/HA to automatically disconnect, if the IDENTIFY message indicates discpriv is set.
  #define CAM_DIS_AUTODISC 0x02000000

- ⇒ Autosave - When set to 1, this bit disables autosave feature for Phase-Cognizant Mode. The default of 0 causes the XPT/SIM to automatically send a SAVE DATA POINTER message on an autodisconnect.
  #define CAM_DIS_AUTOSRP 0x01000000

◊ The following flags shall apply to Host Target mode. The Host Target Mode flags are shall only be valid for the ENABLE LUN, ACCEPT TARGET I/O and CONTINUE TARGET I/O CCBs. See Clause XXX for complete details.

- ⇒ Send Status when set to a 1, this bit directs the SIM/HA that it shall go to status phase after data phase (if there is a data phase for this CCB) and send the SCSI status byte contained within this CCB in the cam_scsi_status member.
  #define CAM_SEND_STATUS 0x80000000

- ⇒ Disconnects Mandatory when set to a 1, this bit directs the SIM/HA that it shall disconnect from the SCSI bus for each CCB processed for the enabled Logical Unit
  #define CAM_DISCONNECT 0x40000000

- ⇒ Terminate I/O when set to a 1, this bit informs the SIM/HA that the Host Target Mode

peripheral driver is supporting the TERMINATE I/O PROCESS SCSI message.
#define CAM_TERM_IO 0x20000000

⇒ Phase-Cognizant mode when set to a 0 shall indicate that the peripheral driver indicates that it wants Host Target mode functionality. If target operations are supported, when set to 0, the SIM shall operate Host Target mode in, otherwise it shall operate in Phase-Cognizant mode.
#define CAM_TGT_PHASE_MODE 0x08000000

– cam_req_map;
The request map information member is a pointer to an OSD data structure which is associated with the original I/O request.

– cam_next_ccb;
This member 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 linking commands.

– cam_cbfcnp;
See Clause XXX for callback on completion of a queued CCB.

– cam_data_ptr;
The cam data buffer pointer member is either a pointer to a logically contiguous buffer or a pointer to an array of SG_ELEM structures. The format type is defined by the CAM flag of CAM_SCATTER_VALID. The data buffers described shall either contain data to be transfer to the Logical Unit or to be used receive data from the Logical Unit.

– cam_dxfer_len;
The cam data transfer length contains the length in bytes of the data to be transferred.

– cam_cdb_len;
For EXECUTE SCSI I/O REQUEST CCBs the CDB length member shall contain the length in bytes of the CDB. For ACCEPT TARGET I/O CCBs and EXECUTE TARGET I/O CCBs this member shall contain the length in bytes of the buffer for CDB placement

– cam_sglist_cnt;
The number of scatter/gather entries member shall contain the number of SG_ELEM(s) pointed to by cam_data_ptr member if the CAM flag of CAM_SCATTER_VALID is a 1. The member shall be 0 if the CAM flag of CAM_SCATTER_VALID is a 0.

– cam_vu_field;
The vendor unique member is defined in the SIM vendor specification.

– cam_scsi_status;
The SCSI status member contains the status byte returned by the SCSI Logical Unit after the command is completed as defined in ANSI X3.131-1994. This field shall be valid for the CAM Status of Request Complete without Error and Request Complete with Error.

– cam_sense_valid;
The cam sense valid member shall contain the number of bytes that have been obtained for autosense.

– cam_resid;

The data residual length member contains the difference in twos complement form of the number of data bytes transferred by the HA for the SCSI command compared with the number of bytes requested by the CCB cam_dxfer_len member. This is calculated by the total number of bytes requested to be transferred by the CCB minus the actual number of bytes transferred by the HA.

– cam_cdb_io;
The cam CDB member either contains the SCSI CDB (command descriptor block), or a pointer to the CDB, to be dispatched. The member shall be define by the CDB_UN data type (union definition).

– cam_timeout;
The timeout member contains the maximum period in seconds that an issued SCSI command 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. The timeout value member is on a per CCB basis, and is measured from successful selection to command completion. If the CCB has timed out and the command has not completed (e.g., COMMAND COMPLETE or LINKED COMMAND COMPLETE message has not been received for the CCB), the SIM/HA shall reselect the addressed Logical Unit and issue an ABORT message or an ABORT TAG message. If the command that timed out is an I_T_L I/O process then an ABORT message shall be issued by the SIM/HA. If the command that timed out is an I_T_L_Q I/O process then an ABORT TAG message for the identified I_T_L_Q I/O process shall be issued by the SIM/HA.

– cam_msg_ptr;
The message buffer member contains a pointer to a buffer containing messages. This pointer is only valid for use in target mode (see Clause XXX for further information).

– cam_msgb_len;
The message buffer length contains the length in bytes of the cam_msg_ptr member which is to be used to hold message information in the event that the peripheral drivers needs to issue any SCSI Messages. This field is exclusive to target mode operation (see Clause XXX for further information).

– cam_vu_flags;
The vendor unique flags member is vendor unique and uses for this member are defined in the SIM vendor specification.

– cam_tag_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. For a description of the tagged command queuing philosophy see SCSI-2 or SCSI-3

When the CAM_QUEUE_ENABLE bit in the cam_flags member is set, the CDB issued by the SIM shall be associated with the queue action specified as:

- 20h = SIMPLE QUEUE TAG request
- 21h = HEAD of QUEUE TAG request
- 22h = ORDERED QUEUE TAG request

– cam_tag_id;
The tag identifier member indicates the SCSI TAG QUEUE ID that this CCB is in response to if tagged queue operation. This field is valid only for Host Target Mode operation

– cam_initiator_id

– The initiator identifier member indicates which SCSI initiator this CCB is in response to. This field is only valid for target mode operation. This member is an array of 2 CAM_U32s to contain the SCSI target specifier. The cam_initiator_id[0] member shall contain the lower 32 bits (least significant portion) of the SCSI target specifier. . The cam_initiator_id[1] member shall contain the upper 32 bits (most significant portion) of the SCSI target specifier.

- cam_sense_len;
  This member contains the number of bytes to request for autosense information. This value shall not exceed the number 256. This value may not be valid for all interconnect protocols. This value is only valid for those interconnects that explicitly denote the transfer size of the sense data. For example SIP but not FCP.

- cam_sim_sense;
  This member is an array of 16 CAM_32s that is available to a SIM as working space for the retrieval of autosense information.

- cam_sense_buf;
  This member is an array of 256 CAM_8s that is available for the placement of autosense information. A SIM shall place all autosense information obtained from a device in this buffer. The SIM shall not place more than 256 CAM_U8s into the cam_sense_buf.

11.8.1.2 Returns for Execute SCSI I/O Request

The final CAM Status shall be one of the following:

- Request Completed without Error: the request has completed and no error condition was encountered.
- Request Aborted by Host: the request was aborted by the SIM/HA as instructed by the peripheral driver.
- Unable to Abort Request: the SIM was unable to abort the request as instructed by the peripheral driver.
- Request Completed with Error: the request has completed and an error condition was encountered.
- CAM Busy: CAM unable to accept request at this time.
- Invalid Request: the request has been rejected because it is invalid.
- Invalid Path ID: indicates that the Path ID is invalid.
- Unable to Terminate I/O Process: the SIM was unable to terminate the request as instructed by the peripheral driver.
- Target Selection Timeout: The target failed to respond to selection.
- Command Timeout: the specified command did not complete within the timer value specified in the CCB. Prior to reporting this status the SIM/HA shall ensure the command is no longer active in the target.
- Message Reject Received: The SIM/HA received a SCSI MESSAGE REJECT message.
- SCSI Bus Reset Sent/Received: The SCSI operation was terminated at some point because the SCSI bus was reset.
- Uncorrectable Parity Error Detected: An uncorrectable SCSI bus parity error was detected.
- Autosense Request Sense Command Failed: The SIM/HA attempted to obtain sense data and failed.
- No HA Detected: HA no longer responding to SIM (assumed to be a hardware problem).
- Data Overrun: target transferred more data bytes than peripheral driver indicated in the CCB.
- Unexpected Bus Free: an unexpected bus free condition occurred.
- Target Bus Phase Sequence Failure: the Logical Unit failed to operate in compliance with ANSI X3.131-1994.
- CCB Length Inadequate: more private data area is required in the CCB (refer to Clause 9.2.3 for further clarification) .

- Cannot Provide Requested Capability: resources are not available to provide the capability requested in the CAM Flags.
- Bus Device Reset Sent: this CCB was terminated because a BUS DEVICE RESET message was sent to the target.
- Terminate I/O Process: this CCB terminated due to a Terminate I/O Process Request CCB was received by the SIM/HA and the CCB was not an I/O Process within the Logical Unit.
- Unrecoverable Host Bus Adapter Error: this CCB was terminated because of a hardware error detected by the HA. The error does not indicate a SCSI bus problem but an error within the HA or host.

11.9 Command Linking (optional)

The SIM/HA supports SCSI's ability to link commands in order to guarantee the sequential execution of several requests. This function requires that both the SIM/HA and the involved Logical Unit support the SCSI link capability. The SIM/HA shall indicate its support for command linking by setting the Linked Commands bit in response to a PATH INQUIRY CCB (see Clause 9.2.3 Path Inquiry for further details).

To utilize linking, a chain of CCBs is built with the next CCB pointer being used to link the CCBs together. The CAM Linked CDB flag bit shall be set in all CCBs but the last in the chain. The first CCB in the linked list of CCBs shall have a valid Callback on Completion field set and the CAM Flag of Disable Callback on completion cleared. When a SCSI Logical Unit returns the LINKED COMMAND COMPLETE message or LINKED COMMAND COMPLETE (WITH FLAG) message, the next CCB is processed, and its associated CDB is dispatched. Any SCSI status other than INTERMEDIATE or INTERMEDIATE CONDITION MET returned by the Logical Unit on a linked command shall break the chain. The SIM/HA shall callback the peripheral driver using the first CCB's Callback on Completion field when the linked list of CCBs is completed or when the chain is broken.

The peripheral driver shall:

- Build a valid list of EXECUTE I/O CCBs with the CAM Linked CDB flag bit set except for the last CCB in the linked CCB chain.

  Note 17

  The peripheral driver is responsible for the correct settings of the Flag and Link bits in the control field of all CDBs within CCBs.

- Call xpt_action() with the address of the first CCB as the argument.

- Wait for completion of the linked CCB list.

- On callback ascertain completion status by starting with the first CCB in the linked list and examining the CAM Status field, then proceeding to the next in the list. The following CCB statuses shall be used to ascertain completion of a individual CCB and the list.

  - The CAM Status of Request completed without Error indicates that this CCB completed successfully and if the CAM Linked CDB flag bit is clear the CCB linked list completed without error.

  - The CAM Status of Request In Progress indicates the chain was broken and this CDB contained within this CCB was not issued to the Logical Unit.

  - The CAM Status of Request Completed with Error indicates that the Logical Unit sent a SCSI status other than INTERMEDIATE or INTERMEDIATE CONDITION MET for this CCB.

  - All other CAM Statuses indicate that another CAM condition occurred while processing this CCB and the chain was broken at this point.

The peripheral driver may:

- Monitor chain CCB processing by examining the CAM Status field of each CCB within the list but shall not attempt to modify any CCB within the list until callback on completion by the SIM/HA.
- Abort the linked CCB chain by issuing an Abort SCSI Command function or Terminate I/O Process Request function to the first CCB within the list.

The SIM/HA shall for linked CCB lists:

- Validate each CCB within the list (e.g. the first CCB in the linked list of CCBs shall have a valid Callback on Completion field set and the CAM flag of Disable Callback on completion cleared and the CAM Linked CDB flag bit shall be set in all CCBs but the last in the chain).
- Establish an I/O Process for the CCB list (e.g. issue the first CCB then proceeding to the next CCB when a LINKED COMMAND COMPLETE message or a LINKED COMMAND COMPLETE (WITH FLAG) message) is received.
- A COMMAND COMPLETE message for the any CCB but the last CCB shall break the chain.
- A COMMAND COMPLETE message for the last CCB shall indicate successful completion of the chain.
- When each CCB within the chain completes, set all appropriate fields within the CCB.
- When the chained CCB list is completed or broken, callback the peripheral driver using the first CCB Callback on Completion field.
- If the peripheral driver terminates or aborts the first CCB within the chain:
  - If the CCB chain to be aborted or terminated has a current I/O process, it shall abort or terminate the CCB as specified by this standard and break the chain (e.g., the current I/O process CCB has a CAM Status of Request Aborted by Host or Terminate I/O Process).
  - If the CCB chain to be aborted or terminated does not have a current I/O Process, it shall break the chain.
  - If the CCB specified for a CCB chain is not the first CCB of a chain, it shall ignore the request.
  - The SIM/HA shall ensure that the SCSI bus and the Logical Unit are not left in a hung state as specified by ANSI X3.131-1994.