ENDL
TEXAS

Date: 16 March 2002
To: T10 Technical Committee
From: Ralph O. Weber
Subject: This is not a revision of SAM-3

**Revision -1**
**16 March 2002**

# SCSI Architecture Model - 3 (SAM-3)

This is a T10 proposal for what SAM-3 might look like. This is not a working draft revision of SAM-3.

The intent of this proposal is to demonstrate how expeditious development of SAM-3 can be facilitated by use of shared source files between SAM-2 and SAM-3 until SAM-2 completes public review. This proposal is produced using shared SAM-2 source files and demonstrates both addition and removal of text for SAM-3 using the FrameMaker Conditional Text feature. See the revision history for details of what has been added and removed.

If this had been an actual SAM-3 revision, this page would have been replaced with a real T10 working draft cover page.

T10 Technical Editor:

Ralph O. Weber
ENDL Texas
18484 Preston Road
Suite 102 PMB 178
Dallas, TX 75252
USA

Telephone: 214-912-1373
Facsimile: 972-596-2775
Email: ROWeber@ACM.org

## Points of Contact:

**T10 Chair**
John B. Lohmeyer
LSI Logic
4420 Arrows West Drive
Colorado Springs, CO 80907-3444
Tel: (719) 533-7560
Fax: (719) 533-7183
Email: lohmeyer@t10.org

**T10 Vice-Chair**
George O. Penokie
IBM
3605 Highway 52 N
MS: 2C6
Rochester, MN 55901
Tel: (507) 253-5208
Fax: (507) 253-2880
Email: gop@us.ibm.com

**INCITS Secretariat**
INCITS Secretariat
1250 Eye Street, NW Suite 200
Washington, DC 20005
Telephone: 202-737-8888
Facsimile: 202-638-4922
Email: INCITS@itic.org

**T10 Web Site** www.t10.org

**T10 Reflector** To subscribe send e-mail to majordomo@T10.org with 'subscribe' in message body
To unsubscribe send e-mail to majordomo@T10.org with 'unsubscribe' in message body
Internet address for distribution via T10 reflector: T10@T10.org

**Document Distribution**

INCITS Online Store
managed by Techstreet
1327 Jones Drive
Ann Arbor, MI 48105
http://www.techstreet.com/INCITS.html
Telephone: 1-734-302-7801 or
1-800-699-9277
Facsimile: 1-734-302-7811

or

Global Engineering
15 Inverness Way East
Englewood, CO 80112-5704
http://global.ihs.com/
Telephone: 1-303-792-2181 or
1-800-854-7179
Facsimile: 1-303-792-2192

ANSI (r)
INCITS.***:200x

Draft

# American National Standards for Information Systems -

# SCSI Architecture Model - 2 (SAM-2)

Secretariat
**National Committee for Information Technology Standards**

Approved mm dd yy

**American National Standards Institute, Inc.**

## Abstract

This standard specifies the SCSI Architecture Model. The purpose of the architecture is to provide a common basis for the coordination of SCSI standards and to specify those aspects of SCSI I/O system behavior that are independent of a particular technology and common to all implementations.

Draft

**American National Standard**

Approval of an American National Standard requires verification by ANSI that the requirements for due process, consensus, and other criteria for approval have been met by the standards developer. Consensus is established when, in the judgment of the ANSI Board of Standards Review, substantial agreement has been reached by directly and materially affected interests. Substantial agreement means much more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered and that effort be made toward their resolution.

The use of American National Standards is completely voluntary; their existence does not in any respect preclude anyone, whether he or she has approved the standards or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not confirming to the standards.

The American National Standards Institute does not develop standards and will in no circumstances give interpretation on any American National Standard in the name of the American National Standards Institute. Requests for interpretations should be addressed to the secretariat or sponsor whose name appears on the title page of this standard.

**CAUTION NOTICE:** This American National Standard may be revised or withdrawn at any time. The procedures of the American National Standards Institute require that action be taken periodically to reaffirm, revise, or withdraw this standard. Purchasers of American National Standards may receive current information on all standards by calling or writing the American National Standards Institute.

CAUTION: The developers of this standard have requested that holders 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, if any, patents may apply to this standard.

As of the date of publication of this standard and following calls for the identification of patents that may be required for the implementation of the standard, no such claims have been made. No further patent search is conducted by the developer or the publisher in respect to any standard it processes. No representation is made or implied that licenses are not required to avoid infringement in the use of this standard.

Published by
**American National Standards Institute**
**11 West 42nd Street, New York, NY 10036**



**Printed in the United States of America**

## Contents

## Tables

## Figures

# Revision Information

## 1 Approved Documents Included

The following T10 approved proposals have been incorporated SAM-3 up to and including this revision:

To the best of the technical editor's knowledge, all proposals for SAM-3 approved by T10 have been included in this revision.

## 2 Revision History

### 2.1 Revision -1 (6 March 2002, 02-119r0)

Revision -1 is just a demonstration of the shared sources concept for SAM-3 during the SAM-2 letter ballot and public review period. It is shared source with SAM-2 revision 23.

For the purposes of demonstration **only**, revision -1 eliminates all normative references to parallel SCSI and makes autosense mandatory.

### 2.2 Revision 0 (?? May 2002)

Revision 0 incorporates the following T10 approved proposals:

## 3 Plans for Future Revisions

This is a list of the work the technical editor considers required in future revisions of SAM-x.

### 3.1 Minor Changes

The terms "call", "procedure", and any related terms should have glossary definitions that clearly identify them as architectural abstractions. All of these concepts are wording conveniences used as shorthand by the architecture and model to express more complex concepts or concepts for which numerous implementations are possible. The technical editor also should search on the terms "call" and "procedure" to locate any uses and edit text at each usage point to clearly identify "call" and "procedure" as architectural model abstractions and not as indications of implementation requirements. In a similar vein, "protocol" is an architectural abstraction, however this may be better understood as an abstraction in the community of SCSI designers.

The term "service" appears to be an architectural abstraction too, it is defined totally on architectural abstractions ("calls" and "objects"). However, careful study is required to determine if "service" has some non-abstract, concrete meaning. If it does, the glossary definition should be changed.

Is it necessary to have definitions for "implementation option", "logical unit option", and "protocol option"? Surely, an option is an option is an option and the context in which the word "option" appears is sufficient to identify whose option is being discussed.

The technical editor wishes to remove the definition of "ended command". Strictly speaking, the term "ended command" is not used anywhere in the working draft, thus allowing removal of the definition as a strictly editorial

change. However, some consideration of the change appears prudent. The word “ended” is used frequently in the working draft, all uses appear to have the standard English meaning, but this thesis needs additional verification. Also, there is an “ended (task state)” that lacks a glossary definition and perhaps should have one.

The glossary definition of “layer” in uninformative, owing in part to the vague usage of “rank”. A clearer, more specific definition is needed.

Is the term “protocol service request” really so general as to require its being defined in terms of “call” (an architectural abstraction)? Also, the technical editor believes that “protocol service response” should be defined as “A reply to the upper level protocol …”, not “A reply from the upper level protocol …” Finally, would it be possible to cast both definitions in terms of specific entities from the SCSI roadmap, instead of “lower level protocol” and “upper level protocol” (see 3.2)?

Although it is used in the Foreword, Scope, and one figure title, the term “reference model” is little more than obfuscated wording for “model”. Could it be replaced?

Is “subsystem” really used as it is defined in the glossary? Many other SCSI standards use subsystem differently.

It seems that task management function names sometimes appear in all capitals and bold, not capitalized and bold as is stated in 3.4.

Why is the following sentence from the end of the first paragraph in 4.2 so important:

> “In such a model, each client or server is a single thread of execution which runs concurrently with all other clients or servers.”

Are the requirements on protocols really contained only in 5.4 and 6.9, as is stated in 4.1?

Change all usage of “remote procedure call” to “procedure call”, since “remote procedure call” is not defined.

Clause 5.3 is a mess. The data delivery services are given individual 5.3.x clauses but the command protocol services are not. 5.3.1 fails to identify the party responsible for establishing the parameters for the transfer of a buffer segment. The rule prohibiting input and output transfers by a single command is buried in a paragraph that starts with a discussion of buffer segmentation. The technical editor was very tempted to rewrite the whole clause during the revision 4 conversion, but wrote this reminder to himself instead.

5.6.4.1 seems to imply that other methods for controlling AER besides the Control mode page are acceptable. Is this really the intent of T10?

## 3.2 Substantial Changes

Is it really necessary for SAM-2 to place requirements on the contents of other standards? Would the SCSI documents set be just as well served if SAM-2 acted as a guide to what readers might expect to find in other SCSI standards? With these thoughts in mind, a few (but not necessarily all) specific instances of needed changes are noted:

a) The Foreword and Introduction clauses need to be modified to remove the work “requirements”; at the time of this writing, “capabilities” is the preferred replacement;
b) Most of 1.1 probably would be obsolete; and
c) A careful audit of the requirements statements will be needed to adjust those placing requirements on other standards.

The technical editor is considering a careful review of the working draft, with an eye toward overly abstract model abstractions. Examples are:

a) Overly general layering terms and discussions; and
b) Discussion of a new application client for each new request or task management function.

The layering seems overly general and thus confusing. SCSI has two (or at most three) layers. The question of two or three layers depends on whether the service delivery port is a layer. The two “main” layers are the command and control layer (application client, device server, and task manager) and the service delivery subsystem. The description appears amenable to substantial simplifications. LLP and ULP could disappear. Generalized interfaces could be replaced with a small number of specific interfaces. Does T10 see value in this kind of simplification?

The terms “SCSI application layer” and “SCSI protocol layer” appear to be redundant. Certainly, “SCSI application layer” is little more than a generalization of “application client”. Perhaps, “SCSI application layer” and “SCSI protocol layer” can be removed. As if this confusion were not enough, the definition of “Upper Layer Protocol” clearly ties it to the application layer. This further suggests that SCSI has only two protocol layers.

The technical editor wonders how useful it is to say that the architectural model presumes the creation of a new application client for each new request or task management function. It is difficult to see how this formalism serves to produce a better understanding of the real-world usage of SCSI. In fact, other text in the working draft acknowledges that this formalism may not relate to reality at all. If this change were made, it might also be possible to simplify the following statements in 4.3:

> “An application client represents a thread of execution whose functionality is independent of the interconnect and SCSI-3 protocol. In an implementation, that thread could correspond to the device driver and any other code within the operating system that is capable of managing I/O requests without requiring knowledge of the interconnect or SCSI-3 protocol.”

Is the following 4.2 statement rigorously true?

> “All allusions to a pending command or task management function in this standard are in the application client's frame of reference.”

The use of “conventional procedure call” in the following 4.2 statement is at odds with the SAM definitions of procedure call as a modeling mechanism.

> “From the client's standpoint, the behavior of a remote service invoked in this manner is indistinguishable from a conventional procedure call.”

If the following two 4.2 statements are true, why are confirmed services defined?

> “In this model, confirmation of successful request or response delivery by the sender is not required. The model assumes that delivery failures will be detected by the client's service delivery port.”

The technical editor suspects that “confirmed service” has multiple definitions.

## Foreword

This foreword is not part of American National Standard INCITS.***:200x.

The purpose of this standard is to provide a basis for the coordination of SCSI standards development and to define requirements, common to all SCSI technologies and implementations that are essential for compatibility with host SCSI application software and device-resident firmware across all SCSI protocols. These requirements are defined through a reference model that specifies the behavior and abstract structure that is generic to all SCSI I/O system implementations.

With any technical document there may arise questions of interpretation as new products are implemented. INCITS has established procedures to issue technical opinions concerning the standards developed by INCITS. These procedures may result in SCSI Technical Information Bulletins being published by INCITS.

These Bulletins, while reflecting the opinion of the Technical Committee that developed the standard, are intended solely as supplementary information to other users of the standard. This standard, ANSI INCITS.***:200x, as approved through the publication and voting procedures of the American National Standards Institute, is not altered by these bulletins. Any subsequent revision to this standard may or may not reflect the contents of these Technical Information Bulletins.

Current INCITS practice is to make Technical Information Bulletins available through:

| | |
|---|---|
| INCITS Online Store | http://www.techstreet.com/INCITS.html |
| managed by Techstreet | Telephone: 1-734-302-7801 or |
| 1327 Jones Drive | 1-800-699-9277 |
| Ann Arbor, MI 48105 | Facsimile: 1-734-302-7811 |

or

| | |
|---|---|
| Global Engineering | http://global.ihs.com/ |
| 15 Inverness Way East | Telephone: 1-303-792-2181 or |
| Englewood, CO 80112-5704 | 1-800-854-7179 |
| | Facsimile: 1-303-792-2192 |

Requests for interpretation, suggestions for improvement and addenda, or defect reports are welcome. They should be sent to the INCITS Secretariat, National Committee for Information Technology Standards, Information Technology Institute, 1250 Eye Street, NW, Suite 200, Washington, DC 20005- 3922.

This standard was processed and approved for submittal to ANSI by the InterNational Committee for Information Technology Standards (INCITS). Committee approval of the standard does not necessarily imply that all committee members voted for approval. At the time of it approved this standard, INCITS had the following members:

<<Insert INCITS member list>>

The INCITS Technical Committee T10 on Lower Level Interfaces, which reviewed this standard, had the following members:

<<Insert T10 member list>>

## Introduction

The SCSI Architecture Model - 3 (SAM-3) standard is divided into seven clauses and one annex:

Clause 1 is the scope.
Clause 2 enumerates the normative references that apply to this standard.
Clause 3 describes the definitions, symbols, and abbreviations used in this standard.
Clause 4 describes the overall SCSI architectural model
Clause 5 describes the SCSI command model element of the SCSI architecture
Clause 6 describes the task management functions common to SCSI devices
Clause 7 describes the task set management capabilities common to SCSI devices
Annex A summarizes the identifier and name definitions of the SCSI Protocols

**American National Standard** **INCITS.***:200x**

# American National Standard for Information Systems - Information Technology - SCSI Architecture Model - 3 (SAM-3)

## 1 Scope

### 1.1 Introduction

The set of SCSI standards consists of this standard and the SCSI implementation standards described in 1.2. This standard defines a reference model that specifies common behaviors for SCSI devices, and an abstract structure that is generic to all SCSI I/O system implementations.

The set of SCSI standards specifies the interfaces, functions, and operations necessary to ensure interoperability between conforming SCSI implementations. This standard is a functional description. Conforming implementations may employ any design technique that does not violate interoperability.

### 1.2 Requirements precedence

This standard defines generic requirements that pertain to SCSI implementation standards, and implementation requirements. An implementation requirement specifies behavior in terms of measurable or observable parameters that apply directly to an implementation. Examples of implementation requirements defined in this document are the command descriptor block format and the status values to be returned upon command completion.

Generic requirements are transformed to implementation requirements by an implementation standard. An example of a generic requirement is the hard reset behavior specified in 5.8.6.

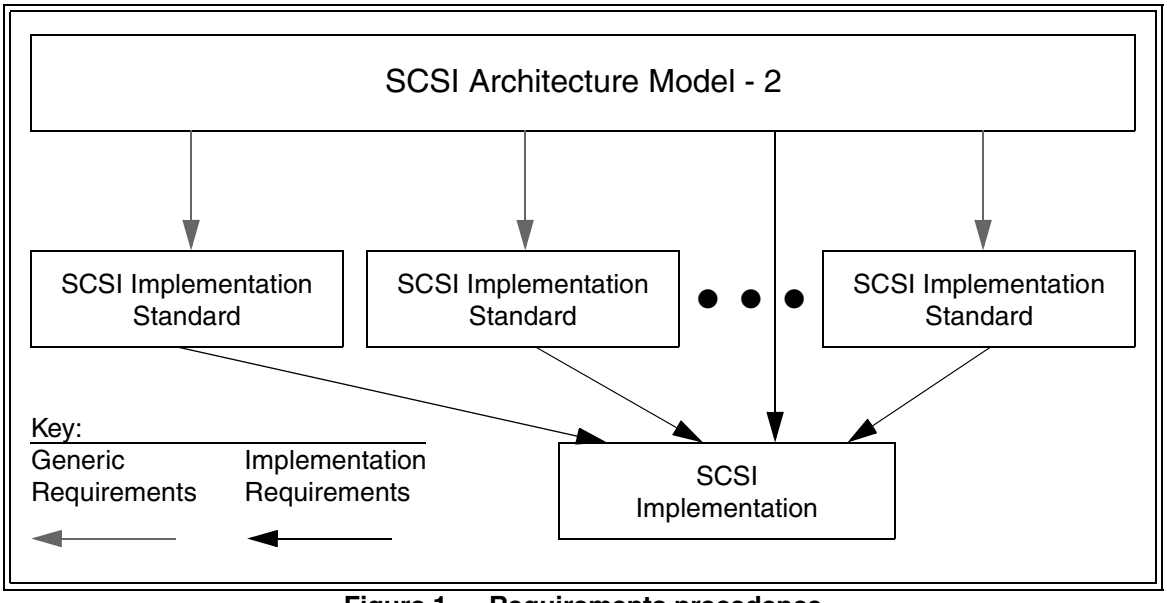



**Figure 1 — Requirements precedence**

As shown in figure 1, all SCSI implementation standards shall reflect the generic requirements defined herein. In addition, an implementation claiming SCSI compliance shall conform to the applicable implementation require-

ments defined in this standard and the appropriate SCSI implementation standards. In the event of a conflict between this document and other SCSI standards under the jurisdiction of technical committee T10, the requirements of this standard shall apply.

## 1.3 SCSI standards family

Figure 2 shows the relationship of this standard to the other standards and related projects in the SCSI family standards as of the publication of this standard.

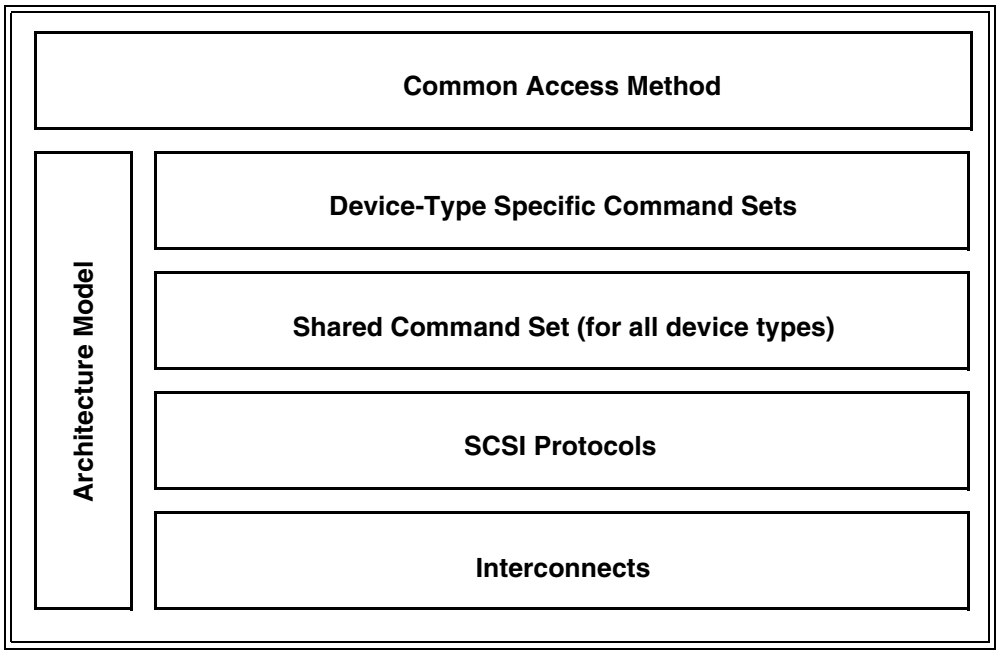



**Figure 2 — SCSI document roadmap**

The roadmap in figure 2 is intended to show the general applicability of the documents to one another. The figure is not intended to imply a relationship such as a hierarchy, protocol stack, or system architecture. It indicates the applicability of a standard to the implementation of a given transport.

The functional areas identified in figure 2 characterize the scope of standards within a group as follows:

**Architecture Model:** Defines the SCSI systems model, the functional partitioning of the SCSI standard set and requirements applicable to all SCSI implementations and implementation standards.

**Common Access Method:** Implementation standard that defines a host architecture and set of services for device access.

**Device-Type Specific Command Sets:** Implementation standards that define specific device types including a device model for each device type. These standards specify the required commands and behavior that is specific to a given device type and prescribe the requirements to be followed by an initiator when sending commands to a device having the specific device type. The commands and behaviors for a specific device type may include by reference commands and behaviors that are shared by all SCSI devices.

**Shared Command Set:** An implementation standard that defines a model for all SCSI device types. This standard specifies the required commands and behavior that is common to all devices, regardless of device type, and prescribe the requirements to be followed by an initiator when sending commands to any device.

**SCSI Protocols:** Implementation standards that define the requirements for exchanging information so that different SCSI devices are capable of communicating.

**Interconnects:** Implementation standards that define the communications mechanism employed by the SCSI Protocols. These standards may describe the electrical and signaling requirements essential for devices to interoperate over a given physical interconnect.

At the time this standard was generated, examples of the SCSI general structure included:

Interconnects:

| | | |
|---|---|---|
| Fibre Channel Arbitrated Loop | FC-AL | [ANSI X3.272-1996] |
| Fibre Channel Arbitrated Loop -2 | FC-AL-2 | [ISO/IEC 14165-122]<br>[ANSI NCITS.332-1999] |
| Fibre Channel Physical and Signalling Interface | FC-PH | [ISO/IEC 14165-111]<br>[ANSI X3.230-1994] |
| Fibre Channel Physical Amendment 1 | | [ANSI X3.230/AM1-1996] |
| Fibre Channel 3rd Generation Physical Interface | FC-PH-3 | [ISO/IEC 14165-113]<br>[ANSI X3.303-1998] |
| Fibre Channel Physical Interfaces | FC-PI | [T11/1235-D] |
| Fibre Channel Framing and Signaling Interface | FC-FS | [T11/1331-D] |
| High Performance Serial Bus | | [ANSI IEEE 1394-1995] |
| High Performance Serial Bus<br>(supplement to ANSI/IEEE 1394-1995) | | [ANSI IEEE 1394a-2000] |
| SCSI Parallel Interface - 2 | SPI-2 | [ISO/IEC 14776-112]<br>[ANSI X3.302-1999] |
| SCSI Parallel Interface - 3 | SPI-3 | [ISO/IEC 14776-113]<br>[ANSI NCITS.336-2000] |
| SCSI Parallel Interface - 4 | SPI-4 | [ISO/IEC 14776-114]<br>[ANSI INCITS.362-200x] |
| SCSI Parallel Interface - 5 | SPI-5 | [ISO/IEC 14776-115]<br>[T10/1525-D] |
| Serial Storage Architecture Physical Layer 1 | SSA-PH | [ANSI X3.293-1996] |
| Serial Storage Architecture Physical Layer 2 | SSA-PH-2 | [ANSI NCITS.307-1998] |

SCSI Protocols:

| | | |
|---|---|---|
| Serial Storage Architecture Transport Layer 1 | SSA-TL-1 | [ANSI X3.295-1996] |
| Serial Storage Architecture Transport Layer 2 | SSA-TL-2 | [ANSI NCITS.308-1998] |
| SCSI-3 Fibre Channel Protocol | FCP | [ISO/IEC 14776-221]<br>[ANSI X3.269-1996] |
| SCSI Fibre Channel Protocol - 2 | FCP-2 | [ISO/IEC 14776-222]<br>[ANSI NCITS.350-200x] |
| Serial Bus Protocol - 2 | SBP-2 | [ISO/IEC 14776-232]<br>[ANSI NCITS.325-1999] |
| Serial Bus Protocol - 3 | SBP-3 | [ISO/IEC 14776-233]<br>[T10/1467-D] |
| Serial Storage Architecture SCSI-3 Protocol | SSA-S3P | [ANSI NCITS.309-1998] |
| SCSI on Scheduled Transfer | SST | [T10/1380-D] |
| SCSI RDMA Protocol | SRP | [T10/1415-D] |

Shared Command Sets:

| | | |
|---|---|---|
| SCSI-3 Primary Commands | SPC | [ISO/IEC 14776-311]<br>[ANSI X3.301-1997] |
| SCSI Primary Commands - 2 | SPC-2 | [ISO/IEC 14776-312]<br>[ANSI NCITS.351-2001] |

| | | |
|---|---|---|
| SCSI Primary Commands - 3 | SPC-3 | [ISO/IEC 14776-313]<br>[T10/1416-D] |
| Device-Type Specific Command Sets: | | |
| SCSI-3 Block Commands | SBC | [ISO/IEC 14776-321]<br>[ANSI NCITS.306-1998] |
| SCSI Block Commands - 2 | SBC-2 | [ISO/IEC 14776-322]<br>[T10/1417-D] |
| SCSI-3 Stream Commands | SSC | [ISO/IEC 14776-331]<br>[ANSI NCITS.335-2000] |
| SCSI Stream Commands - 2 | SSC-2 | [ISO/IEC 14776-332]<br>[T10/1434-D] |
| SCSI-3 Medium Changer Commands | SMC | [ISO/IEC 14776-351]<br>[ANSI NCITS.314-1998] |
| SCSI Medium Changer Commands - 2 | SMC-2 | [ISO/IEC 14776-352]<br>[T10/1383-D] |
| SCSI-3 Multimedia Command Set | MMC | [ANSI X3.304-1997] |
| SCSI Multimedia Command Set - 2 | MMC-2 | [ISO/IEC 14776-362]<br>[ANSI NCITS.333-2000] |
| SCSI Multimedia Command Set - 3 | MMC-3 | [ISO/IEC 14776-363]<br>[T10/1363-D] |
| SCSI-3 Controller Commands | SCC | [ISO/IEC 14776-341]<br>[ANSI X3.276-1997] |
| SCSI Controller Commands - 2 | SCC-2 | [ISO/IEC 14776-342]<br>[ANSI NCITS.318-1998] |
| SCSI Reduced Block Commands | RBC | [ISO/IEC 14776-326]<br>[ANSI NCITS.330-2000] |
| SCSI-3 Enclosure Services Commands | SES | [ISO/IEC 14776-371]<br>[ANSI NCITS.305-1998] |
| SCSI Specification for Optical Card Reader/Writer | OCRW | [ISO/IEC 14776-381] |
| Object-based Storage Devices Commands | OSD | [T10/1355-D] |
| SCSI Management Server Commands | MSC | [T10/1528-D] |
| Architecture Model: | | |
| SCSI-3 Architecture Model | SAM | [ISO/IEC 14776-411]<br>[ANSI X3.270-1996] |
| SCSI Architecture Model - 2 | SAM-2 | [ISO/IEC 14776-412]<br>[T10/1157-D] |

The term SCSI is used to refer to the family of standards described in this subclause.

# 2 Normative references

## 2.1 Normative references

The following standards contain provisions that, by reference in the text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

Copies of the following documents may be obtained from ANSI: approved ANSI standards, approved and draft international and regional standards (ISO, IEC, CEN/CENELEC, ITUT), and approved and draft foreign standards (including BSI, JIS, and DIN). For further information, contact ANSI Customer Service Department at 212-642-4900 (phone), 212-302-1286 (fax) or via the World Wide Web at http://www.ansi.org.

## 2.2 Approved references

*ISO/IEC 60027-2-am2 (1999-01),* Letter symbols to be used in electrical technology - Part 2: Telecommunications and electronics (Amendment 2)

*ISO/IEC 14776-312,* SCSI Primary Commands - 2 (SPC-2) [ANSI NCITS.351-2001]

## 2.3 References under development

At the time of publication, the following referenced standards were still under development. For information on the current status of the document, or regarding availability, contact the relevant standards body or other organization as indicated.

*ISO/IEC 14776-313,* SCSI Primary Commands - 3 (SPC-3) [T10/1416-D]

# 3 Definitions, symbols, abbreviations, and conventions

## 3.1 Definitions

**3.1.1 aborted command:** A SCSI command that has been ended by aborting the task created to process it.

**3.1.2 ACA command:** A command performed by a task with the ACA attribute (see 3.1.5, 4.9 and 7.5.4).

**3.1.3 additional sense code:** A combination of the ADDITIONAL SENSE CODE and ADDITIONAL SENSE CODE QUALIFIER fields in the sense data (see 3.1.107 and SPC-2).

**3.1.4 application client:** An object that is the source of SCSI commands.

**3.1.5 auto contingent allegiance (ACA):** One of the possible conditions of a task set following the return of a CHECK CONDITION status. See 5.8.1.

**3.1.6 blocked task state** The state of a task that is prevented from completing due to an ACA condition.

**3.1.7 blocking boundary:** A task set boundary denoting a set of conditions that inhibit tasks outside the boundary from entering the enabled task state.

**3.1.8 byte:** An 8-bit construct.

**3.1.9 call:** The act of invoking a procedure.

**3.1.10 client-server:** A relationship established between a pair of distributed objects where one (the client) requests the other (the server) to perform some operation or unit of work on the client's behalf.

**3.1.11 client:** An object that requests a service from a server.

**3.1.12 code value:** A one or a series of defined numeric values each representing an identified and described instance or condition. Code values are defined to be used in a specific field (see 3.1.37), in a procedure input data object (see 3.6.2), in a procedure output data object, or in a procedure result.

**3.1.13 command:** A request describing a unit of work to be performed by a device server.

**3.1.14 command descriptor block (CDB):** A structure used to communicate a command from an application client to a device server. A CDB may have a fixed length of up to 16 bytes or a variable length of between 12 and 260 bytes.

**3.1.15 completed command:** A command that has ended by returning a status and service response of TASK COMPLETE or LINKED COMMAND COMPLETE.

**3.1.16 completed task:** A task that has ended by returning a status and service response of TASK COMPLETE. The actual events comprising the TASK COMPLETE response are protocol specific.

**3.1.17 confirmation:** A response returned to an object that signals the completion of a service request.

**3.1.18 confirmed SCSI protocol service:** A service available at the SCSI protocol service interface that includes a confirmation of completion.

**3.1.19 contingent allegiance (CA):** One of the possible conditions of a task set following the return of a CHECK CONDITION status. See 5.8.1.

**3.1.20 Control mode page:** The Control mode page that identifies the settings of several device server behaviors that may be of interest to an application client or may be changed by an application client. Fields in the Control mode page are referenced by name in this standard and SPC-2 contains a complete definition of the Control mode page.

**3.1.21 current task:** A task that has a data transfer SCSI protocol service request in progress (see 5.4.3) or is in the process of sending command status. Each SCSI protocol standard shall define the protocol specific conditions under which a task is considered a current task.

**3.1.22 dependent logical unit:** A logical unit that is addressed via some other logical unit(s) in a hierarchical logical unit structure (see 3.1.40), also a logical unit that is at a higher numbered level in the hierarchy than the referenced logical unit (see 4.12).

**3.1.23 destination device:** The SCSI device to which a service delivery transaction is addressed. See source device (3.1.115).

**3.1.24 device identifier:** Synonymous with SCSI port identifier (see 3.1.95).

**3.1.25 device model:** The description of a type of SCSI target device (e.g., block, stream).

**3.1.26 device server:** An object within the logical unit that processes SCSI tasks according to the requirements for task management described in clause 7.

**3.1.27 device service request:** A request, submitted by an application client, conveying a SCSI command to a device server.

**3.1.28 device service response:** The response returned to an application client by a device server on completion of a SCSI command.

**3.1.29 domain:** An I/O system consisting of a set of SCSI devices that interact with one another by means of a service delivery subsystem.

**3.1.30 dormant task state:** The state of a task that is prevented from starting processing due to the presence of certain other tasks in the task set.

**3.1.31 enabled task state:** The state of a task that may complete at any time. Alternatively, the state of a task that is waiting to receive the next command in a series of linked commands.

**3.1.32 ended command:** A command that has completed or aborted.

**3.1.33 faulted initiator:** The initiator to which a CHECK CONDITION status was returned. The faulted initiator condition disappears when the ACA or CA condition resulting from the CHECK CONDITION status is cleared.

**3.1.34 faulted task set:** A task set that contains a faulting task. The faulted task set condition disappears when the ACA or CA condition resulting from the CHECK CONDITION status is cleared.

**3.1.35 faulting command:** A command that completed with a status of CHECK CONDITION.

**3.1.36 faulting task:** A task that has completed with a status of CHECK CONDITION.

**3.1.37 field:** A group of one or more contiguous bits, part of a larger structure such as a CDB (see 3.1.14) or sense data (see 3.1.107).

**3.1.38 function complete:** A logical unit response indicating that a task management function has finished. The actual events comprising this response are protocol specific.

**3.1.39 hard reset:** A target action in response to a reset event in which the target port performs the operations described in 5.8.6.

**3.1.40 hierarchical logical unit:** An inverted tree structure for forming and parsing logical unit numbers (see 3.1.63) containing up to four addressable levels (see 4.12).

**3.1.41 I_T nexus:** A nexus between an initiator and a target (see 4.10).

**3.1.42 I_T_L nexus:** A nexus between an initiator, a target, and a logical unit (see 4.10).

**3.1.43 I_T_L_Q nexus:** A nexus between an initiator, a target, a logical unit, and a tagged task (see 4.10).

**3.1.44 I_T_L_x nexus:** Either an I_T_L nexus or an I_T_L_Q nexus (see 4.10).

**3.1.45 I/O operation:** An operation defined by an unlinked SCSI command, a series of linked SCSI commands or a task management function.

**3.1.46 implementation:** The physical realization of an object.

**3.1.47 implementation specific:** A requirement or feature that is defined in a SCSI standard but whose implementation may be specified by the system integrator or vendor.

**3.1.48 implementation option:** An option whose actualization within an implementation is at the discretion of the implementor.

**3.1.49 initiator:** Synonymous with SCSI initiator port (see 3.1.93).

**3.1.50 initiator device name:** A SCSI device name of a SCSI initiator device (see 4.7.1).

**3.1.51 initiator identifier:** Synonymous with initiator port identifier (see 4.7.1).

**3.1.52 initiator port identifier:** A value by which a SCSI initiator port is referenced within a domain (see 4.7.1).

**3.1.53 initiator port name:** A SCSI port name (see 3.1.96) of a SCSI initiator port or of a SCSI target/initiator port when operating as a SCSI initiator port (see 4.7.1).

**3.1.54 interconnect subsystem:** One or more physical interconnects that appear as a single path for the transfer of information between SCSI devices in a domain.

**3.1.55 in transit:** Information that has been sent to a remote object but not yet received.

**3.1.56 layer:** A subdivision of the architecture constituted by subsystems of the same rank.

**3.1.57 linked CDB:** A CDB with the LINK bit in the CONTROL byte set to one.

**3.1.58 linked command:** One in a series of SCSI commands processed by a single task that collectively make up a discrete I/O operation. In such a series, each command is represented by the same I_T_L_x nexus, and all, except the last, have the LINK bit in the CDB CONTROL byte set to one.

**3.1.59 logical unit:** A target-resident object that implements a device model and processes SCSI commands sent by an application client.

**3.1.60 logical unit reset:** A logical unit action in response to a logical unit reset event in which the logical unit performs the operations described in 5.8.7.

**3.1.61 logical unit reset event:** An event that triggers a logical unit reset from a logical unit as described in 5.8.7.

**3.1.62 logical unit inventory:** The list of the logical unit numbers reported by a REPORT LUNS command (see SPC-2).

**3.1.63 logical unit number (LUN):** A 64-bit identifier for a logical unit.

**3.1.64 logical unit option:** An option pertaining to a logical unit, whose actualization is at the discretion of the logical unit implementor.

**3.1.65 lower level protocol (LLP):** A protocol used to carry the information representing upper level protocol transactions.

**3.1.66 media information:** Information stored within a SCSI device that is non-volatile (retained through a power cycle) and accessible to an initiator through the processing of SCSI commands.

**3.1.67 name:** A label of an object that is unique within a specified context and should never change (e.g., the term name and world wide identification (WWID) may be interchangeable).

**3.1.68 nexus:** A relationship between two SCSI devices and the initiator and target objects within those SCSI devices (see 4.10).

**3.1.69 object:** An architectural abstraction or container that encapsulates data types, services, or other objects that are related in some way.

**3.1.70 peer-to-peer protocol service:** A service used by an upper level protocol implementation to exchange information with its peer.

**3.1.71 peer entities:** Entities within the same layer.

**3.1.72 pending task:** A task that is not a current task.

**3.1.73 physical interconnect:** A single physical pathway for the transfer of information between SCSI devices in a domain.

**3.1.74 port:** Synonymous with SCSI port (see 3.1.94).

**3.1.75 procedure:** An operation that is invoked through an external calling interface.

**3.1.76 protocol:** The requirements governing the content and exchange of information passed between distributed objects through the service delivery subsystem.

**3.1.77 protocol option:** An function whose definition within a SCSI protocol standard is optional.

**3.1.78 queue:** The arrangement of tasks within a task set (see 3.1.134), usually according to the temporal order in which they were created.

**3.1.79 receiver:** A client or server that is the recipient of a service delivery transaction.

**3.1.80 reference model:** A standard model used to specify system requirements in an implementation-independent manner.

**3.1.81 request:** A transaction invoking a service.

**3.1.82 request-response transaction:** An interaction between a pair of distributed, cooperating objects, consisting of a request for service submitted to an object followed by a response conveying the result.

**3.1.83 request-confirmation transaction:** An interaction between a pair of cooperating objects, consisting of a request for service submitted to an object followed by a response from the object confirming request completion.

**3.1.84 reset event:** A protocol specific event that triggers a hard reset from a SCSI device as described in 5.8.6.

**3.1.85 response:** A transaction conveying the result of a request.

**3.1.86 SCSI application layer:** The protocols and procedures that implement or issue SCSI commands and task management functions by using services provided by a SCSI protocol layer.

**3.1.87 SCSI device:** A device that contains one or more SCSI ports that are connected to a service delivery subsystem and supports a SCSI application protocol.

**3.1.88 SCSI device identifier:** Synonymous with SCSI port identifier (see 3.1.95).

**3.1.89 SCSI device name:** A name (see 3.1.67) of a SCSI device that is world wide unique within the protocol of a SCSI domain in which the SCSI device has SCSI ports (see 4.7.6). The SCSI device name may be made available to other SCSI devices or SCSI ports in that SCSI domain in protocol specific ways.

**3.1.90 SCSI I/O system:** An I/O system, consisting of two or more SCSI devices, a SCSI interconnect and a SCSI protocol that collectively interact to perform SCSI I/O operations.

**3.1.91 SCSI identifier:** Synonymous with SCSI port identifier (see 3.1.95).

**3.1.92 SCSI initiator device:** A SCSI device containing application clients and SCSI initiator ports that originate device service and task management requests to be processed by a target SCSI device. When used this term refers to SCSI initiator devices or SCSI target/initiator devices that are using the SCSI target/initiator port as a SCSI initiator port.

**3.1.93 SCSI initiator port:** A SCSI initiator device object acts as the connection between application clients and the service delivery subsystem through which requests and responses are routed. In all cases when this term is used it refers to an initiator port or a SCSI target/initiator port operating as a SCSI initiator port.

**3.1.94 SCSI port:** A device-resident object that connects the application client, device server or task manager to the service delivery subsystem through which requests and responses are routed. SCSI port is synonymous with port and either a SCSI initiator port (see 3.1.93) or a SCSI target port (see 3.1.103).

**3.1.95 SCSI port identifier:** A value by which a SCSI port is referenced within a domain. The SCSI port identifier is either an initiator port identifier (see 3.1.52) or a target port identifier (see 3.1.122).

**3.1.96 SCSI port name:** A name (see 3.1.67) of a SCSI port that is world wide unique within the protocol of the SCSI domain of that SCSI port (see 4.7.7). The name may be made available to other SCSI devices or SCSI ports in that SCSI domain in protocol specific ways.

**3.1.97 SCSI protocol layer:** The protocol and services used by a SCSI application layer to transport data representing a SCSI application protocol transaction.

**3.1.98 SCSI protocol service confirmation:** A signal from the lower level SCSI protocol layer notifying the upper layer that a SCSI protocol service request has completed.

**3.1.99 SCSI protocol service indication:** A signal from the lower level SCSI protocol layer notifying the upper level that a SCSI protocol transaction has occurred.

**3.1.100 SCSI protocol service request:** A call to the lower level SCSI protocol layer to begin a SCSI protocol service transaction.

**3.1.101 SCSI protocol service response:** A reply from the upper level protocol layer in response to a SCSI protocol service indication.

**3.1.102 SCSI target device:** A SCSI device containing logical units and SCSI target ports that receives device service and task management requests for processing. When used this term refers to SCSI target devices or SCSI target/initiator devices that are using the SCSI target/initiator port as a SCSI target port.

**3.1.103 SCSI target port:** A SCSI target device object that contains a task router and acts as the connection between device servers and task managers and the service delivery subsystem through which requests and responses are routed. When this term is used it refers to a SCSI target port or a SCSI target/initiator port operating as a SCSI target port.

**3.1.104 SCSI target/initiator device:** A device that has all the characteristics of a SCSI target device and a SCSI initiator device.

**3.1.105 SCSI target/initiator port:** A device-resident object that has all the characteristics of a SCSI target port and a SCSI initiator port.

**3.1.106 sender:** A client or server that originates a service delivery transaction.

**3.1.107 sense data:** Data returned to an application client as a result of an autosense operation, asynchronous event report, or REQUEST SENSE command (see 5.8.4). Fields in the sense data are referenced by name in this standard. See SPC-2 for a complete sense data format definition.

**3.1.108 sense key:** A field in the sense data. See 3.1.107 and SPC-2.

**3.1.109 server:** A SCSI object that performs a service on behalf of a client.

**3.1.110 service:** Any operation or function performed by a SCSI object that is invoked by other SCSI objects.

**3.1.111 service delivery failure:** Any non-recoverable error causing the corruption or loss of one or more service delivery transactions while in transit.

**3.1.112 service delivery subsystem:** That part of a SCSI I/O system that transmits service requests to a logical unit or target and returns logical unit or target responses to an initiator.

**3.1.113 service delivery transaction:** A request or response sent through the service delivery subsystem.

**3.1.114 signal:** (n) A detectable asynchronous event possibly accompanied by descriptive data and parameters. (v) The act of generating such an event.

**3.1.115 source device:** The SCSI device from which a service delivery transaction originates. See destination device (see 3.1.23).

**3.1.116 standard INQUIRY data:** Data returned to an application client as a result of an INQUIRY command. Fields in the standard INQUIRY data are referenced by name in this standard and SPC-2 contains a complete definition of the standard INQUIRY data format.

**3.1.117 subsystem:** An element in a hierarchically partitioned system that interacts directly only with elements in the next higher division or the next lower division of that system.

**3.1.118 suspended information:** Information within a logical unit that is not available to any pending task.

**3.1.119 target:** Synonymous with SCSI target port (see 3.1.103).

**3.1.120 target device name:** A SCSI device name (see 3.1.89) of a SCSI target device (see 4.7.2).

**3.1.121 target identifier:** Synonymous with target port identifier (see 4.7.2).

**3.1.122 target port identifier:** A value by which a SCSI target port is referenced within a domain (see 4.7.2).

**3.1.123 target port name:** A SCSI port name of a SCSI target port or of a SCSI target/initiator port when operating as a SCSI target port (see 4.7.2).

**3.1.124 target/initiator device name:** A SCSI device name (see 3.1.89) of a SCSI target/initiator device (see 4.7.3).

**3.1.125 task:** An object within the logical unit representing the work associated with a command or a group of linked commands.

**3.1.126 task abort event:** An event or condition indicating that the task has been aborted by means of a task management function (see 7.3).

**3.1.127 task completion event:** An event or condition indicating that the task has ended with a service response of TASK COMPLETE (see 7.3).

**3.1.128 task ended event:** An event or condition indicating that the task has completed or aborted (see 7.3).

**3.1.129 task management function:** A task manager service capable of being requested by an application client to affect the processing of one or more tasks.

**3.1.130 task management request:** A request submitted by an application client, invoking a task management function to be processed by a task manager.

**3.1.131 task management response:** The response returned to an application client by a task manager on completion of a task management request.

**3.1.132 task manager:** A server within a logical unit that processes task management functions.

**3.1.133 task router:** Routes commands and task management functions between the service delivery subsystem (see 3.1.112) and the appropriate logical unit's task manager (see 3.1.132).

**3.1.134 task set:** A group of tasks within a logical unit, whose interaction is dependent on the task management (queuing), CA, and ACA requirements (see 4.8).

**3.1.135 third-party command:** A SCSI command that requires a logical unit within a SCSI target device to assume the initiator role and send SCSI command(s) to another SCSI target device.

**3.1.136 transaction:** A cooperative interaction between two objects, involving the exchange of information or the processing of some service by one object on behalf of the other.

**3.1.137 unconfirmed protocol service:** A service available at the protocol service interface that does not result in a completion confirmation.

**3.1.138 unlinked command:** A SCSI command having the LINK bit set to zero in the CDB CONTROL byte.

**3.1.139 upper level protocol (ULP):** An application specific protocol processed through services provided by a lower level protocol.

3.1.140 **wakeup:** A SCSI target port returning from the sleep power condition to the active power condition (see SPC-3).

**3.1.141 wakeup event:** An event that triggers a wakeup from a SCSI target port as described in SPC-3.

**3.1.142 well known logical unit:** A logical unit that only does specific functions (see 4.13.3). Well known logical units allow an application client to issue requests to receive and manage specific information usually relating to a SCSI target.

**3.1.143 well known logical unit number (W-LUN):** The logical unit number that identifies a well known logical unit.

## 3.2 Acronyms

| | |
|---|---|
| ACA | Auto Contingent Allegiance (see 3.1.5) |
| AER | Asynchronous Event Reporting |
| CA | Contingent Allegiance (see 3.1.19) |
| CDB | Command Descriptor Block (see 3.1.14) |
| LLP | Lower Level Protocol (see 3.1.65) |
| LUN | Logical Unit Number (see 3.1.63) |
| MMC-2 | SCSI Multi-Media Commands -2 (see 1.3) |
| n/a | Not Applicable |
| SBC | SCSI-3 Block Commands (see 1.3) |
| SCSI | The architecture defined by the family of standards described in 1.3 |
| SSC | SCSI-3 Stream Commands (see 1.3) |
| SPI-4 | SCSI Parallel Interface -4 (see 1.3) |
| SPC-2 | SCSI Primary Commands -2 (see 1.3) |
| SPC-3 | SCSI Primary Commands -3 (see 1.3) |
| SSC | SCSI-3 Stream Commands (see 1.3) |
| ULP | Upper Level Protocol (see 3.1.139) |
| VPD | Vital Product Data (see SPC-2) |
| W-LUN | Well known logical unit number (see 3.1.143) |

## 3.3 Keywords

**3.3.1 expected:** A keyword used to describe the behavior of the hardware or software in the design models assumed by this standard. Other hardware and software design models may also be implemented.

**3.3.2 invalid:** A keyword used to describe an illegal or unsupported bit, byte, word, field or code value. Receipt by a device server of an invalid bit, byte, word, field or code value shall be reported as error.

**3.3.3 mandatory:** A keyword indicating an item that is required to be implemented as defined in this standard.

**3.3.4 may:** A keyword that indicates flexibility of choice with no implied preference (synonymous with "may or may not").

**3.3.5 may not:** A keyword that indicates flexibility of choice with no implied preference (synonymous with "may or may not").

**3.3.6 obsolete:** A keyword indicating that an item was defined in prior SCSI standards but has been removed from this standard.

**3.3.7 option, optional:** Keywords that describe features that are not required to be implemented by this standard. However, if any optional feature defined by this standard is implemented, then it shall be implemented as defined in this standard.

**3.3.8 protocol specific:** Implementation of the referenced item is defined by a SCSI protocol standard (see 1.3).

**3.3.9 reserved:** A keyword referring to bits, bytes, words, fields and code values that are set aside for future standardization. A reserved bit, byte, word or field shall be set to zero, or in accordance with a future extension to this standard. Recipients are not required to check reserved bits, bytes, words or fields for zero values. Receipt of reserved code values in defined fields shall be reported as error.

**3.3.10 shall:** A keyword indicating a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other products that conform to this standard.

**3.3.11 should:** A keyword indicating flexibility of choice with a strongly preferred alternative; equivalent to the phrase "it is strongly recommended".

**3.3.12 vendor specific:** Specification of the referenced item is determined by the device vendor.

## 3.4 Editorial Conventions

Certain words and terms used in this standard have a specific meaning beyond the normal English meaning. These words and terms are defined either in the glossary or in the text where they first appear.

Upper case is used when referring to the name of a numeric value defined in this specification or a formal attribute possessed by an object. When necessary for clarity, names of objects, procedures, parameters or discrete states are capitalized or set in bold type. Names of fields are identified using small capital letters (e.g., NACA bit).

Callable procedures are identified by a name in bold type, such as **Execute Command** (see clause 5). Names of procedural arguments are denoted by capitalizing each word in the name. For instance, Sense Data is the name of an argument in the **Execute Command** procedure call.

Quantities having a defined numeric value are identified by large capital letters. CHECK CONDITION, for example, refers to the numeric quantity defined in table 22 (see 5.3.1). Quantities having a discrete but unspecified value are identified using small capital letters. As an example, TASK COMPLETE, indicates a quantity returned by the **Execute Command** procedure call (see clause 5). Such quantities are usually associated with an event or indication whose observable behavior or value is specific to a given implementation standard.

Lists sequenced by letters (e.g., a-red, b-blue, c-green) show no priority relationship between the listed items. Numbered lists (e.g., 1-red, 2-blue, 3-green) show a priority ordering between the listed items.

If a conflict arises between text, tables, or figures, the order of precedence to resolve the conflicts is text; then tables; and finally figures. Not all tables or figures are fully described in the text. Tables show data format and values.

Notes do not constitute any requirements for implementors.

## 3.5 Numeric Conventions

Digits 0-9 in the text of this standard that are not immediately followed by lower-case "b" or "h" are decimal values. Digits 0 and 1 immediately followed by lower case "b" are binary values. Digits 0-9 and the upper case letters "A"-"F" immediately followed by lower-case "h" are hexadecimal values.

Large numbers are separated by spaces (e.g., 12 345, not 12,345).

## 3.6 Notation Conventions

### 3.6.1 Hierarchy diagram conventions

Hierarchy diagrams show how objects are related to each other. The hierarchy diagram of figure 3, for example, shows the relationships among the objects comprising an object called Book. For this example, a Book object is defined as containing a Table of Contents, an optional Preface, one or more Chapter(s), and an optional Index. Further contents definitions are provided for Preface and Chapter. A Preface contains zero or more Figure(s) as well as one instance of Outline or one instance of Introductory Text or one instance of Outline and one instance of Introductory Text. A Chapter contains one or more Section(s) and zero or more Figure(s).

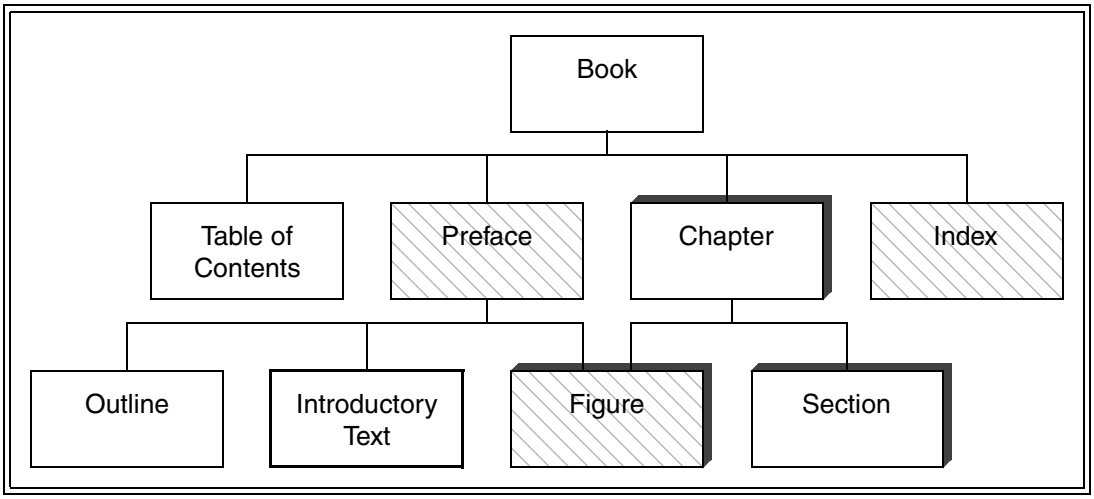



**Figure 3 — Example hierarchy diagram**

In the corresponding hierarchy diagram, labeled boxes denote the above objects. The composition and relation of one object to others is shown by the connecting lines. In this case, the connecting lines indicate the relationship between the Book object and its constituent objects Table of Contents, Preface, Chapter and Index. Similarly, connecting lines show that a Chapter object contains the objects Section and Figure. Note that the Figure object also may be a component of the Preface object.

In the hierarchy diagram, objects that are required to have one and only one instance are shown as simple boxes, as is the case for the Book and Table of Contents objects. The hierarchy diagram also shows multiple instances of

an object by the presence of a shadow, as is the case for the Chapter, Figure and Section objects. Objects that are optional are indicated by light diagonal lines, as is the case for the Preface, Figure and Index objects. An object that may not have any instances, have only one instance, or have multiple instances is shown with both diagonal lines and a shadow, as is the case for the Figure object. The instance indications shown in a hierarchy diagram are approximate, detailed requirements appear in the accompanying text.

### 3.6.2 Notation for procedures and functions

In this standard, the model for functional interfaces between objects is the callable procedure. Such interfaces are specified using the following notation:

**[Result =] Procedure Name (IN ([input-1] [,input-2] …]), OUT ([output-1] [,output-2] …))**

Where:

Result: A single value representing the outcome of the procedure or function.

Procedure Name: A descriptive name for the function to be performed.

Input-1, Input-2, …: A comma-separated list of names identifying caller-supplied input data objects.

Output-1, Output-2, …: A comma-separated list of names identifying output data objects to be returned by the procedure.

"[ …]": Brackets enclosing optional or conditional parameters and arguments.

This notation allows data objects to be specified as inputs and outputs. The following is an example of a procedure specification:

Found = **Search** (IN (Pattern, Item List), OUT ([Item Found]))

Where:

**Found = Flag**
**Flag**, if set, indicates that a matching item was located.

Input Arguments:

**Pattern = …** /* Definition of **Pattern** object */
Object containing the search pattern.

**Item List = Item<NN>** /* Definition of **Item List** as an array of NN **Item** objects*/
Contains the items to be searched for a match.

Output Arguments:

**Item Found = Item …** /* Item located by the search procedure */
This object is only returned if the search succeeds.

### 3.6.3 Notation for state diagrams

All state diagrams use the notation shown in figure 4.

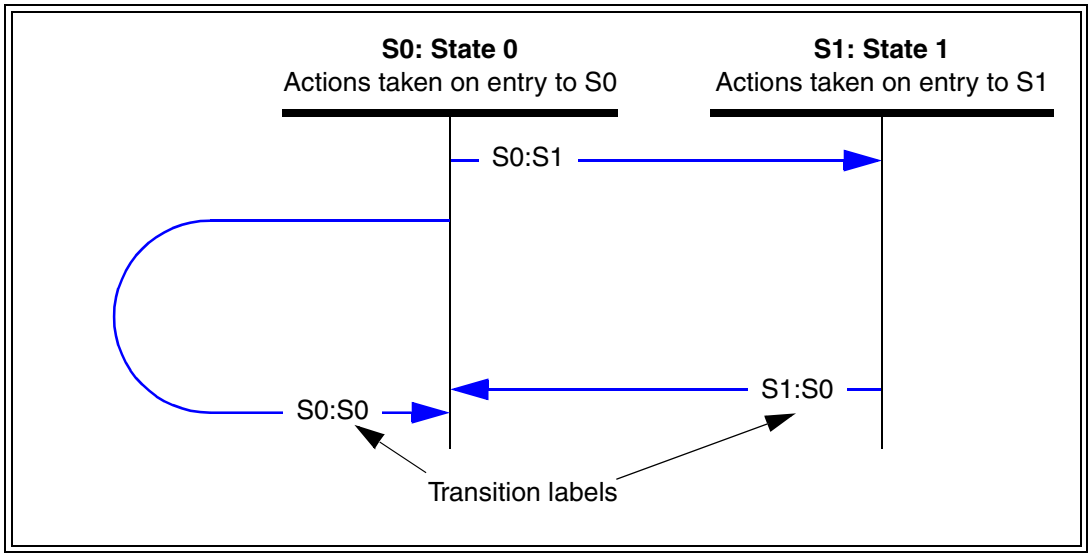



**Figure 4 — Example state diagram**

The state diagram is followed by a list of the state transitions, using the transition labels. Each transition is described in the list with particular attention to the conditions that cause the transition to occur and special conditions related to the transition. Using figure 4 as an example, the transition list might read as follows:

**Transition S0:S1:** This transition occurs when state S0 is exited and state S1 is entered.

**Transition S1:S0:** This transition occurs when state S1 is exited and state S0 is entered.

**Transition S0:S0:** This transition occurs when state S0 transitions to itself. It is particularly important to note that the actions taken whenever state S0 is entered are repeated every time this transition occurs.

A system specified in this manner has the following properties:

a) Time elapses only within discrete states;
b) State transitions are logically instantaneous; and
c) Every time a state is entered, the actions of that state are started. Note that this means that a transition that points back to the same state restarts the actions from the beginning.

# 4 SCSI Architecture Model

## 4.1 Introduction

The purpose of the SCSI architecture model is to:

a) Provide a basis for the coordination of SCSI standards development that allows each standard to be placed into perspective within the overall SCSI Architecture model;
b) Identify areas for developing standards and provide a common reference for maintaining consistency among related standards so that independent teams of experts may work productively and independently on the development of standards within each functional area; and
c) Provide the foundation for application compatibility across all SCSI interconnect and protocol environments by specifying generic requirements that apply uniformly to all implementation standards within each functional area.

The development of this standard is assisted by the use of an abstract model. To specify the external behavior of a SCSI system, elements in a system are replaced by functionally equivalent components within this model. Only externally observable behavior is retained as the standard of behavior. The description of internal behavior in this standard is provided only to support the definition of the observable aspects of the model. Those aspects are limited to the generic properties and characteristics needed for host applications to interoperate with SCSI devices in any SCSI interconnect and protocol environment. The model does not address other requirements that may be essential to some I/O system implementations such as the mapping from SCSI device addresses to network addresses, the procedure for discovering SCSI devices on a network and the definition of network authentication policies for SCSI initiators or targets. These considerations are outside the scope of the architecture model.

The set of SCSI standards specifies the interfaces, functions, and operations necessary to ensure interoperability between conforming SCSI implementations. This standard is a functional description. Conforming implementations may employ any design technique that does not violate interoperability.

The SCSI architecture model is described in terms of objects (see 3.1.69), protocol layers and service interfaces between objects. As used in this standard, objects are abstractions, encapsulating a set of related functions, data types, and other objects. Certain objects, such as an interconnect, are defined by SCSI, while others, such as a task, are needed to understand the functioning of SCSI but have implementation definitions outside the scope of SCSI. That is, although such objects exhibit well-defined, observable behaviors, they do not exist as separate physical elements. An object may be a single numeric parameter, such as a logical unit number, or a complex entity that performs a set of operations or services on behalf of another object.

Service interfaces are defined between distributed objects and protocol layers. The template for a distributed service interface is the client-server model described in 4.2. The structure of a SCSI I/O system is specified in 4.4 by defining the relationship among objects. The set of distributed services to be provided are specified in clause 5 and clause 6.

Requirements that apply to each SCSI protocol standard are specified in the SCSI protocol service model described in 5.4 and 6.9. The model describes required behavior in terms of layers, objects within layers and protocol service transactions between layers.

## 4.2 The SCSI distributed service model

Service interfaces between distributed objects are represented by the client-server model shown in figure 5. Dashed horizontal lines with arrowheads denote a single request-response transaction as it appears to the client and server. The solid lines with arrowheads indicate the actual transaction path through the service delivery subsystem. In such a model, each client or server is a single thread of processing that runs concurrently with all other clients or servers.

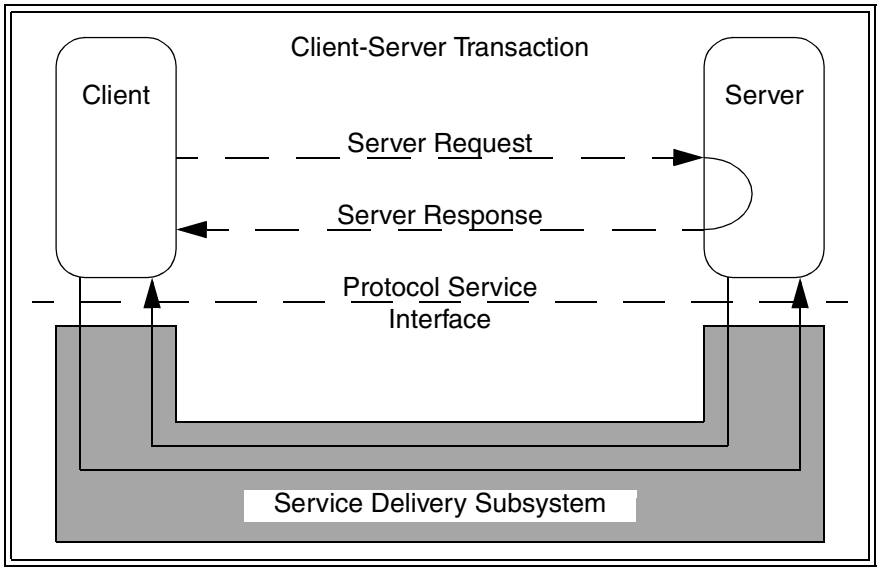



**Figure 5 — Client-Server model**

A client-server transaction is represented as a remote procedure call with inputs supplied by the caller (the client). The procedure is processed by the server returns outputs and a procedure status. A client directs requests to a remote server, via the client's service delivery subsystem, and receives a completion response or a failure notification. The request identifies the server and the service to be performed and includes the input data. The response conveys the output data and request status. The function of the service delivery subsystem is to transport an error-free copy of the request or response between sender and receiver. A failure notification indicates that a condition has been detected, such as a reset, or service delivery failure, that precludes request completion.

As seen by the client, a request becomes pending when it is passed to the service delivery subsystem for transmission. The request is complete when the server response is received or when a failure notification is sent. As seen by the server, the request becomes pending upon receipt and completes when the response is passed to its service delivery subsystem for return to the client. As a result there may be a time skew between the server and client's perception of request status and logical unit state. All references to a pending command or task management function in this standard are in the application client's point of view.

Client-server relationships are not symmetrical. A client may only originate requests for service. A server may only respond to such requests. The client calls the server-resident procedure and waits for completion. From the client's point of view, the behavior of a remote service invoked in this manner is indistinguishable from a conventional procedure call. In this model, confirmation of successful request or response delivery by the sender is not required. The model assumes that delivery failures are detected by the client's SCSI port or within service delivery subsystem.

## 4.3 The SCSI client-server model

As shown in figure 6, each SCSI target device provides device services performed by the logical units under the control of the target and task management functions performed by the task manager. A logical unit is an object that implements one of the device functional models described in the SCSI command standards and processes SCSI commands such as reading from or writing to the media. Each pending SCSI command or series of linked commands defines a unit of work to be performed by the logical unit. Each unit of work is represented within the target by a task that may be externally referenced and controlled through requests issued to the task manager.

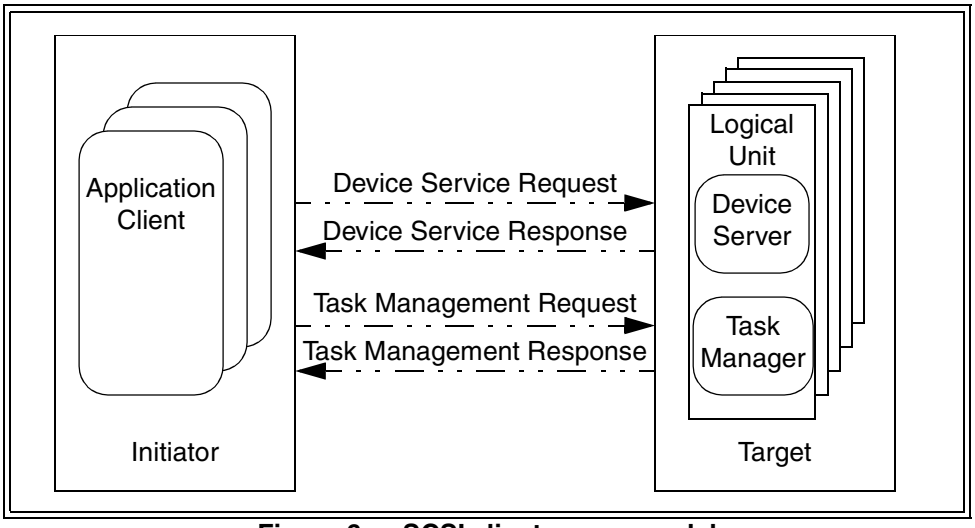



**Figure 6 — SCSI client-server model**

All requests originate from application clients residing within a SCSI initiator device. An application client represents a thread of processing whose functionality is independent of the interconnect and SCSI protocol. In an implementation, that thread could correspond to the device driver and any other code within the operating system that is capable of managing I/O requests without requiring knowledge of the interconnect or SCSI protocol. In the architecture model, an application client is created to issue a single SCSI command or task management function. An application client ceases to exist once the command or task management function ends. Consequently, there is one application client for each pending command or task management request. Within the initiator, one or more controlling entities, whose definition is outside the scope of the architecture model, oversee the creation of and interaction among application clients.

As described in 4.2, each request takes the form of a procedure call with arguments and a status to be returned. An application client may request processing of a SCSI command through a request directed to the device server within a logical unit. Each device service request contains a CDB, defining the operation to be performed, along with a list of command specific inputs and other parameters specifying how the command is to be processed. If supported by a logical unit, a sequence of linked commands may be used to define an extended I/O operation.

A task is an object within the logical unit representing the work associated with a command or series of linked commands. A new command or the first in a series of linked commands causes the creation of a task. The task persists until a command completion response is sent or until the task is ended by a task management function or exception condition. For an example of the processing for a single command see 5.7.1. For an example of linked command processing see 5.7.2.

An application client may request processing of a task management function through a request directed to the task manager within the logical unit. The interactions between the task manager and application client when a task management request is processed are shown in 6.10.

## 4.4 The SCSI structural model

The SCSI structural model represents a view of the elements comprising a SCSI I/O system as seen by the application clients interacting with the system. As shown in figure 7, the fundamental object is the SCSI domain that represents an I/O system. A domain is made up of SCSI devices and a service delivery subsystem that transports commands and data. A SCSI device contains application clients or device servers or both and the infrastructure to support them.

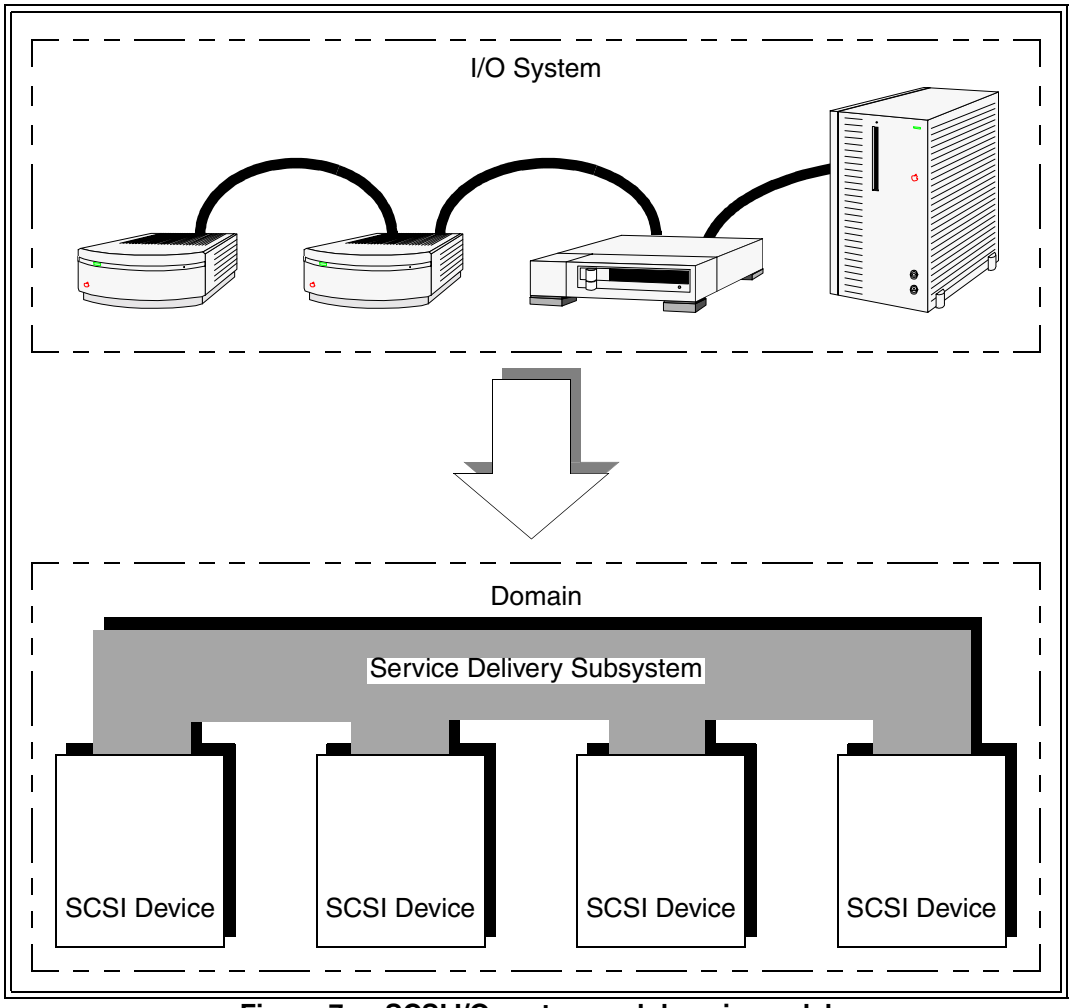



**Figure 7 — SCSI I/O system and domain model**

Figure 8 shows the main functional components of the SCSI domain. The following clauses define these components in greater detail.

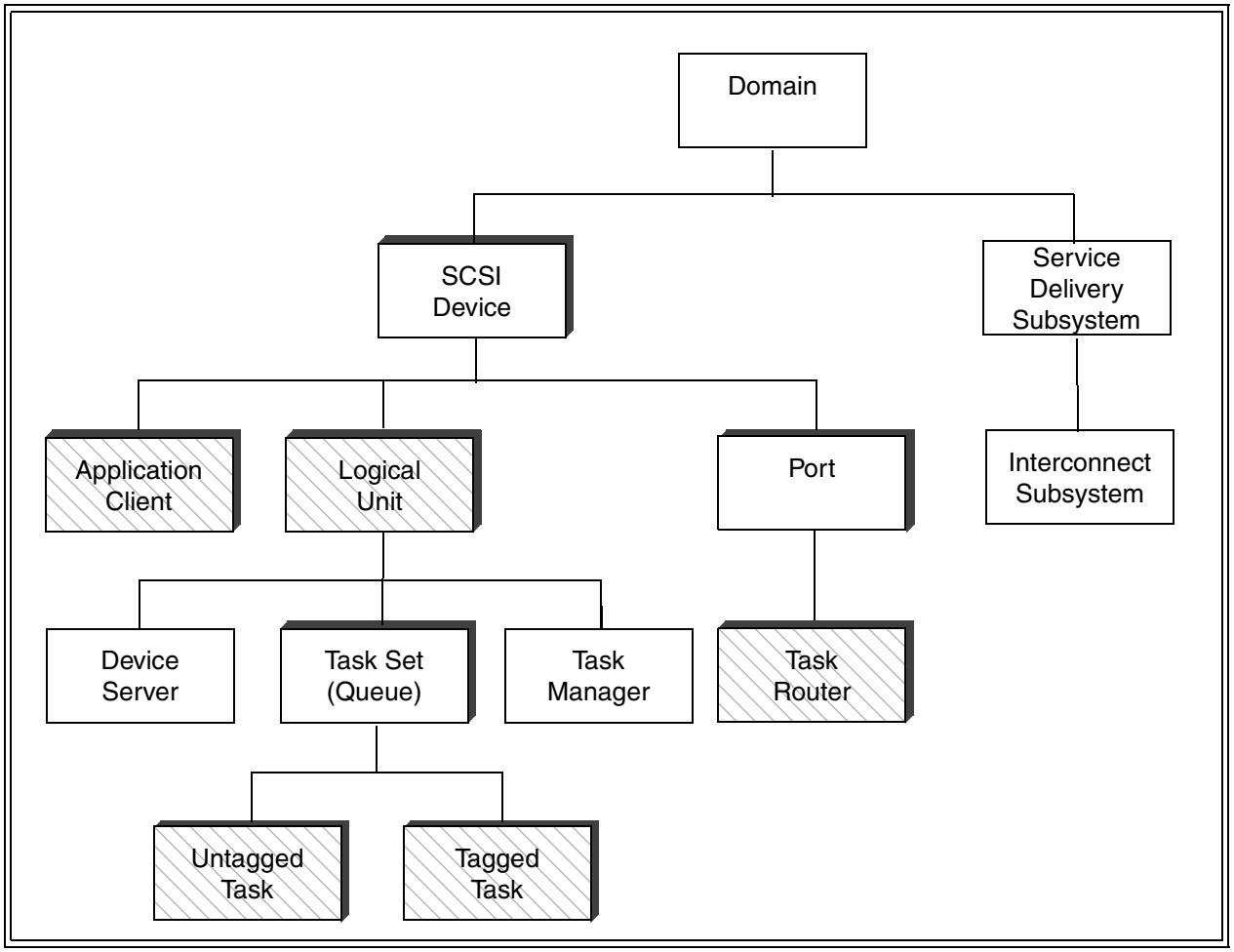



**Figure 8 — Overall SCSI domain model**

## 4.5 SCSI domain

A SCSI domain is composed of at least one SCSI device, at least one target port and at least one initiator port interconnected by a service delivery subsystem (see figure 9).

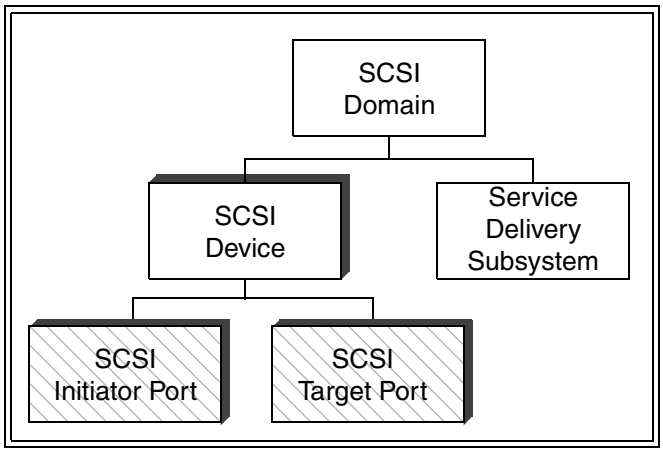



**Figure 9 — SCSI domain model**

A SCSI device is an object that originates or services SCSI commands. As described in 4.7, when a SCSI device originates a command it is called a SCSI initiator device and that command is transmitted through an initiator port or a SCSI target/initiator port. A SCSI device containing logical units that service commands is called a SCSI target device and receives commands through a SCSI target port or a SCSI target/initiator port. The service delivery subsystem connects all the SCSI ports in the SCSI domain, providing a subsystem through which application clients and device servers communicate (see 4.6). The boundaries of a SCSI domain are established by the system implementor, within the constraints of a specific SCSI protocol and interconnect standards.

## 4.6 The service delivery subsystem

The service delivery subsystem connects SCSI ports (see 3.1.94) and is composed of an interconnect subsystem (see figure 10).

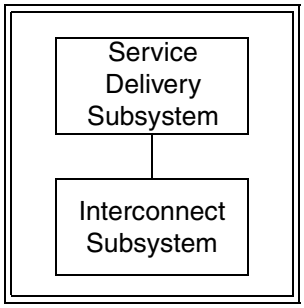



**Figure 10 — Service delivery subsystem model**

The interconnect subsystem is a set of one or more physical interconnects that appear to a client or server as a single path for the transfer of requests, responses, and data between SCSI Devices.

The service delivery subsystem is assumed to provide error-free transmission of requests and responses between client and server. Although a device driver in a SCSI implementation may perform these transfers through several interactions with its SCSI protocol layer, the architecture model portrays each operation, from the viewpoint of the application client, as occurring in one discrete step. In this model, the data comprising an outgoing request is sent

in a single package containing all the information required to process the remote procedure call. Similarly, an incoming server response is returned in a package enclosing the output data and status. The request or response package is sent when it is passed to the SCSI port for transmission; it is in transit until delivered and received when it has been forwarded to the receiver via the destination device's SCSI port.

### 4.6.1 Synchronizing client and server states

The client is usually informed of changes in server state through the arrival of server responses. In the architecture model such state changes occur after the server has sent the associated response and possibly before the response has been received by the SCSI initiator device. Some SCSI protocols, however, may require the SCSI target device to verify that the response has been received successfully before completing a state change. State changes controlled in this manner are said to be synchronized. Since synchronized state changes are not assumed or required by the architecture model, there may be a time lag between the occurrence of a state change within the SCSI target device and the SCSI initiator device's awareness of that change.

The model assumes that state synchronization, if required by a SCSI protocol standard, is enforced by the service delivery subsystem transparently to the server. That is, whenever the server invokes a SCSI protocol service to return a response as described in 6.9 and 5.4, it is assumed that the service delivery port for such a protocol does not return control to the server until the response has been successfully delivered to the SCSI initiator device.

### 4.6.2 Request/Response ordering

In this standard, request or response transactions are said to be in order if, relative to a given pair of sending and receiving SCSI ports, transactions are delivered in the order they were sent.

A sender may occasionally require control over the order in which its requests or responses are presented to the receiver (e.g., the sequence in which requests are received is often important whenever a SCSI initiator device issues a series of SCSI commands with the ORDERED attribute to a logical unit as described in clause 7). In this case, the order in which these commands are completed, and hence the final state of the logical unit, may depend on the order in which these commands are received. Similarly, the SCSI initiator device acquires knowledge about the state of pending commands and task management functions and may subsequently take action based on the nature and sequence of SCSI target device responses (e.g., if the SCSI initiator device aborts a command whose completion response is in transit and the abort response is received out of order, the SCSI initiator device could incorrectly conclude that no further responses are expected from that command).

The manner in which ordering constraints are established is vendor specific. An implementation may choose to delegate this responsibility to the application client (e.g., the device driver). In some cases, in-order delivery may be an intrinsic property of the service delivery subsystem or a requirement established by the SCSI protocol standard.

The SCSI architecture model assumes in-order delivery to be a property of the service delivery subsystem. This assumption is made to simplify the description of behavior and does not constitute a requirement. In addition, this specification makes no assumption about, or places any requirement on the ordering of requests or responses between tasks or task management functions received from different SCSI initiator ports.

## 4.7 SCSI devices

A SCSI device is a SCSI target device, a SCSI initiator device, or a SCSI target/initiator device.

A SCSI initiator device contains at least one SCSI initiator port and is capable of originating SCSI commands and task management requests (see 4.7.1). A SCSI target device contains at least one SCSI target port and is capable of processing SCSI commands and task management requests (see 4.7.2). A SCSI target/initiator device contains at least one SCSI target/initiator port and is capable of originating and processing SCSI commands and task management requests (see 4.7.3). To be functional, a SCSI domain needs to contain a SCSI target port or a SCSI target/initiator port operating as a SCSI target port and a SCSI initiator port or SCSI target/initiator port operating as a SCSI initiator port.

### 4.7.1 SCSI initiator device

A SCSI initiator device (see figure 11) contains:

a) Zero or more initiator device names;
b) One or more SCSI initiator ports each containing an initiator port identifier and an optional initiator port name;
c) Zero or more application clients.

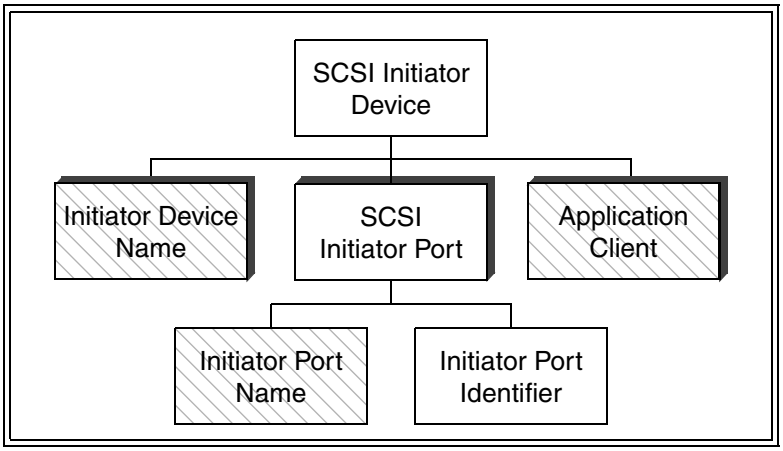



**Figure 11 — SCSI initiator device model**

An initiator port identifier is a value that is the SCSI port identifier (see 4.7.4) for an initiator port.

An initiator device name is a name (see 3.1.67) that is a SCSI device name (see 4.7.6) for a SCSI initiator device. A SCSI initiator device shall have no more than one initiator device name for each supported SCSI protocol. A SCSI protocol standard may place additional requirements on initiator device names.

An initiator port name is a name (see 3.1.67) that is the SCSI port name (see 4.7.7) for the initiator port. A SCSI protocol standard may place additional requirements on initiator port names.

An application client is the source of commands and task management functions. This model assumes that a SCSI initiator device contains one application client for each pending command or task management function.

### 4.7.2 SCSI target device

A SCSI target device (see figure 12) contains:

a) Zero or more target device names;
b) One or more SCSI target ports each containing a task router, SCSI target port identifier, and an optional target port name; and
c) One or more logical units.

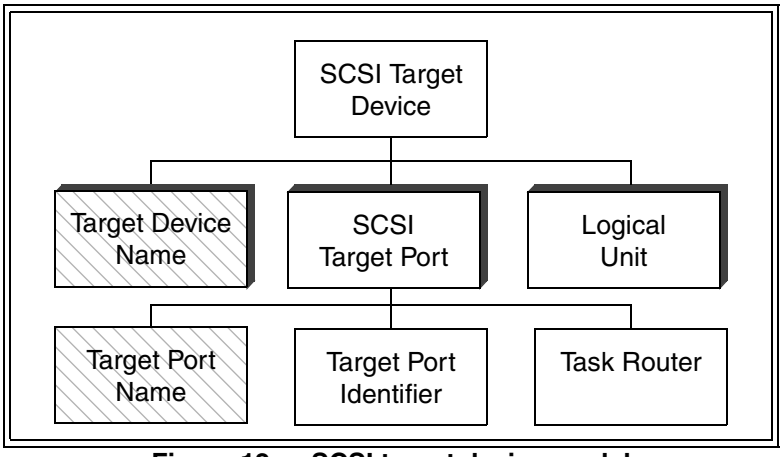



**Figure 12 — SCSI target device model**

A SCSI target port identifier is a value that is a SCSI port identifier (see 4.7.4) for a SCSI target port.

A target device name is a name (see 3.1.67) that is a SCSI device name (see 4.7.6) for a SCSI target device. A SCSI target device shall have no more than one target device name for each supported SCSI protocol. A SCSI protocol standard may place additional requirements on target device names.

A target port name is a name (see 3.1.67) that is the SCSI port name (see 4.7.7) for the target port. A SCSI protocol standard may place additional requirements on target port names.

A task router routes commands and task management functions between the service delivery subsystem and the appropriate logical unit's task manager (see 4.7.5).

A logical unit is the object to which SCSI commands are addressed. One of the logical units within the SCSI target device shall be accessed using the logical unit number zero. See 4.8 for a description of the logical unit.

### 4.7.3 SCSI target/initiator device

A SCSI target device (see figure 13) contains:

a) Zero or more target/initiator device names;
b) One or more SCSI target/initiator ports each containing a task router, target port identifier, an initiator port identifier, an optional target port name and an optional initiator port name;
c) One or more logical units; and
d) Zero or more application clients.

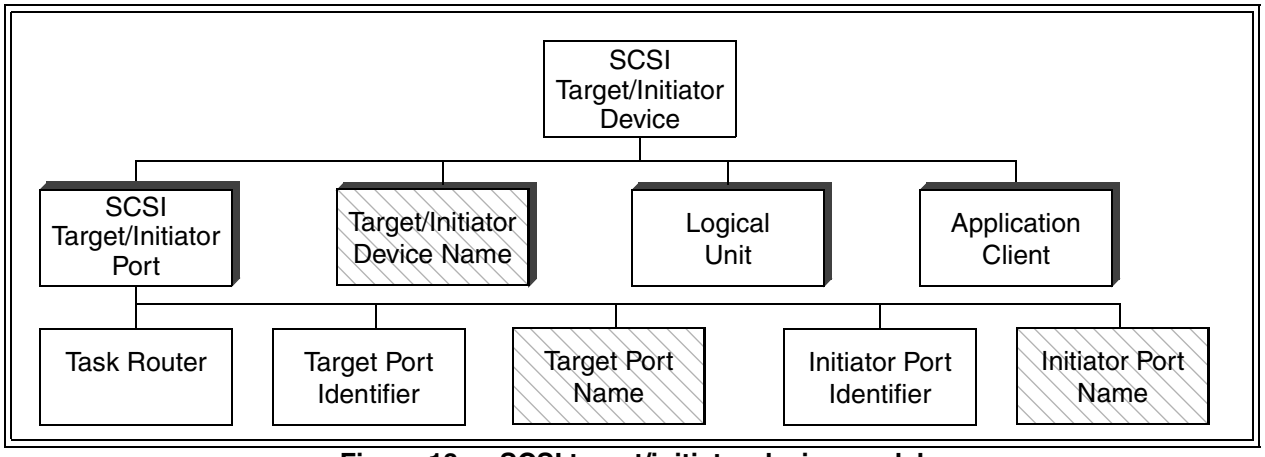



**Figure 13 — SCSI target/initiator device model**

The target port identifier and the initiator port identifier are values containing a SCSI port identifier (see 4.7.4) for a SCSI target/initiator port. The target port identifier and the initiator port identifier may or may not be identical.

A target/initiator device name is a name (see 3.1.67) that is a SCSI device name (see 4.7.6) for a SCSI target/initiator device. A SCSI target/initiator device shall have no more than one target/initiator device name for each supported SCSI protocol. A SCSI protocol standard may place additional requirements on target/initiator device names.

The target port name and initiator port name are names (see 3.1.67) that are the SCSI port name (see 4.7.7) for the target/initiator port when operating as a target port and initiator port, respectively. The target port name and the initiator port name may or may not be identical. A SCSI protocol standard may place additional requirements on target port names and initiator port names.

When the SCSI target/initiator device is operating as a SCSI target device a task router routes the commands and task management functions between the service delivery subsystem and the appropriate logical unit (see 4.7.5). A logical unit is the object to which SCSI commands are sent. One of the logical units within the SCSI target/initiator device shall be accessed using the logical unit number zero. See 4.8 for a description of the logical unit.

When the SCSI target/initiator device is operating as a SCSI initiator device an application client is the source of commands and task management functions. This model assumes that there is one application client for each pending command or task management function.

### 4.7.4 SCSI port identifier

The SCSI port identifier is equivalent to SCSI identifier and is the object name used to represent either an initiator port identifier for an initiator, or a target port identifier for a target. SCSI port identifier is used when either a SCSI initiator port or SCSI target port might be applicable or when other context in the description identifies the SCSI initiator port or SCSI target port usage.

### 4.7.5 SCSI task router

The task router routes tasks and task management functions to the selected logical unit. Any task that is sent to a logical unit that is not known to the task router shall be routed to a default logical unit (e.g., LUN 0). Any task management function that is not sent to a specific logical unit shall be broadcast to all logical units known to the task router.

### 4.7.6 SCSI device name

A SCSI device name is an optional name (see 3.1.67) for a SCSI device that is world wide unique within the protocol of a SCSI domain in which the SCSI device has SCSI ports. A SCSI device may have more than one name if that device has SCSI ports in different SCSI protocol domains. A SCSI device shall have no more than one name for each supported SCSI protocol. A SCSI device name shall never change and may be used to persistently identify a SCSI device in contexts where specific references to port names or port identifiers is not required.

A SCSI protocol standard may require that a SCSI device include a SCSI device name if the SCSI device has SCSI ports in a SCSI domain of that protocol. The SCSI device name may be made available to other SCSI devices or SCSI ports in a given SCSI domain in protocol specific ways.

### 4.7.7 SCSI port name

A SCSI port name is an optional name (see 3.1.67) of a SCSI port that is world wide unique within the protocol of the SCSI domain of that SCSI port. A SCSI port may have at most one name. A SCSI port name shall never change and may be used to persistently identify a SCSI initiator port or SCSI target port in contexts similar to those where a SCSI port identifier (see 4.7.4) may be used.

A SCSI protocol standard may require that a SCSI port include a SCSI port name if the SCSI port is in a SCSI domain of that protocol. The SCSI port name may be made available to other SCSI devices or SCSI ports in the given SCSI domain in protocol specific ways.

## 4.8 Logical units

A logical unit (see figure 14) contains:

a) A logical unit number;
b) A device server;
c) A task manager; and
d) One or more task sets each may contain zero or more untagged tasks or a combination of zero or more tagged tasks and zero or more untagged tasks.

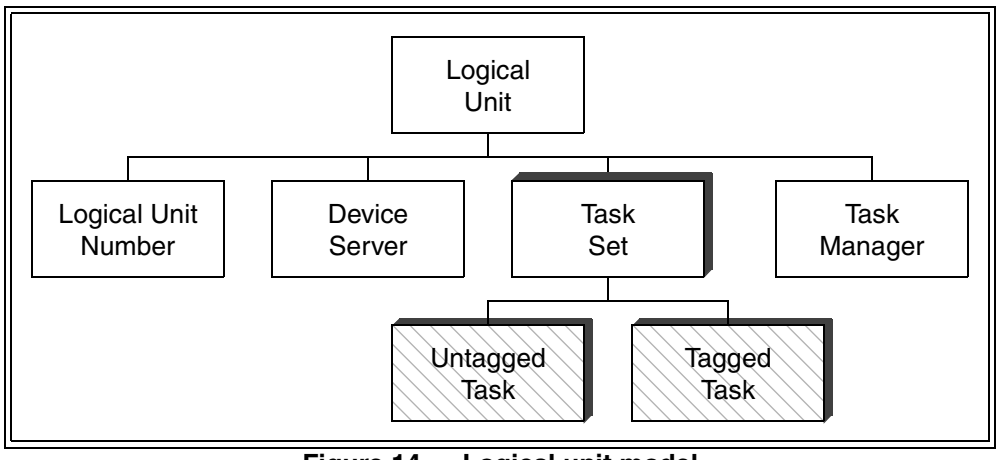



**Figure 14 — Logical unit model**

A logical unit number is a field containing up to 64 bits that identifies the logical unit within a SCSI target device. If a SCSI target device contains 256 or fewer logical units none of which are dependent logical units (see 4.12) or extended addressing logical units (see 4.13), then its Logical Unit Numbers shall have the format shown in table 1, which is a single level subset of the format described in 4.12.

**Table 1 — Single Level LUN structure**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | ADDRESS METHOD (00b) | | BUS IDENTIFIER (00h) | | | | | |
| 1 | SINGLE LEVEL LUN (00h to FFh, inclusive) | | | | | | | |
| 2 | (MSB) | | Null second level LUN (0000h) | | | | | |
| 3 | | | | | | | | (LSB) |
| 4 | (MSB) | | Null third level LUN (0000h) | | | | | |
| 5 | | | | | | | | (LSB) |
| 6 | (MSB) | | Null forth level LUN (0000h) | | | | | |
| 7 | | | | | | | | (LSB) |

In the single level subset format, all LUN structure fields shall be zero except the SINGLE LEVEL LUN field (see table 1). The value in the single level LUN field shall be between 0 and 255. The 00b in the ADDRESS METHOD field and the 00h in the BUS IDENTIFIER field indicate addressing for a logical unit at the current level (see 4.12.3). When the single level subset format is used, the HISUP bit shall be set to one in the standard INQUIRY data (see SPC-2) returned by logical unit 0.

If any Logical Unit within the scope of a SCSI target device includes dependent logical units in its composition, all logical unit numbers within the scope of the SCSI target device shall have the format described in 4.12.

A device server is the object that processes the operations requested by the received commands.

The task manager controls the sequencing of one or more tasks within a logical unit. The task manager also carries out the task management functions specified in clause 6. There is one task manager per logical unit.

The order in which task management requests are processed is not specified by this standard. This standard does not require in-order delivery of such requests, as defined in 4.6.2, or processing by the task manager in the order received. To guarantee the processing order of task management requests referencing a specific logical unit, an initiator should, therefore, not have more than one such request pending to that logical unit.

A task set is composed of zero or more untagged tasks or a combination of zero or more tagged tasks and zero or more untagged tasks. See 4.9 for additional restrictions on the untagged tasks and tagged tasks in a task set.

For convenience, task (see 4.9) refers to either a tagged task or an untagged task. The interactions among the tasks in a task set are determined by the requirements for task set management specified in clause 7 and the ACA and CA requirements specified in 5.8.1. The number of task sets per logical unit and the boundaries between task sets are governed by the TST field in the Control mode page (see SPC-2).

## 4.9 Tasks

### 4.9.1 The task object

The task object represents either a tagged task or an untagged task. The composition of a task includes a definition of the work to be performed by the logical unit in the form of a command or a group of linked commands. A tagged task is represented by an I_T_L_Q nexus (see 4.10) and is composed of a definition of the work to be performed by the logical unit, and a task attribute (see 7.5). An untagged task is represented by an I_T_L nexus (see 4.10) and is composed of a definition of the work to be performed by the logical unit, and implicitly a SIMPLE task attribute (see 7.5).

The I_T_L_Q nexus representing a tagged task includes a tag (see 4.9.2) allowing many uniquely identified tagged tasks to be present concurrently in a single task set. A tagged task also includes one of the task attributes described in 7.5 that allows the initiator to specify processing relationships between various tagged tasks. An untagged task does not include a tag in its I_T_L nexus, thus restricting the number of concurrent untagged tasks in a single task set to one per initiator. Also, an untagged task is assumed to have a SIMPLE task attribute, leaving the initiator no control over its relationship to other tasks in the task set.

Every SCSI protocol shall support tagged tasks and may support untagged tasks. If the SCSI protocol upon which a SCSI device operates supports untagged tasks, the SCSI device is not required to support tagged tasks.

An I_T_L_x nexus that is in use shall be unique as seen by the initiator originating the command and the logical unit to which the command was addressed, otherwise an overlapped command condition exists (see 5.8.2). An I_T_L_x nexus is in use over the interval bounded by the events specified in 5.5). An I_T_L_x nexus is unique if one or more of its components is unique within the specified time interval. An untagged task shall be unique with respect to all tagged tasks in the task set.

By implication, therefore, an initiator shall not cause the creation of more than one untagged task having identical values for the target identifier and logical unit number. An initiator also shall not create more than one task having identical values for the target identifier, logical unit number, and tag.

### 4.9.2 Task tags

A tag is a field containing up to 64 bits that is a component of an I_T_L_Q nexus. An initiator assigns tag values in each I_T_L_Q nexus in a way that ensures that the nexus uniqueness requirements stated in 4.9.1 are met.

## 4.10 The nexus object

The nexus object is a relationship between a SCSI initiator port, a SCSI target port, optionally a logical unit. and optionally a task.

The nexus object may refer to any one or all of the following relationships:

a) One SCSI initiator port to one SCSI target port (an I_T nexus);
b) One SCSI initiator port to one SCSI target port to one logical unit (an I_T_L nexus);
c) One SCSI initiator port to one SCSI target port to one logical unit to one tagged task (an I_T_L_Q nexus); or
d) Either an I_T_L nexus or an I_T_L_Q nexus (denoted as an I_T_L_x nexus).

Table 2 maps the nexus object to other identifier objects.

**Table 2 — Mapping nexus to SAM-2 identifiers**

| Nexus | Identifiers that form nexus | Reference |
|---|---|---|
| I_T | Initiator Port Identifier<br>Target Port Identifier | 4.7.1<br>4.7.2 |
| I_T_L | Initiator Port Identifier<br>Target Port Identifier<br>Logical Unit Number | 4.7.1<br>4.7.2<br>4.8 |
| I_T_L_Q | Initiator Port Identifier<br>Target Port Identifier<br>Logical Unit Number<br>Tag | 4.7.1<br>4.7.2<br>4.8<br>4.9.2 |

## 4.11 SCSI ports

### 4.11.1 SCSI port configurations

A SCSI device may contain only SCSI target ports, only SCSI initiator ports, only SCSI target/initiator ports or any combination of ports. Some of the port configurations possible for a SCSI device are shown in figure 15.

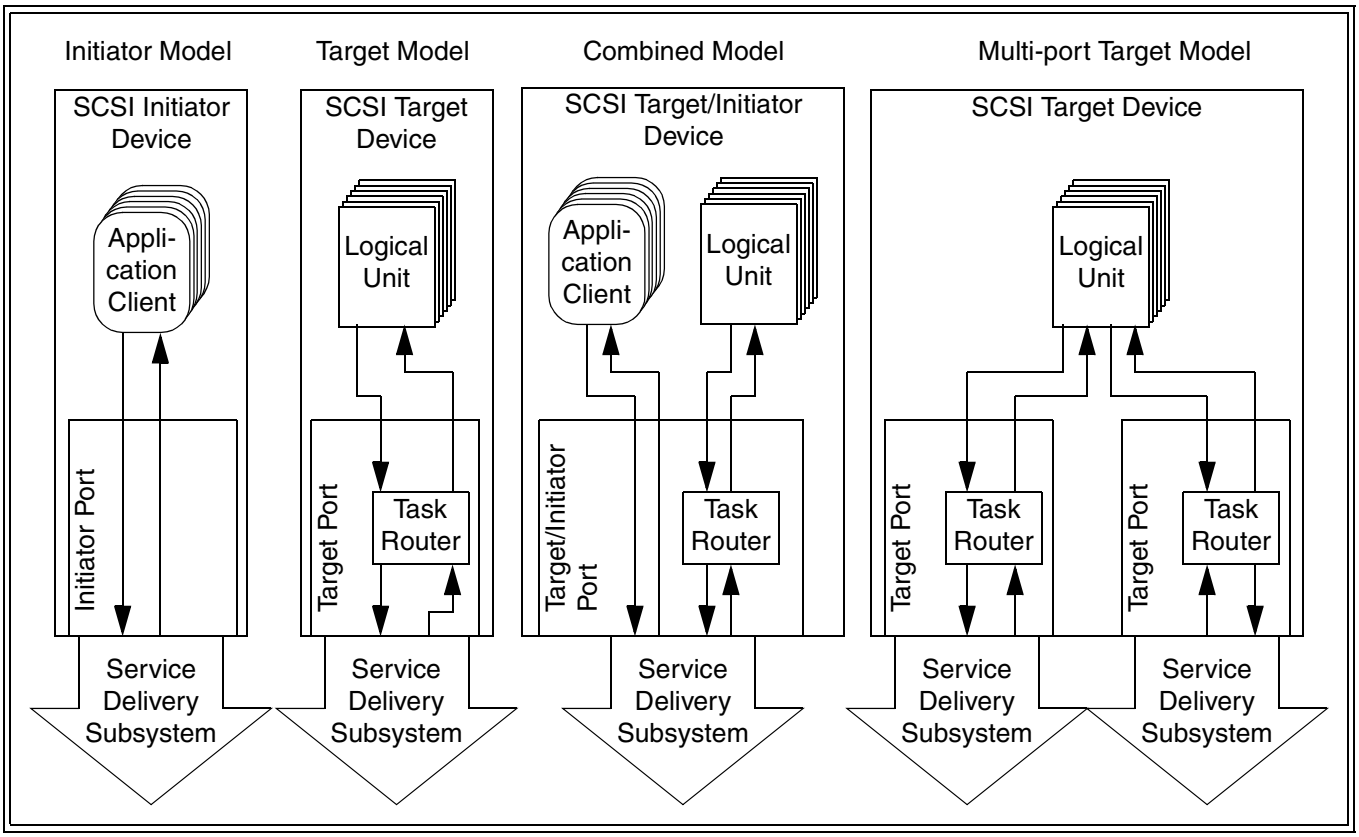



**Figure 15 — SCSI device functional models**

A target/initiator SCSI device is referred to by the role it's port takes when it participates in an I/O operation. When a SCSI target/initiator device receives SCSI commands or task management functions, the SCSI target/initiator device takes on the characteristics of and is referred to as a SCSI target device. When a SCSI target/initiator device issues SCSI commands or task management functions, the SCSI target/initiator device takes on the characteristics of and is referred to as a SCSI initiator device.

### 4.11.2 SCSI devices with multiple ports

The model for a SCSI device with multiple ports is a single SCSI target device (see 4.7.2), SCSI initiator device (see 4.7.1), or SCSI target/initiator device (see 4.7.3) with multiple ports. Similarly, a single SCSI target port or SCSI initiator port may respond to multiple SCSI identifiers. The model for such a SCSI device also is one of multiple SCSI target ports or SCSI initiator ports, one for each SCSI identifier.

The SCSI identifiers representing the ports shall meet the requirements for initiator port identifiers (see 4.7.1) or target port identifiers (see 4.7.2) or both. SCSI target/initiator devices with multiple ports implement both target and initiator models and combine the SCSI target/initiator port structures in vendor specific ways that meet product requirements while maintaining the multi-port model for the target and initiator functions performed by the product. How a multiple port SCSI device is viewed by counterpart SCSI devices in the SCSI domain also depends on whether a SCSI initiator port is examining a SCSI target port or SCSI target/initiator port, or a SCSI target port is servicing a SCSI initiator port or SCSI target/initiator port. The structures and views of SCSI devices are asymmetric for SCSI target ports, and SCSI initiator ports.

### 4.11.3 Multiple port target SCSI device structure

Figure 16 shows the structure of a SCSI target device with multiple SCSI target ports. Each SCSI target port consists of a task router that is shared by a collection of logical units. Each logical unit contains a single task manager and a device server.

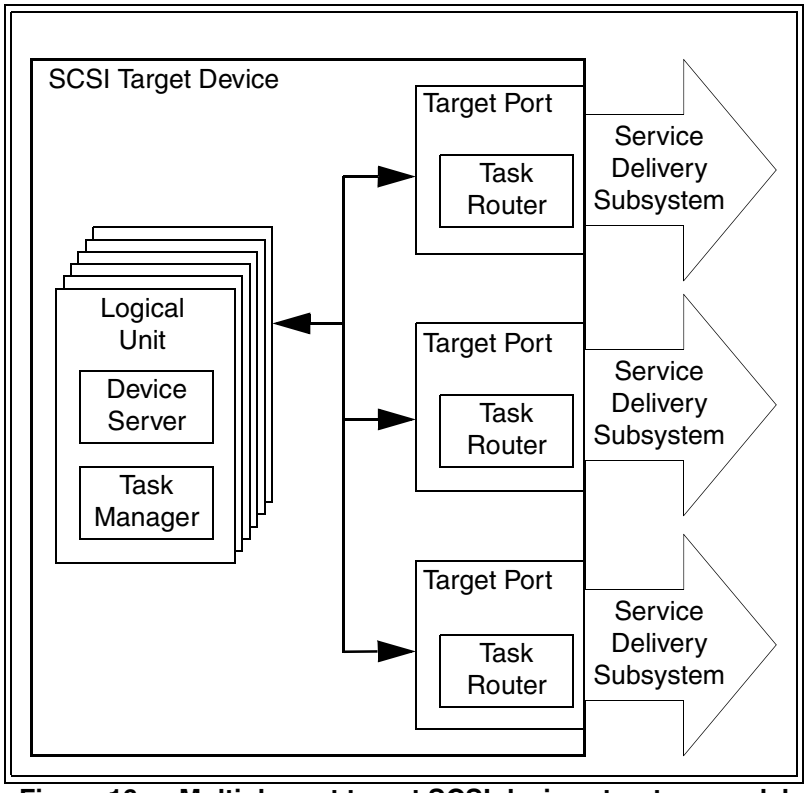



**Figure 16 — Multiple port target SCSI device structure model**

Two-way communications shall be possible between all logical units and all SCSI target ports, however, communications between any logical unit and any SCSI target port may occasionally be inactive. Two-way communications shall be available between each task manager and all task routers. Each SCSI target port shall accept commands sent to LUN 0 and the task router shall route them to a device server for processing. The REPORT LUNS commands (see SPC-2) shall be accepted by logical unit 0 from any SCSI target port and shall return the logical unit inventory available via that SCSI target port. The availability of the same logical unit through multiple SCSI target ports is discovered by matching SCSI port identifier values in the INQUIRY command Device Identification VPD page (see SPC-2).

### 4.11.4 Multiple port initiator SCSI device structure

Figure 17 shows the structure of a SCSI initiator device with multiple SCSI initiator ports. Each SCSI initiator port is shared by a collection of application clients.

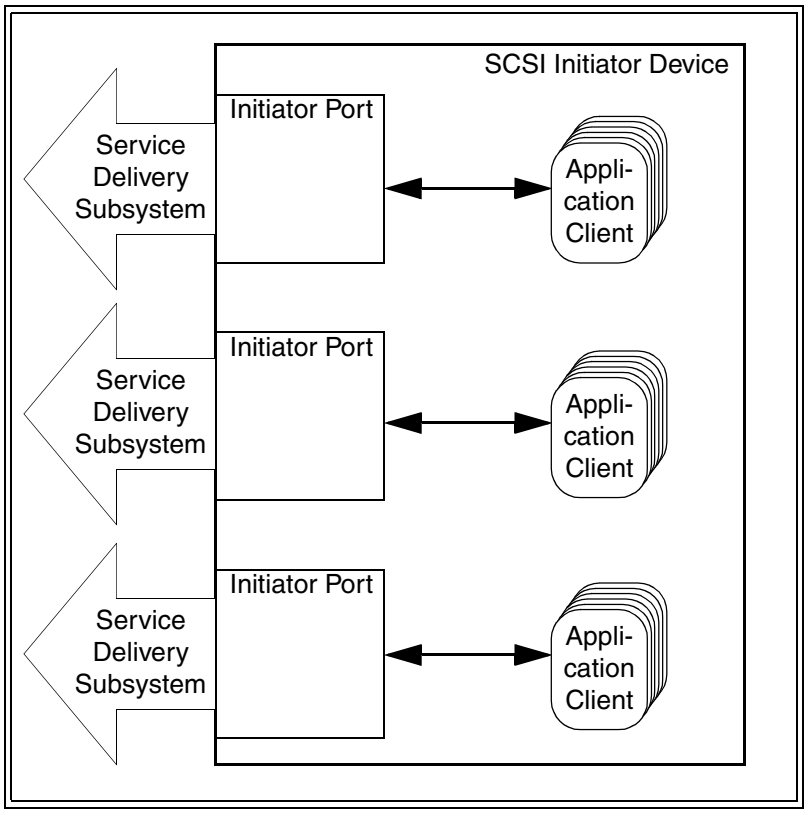



**Figure 17 — Multiple port SCSI initiator device structure model**

Two-way communications shall be possible between an application client and it's associated SCSI initiator port. Mechanisms by which a SCSI target device would have the ability to discover that it is communicating with multiple ports on a single SCSI initiator device are beyond the scope of any standards in the SCSI family of standards.

### 4.11.5 Multiple port target/initiator SCSI device structure

Figure 18 shows the structure of a SCSI target/initiator device with multiple SCSI target/initiator ports. Each SCSI target/initiator port consists of a task router and is shared by a collection of logical units and application clients. Each logical unit contains a single task manager and a device server.

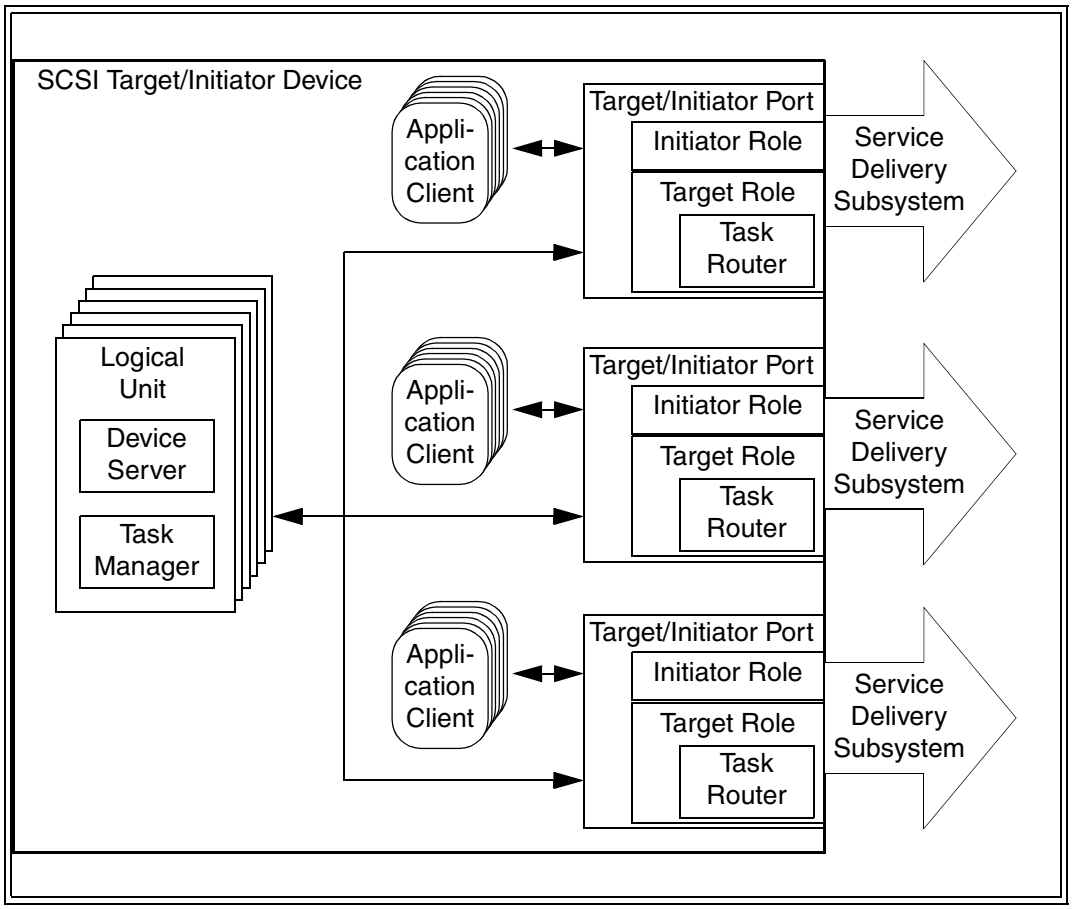



**Figure 18 — Multiple port target/initiator SCSI device structure model**

Two-way communications shall be possible between all logical units and all SCSI target/initiator ports, however, communications between any logical unit and any SCSI target/initiator port may occasionally be inactive. Two-way communications shall be possible between an application client and it's associated SCSI target/initiator port. Each SCSI target/initiator port shall accept commands sent to LUN 0 and the task router shall route them to a device server for processing. The REPORT LUNS commands (see SPC-2) shall be accepted by logical unit 0 from any SCSI target/initiator port and shall return the logical unit inventory available via that SCSI target/initiator port. The availability of the same logical unit through multiple SCSI target/initiator ports is discovered by matching SCSI port identifier values in the INQUIRY command Device Identification VPD page (see SPC-2).

Mechanisms by which a SCSI target device would have the ability to discover that it is communicating with multiple ports on a SCSI target/initiator device are beyond the scope of any standards in the SCSI family of standards.

### 4.11.6 SCSI initiator device view of a multiple port SCSI target device

A SCSI target device may be connected to multiple domains such that a SCSI initiator port is only allowed to only communicate with logical units using a single SCSI target port. This would restrict application clients from determining if a SCSI target device has multiple SCSI ports.

However, SCSI target devices with multiple SCSI ports may be configured where application clients have the ability to discover that one or more logical units are accessible via multiple SCSI target ports. Figure 19 and figure 20 show two examples of such configurations.

Figure 19 shows a SCSI target device with multiple SCSI target ports participating in a single SCSI domain with two SCSI initiator devices. There are three SCSI devices, one of which has two SCSI target ports, one with one SCSI initiator port, and one with one SCSI initiator port. There are two SCSI target port identifiers and two initiator port identifiers in this SCSI domain. Using the INQUIRY command Device Identification VPD page (see SPC-2), the application clients in each of the SCSI initiator devices have the ability to discover the logical units in the SCSI target devices are accessible via multiple SCSI target port identifiers (i.e., SCSI target ports) and map the configuration of the SCSI target devices.

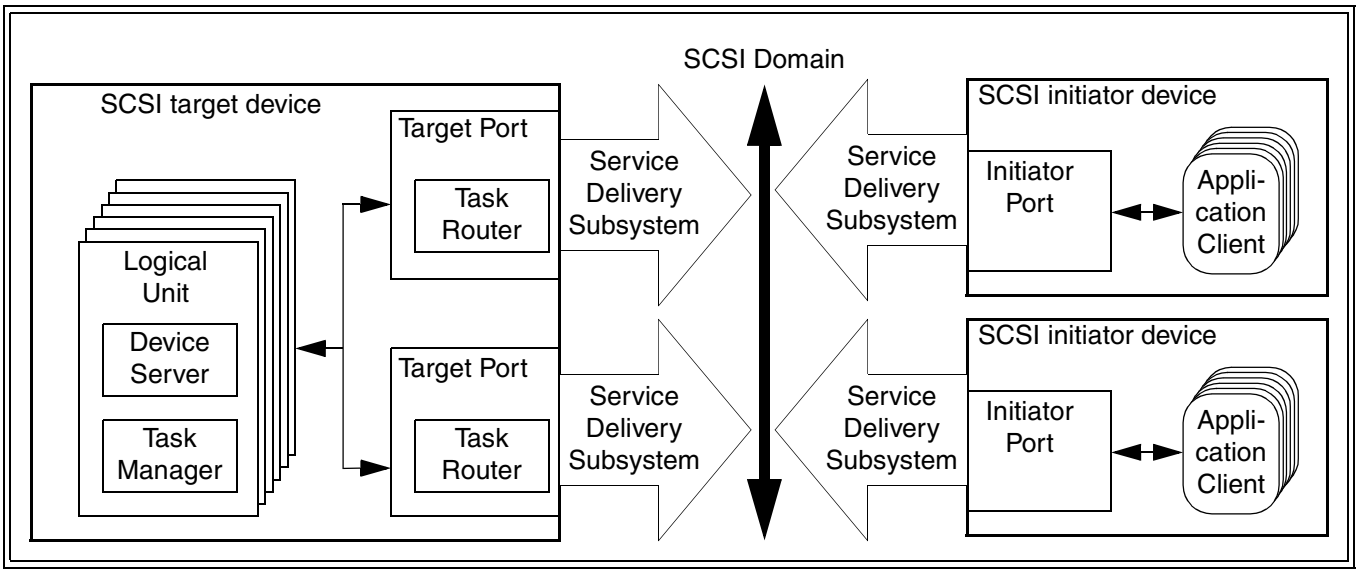



**Figure 19 — SCSI target device configured in a single SCSI domain**

Figure 20 shows a SCSI target device with multiple SCSI target ports participating in two SCSI domains and a SCSI initiator device with multiple SCSI initiator ports participating in the same two SCSI domains. There is one SCSI target device with two SCSI target ports and one SCSI initiator device with two SCSI initiator ports. There is one SCSI target port identifier and one initiator port identifier in each of the two SCSI domains. Using the INQUIRY

command Device Identification VPD page (see SPC-2), the application clients in the SCSI initiator device have the ability to discover that logical units in the SCSI target device are accessible via multiple ports and map the configuration. However, the methods available to application clients to distinguish between the configuration shown in figure 20 and the configuration shown in figure 19 are beyond the scope of the SCSI family of standards.

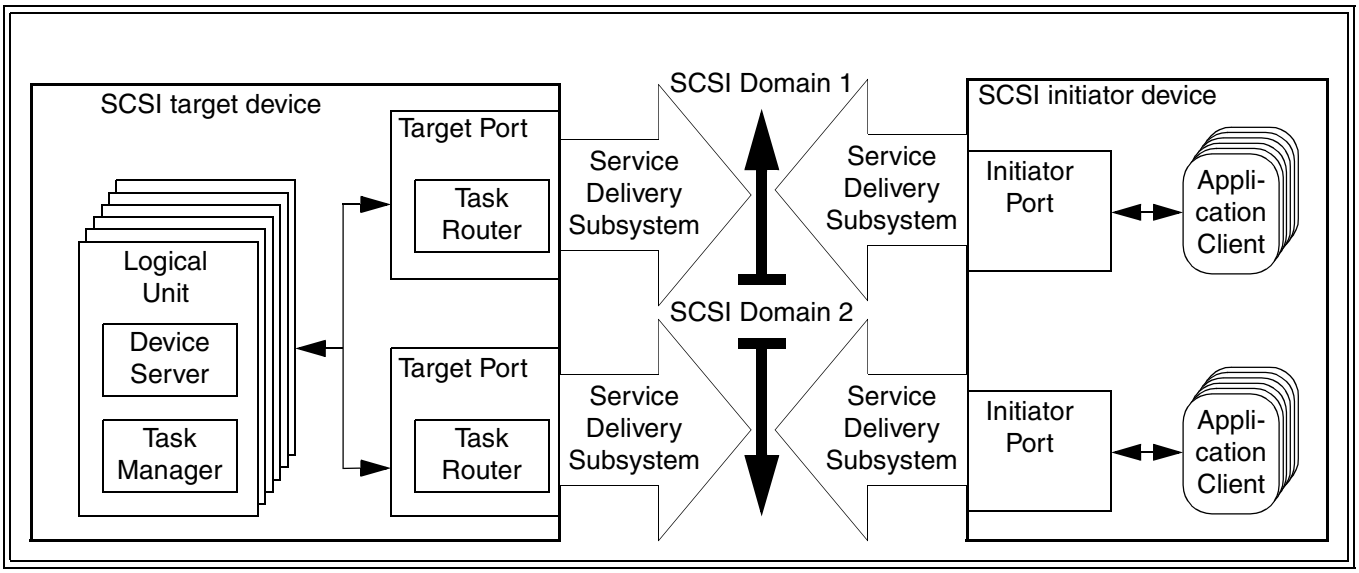



**Figure 20 — SCSI target device configured in multiple SCSI domains**

Figure 21 shows the same configuration as figure 20 except that the two SCSI domains have been replaced by a single SCSI domain.

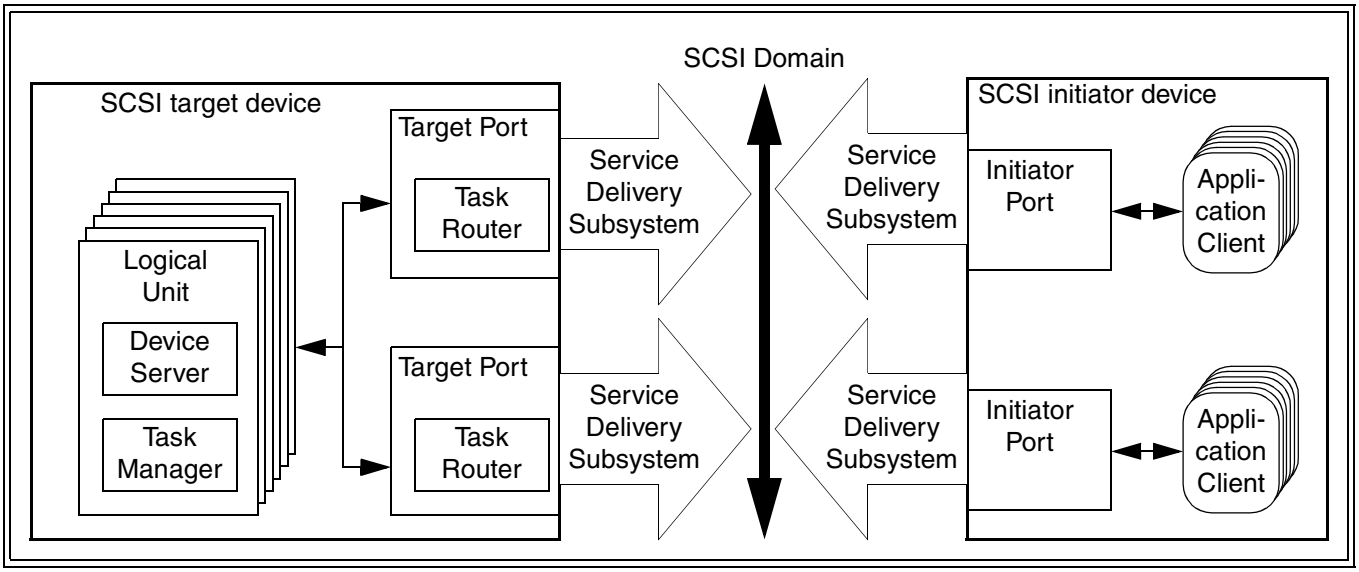



**Figure 21 — SCSI target device and SCSI initiator device configured in a single SCSI domain**

This model for application client determination of multiple SCSI target port configurations relies on information that is available only to the application clients via SCSI commands. The SCSI initiator ports in the SCSI initiator devices (figure 19) or SCSI initiator device (figure 20 and figure 21) are unable to distinguish the multiple SCSI target ports from individual SCSI target ports in two separate SCSI target devices.

### 4.11.7 SCSI target device view of a multiple port SCSI initiator device

An SCSI target device does not have the ability to detect the presence of an SCSI initiator device with multiple SCSI initiator ports. Therefore, a SCSI target device handles an SCSI initiator device with multiple SCSI initiator ports exactly as it would handle multiple separate SCSI initiator devices. For example, an SCSI target device handles the configurations shown in figure 20 and figure 21 in exactly the same way it handles the configuration show in figure 19.

NOTE 1 - The implications of this view of an SCSI initiator device are more far reaching than are immediately apparent. For example, if an SCSI initiator device makes an exclusive access reservation via one SCSI initiator port, then access will be denied to the other SCSI initiator port(s) on that same SCSI initiator device.

## 4.12 Model for dependent logical units

### 4.12.1 Introduction

Optionally, the model for a logical unit (see 4.8) may include one or more unique logical units embedded within another logical unit. The embedded logical units are called dependent logical units (see 3.1.22). In such cases, the model hierarchy diagram in 4.8 is enhanced to become the diagram shown in figure 22.

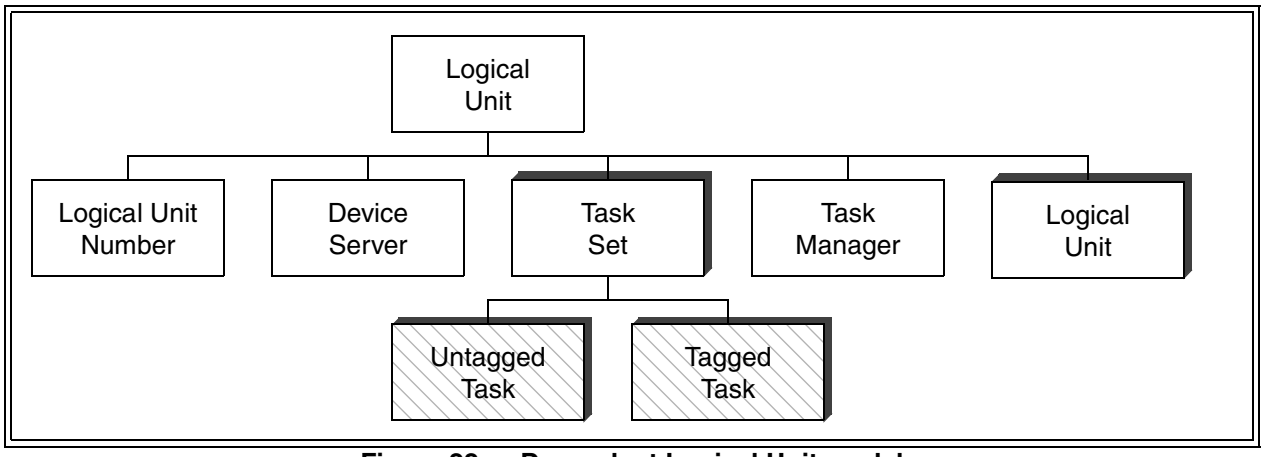



**Figure 22 — Dependent Logical Unit model**

When the dependent logical unit model is utilized, the hierarchical logical unit structure defined in this subclause shall be used. If any logical unit within a SCSI target device includes dependent logical units, all logical unit numbers within the SCSI target device shall have the format described in this subclause. A device server that implements the hierarchical structure for dependent logical units described in this subclause shall set the HISUP bit to one in the standard INQUIRY data returned by logical unit 0 (see SPC-2). In the cases defined by 4.8, SCSI target devices that do not implement dependent logical units are required to implement a subset of the logical unit structure described in this subclause.

As shown in figure 23, the hierarchical logical unit structure is an inverted tree containing up to four addressable levels. The example in figure 23 is a three-level system that consists of:

a) One initiator that has three SCSI devices attached on a single SCSI bus that is not expandable. One of the SCSI devices is a dual ported SCSI bridge controller.
b) One initiator has three SCSI devices attached on a single SCSI bus that is expandable. One of the SCSI devices contains a dual ported SCSI bridge controller.

c) The SCSI bridge controller has three SCSI buses with SCSI devices attached and is capable of driving more SCSI buses.
   A) Two of the SCSI buses contain two SCSI devices each and these SCSI buses are not expandable. One of the SCSI devices contains a SCSI bridge controller.
   B) One of the SCSI buses contains two SCSI devices and is expandable.
   C) The SCSI bridge controller has three SCSI buses with SCSI devices attached and is capable of driving more SCSI buses.
      a) Two of the SCSI buses contain two SCSI devices each and these SCSI buses are not expandable.
      b) One of the SCSI buses contains two SCSI devices and is expandable.

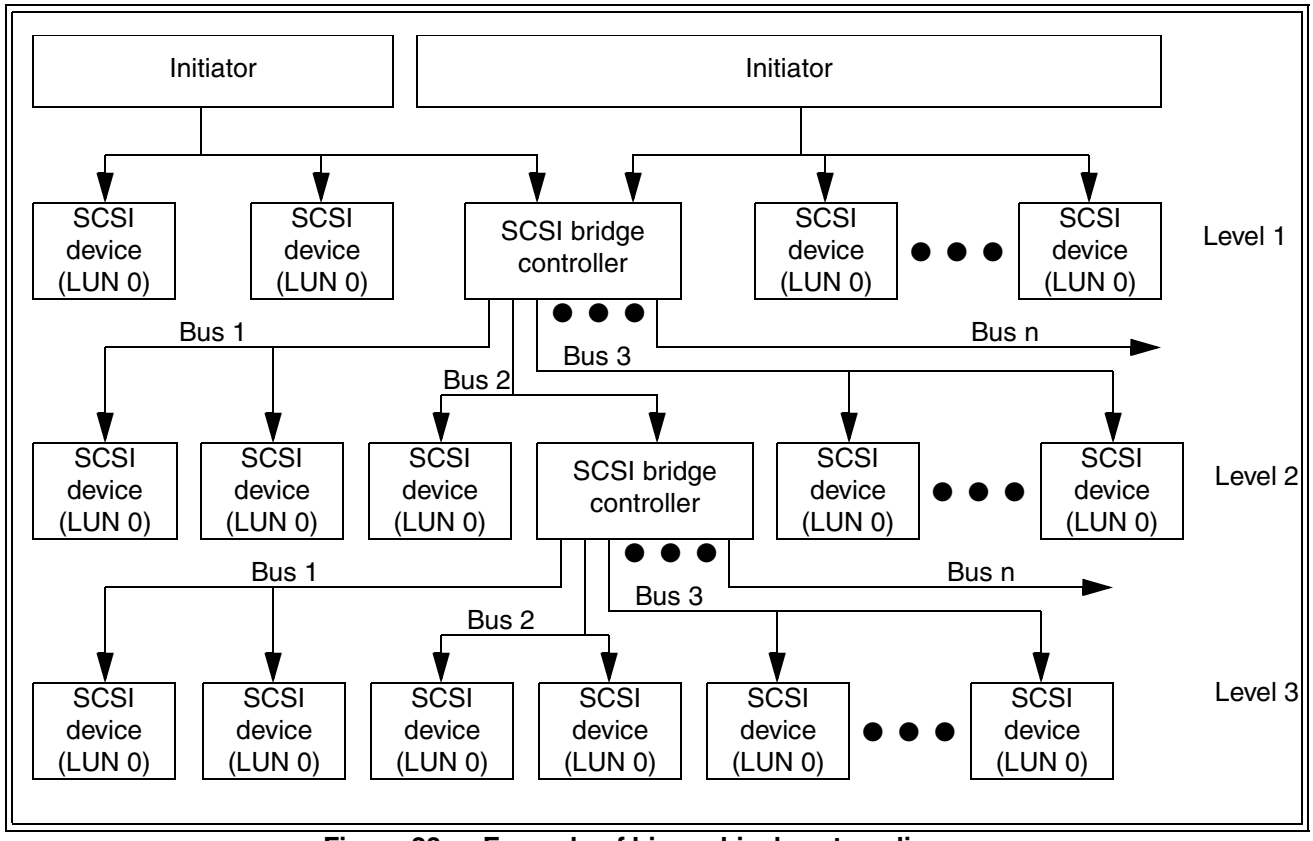



**Figure 23 — Example of hierarchical system diagram**

Devices at each level in the tree are referenced by one of the following address methods:

a) Logical unit address method (see 4.12.4);
b) Peripheral device address method (see 4.12.5); and
c) Device type specific.

All peripheral device addresses, except LUN 0 (see 4.12.2), default to vendor specific values. All addressable entities may default to vendor specific values or may be defined by an application client (e.g., by the use of SCC-2 configuration commands).

Within the hierarchical system there may be SCSI target devices that have multiple logical units connected to them through separate physical interconnects. These physical interconnects are referred to as buses. A SCSI target device that has SCSI devices attached to these buses shall assign numbers, other than zero, to those buses. The bus numbers shall be used as components of the logical unit numbers to the logical units attached to those buses, as described in the clauses below.

SCSI target devices shall assign a bus number of zero to all the logical units under control by the SCSI target device that are not connected through a separate physical interconnect.

### 4.12.2 LUN 0 address

All SCSI devices shall accept LUN 0 as a valid address. For SCSI devices that support the hierarchical addressing model the LUN 0 shall be the logical unit that an application client addresses to determine information about the target and the logical units contained within the target.

To address the LUN 0 of a SCSI device the peripheral device address method shall be used.

### 4.12.3 Eight byte LUN structure

The eight byte LUN structure (see table 4) allows up to four levels of devices to be addressed under a single target. Each level shall use byte 0 and byte 1 to define the address and/or location of the SCSI device to be addressed on that level.

If the LUN indicates that the command is to be relayed to the next layer then the current layer shall use byte 0 and byte 1 of the eight byte LUN structure to determine the address of the device to which the command is to be sent. When the command is sent to the target the eight byte LUN structure that was received shall be adjusted to create a new eight byte LUN structure (see table 3 and figure 24).

Devices shall keep track of the addressing information necessary to transmit information back through all intervening layers to the task's originating initiator.

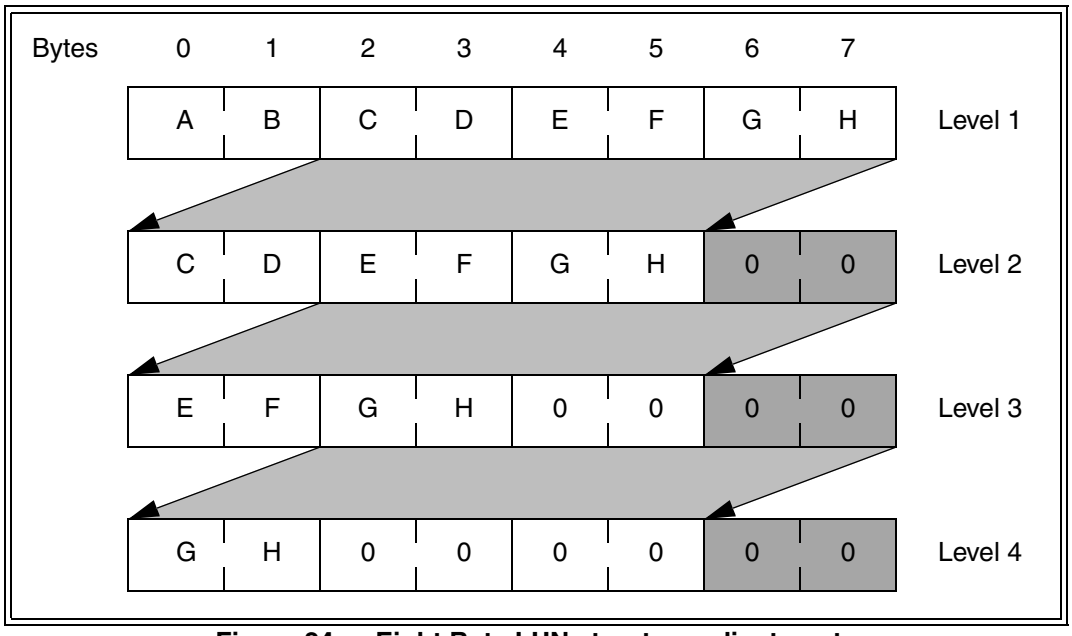



**Figure 24 — Eight Byte LUN structure adjustments**

**Table 3 — Eight byte LUN structure adjustments**

| Byte position | | |
|---|---|---|
| **Old** | | **New** |
| 0 & 1 | Moves to | Not Used |
| 2 & 3 | Moves to | 0 & 1 |
| 4 & 5 | Moves to | 2 & 3 |
| 6 & 7 | Moves to | 4 & 5 |
| N/A | zero fill | 6 & 7 |

The eight byte LUN structure requirements as viewed from the application client are shown in table 4.

**Table 4 — Eight Byte LUN structure**

| **Bit Byte** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (MSB) | | | | | | | |
| 1 | | | | FIRST LEVEL ADDRESSING | | | | (LSB) |
| 2 | (MSB) | | | | | | | |
| 3 | | | | SECOND LEVEL ADDRESSING | | | | (LSB) |
| 4 | (MSB) | | | | | | | |
| 5 | | | | THIRD LEVEL ADDRESSING | | | | (LSB) |
| 6 | (MSB) | | | | | | | |
| 7 | | | | FOURTH LEVEL ADDRESSING | | | | (LSB) |

The FIRST LEVEL ADDRESSING field indicates the first level address of a device. See table 5 for a definition of the FIRST LEVEL ADDRESSING field.

The SECOND LEVEL ADDRESSING field indicates the second level address of a device. See table 5 for a definition of the SECOND LEVEL ADDRESSING field.

The THIRD LEVEL ADDRESSING field indicates the third level address of a device. See table 5 for a definition of the THIRD LEVEL ADDRESSING field.

The FOURTH LEVEL ADDRESSING field indicates the fourth level address of a device. See table 5 for a definition of the FOURTH LEVEL ADDRESSING field.

The device pointed to in the FIRST LEVEL ADDRESSING, SECOND LEVEL ADDRESSING, THIRD LEVEL ADDRESSING, and FOURTH LEVEL ADDRESSING fields may be any physical or logical device addressable by an application client.

**Table 5 — Format of addressing fields**

| **Bit Byte** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| n-1 | ADDRESS METHOD | | (MSB) | | | | | |
| n | | | | ADDRESS METHOD SPECIFIC | | | | (LSB) |

The ADDRESS METHOD field defines the contents of the ADDRESS METHOD SPECIFIC field. See table 6 for the address methods defined for the ADDRESS METHOD field. The ADDRESS METHOD field only defines address methods for entities that are directly addressable by an application client.

**Table 6 — ADDRESS METHOD field values**

| Code | Description | Reference |
|---|---|---|
| 10b | Logical unit addressing method | 4.12.4 |
| 00b | Peripheral device addressing method | 4.12.5 |
| 01b | Flat space addressing method | 4.12.6 |
| 11b | Extended logical unit addressing method | 4.13 |

### 4.12.4 Logical unit addressing method

All SCSI commands are allowed when the logical unit address method is selected, however logical units are only required to support mandatory SCSI commands. Devices are not required to relay commands, from the application client, to a dependent logical unit. Any command that is not supported or relayed to a lower addressing layer shall be terminated with a CHECK CONDITION status. The sense key shall be set to ILLEGAL REQUEST and the additional sense code shall be set to INVALID COMMAND OPERATION CODE.

If the logical unit addressing method is selected the device shall relay the received command, if not filtered, to the addressed logical unit.

NOTE 2 - A SCSI device may filter commands to prevent an application client from issuing (e.g., a write command to a specific logical unit). A reason for doing this would be to prevent an application client from bypassing configuration requirements at an intermediate level of the hierarchy.

See table 7 for the definition of the ADDRESS METHOD SPECIFIC field used when the logical unit addressing method is selected.

**Table 7 — Logical unit addressing**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| n-1 | 1 | 0 | TARGET | | | | | |
| n | BUS NUMBER | | | LUN | | | | |

The TARGET field, BUS NUMBER field, and LUN field address the logical unit to which the received command shall be relayed. The command shall be relayed to the logical unit (LUN field value) within target (TARGET field value) located on bus (BUS NUMBER field value). The target information in the TARGET field may be a target identifier (see 4.7.2) or it may be a mapped representation of a target identifier, when the range of possible target identifiers is too large to fit in the TARGET field.

NOTE 3 - The value of targets within the TARGET field are defined by individual standards. (e.g., SCSI Parallel Interface -2 standard defines targets to be in the range 0 to 7, 0 to 15, and 0 to 31).

### 4.12.5 Peripheral device addressing method

All SCSI commands are allowed when the peripheral device address method is selected, however peripheral devices are only required to support mandatory SCSI commands. Devices are not required to relay commands, from the application client, to a lower layer. Any command that is not supported or relayed shall be terminated with a CHECK CONDITION status. The sense key shall be set to ILLEGAL REQUEST and the additional sense code shall be set to INVALID COMMAND OPERATION CODE.

If the peripheral device addressing method is selected the device shall relay the received command, if not filtered, to the addressed peripheral device.

NOTE 4 - A SCSI device may filter commands to prevent an application client from issuing (e.g., a write command to a specific peripheral device). A reason for doing this would be to prevent an application client from bypassing configuration requirements at an intermediate level of the hierarchy.

See table 8 for the definition of the ADDRESS METHOD SPECIFIC field used when the peripheral device addressing method is selected.

**Table 8 — Peripheral device addressing**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| n-1 | 0 | 0 | BUS IDENTIFIER | | | | | |
| n | TARGET/LUN | | | | | | | |

The BUS IDENTIFIER field identifies the bus or path that the SCSI device shall use to relay the received command. The BUS IDENTIFIER field may use the same value encoding as the BUS NUMBER field (see 4.12.4). However, bus identifier zero shall indicate that the command is to be relayed to a logical unit within the SCSI device at the current level.

The TARGET/LUN field indicates the address of the peripheral device to which the SCSI device shall relay the received command. The meaning and usage of the TARGET/LUN field depends on whether the BUS IDENTIFIER field contains zero.

A BUS IDENTIFIER field of zero indicates a logical unit at the current level. This representation of a logical unit may be used either when the SCSI device at the current level does not use hierarchical addressing for assigning LUNs to entities or when the SCSI device at the current level includes entities that need LUNs but are not attached to SCSI buses (e.g., fans, cache, controllers, etc.). When the BUS IDENTIFIER field contains zero, the command shall be relayed to the current level logical unit (TARGET/LUN field value) within or joined to the current level SCSI device.

A bus identifier field greater than zero represents physical SCSI interconnect that connects a group of SCSI devices to the current level SCSI device. Each physical interconnect shall be assigned a unique number from 1 to 63. These bus identifiers shall be used in the BUS IDENTIFIER field when assigning addresses to peripheral devices attached to the physical interconnects. When the BUS IDENTIFIER field is greater than zero, the command shall be relayed to the logical unit zero within target (TARGET/LUN field value) located physical interconnect (BUS IDENTIFIER field value). The target information in the TARGET/LUN field may be a target identifier (see 4.7.2) or it may be a mapped representation of a target identifier, when the range of possible target identifiers is too large to fit in the TARGET/LUN field.

NOTE 5 - The value of target identifiers within the TARGET/LUN field are defined by individual standards. (e.g., SCSI Parallel Interface -2 standard defines targets to be in the range 0 to 7, 0 to 15, and 0 to 31).

The SCSI device located within the current level shall be addressed by a BUS IDENTIFIER field and a TARGET/LUN field of all zeros, also known as LUN 0 (see 4.12.2).

### 4.12.6 Flat Space Addressing Method

All SCSI commands are allowed when the flat space addressing method is used, however, the addressed logical unit is not required to support all SCSI commands. Any command that is not supported shall be terminated with a CHECK CONDITION status. The sense key shall be set to ILLEGAL REQUEST and the additional sense code shall be set to INVALID COMMAND OPERATION CODE.

In the response to an INQUIRY command the addressed logical unit shall return a valid SCSI peripheral device type.(e.g., direct access device, streaming device).

See table 9 for the definition of the ADDRESS METHOD SPECIFIC field used when the flat space addressing method is selected.

**Table 9 — Flat space addressing**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| n-1 | 0 | 1 | (MSB) | | | | | |
| n | LUN | | | | | | | (LSB) |

The LUN field indicates the address of the logical unit the current level shall direct the received command to.

## 4.13 Model for extended logical unit addressing

### 4.13.1 Introduction to extended logical unit addressing

Extended logical unit addressing builds on the formats defined for dependent logical units (see 4.12) but may be used by SCSI devices having single level logical unit structure. In dependent logical unit addressing, the logical unit information at each level fits in exactly two bytes. Extended logical unit addresses have sizes of two bytes, four bytes, six bytes, or eight bytes.

### 4.13.2 Extended logical unit addressing formats

Extended logical units are identified by the ADDRESS METHOD field (see table 6 in 4.12) in the same manner as is the case for dependent logical units. An ADDRESS METHOD field value of 11b specifies the extended logical unit addressing method.

See table 10 for the definition of the ADDRESS METHOD SPECIFIC field used when the extended logical unit addressing method is selected.

**Table 10 — Extended logical unit addressing**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| n | 1 | 1 | LENGTH | | EXTENDED ADDRESS METHOD | | | |
| m | EXTENDED ADDRESS METHOD SPECIFIC | | | | | | | |

The LENGTH field (see table 11) specifies the length of the EXTENDED ADDRESS METHOD SPECIFIC field.

**Table 11 — LENGTH field values**

| Value | Length of the EXTENDED ADDRESS METHOD SPECIFIC Field | Reference |
|---|---|---|
| 00b | One byte | table 12 |
| 01b | Three bytes | table 13 |
| 10b | Five bytes | table 14 |
| 11b | Seven bytes | table 15 |

Table 12, table 13, table 14, and table 15 show the four extended logical unit addressing formats.

**Table 12 — Two byte extended logical unit addressing format**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| n | 1 | 1 | LENGTH (00b) | | | EXTENDED ADDRESS METHOD | | |
| n+1 | EXTENDED ADDRESS METHOD SPECIFIC | | | | | | | |

**Table 13 — Four byte extended logical unit addressing format**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| n | 1 | 1 | LENGTH (01b) | | | EXTENDED ADDRESS METHOD | | |
| n+1 | (MSB) | | | | | | | |
| | EXTENDED ADDRESS METHOD SPECIFIC | | | | | | | |
| n+3 | | | | | | | | (LSB) |

**Table 14 — Six byte extended logical unit addressing format**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| n | 1 | 1 | LENGTH (10b) | | | EXTENDED ADDRESS METHOD | | |
| n+1 | (MSB) | | | | | | | |
| | EXTENDED ADDRESS METHOD SPECIFIC | | | | | | | |
| n+5 | | | | | | | | (LSB) |

**Table 15 — Eight byte extended logical unit addressing format**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | LENGTH (01b) | | | EXTENDED ADDRESS METHOD | | |
| 1 | (MSB) | | | | | | | |
| | EXTENDED ADDRESS METHOD SPECIFIC | | | | | | | |
| 7 | | | | | | | | (LSB) |

The EXTENDED ADDRESS METHOD field combined with the LENGTH field (see table 16) specifies the type and size of extended logical unit address found in the EXTENDED ADDRESS METHOD SPECIFIC field.

**Table 16 — Logical unit extended address methods**

| **EXTENDED ADDRESS METHOD** Code | **LENGTH Code(s)** | **Description** | **Reference** |
|---|---|---|---|
| 0h | 00b - 11b | Reserved | |
| 1h | 00b | Well known logical unit | 4.13.3 |
| 1h | 01b - 11b | Reserved | |
| 2h - Fh | 00b - 11b | Reserved | |

### 4.13.3 Well known logical unit addressing

A SCSI target device may support zero or more W-LUNs. A single SCSI target device shall only support one instance of each supported well known logical unit. All W-LUNs within a SCSI target device shall be accessible from all target ports contained within the SCSI target device.

See table 17 for the definition of the EXTENDED ADDRESS METHOD SPECIFIC field used when the well known logical unit extended address method is selected.

**Table 17 — Well known logical unit extended address format**

| **Bit Byte** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| n | 1 | 1 | LENGTH (00b) | | | Well known logical unit (1h) | | |
| n+1 | W-LUN | | | | | | | |

The W-LUN field specifies well known logical unit to be addressed (see SPC-3).

## 4.14 The SCSI model for distributed communications

The SCSI model for communications between distributed objects is based on the technique of layering. In the layering technique, the initiator and target I/O systems are viewed as being logically composed of the ordered set of subsystems represented for convenience by the vertical sequence shown in figure 25.

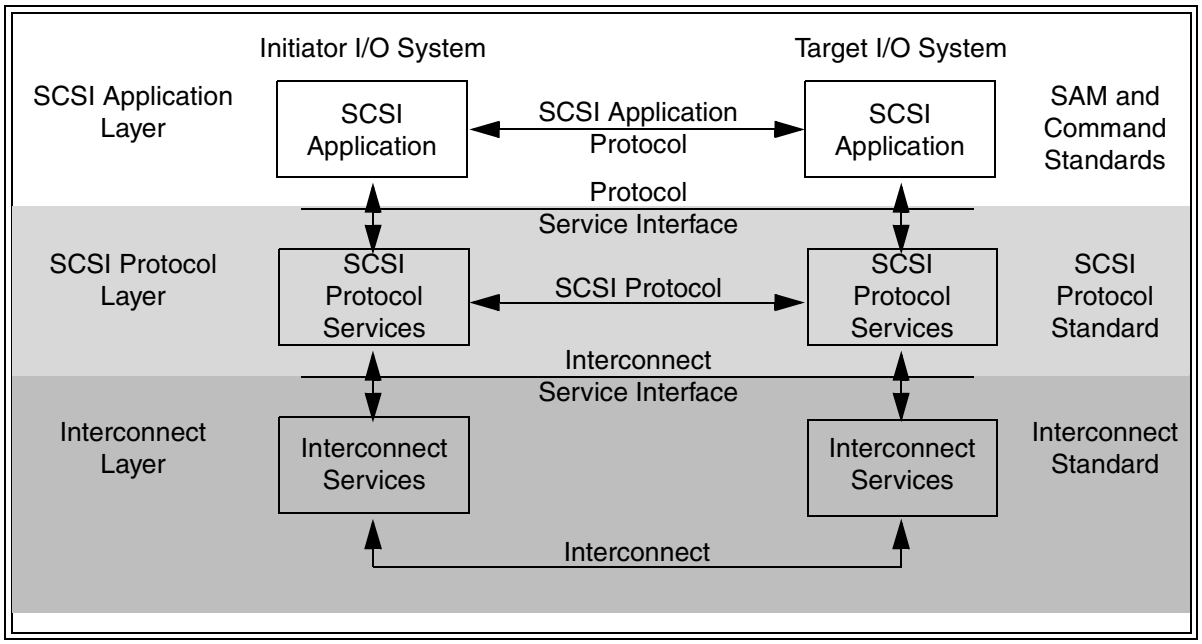



**Figure 25 — Protocol service reference model**

The layers comprising this model and the specifications defining the functionality of each layer are denoted by horizontal sequences. A layer consists of peer entities that communicate with one another by means of a protocol. Except for the physical interconnect layer, such communication is accomplished by invoking services provided by the adjacent lower layer. By convention, the layer from which a request for service originates is called the upper level protocol layer or ULP layer. The layer providing the service is referred to as the lower level protocol layer or LLP layer. The following layers are defined:

**SCSI application layer:** Contains the clients and servers that originate and process SCSI I/O operations by means of a SCSI application protocol.

**SCSI protocol layer:** Consists of the services and protocols through which clients and servers communicate; and

**Physical interconnect layer:** Comprised of the services, signaling mechanism and interconnect subsystem needed for the physical transfer of data from sender to receiver. In the SCSI model, the physical interconnect layer is known as the service deliver subsystem.

The set of protocol services implemented by the service delivery subsystem are intended to identify external behavioral requirements that apply to SCSI protocol standards. While these protocol services may serve as a guide for designing reusable software or firmware that is adaptable to different SCSI protocols, there is no requirement for an implementation to provide the service interfaces specified in this standard.

Interactions between the ULP and LLP layers are defined with respect to the ULP layer and may originate in either layer. An outgoing interaction is modeled as a procedure call invoking an LLP service. An incoming interaction is modeled as a signal sent by the LLP layer that may be accompanied by parameters or data. Both types of interaction are described using the notation for procedures specified in 3.6.2. In this model, input arguments are defined relative to the layer receiving an interaction (i.e., an input is a parameter supplied to the receiving layer by the layer initiating the interaction).

The following types of service interactions between layers are defined:

SCSI **Protocol service request:** A request from the ULP layer invoking a service provided by the LLP layer.

SCSI **Protocol service indication:** A signal from the LLP layer informing the ULP layer that an asynchronous event has occurred (e.g., a reset or the receipt of a peer-to-peer protocol transaction).

SCSI **Protocol service response:** A call to the LLP layer invoked by the ULP layer in response to a SCSI protocol service indication. A SCSI protocol service response may be invoked to return a reply to the ULP peer.

SCSI **Protocol service confirmation:** A signal from the LLP layer notifying the ULP layer that a SCSI protocol service request has completed. A confirmation may communicate parameters that indicate the completion status of the SCSI protocol service request or any other status. A SCSI protocol service confirmation may be used to convey a response from the ULP peer.

The services provided by an LLP layer are either confirmed or unconfirmed. A ULP service request invoking a confirmed service always results in a confirmation from the LLP layer.

Figure 26 shows the relationships between the four protocol service types.

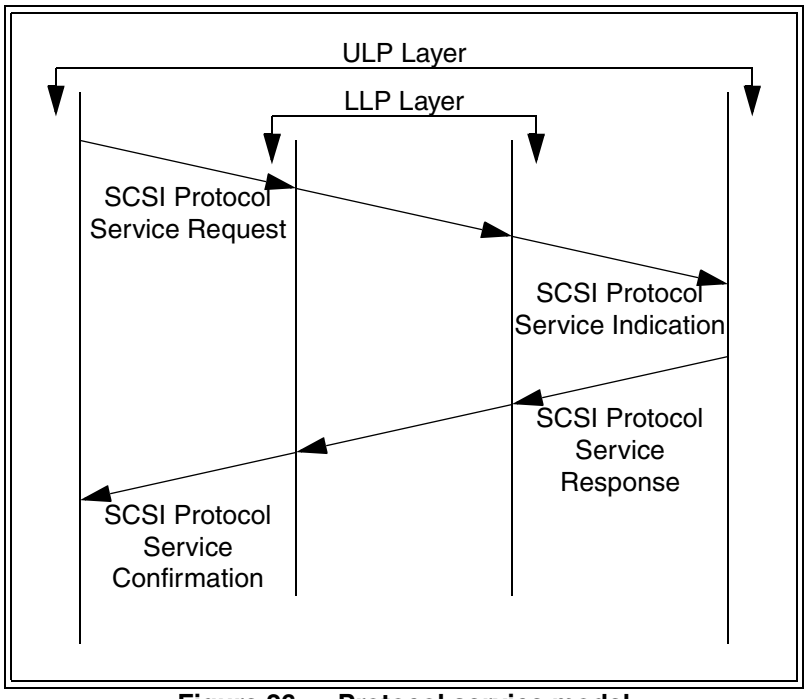



**Figure 26 — Protocol service model**

Figure 27 shows how protocol services may be used to process a client-server request-response transaction at the SCSI application layer.

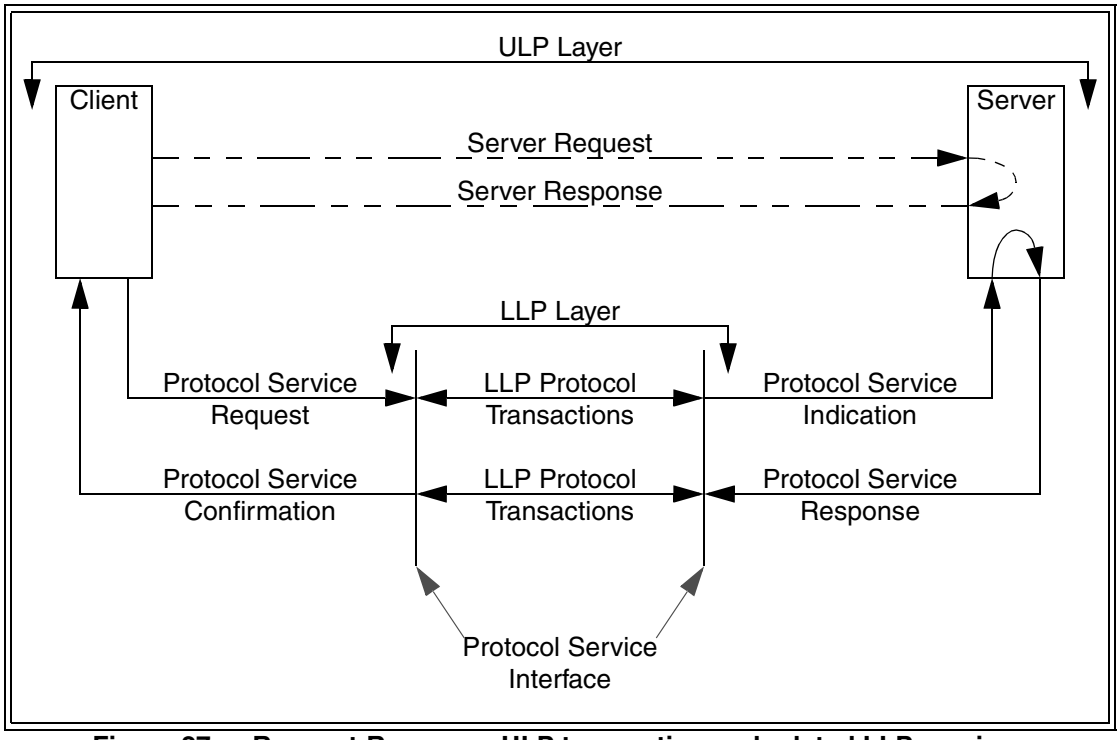



**Figure 27 — Request-Response ULP transaction and related LLP services**

The dashed lines in figure 27 show a SCSI application protocol transaction as it might appear to sending and receiving entities within the client and server. The solid lines in figure 27 show the corresponding protocol services and LLP transactions that are used to physically transport the data.

# 5 SCSI Command Model

## 5.1 The Execute Command remote procedure

An application client invokes the following remote procedure to process a SCSI command:

**Service response =Execute Command (IN (I_T_L_x Nexus, CDB, [Task Attribute], [Data-In Buffer Size], [Data-Out Buffer], [Data-Out Buffer Size], [Autosense Request], [Command Reference Number]), OUT ([Data-In Buffer], [Sense Data], Status))**

Input Arguments:

**I_T_L_x Nexus:** Either an I_T_L nexus or an I_T_L_Q nexus (see 4.10).

**CDB:** Command descriptor block (see 5.2).

**Task Attribute:** A value specifying one of the task attributes defined in 7.5. This argument shall not be specified for an untagged command or the second and subsequent commands in a sequence of linked commands. (Untagged tasks shall implicitly have the SIMPLE attribute.) The attribute of a task that processes linked commands shall be set according to the Task Attribute argument specified for the first command in the sequence.

**Data-In Buffer Size:** The number of bytes available for data transfers to the Data-In Buffer (see 5.4.3).

**Data-Out Buffer:** A buffer containing command specific information to be sent to the logical unit, such as data or parameter lists needed to service the command. The content of the Data-Out Buffer shall not change during the lifetime of the command (see 5.5) as viewed by the initiator.

**Data-Out Buffer Size:** The number of bytes available for data transfers from the Data-Out Buffer (see 5.4.3).

**Autosense Request:** An argument requesting the automatic return of sense data by means of the autosense mechanism specified in 5.8.4.3. It is not an error for the application client to provide this argument when autosense is not supported by the SCSI protocol or logical unit. Protocols may require that the Autosense Request argument always request automatic return of the sense data.

**Command Reference Number (CRN):** When this argument is used, all sequential commands of an I_T_L nexus shall include a CRN argument that is incremented by one. The initial, wrap, and reset CRN values shall be one. The CRN value zero shall be reserved for use as defined by the SCSI protocol. It is not an error for the application client to provide this argument when CRN is not supported by the SCSI protocol or logical unit.

Output Arguments:

**Data-In Buffer:** A buffer to contain command specific information returned by the logical unit on command completion. The application client shall not assume that the buffer contents are valid unless the command completes with a status of GOOD, INTERMEDIATE, or INTERMEDIATE-CONDITION MET. While some valid data may be present for other values of status, the application client should obtain additional information from the logical unit, such as sense data, to determine the state of the buffer contents. If the command ends with a service response of SERVICE DELIVERY OR TARGET FAILURE, the application client shall consider this parameter to be undefined.

**Sense Data:** A buffer to contain sense data returned by means of the autosense mechanism (see 5.8.4.3). If the command ends with a service response of SERVICE DELIVERY OR TARGET FAILURE, the application client shall consider this parameter to be undefined.

**Status:** A one-byte field containing command completion status (see 5.3). If the command ends with a service response of SERVICE DELIVERY OR TARGET FAILURE, the application client shall consider this parameter to be undefined.

**Service Response** assumes one of the following values:

**TASK COMPLETE:** A logical unit response indicating that the task has ended. The status parameter shall have one of the values specified in 5.3 other than INTERMEDIATE or INTERMEDIATE-CONDITION MET.

**LINKED COMMAND COMPLETE:** Logical unit responses indicating that a linked command has completed successfully. As specified in 5.3, the status parameter shall have a value of INTERMEDIATE or INTERMEDIATE-CONDITION MET.

**SERVICE DELIVERY OR TARGET FAILURE:** The command has been ended due to a service delivery failure (see 3.1.111) or SCSI target device malfunction. All output parameters are invalid.

The actual protocol events corresponding to a response of TASK COMPLETE, LINKED COMMAND COMPLETE or SERVICE DELIVERY OR TARGET FAILURE shall be specified in each SCSI protocol standard.

An application client requests processing of a linked command by setting the LINK bit to one in the CDB CONTROL byte as specified in 5.2.3. The task attribute is determined by the Task Attribute argument specified for the first command in the sequence. Upon receiving a response of LINKED COMMAND COMPLETE, an application client may issue the next command in the series through an **Execute Command** remote procedure call having the same I_T_L_x nexus and omitting the Task Attribute argument. If the application client issues the next command without waiting for one of the linked command complete responses, the overlapped command condition described in 5.8.2 may result.

## 5.2 Command Descriptor Block (CDB)

### 5.2.1 CDB Format

The CDB defines the operation to be performed by the device server. For some commands, the CDB is accompanied by a list of command parameters contained in the Data-Out Buffer defined in clause 5. The parameters required for each command are specified in the applicable SCSI command standards.

If a logical unit validates reserved CDB fields and receives a reserved field within the CDB that is not zero or receives a reserved CDB code value, the logical unit shall terminate the command with CHECK CONDITION status; the sense key shall be set to ILLEGAL REQUEST with an additional sense code of INVALID FIELD IN CDB (see SPC-2). Also, a logical unit may interpret a field or code value in accordance with a future revision to a SCSI standard.

For all commands, if the logical unit detects an invalid parameter in the CDB, then the logical unit shall complete the command without altering the medium.

All CDBs shall have an OPERATION CODE as the first byte. All CDBs (except the CDB for operation code 7Fh) shall have a CONTROL byte as the last byte. The format for the CDBs with operation code 7Fh is defined in SPC-2.

The general format for all CDBs except the CDB for operation code 7Fh is shown in table 18. The remaining parameters depend on the command to be processed. All SCSI protocol standards shall accept CDBs less than or equal to 16 bytes in length. CDBs using the format shown in table 18 shall not exceed sixteen bytes in length.

**Table 18 — Command Descriptor Block (CDB) Format**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | OPERATION CODE | | | | | | | |
| 1 | Command specific parameters | | | | | | | |
| n-1 | | | | | | | | |
| n | CONTROL | | | | | | | |

### 5.2.2 OPERATION CODE byte

The first byte of a SCSI CDB shall contain an operation code. The OPERATION CODE (see table 19) of the CDB has a GROUP CODE field and a COMMAND CODE field. The three-bit GROUP CODE field provides for eight groups of command codes. The five-bit COMMAND CODE field provides for thirty-two command codes in each group. A total of 256 possible operation codes exist. Operation codes are defined in the SCSI command standards. The group code value shall determine the length of the CDB (see table 20).

**Table 19 — OPERATION CODE byte**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | GROUP CODE | | | COMMAND CODE | | | | |

The value in the GROUP CODE field specifies one of the groups shown in table 20.

**Table 20 — Group Code values**

| Group Code | Meaning |
|---|---|
| 000b | 6 byte commands |
| 001b | 10 byte commands |
| 010b | 10 byte commands |
| 011b | reserved [a] |
| 100b | 16 byte commands |
| 101b | 12 byte commands |
| 110b | vendor specific |
| 111b | vendor specific |

[a] The format commands using the group code 011b and operation code 7Fh is described in SPC-2. With the exception of operation code 7Fh, all group code 011b operation codes are reserved.

### 5.2.3 CONTROL byte

The CONTROL byte is the last byte of every CDB. The CONTROL byte is defined in table 21.

**Table 21 — CONTROL byte**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | Vendor specific | | Reserved | | | NACA | Obsolete | LINK |

All SCSI protocol standards shall define as mandatory the functionality needed for a logical unit to implement the NACA bit and LINK bit.

The NACA (Normal ACA) bit is used to select whether a contingent allegiance (CA) or an auto contingent allegiance (ACA) is established if the command returns with CHECK CONDITION status. An NACA bit of one indicates that an ACA shall be established. An NACA bit of zero indicates that a CA shall be established. The actions for ACA and CA are specified in 5.8.1.2. All logical units shall implement support for the NACA value of zero (i.e., CA) and may support the NACA value of one (i.e., ACA). The ability to support a NACA value of one is indicated with the NORMACA bit in the standard INQUIRY data (see SPC-2).

If the NACA bit is set to one but the logical unit does not support ACA, the logical unit shall complete the command with a CHECK CONDITION status, sense key of ILLEGAL REQUEST, an additional sense code of INVALID FIELD IN CDB and establish a CA condition. The requirements for handling the resulting ACA condition shall be in accordance with the supported bit value.

The LINK bit is used to continue the task across multiple commands. Support for the LINK bit is optional. The initiator sets the LINK bit to one to specify a request for continuation of the task across two or more SCSI commands. If the LINK bit is one and the command completes successfully, a logical unit that supports the LINK bit shall continue the task and return a status of INTERMEDIATE or INTERMEDIATE-CONDITION MET and a service response of LINKED COMMAND COMPLETE (see 5.3). The logical unit shall complete the command with a status of CHECK CONDITION and a sense key of ILLEGAL REQUEST if the LINK bit is set to one and the logical unit does not support linked commands.

Bit 1 provides an obsolete way to request interrupts between linked commands.

## 5.3 Status

### 5.3.1 Status codes

The status codes are specified in table 22. Status shall be sent from the logical unit to the application client whenever a command ends with a service response of TASK COMPLETE or LINKED COMMAND COMPLETE. The receipt of any status, except INTERMEDIATE or INTERMEDIATE-CONDITION MET, shall indicate that the associated task has ended.

**Table 22 — Status codes**

| Status Code | Status |
|---|---|
| 00h | GOOD |
| 02h | CHECK CONDITION |
| 04h | CONDITION MET |
| 08h | BUSY |
| 10h | INTERMEDIATE |
| 14h | INTERMEDIATE-CONDITION MET |
| 18h | RESERVATION CONFLICT |
| 22h | Obsolete |
| 28h | TASK SET FULL |
| 30h | ACA ACTIVE |
| 40h | TASK ABORTED |
| All other codes | Reserved |

Definitions for each status code are as follows:

**GOOD.** This status indicates that the device server has successfully completed the task.

**CHECK CONDITION.** This status indicates that an ACA or CA condition has occurred (see 5.8.1). Autosense data may be delivered (see 5.8.4.3).

**CONDITION MET.** This status shall be returned whenever the requested operation specified by an unlinked command is satisfied (see the PRE-FETCH commands in the SBC standard).

**BUSY.** This status indicates that the logical unit is busy. This status shall be returned whenever a logical unit is unable to accept a command from an otherwise acceptable initiator (i.e., no reservation conflicts). The recommended initiator recovery action is to issue the command again at a later time. If the UA_INTLCK_CTRL field in the Control mode page contains 11b (see SPC-3), termination of a command with BUSY status shall cause a unit attention condition to be established for the initiator that sent the command with an additional sense code of PREVIOUS BUSY STATUS unless such a unit attention condition is already pending.

**INTERMEDIATE.** This status or INTERMEDIATE-CONDITION MET shall be returned for each successfully completed command in a series of linked commands (except the last command), unless the command is terminated with CHECK CONDITION, RESERVATION CONFLICT, TASK SET FULL, BUSY status. If INTERMEDIATE or INTERMEDIATE-CONDITION MET status is not returned, the series of linked commands is terminated and the task is ended.

**INTERMEDIATE-CONDITION MET.** This status is returned whenever the operation requested by a linked command is satisfied (see the PRE-FETCH commands in the SBC standard), unless the command is terminated with CHECK CONDITION, RESERVATION CONFLICT, TASK SET FULL, BUSY status. If INTERMEDIATE or INTERMEDIATE-CONDITION MET status is not returned, the series of linked commands is terminated and the task is ended.

**RESERVATION CONFLICT.** This status shall be returned whenever an initiator attempts to access a logical unit or an element of a logical unit that is reserved with a conflicting reservation type for another SCSI initiator. (See the RESERVE, RELEASE, PERSISTENT RESERVE OUT and PERSISTENT RESERVE IN commands in SPC-2). The recommended initiator recovery action is to issue the command again at a later time. Removing a persistent reservation belonging to a failing initiator may require the processing of a PERSISTENT RESERVE OUT command with the Preempt or Preempt and Clear service actions (see SPC-2).

If the UA_INTLCK_CTRL field in the Control mode page contains 11b (see SPC-3), termination of a command with RESERVATION CONFLICT status shall cause a unit attention condition to be established for the initiator that sent the command with an additional sense code of PREVIOUS RESERVATION CONFLICT STATUS unless such a unit attention condition is already pending.

**TASK SET FULL.** This status shall be implemented if the logical unit supports the creation of tagged tasks (see 4.9). This status shall not be implemented if the logical unit does not support the creation of tagged tasks.

When the logical unit has at least one task in the task set for an initiator and a lack of task set resources prevents accepting a received tagged task from that initiator in the task set, TASK SET FULL shall be returned. When the logical unit has no task in the task set for an initiator and a lack of task set resources prevents accepting a received tagged task from that initiator in the task set, BUSY should be returned.

When the logical unit has at least one task in the task set and a lack of task set resources prevents accepting a received untagged task in the task set, BUSY should be returned.

The logical unit should allow at least one queued command for each supported initiator that has identified itself to the target by a protocol specific procedure or by the successful transmission of a command.

If the UA_INTLCK_CTRL field in the Control mode page contains 11b (see SPC-3), termination of a command with TASK SET FULL status shall cause a unit attention condition to be established for the initiator that sent the command with an additional sense code of PREVIOUS TASK SET FULL STATUS unless such a unit attention condition is already pending.

**ACA ACTIVE.** This status shall be returned when an ACA exists within a task set and an initiator issues a command for that task set when at least one of the following is true:

a) There is a task with the ACA attribute (see 7.5.4) in the task set;
b) The initiator issuing the command did not cause the ACA condition; or
c) The task created to process the command did not have the ACA attribute and the NACA bit was set to one in the CDB CONTROL byte of the faulting command (see 5.8.1).

The initiator may reissue the command after the ACA condition has been cleared.

**TASK ABORTED.** This status shall be returned when a task is aborted by another initiator and the Control mode page TAS bit is one (see 5.6.3).

### 5.3.2 Status precedence

If more than one condition applies to a completed task, the report of a BUSY, RESERVATION CONFLICT, ACA ACTIVE or TASK SET FULL status shall take precedence over the return of any other status for that task.

## 5.4 SCSI Protocol Services in Support of Execute Command

### 5.4.1 Overview

The SCSI protocol services that support the **Execute Command** remote procedure call are described in 5.4. Two groups of protocol services are described. The protocol services that support the request and confirmation for the **Execute Command** remote procedure call are described in 5.4.2. The protocol services that support the data transfers associated with processing a SCSI command are described in 5.4.3.

### 5.4.2 Execute Command Request/Confirmation Protocol Services

All SCSI protocol standards shall define the protocol specific requirements for implementing the Send SCSI Command SCSI protocol service request and the Command Complete Received confirmation. Support for the SCSI Command Received indication and Send Command Complete response by a SCSI protocol standard is optional. All SCSI I/O systems shall implement these protocols as defined in the applicable protocol specification.

**SCSI Protocol Service Request:**

**Send SCSI Command (IN (I_T_L_x Nexus, CDB, [Task Attribute], [Data-In Buffer Size], [Data-Out Buffer], [Data-Out Buffer Size], [Autosense Request], [Command Reference Number] ))**

Input Arguments:

**I_T_L_x Nexus:** Either an I_T_L nexus or an I_T_L_Q nexus (see 4.10).

**CDB:** Command descriptor block (see 5.2).

**Task Attribute:** A value specifying one of the task attributes defined in 7.5. For specific requirements on the Task Attribute argument see 5.1.

**Data-In Buffer Size:** The number of bytes available for data transfers to the Data-In Buffer (see 5.4.3).

**Data-Out Buffer:** A buffer containing command specific information to be sent to the logical unit, such as data or parameter lists needed to service the command (see 5.1). The content of the Data-Out Buffer shall not change during the lifetime of the command (see 5.5) as viewed by the initiator.

**Data-Out Buffer Size:** The number of bytes available for data transfers from the Data-Out Buffer (see 5.4.3).

**Autosense Request:** An argument (see 5.1) requesting the automatic return of sense data by means of the autosense mechanism specified in 5.8.4.3.

**Command Reference Number (CRN):** When this argument is used, all sequential commands of an I_T_L nexus shall include a CRN argument that is incremented by one (see 5.1).

**SCSI Protocol Service Indication:**

**SCSI Command Received (IN (I_T_L_x Nexus, CDB, [Task Attribute], [Autosense Request], [Command Reference Number] ))**

Input Arguments:

**I_T_L_x Nexus:** Either an I_T_L nexus or an I_T_L_Q nexus (see 4.10).

**CDB:** Command descriptor block (see 5.2).

**Task Attribute:** A value specifying one of the task attributes defined in 7.5. For specific requirements on the Task Attribute argument see 5.1.

**Autosense Request:** This parameter is only present if the **Autosense Request** parameter was specified in the **Send SCSI Command** call and autosense delivery is supported by the SCSI protocol and logical unit.

**Command Reference Number (CRN):** When this argument is used, all sequential commands of an I_T_L nexus shall include a CRN argument that is incremented by one (see 5.1).

**SCSI Protocol Service Response** (from device server)**:**

**Send Command Complete (IN (I_T_L_x Nexus, [Sense Data], Status, Service Response ))**

Input Arguments:

**I_T_L_x Nexus:** Either an I_T_L nexus or an I_T_L_Q nexus (see 4.10).

**Sense Data:** If present, this argument instructs the target's service delivery port to return sense information to the initiator automatically (see 5.8.4.3).

**Status:** Command completion status (see 5.1).

**Service Response:** Possible service response information for the command (see 5.1).

**SCSI Protocol Service Confirmation:**

**Command Complete Received (IN (I_T_L_x Nexus, [Data-In Buffer], [Sense Data], Status, Service Response ))**

Input Arguments:

**I_T_L_x Nexus:** Either an I_T_L nexus or an I_T_L_Q nexus (see 4.10).

**Data-In Buffer:** A buffer containing command specific information returned by the logical unit on command completion (see 5.1).

**Sense Data:** Autosense data (see 5.8.4.3).

**Status:** Command completion status (see 5.1).

**Service Response:** Service response for the command (see 5.1).

### 5.4.3 Data Transfer Protocol Services

#### 5.4.3.1 Introduction

The data transfer services described in 5.4.3 provide mechanisms for moving data to and from the initiator in response to commands transmitted using the **Execute Command** remote procedure call. All SCSI protocol standards shall define the protocols required to implement these services.

The application client's Data-In Buffer and/or Data-Out Buffer each appears to the device server as a single, logically contiguous block of memory large enough to hold all the data required by the command (see figure 28). The model allows either unidirectional or bidirectional data transfer. The processing of a SCSI command may require the transfer of data from the application client using the Data-Out Buffer, or to the application client using the Data-In Buffer, or both to and from the application client using both the Data-In Buffer and the Data-Out Buffer.

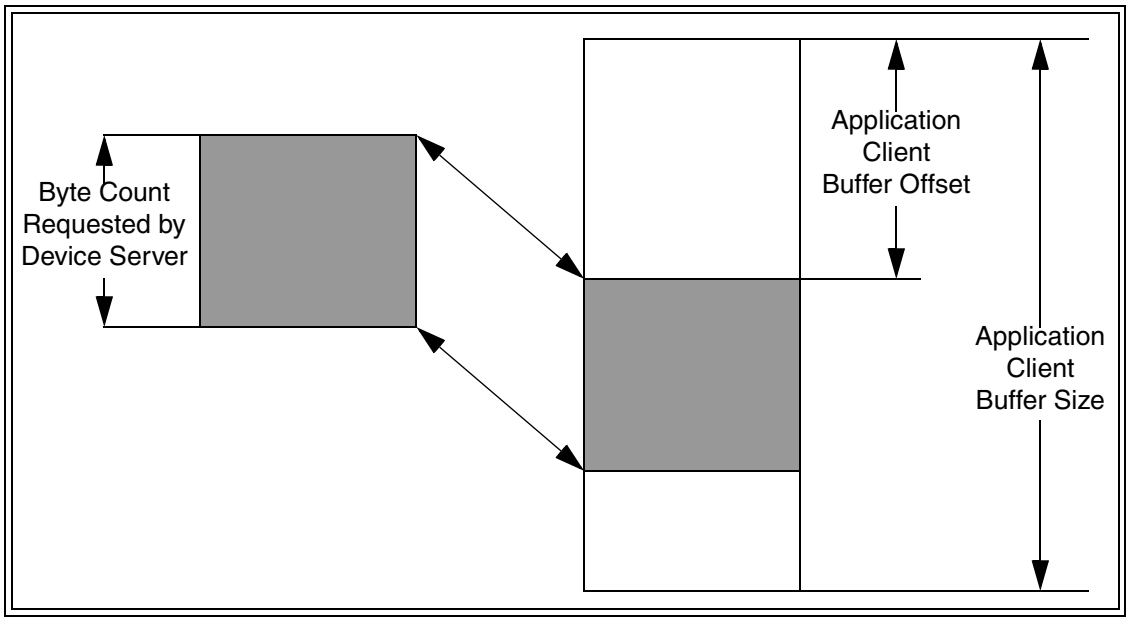



**Figure 28 — Model for Data-In and Data-Out data transfers**

It is assumed that the buffering resources available to the logical unit are limited and may be less than the amount of data that is capable of being transferred in one SCSI command. Such data needs to be moved between the application client and the media in segments that are smaller than the transfer size specified in the SCSI command. The amount of data moved per segment is usually a function of the buffering resources available to the logical unit. Figure 28 shows the model for such incremental data transfers.

The movement of data between the application client and device server is controlled by the following arguments:

**Application Client Buffer Size:** The total number of bytes in the application client's buffer (Data-In or Data-Out).

**Application Client Buffer Offset:** Offset in bytes from the beginning of the application client's buffer (Data-In or Data-Out) to the first byte of transferred data.

**Byte Count Requested by Device Server:** Number of bytes to be moved by the data transfer request.

For any specific data transfer SCSI protocol service request, the **Byte Count Requested by Device Server** is less than or equal to the combination of **Application Client Buffer Size** minus the **Application Client Buffer Offset**.

If a SCSI protocol supports random buffer access, the offset and byte count specified for each data segment to be transferred may overlap. In this case the total number of bytes moved for a command is not a reliable indicator of highest byte transferred and shall not be used by an initiator or target implementation to determine whether all data has been transferred.

All SCSI protocol standards shall define support for a resolution of one byte for the above arguments. A SCSI initiator device shall support a resolution of one byte. A SCSI target device may support any resolution.

Random buffer access occurs when the device server requests data transfers to or from segments of the application client's buffer that have an arbitrary offset and byte count. Buffer access is sequential when successive transfers access a series of monotonically increasing, adjoining buffer segments. Support for random buffer access by a SCSI protocol standard is optional. A device server implementation designed for any SCSI protocol implementation should be prepared to use sequential buffer access when necessary.

The LLP confirmed services specified in 5.4.3.2 and 5.4.3.3 are used by the device server to request the transfer of command data to or from the application client. The initiator SCSI protocol service interactions are unspecified.

#### 5.4.3.2 Data-In Delivery Service

**Request:**

**Send Data-In (IN (I_T_L_x Nexus, Device Server Buffer, Application Client Buffer Offset, Request Byte Count ))**

Argument descriptions:

**I_T_L_x Nexus:** either an I_T_L nexus or an I_T_L_Q nexus (see 4.10).

**Device Server Buffer:** Buffer from which data is to be transferred.

**Application Client Buffer Offset:** Offset in bytes from the beginning of the application client's buffer to the first byte of transferred data.

**Request Byte Count:** Number of bytes to be moved by this request.

**Confirmation:**

**Data-In Delivered (IN (I_T_L_x Nexus ))**

This confirmation notifies the device server that the specified data was successfully delivered to the application client buffer.

Argument descriptions:

**I_T_L_x Nexus:** either an I_T_L nexus or an I_T_L_Q nexus (see 4.10).

#### 5.4.3.3 Data-Out Delivery service

**Request:**

**Receive Data-Out (IN (I_T_L_x Nexus, Application Client Buffer Offset, Request Byte Count, Device Server Buffer ))**

Argument descriptions:

**I_T_L_x Nexus:** either an I_T_L nexus or an I_T_L_Q nexus (see 4.10).

**Device Server Buffer:** Buffer from which data is to be transferred.

**Application Client Buffer Offset:** Offset in bytes from the beginning of the application client's buffer to the first byte of transferred data.

**Request Byte Count:** Number of bytes to be moved by this request.

**Confirmation:**

**Data-Out Received (IN (I_T_L_x Nexus ))**

This confirmation notifies the device server that the requested data has been successfully delivered to its buffer.

Argument descriptions:

**I_T_L_x Nexus:** either an I_T_L nexus or an I_T_L_Q nexus (see 4.10).

## 5.5 Task and command lifetimes

This subclause specifies the events delimiting the beginning and end (i.e., lifetime) of a task or tendered SCSI command from the viewpoint of the device server and application client.

The device server shall create a task upon receiving a SCSI Command Received indication unless the command represents a continuation of a linked command as described in 5.1.

The task shall exist until:

a) The device server sends a SCSI protocol service response for the task of TASK COMPLETE; or
b) The task is aborted as described in 5.6.

The application client assumes that the task exists from the time the **Send SCSI Command** SCSI protocol service request is invoked until it receives one of the following target responses:

a) A service response of TASK COMPLETE for that task;
b) Notification of a unit attention condition with one of the following additional sense codes:
   A) COMMANDS CLEARED BY ANOTHER INITIATOR (if in reference to the task set containing the task);
   B) Any additional sense code whose ADDITIONAL SENSE CODE field contains 29h (e.g., POWER ON, RESET, OR BUS DEVICE RESET OCCURRED; POWER ON OCCURRED; SCSI BUS RESET OCCURRED; BUS DEVICE RESET FUNCTION OCCURRED; DEVICE INTERNAL RESET; TRANSCEIVER MODE CHANGED TO SINGLE-ENDED; or TRANSCEIVER MODE CHANGED TO LVD);
c) A service response of SERVICE DELIVERY OR TARGET FAILURE for the command. In this case, system implementations shall guarantee that the task associated with the failed command has ended;

d) A service response of FUNCTION COMPLETE following an ABORT TASK task management request directed to the specified task;
e) A service response of FUNCTION COMPLETE following an ABORT TASK SET or a CLEAR TASK SET task management function directed to the task set containing the specified task; or
f) A service response of FUNCTION COMPLETE in response to a LOGICAL UNIT RESET or TARGET RESET.

To the application client, the command is tendered from the time it calls the **Send SCSI Command** SCSI protocol service until one of the above responses or a service response of linked command complete is received.

When a SCSI protocol does not require state synchronization (see 4.6.1), there may be a time skew between the completion of a device server request-response transaction as seen by the application client and device server. As a result, the lifetime of a task or command as it appears to the application client normally is different from the lifetime observed by the device server.

## 5.6 Aborting tasks

### 5.6.1 Mechanisms that cause tasks to be aborted

A task is aborted when an event or initiator action causes termination of the task prior to its normal successful completion.

The following events cause a task or several tasks to be aborted:

a) The return of an **Execute Command** service response of SERVICE DELIVERY OR TARGET FAILURE as described in 5.1;
b) A power on condition; or
c) Protocol specific events.

The action of an initiator may abort task(s) created by the initiator itself or task(s) created by another initiator or both its own tasks and other initiator(s) task(s).

The following initiator actions affect only the task(s) created by the initiator that takes the action:

a) Completion of an ABORT TASK task management function directed to the specified task;
b) Completion of an ABORT TASK SET task management function under the conditions specified in 6.3;
c) An ACA or CA condition was established (see 5.8.1.2) and the QERR field was set to 01b or 11b in the Control mode page (see SPC-2); or
d) An ACA condition was cleared and the task had the ACA attribute (see 6.4).

The following initiator actions affect the task(s) created by the initiator that takes the action and/or task(s) created by other initiators:

a) Completion of a CLEAR TASK SET task management function referencing the task set containing the specified task;
b) An ACA or CA condition was established (see 5.8.1.2) and the QERR field was set to 01b in the Control mode page (see SPC-2);
c) Completion of a PERSISTENT RESERVE OUT command with a PREEMPT AND ABORT service action directed to the initiator that created the task (see SPC-2);
d) A logical unit reset (see 5.8.7); or
e) A hard reset (see 5.8.6).

### 5.6.2 When an initiator aborts its own tasks

When an initiator causes its own task(s) to be aborted, no notification that the task(s) have been aborted shall be returned to the initiator other than the completion response for the command or task management function action that caused the task(s) to be aborted and notification(s) associated with related effects of the action (e.g., a target reset unit attention condition).

### 5.6.3 When an initiator aborts another initiator's tasks

When an initiator causes the task(s) of another initiator to be aborted, the other initiator shall be notified that the task(s) have been aborted. The method of notifying the other initiator shall depend on the setting of the TAS bit in the Control mode page (see SPC-2) that applies to the other initiator.

If the TAS bit is zero, the method of notification shall be a unit attention condition. The additional sense code set for the unit attention condition depends on the action that caused the task(s) to be aborted.

If the TAS bit is one, the method of notification shall be the termination of each aborted task with a TASK ABORTED status. The COMMANDS CLEARED BY ANOTHER INITIATOR unit attention condition shall not be established, however, the establishment of any other applicable unit attention condition shall not be affected.

When a device server is aborting one or more tasks from an initiator with the TASK ABORTED status it should complete all of those tasks before entering additional tasks from that initiator into the task set.

## 5.7 Command processing examples

### 5.7.1 Unlinked command example

An unlinked command is used to show the events associated with the processing of a single device service request (see figure 29). This example does not include error or exception conditions.

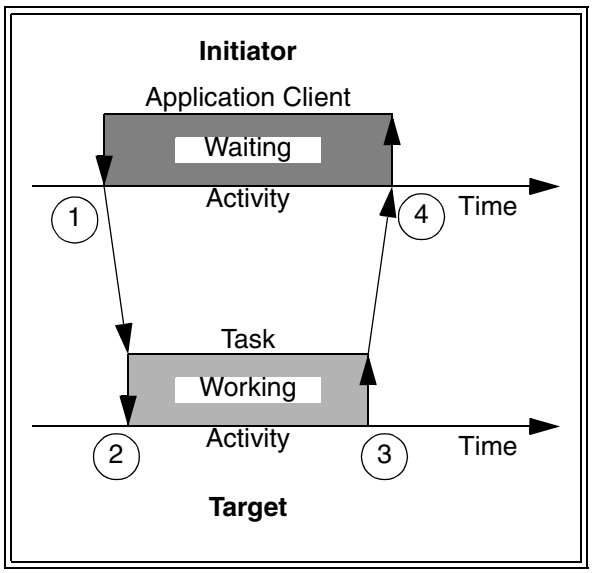



**Figure 29 — Command processing events**

The numbers in figure 29 identify the events described as follows:

1) The application client performs an **Execute Command** remote procedure call by invoking the **Send SCSI Command** SCSI protocol service to send the CDB and other input parameters to the logical unit.

2) The device server is notified through a SCSI **Command Received** indication containing the CDB and command parameters. A task is created and entered into the task set. The device server may invoke the appropriate data delivery service one or more times to complete command processing.

3) The task ends upon completion of the command. On command completion, the **Send Command Complete** SCSI protocol service is invoked to return a status of GOOD and a service response of TASK COMPLETE.

4) A confirmation of **Command Complete Received** is passed to the ULP by the initiator's service delivery subsystem.

### 5.7.2 Linked command example

A task may consist of multiple commands linked together. After the logical unit notifies the application client that a linked command has successfully completed, the application client issues the next command in the series.

The example in figure 30 shows the events in a sequence of two linked commands.

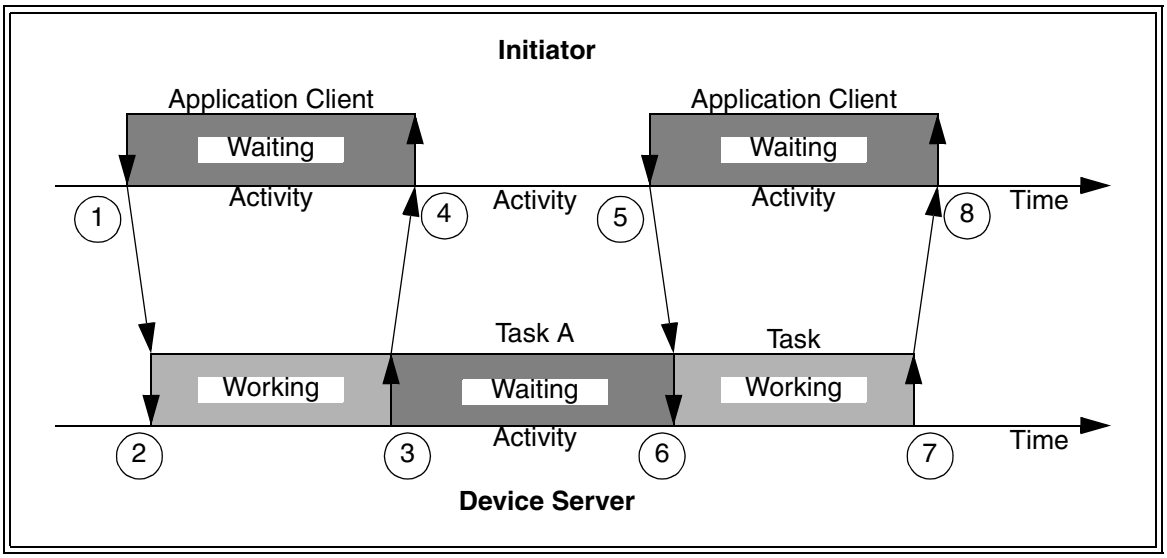



**Figure 30 — Linked command processing events**

The numbers in figure 30 Identify the events described as follows:

1) The application client performs an **Execute Command** remote procedure call by invoking the **Send SCSI Command** SCSI protocol service to send the CDB and other input parameters to the logical unit. The LINK bit is set to one in the CDB CONTROL byte (see 5.2.3).

2) The target's service delivery port issues **SCSI Command Received** to the device server. The device server creates a task (Task A) and enters it into the task set.

3) Upon completion of the first command, the device server invokes the **Send Command Complete** SCSI protocol service with the Status argument set to INTERMEDIATE or INTERMEDIATE-CONDITION MET and a Service Response of LINKED COMMAND COMPLETE. Task A is not terminated.

4) The LLP returns the status and service response to the ULP by means of a **Command Complete Received** confirmation.

5) The application client performs an **Execute Command** remote procedure call by means of the **Send SCSI Command** SCSI protocol service as described in 1). The Task Attribute argument is omitted. The LINK bit in the CDB CONTROL byte is zero.

6) The device server receives the last command in the sequence and processes the operation.

7) The command completes successfully. Task A is terminated. A **Send Command Complete** SCSI protocol service response of TASK COMPLETE, with status GOOD, is sent to the application client.

8) The LLP delivers an **Command Complete Received** confirmation to the application client that contains the service response and status.

## 5.8 Command processing considerations and exception conditions

### 5.8.1 Contingent Allegiance (CA) and Auto Contingent Allegiance (ACA)

#### 5.8.1.1 Overview

There are two mechanisms for returning sense data when a command is terminated with a CHECK CONDITION status: autosense (see 5.8.4.3) and the REQUEST SENSE command (see SPC-2). There are two mechanisms for altering task processing when a command is terminated with a CHECK CONDITION status: CA and ACA. CA alters task processing so that sense data is preserved for subsequent delivery. ACA alters task processing until a CLEAR ACA task management function (see 6.4) is requested. Table 23 provides an overview of how autosense, CA, and ACA interact.

**Table 23 — Autosense, CA, and ACA Interactions**

| Autosense Requested [a] | NACA Value [b] | Tasks Blocked [c] | |
|---|---|---|---|
| | | From | To [d] |
| No | 0 (i.e., CA) | Termination of a command with CHECK CONDITION status | Receipt of a command [e] |
| | 1 (i.e., ACA) | | Receipt of CLEAR ACA [g] |
| Yes | 0 (i.e., CA) | | Transmission of autosense data [f] |
| | 1 (i.e., ACA) | | Receipt of CLEAR ACA [g] |

[a] Autosense is requested via the **Execute Command** remote procedure call (see 5.1).

[b] The NACA bit is in the CONTROL byte in the CDB (see 5.2.3).

[c] The blocking of tasks is described in 5.8.1.2. If the QERR field in the Control mode page (see SPC-2) contains 01b or 11b, tasks are aborted instead of being blocked. If the TST field in the Control mode page contains 000b, tasks from all initiators are blocked or aborted. If the TST field in the Control mode page contains 001b, only tasks from the faulted initiator are blocked or aborted.

[d] This table covers only the normal methods for clearing a CA or ACA as seen by the faulted initiator. Exception handling methods for clearing CA and ACA are described in 5.8.1.6 and 5.8.1.7.

[e] The intent is that the next command from the faulted initiator be a REQUEST SENSE command but the next command received clears the CA condition, regardless of what command that is.

[f] Since the autosense data is transmitted coincident with the delivery of the CHECK CONDITION status (see 5.8.4.3), the interval during which tasks are blocked is not detectable by the initiator. If the QERR field in the Control mode page (see SPC-2) contains 01b or 11b, the specified blocked tasks are aborted, an action that makes the CA condition detectable by the initiator.

[g] The CLEAR ACA task management function is described in 6.4. During ACA new tasks received by the logical unit are not allowed to enter the task set unless they have the ACA task attribute (see 7.5.4). One of the results of the ACA task attribute requirement is that commands in-flight when the CHECK CONDITION status occurs are returned unprocessed to the initiator with an ACA ACTIVE status. Multiple commands may be sent one at a time using the ACA task attribute to recover from the CHECK CONDITION that caused the ACA condition without clearing the ACA.

#### 5.8.1.2 Establishing a CA or ACA

When a device server terminates a command with a CHECK CONDITION status, either an ACA or CA condition is established within the task set. If the NACA bit was zero in the CONTROL byte (see 5.2.3) of the faulting command, the device server shall create a CA condition. If the NACA bit was one in the CONTROL byte of the faulting command, the device server shall create an ACA condition.

When a CA or ACA condition is established, tasks in the dormant and enabled task states (see 7.4) shall either be aborted or blocked based on the contents of the TST and QERR field in the Control mode page (see SPC-2) as shown in table 24.

**Table 24 — Blocking and aborting tasks when a CA or ACA is established**

| QERR | TST | Action |
|---|---|---|
| 00b | 000b | All enabled tasks from all initiators shall transition to the blocked task state (see 7.6). All dormant tasks from all initiators shall remain in the dormant task state. |
| | 001b | All enabled tasks from the faulted initiator shall transition to the blocked task state (see 7.6). All dormant tasks from the faulted initiator shall remain in the dormant task state. All tasks from initiators other than the faulted initiator shall not be affected by the establishment of this CA or ACA condition. |
| 01b | n/a | All enabled and dormant tasks from all initiators shall be aborted (see 5.6). |
| 11b | 000b | All enabled and dormant tasks from the faulted initiator shall be aborted (see 5.6). All enabled tasks from initiators other than the faulted initiator shall transition to the blocked task state (see 7.6). All dormant tasks from initiators other than the faulted initiator shall remain in the dormant task state. |
| | 001b | All enabled and dormant tasks from the faulted initiator shall be aborted (see 5.6). All tasks from initiators other than the faulted initiator shall not be affected by the establishment of this CA or ACA condition. |

After the CA or ACA conditions is established:

a) Tasks from the faulted initiator shall be handled as described in 5.8.1.4, and
b) Tasks from initiators other than the faulted initiator shall be handled as described in 5.8.1.5.

A CA or ACA condition shall not cross task set boundaries and shall be preserved until it is cleared as described in 5.8.1.6 or 5.8.1.7. If requested by the application client and supported by the SCSI protocol and logical unit, sense data shall be returned via autosense as described in 5.8.4.3.

If the SCSI protocol does not enforce state synchronization as described in 4.6.1, there may be a time delay between the occurrence of the CA or ACA condition and the time at which the initiator becomes aware of the condition.

#### 5.8.1.3 Handling tasks when neither CA or ACA is in effect

Table 25 describes the handling of tasks when neither a CA nor an ACA condition is in effect for the task set. The number of initiators in the task set is influenced by the TST field in the Control mode page (see SPC-2).

**Table 25 — Task handling when neither CA nor ACA is in effect**

| New Task Properties | | Device Server Action | Condition Established if New Task Terminates with a CHECK CONDITION status |
|---|---|---|---|
| Attribute [a] | NACA Value [b] | | |
| Any Attribute Except ACA | 0 | Process the task. [c] | CA |
| | 1 | | ACA |
| ACA | 0 | Terminate the command with CHECK CONDITION status, sense key of ILLEGAL REQUEST and additional sense code of INVALID MESSAGE ERROR. | CA |
| | 1 | | ACA |

[a] Task attributes are described in 7.5.

[b] The NACA bit is in the CONTROL byte in the CDB (see 5.2.3).

[c] All the conditions that affect the processing of commands (e.g., reservations) still apply.

#### 5.8.1.4 Handling new tasks from the faulted initiator when CA or ACA is in effect

Table 26 describes the handling of new tasks from the faulted initiator when CA is in effect.

**Table 26 — Handling for new tasks from a faulted initiator during CA**

| New Task Properties | | Device Server Action | Condition Established If New Task Terminates with a CHECK CONDITION status [c] |
|---|---|---|---|
| Attribute [a] | NACA Value [b] | | |
| Any Attribute Except ACA | 0 | Process the task. [d] | CA |
| | 1 | | ACA |
| ACA | 0 | Terminate the command with CHECK CONDITION status, sense key of ILLEGAL REQUEST, and additional sense code of INVALID MESSAGE ERROR. | CA |
| | 1 | | ACA |

[a] Task attributes are described in 7.5.

[b] The NACA bit is in the CONTROL byte in the CDB (see 5.2.3).

[c] The CA condition is cleared upon completion of any new task regardless of status. Termination of that new task with CHECK CONDITION status shall result in the establishment of a new CA or ACA based on the value of the NACA bit.

[d] All the conditions that affect the processing of commands (e.g., reservations) still apply.

Table 27 describes the handling of new tasks from the faulted initiator when ACA is in effect.

**Table 27 — Handling for new tasks from a faulted initiator during ACA**

| New Task Properties | | ACA Task Present in the Task Set | Device Server Action | Condition Established If New Task Terminates with a CHECK CONDITION status |
|---|---|---|---|---|
| Attribute [a] | NACA Value [b] | | | |
| ACA | 0 | No | Process the task. [d] | CA [c] |
| | 1 | No | | ACA [c] |
| | 0 or 1 | Yes | Terminate the task with ACA ACTIVE status. | n/a |
| Any Attribute Except ACA | 0 or 1 | n/a | Terminate the task with ACA ACTIVE status. | n/a |

[a] Task attributes are described in 7.5.

[b] The NACA bit is in the CONTROL byte in the CDB (see 5.2.3).

[c] If a task with the ACA attribute terminates with a CHECK CONDITION status, the existing ACA condition shall be cleared and a new CA or ACA condition shall be established based on the value of the NACA bit.

[d] All the conditions that affect the processing of commands (e.g., reservations) still apply.

#### 5.8.1.5 Handling new tasks from initiators other than the faulted initiator when CA or ACA is in effect

The handling of tasks created by initiators other than the faulted initiator depends on the value in the TST field in the Control mode page (see SPC-2).

Table 28 describes the handling of new tasks from initiators other than the faulted initiator when CA is in effect.

**Table 28 — Handling for new tasks from non-faulted initiators during CA**

| TST Field Value in Control mode page | New Task Properties | | New Command Permitted During CA [c] | Device Server Action | Condition Established If New Task Terminates with a CHECK CONDITION status |
|---|---|---|---|---|---|
| | Attribute [a] | NACA Value [b] | | | |
| 000b | ACA | n/a | n/a | Terminate the task with BUSY status. | n/a |
| | Any Attribute Except ACA | 0 or 1 | No | Terminate the task with BUSY status. | n/a |
| | | 0 | Yes | Process the task. | CA [d] |
| | | 1 | Yes | | ACA [d] |
| 001b | ACA | 0 | n/a | Terminate the command with CHECK CONDITION status, sense key of ILLEGAL REQUEST and additional sense code of INVALID MESSAGE ERROR. | CA |
| | | 1 | | | ACA |
| | Any Attribute Except ACA | 0 or 1 | n/a | Process the task. [e] | See 5.8.1.3. |

[a] Task attributes are described in 7.5.

[b] The NACA bit is in the CONTROL byte in the CDB (see 5.2.3).

[c] The device server shall permit (i.e., not terminate) the processing of specified commands from initiators other than the faulted initiator while a CA condition is established. The device server shall process a PERSISTENT RESERVE OUT command with a PREEMPT AND ABORT service action (see SPC-2) from an initiator other than the faulted initiator during a CA condition.

[d] If a permitted command terminates with a CHECK CONDITION status, the existing CA condition shall be cleared and a new CA or ACA condition shall be established for a new faulted initiator based on the value of the NACA bit.

[e] When the TST field in the Control mode page contains 001b, commands from initiators other than the faulted initiator shall be processed as if the CA condition does not exist (see 5.8.1.3). In this case, the logical unit shall be capable of handling concurrent CA conditions and sense data for all initiators.

Table 29 describes the handling of new tasks from initiators other than the faulted initiator when ACA is in effect.

**Table 29 — Handling for new tasks from non-faulted initiators during ACA**

| TST Field Value in Control mode page | New Task Properties | | New Command Permitted During ACA [c] | Device Server Action | Condition Established If New Task Terminates with a CHECK CONDITION status |
|---|---|---|---|---|---|
| | Attribute [a] | NACA Value [b] | | | |
| 000b | ACA | n/a | n/a | Terminate the task with ACA ACTIVE status. | n/a |
| | Any Attribute Except ACA | 0 | No | Terminate the task with BUSY status. | n/a |
| | | 1 | No | Terminate the task with ACA ACTIVE status. | n/a |
| | | 0 | Yes | Process the task. | CA [d] |
| | | 1 | Yes | | ACA [d] |
| 001b | ACA | 0 | n/a | Terminate the command with CHECK CONDITION status, sense key of ILLEGAL REQUEST and additional sense code of INVALID MESSAGE ERROR. | CA |
| | | 1 | | | ACA |
| | Any Attribute Except ACA | 0 or 1 | n/a | Process the task. [e] | See 5.8.1.3. |

[a] Task attributes are described in 7.5.

[b] The NACA bit is in the CONTROL byte in the CDB (see 5.2.3).

[c] The device server shall permit (i.e., not terminate) the processing of specified commands from initiators other than the faulted initiator while an ACA condition is established. The device server shall process a PERSISTENT RESERVE OUT command with a PREEMPT AND ABORT service action (see SPC-2) from an initiator other than the faulted initiator during an ACA condition.

[d] If a permitted command terminates with a CHECK CONDITION status, the existing ACA condition shall be cleared and a new CA or ACA condition shall be established for a new faulted initiator based on the value of the NACA bit.

[e] When the TST field in the Control mode page contains 001b, commands from initiators other than the faulted initiator shall be processed as if the ACA condition does not exist (see 5.8.1.3). In this case, the logical unit shall be capable of handling concurrent ACA conditions and sense data for all initiators.

### 5.8.1.6 Clearing a CA condition

A CA condition shall only be cleared:

a) As a result of a power on or logical unit reset (see 5.8.7);
b) By an ABORT TASK SET task management function (see 6.3) from the faulted initiator;
c) By a CLEAR TASK SET task management function (see 6.5) from any initiator including the faulted initiator if the TST field in the Control mode page (see SPC-2) contains 000b;

d) By a CLEAR TASK SET task management function from the faulted initiator if the TST field in the Control mode page (see SPC-2) contains 001b;
e) By a PERSISTENT RESERVE OUT command with a PREEMPT AND ABORT service action from another initiator that clears the tasks of the faulted initiator (see SPC-2);
f) When a PERSISTENT RESERVE OUT command with a PREEMPT AND ABORT service action from another initiator terminates in a CHECK CONDITION status;
g) Upon completion of a subsequent REQUEST SENSE command for the I_T_L nexus;
h) Upon accepting any subsequent command other than a REQUEST SENSE command for the I_T_L nexus; or
i) Upon sending sense data by means of the autosense mechanism (see 5.8.4.3).

NOTE 6 - Case f) results in the establishment of a new CA or ACA for a new faulted initiator based on the value of the NACA bit.

When a CA condition is cleared and no new CA or ACA condition is established, the state of all tasks in the task set shall be modified as described in clause 7.

#### 5.8.1.7 Clearing an ACA condition

An ACA condition shall only be cleared:

a) As the result of a power on or a logical unit reset (see 5.8.7);
b) By a CLEAR ACA task management function (see 6.4) from the faulted initiator;
c) By a PERSISTENT RESERVE OUT command with a PREEMPT AND ABORT service action with the ACA task attribute from the faulted initiator that clears the tasks of the faulted initiator (see SPC-2);
d) By a PERSISTENT RESERVE OUT command with a PREEMPT AND ABORT service action with a task attribute other than ACA from an initiator other than the faulted initiator that clears the tasks of the faulted initiator;
e) When a command with the ACA task attribute from the faulted initiator terminates with a CHECK CONDITION status; or
f) When a PERSISTENT RESERVE OUT command with a PREEMPT AND ABORT service action terminates in a CHECK CONDITION status.

NOTE 7 - Cases e) and f) result in the establishment of a new CA or ACA based on the value of the NACA bit.

When an ACA condition is cleared and no new CA or ACA condition is established, the state of all tasks in the task set shall be modified as described in clause 7.

### 5.8.2 Overlapped commands

An overlapped command occurs when a task manager detects the use of a duplicate I_T_L_x nexus (see 4.9.1) in a command before a pending task holding that I_T_L_x nexus completes its task lifetime (see 5.5). Each SCSI protocol standard shall specify whether or not a task manager is required to detect overlapped commands.

A task manager that detects an overlapped command shall abort all tasks for the faulted initiator in the task set and the device server shall return CHECK CONDITION status for that command. The sense key shall be set to ABORTED COMMAND and the additional sense code shall be set to OVERLAPPED COMMANDS ATTEMPTED.

NOTES
8 An overlapped command may be indicative of a serious error and, if not detected, could result in corrupted data. This is considered a catastrophic failure on the part of the initiator. Therefore, vendor specific error recovery procedures may be required to guarantee the data integrity on the medium. The target logical unit

may return additional sense data to aid in this error recovery procedure (e.g., sequential-access devices may return the residue of blocks remaining to be written or read at the time the second command was received).

9 Some logical units may not detect an overlapped command until after the CDB has been received.

### 5.8.3 Incorrect Logical Unit selection

The target's response to an incorrect logical unit number is described in this subclause.

The logical unit number may be incorrect because:

a) The target does not support the logical unit (e.g., some targets support only one peripheral device).

In response to any other command except REQUEST SENSE and INQUIRY, the target shall terminate the command with CHECK CONDITION status. Sense key and additional sense code shall be set to the values specified for the REQUEST SENSE command in item b);

b) The target supports the logical unit, but the peripheral device is not currently attached to the target.

In response to an INQUIRY command the target shall return the INQUIRY data with the peripheral qualifier set to the value required in SPC-2. In response to a REQUEST SENSE command, the target shall return sense data. The sense key shall be set to ILLEGAL REQUEST and the additional sense code shall be set to LOGICAL UNIT NOT SUPPORTED.

In response to any other command except REQUEST SENSE and INQUIRY, the target shall terminate the command with CHECK CONDITION status. Sense key and additional sense code shall be set to the values specified for the REQUEST SENSE command in item b);

c) The target supports the logical unit and the peripheral device is attached, but not operational.

In response to an INQUIRY command the target shall return the INQUIRY data with the peripheral qualifier set to the value required in SPC-2. In response to REQUEST SENSE, the target shall return sense data.

The target's response to any command other than INQUIRY and REQUEST SENSE is vendor specific; or

d) The target supports the logical unit but is incapable of determining if the peripheral device is attached or is not operational when it is not ready.

In response to an INQUIRY command the target shall return the INQUIRY data with the peripheral qualifier set to the value specified in SPC-2. In response to a REQUEST SENSE command the target shall return the REQUEST SENSE data with a sense key of NO SENSE unless an ACA exists.

The target's response to any other command is vendor specific.

### 5.8.4 Sense data

#### 5.8.4.1 Sense data introduction

Sense data shall be made available by the logical unit in the event a command completes with a CHECK CONDITION status or other conditions. The format, content and conditions under which sense data shall be prepared by the logical unit are specified in this standard, SPC-2, the applicable device command standard and applicable SCSI protocol standard.

Sense data shall be preserved by the logical unit for the initiator until it is transferred by one of the methods listed below or until another task from that initiator is entered into the task set.

The sense data may be transferred to the initiator through any of the following methods:

a) The REQUEST SENSE command (see SPC-2);
b) An asynchronous event report (see 5.8.4.2); or
c) Autosense delivery (see 5.8.4.3).

The following clauses describe the last two transfer methods.

### 5.8.4.2 Asynchronous Event Reporting

Asynchronous Event Reporting is used by a logical unit to signal another device that an asynchronous event has occurred. The mechanism automatically returns sense data associated with the event. Each SCSI protocol standard shall describe a mechanism for Asynchronous Event Reporting. (In this subclause, references to Asynchronous Event Reporting assume that the device to be notified has enabled asynchronous event reports from the target.) Support for asynchronous event reporting is a logical unit option.

NOTE 10 - A SCSI device that is capable of producing asynchronous event reports at initialization time should provide means to defeat these reports. This may be done with a switch or jumper wire. Devices that implement saved parameters may alternatively save the asynchronous event reporting permissions either on a per SCSI device basis or as a system wide option.

Parameters managing the use of asynchronous event reporting are contained in the Control mode page (see SPC-2).

Asynchronous Event Reporting is used to signal a device that one of the four events listed below has occurred:

a) An exception condition was encountered after command completion;
b) A newly initialized device is available;
c) Some other type of unit attention condition has occurred; or
d) An asynchronous event has occurred.

An example of a) occurs in a device that implements a write cache. If the target is unable to write cached data to the medium, it may use an asynchronous event report to inform the initiator of the failure.

An example of b) is a logical unit that generates an asynchronous event report, following a power-on cycle, to notify other SCSI devices that it is ready to accept I/O commands.

An example of c) occurs in a device that supports removable media. Asynchronous event reporting may be used to inform an initiator of a not-ready-to-ready transition (medium changed) or of an operator initiated event (e.g., activating a write protect switch or activating a start or stop switch).

An example of d) is a sequential-access device performing a REWIND command with the IMMEDIATE bit set to one (see SSC). An asynchronous event report may be used to inform an initiator that the beginning of medium has been reached. Completion of a CD-ROM AUDIO PLAY command (see MMC-2) started in the immediate mode is another example of this case.

Sense data accompanying the report identifies the condition (see 5.8.4.1).

An exception condition encountered after command completion shall be reported to a specific initiator once per occurrence of the event causing it. The logical unit may choose to use an asynchronous event report or to return CHECK CONDITION status on a subsequent command, but not both. Notification of an exception condition encountered after command completion shall be reported only to the initiator or initiators that sent the affected task or tasks.

Asynchronous event reports may be used to notify devices that a system resource has become available. If a logical unit uses this method of reporting, the sense key in the AER sense data shall be set to UNIT ATTENTION.

#### 5.8.4.3 Autosense

Autosense is the automatic return of sense data to the application client coincident with the completion of a SCSI command under the conditions described in this subclause. All protocols shall support autosense.

If supported by the protocol and logical unit and requested by the **Execute Command** remote procedure call (see 5.1), the device server shall only return sense data in this manner coincident with the completion of a command with a status of CHECK CONDITION. After autosense data is sent the following shall be cleared:

a) The CA condition (see 5.8.1.6), if any; and
b) The sense data, except sense data associated with a unit attention condition when the UA_INTLCK_CTRL field in the Control mode page (see SPC-3) contains 10b or 11b.

Autosense shall not affect ACA (see 5.8.1) or the sense data associated with a unit attention condition when the UA_INTLCK_CTRL field contains 10b or 11b.

SCSI protocol standards that support autosense shall require an autosense implementation to:

a) Notify the logical unit when autosense data has been requested for a command; and
b) Inform the application client when autosense data has been returned upon command completion (see 5.1).

It is not an error for the application client to request the automatic return of sense data when autosense is not supported by the SCSI protocol or logical unit implementation. If the application client requested the return of sense data through the autosense facility and the SCSI protocol layer does not support this feature, then the confirmation returned by the SCSI initiator port should indicate that no sense data was returned. If the SCSI protocol layer supports autosense but the logical unit does not, then the target should indicate that no sense data was returned. In either case, sense information shall be preserved and the application client may issue a command to retrieve it.

### 5.8.5 Unit Attention condition

Each logical unit shall generate a unit attention condition whenever the logical unit has been reset as described in 5.8.7 or by a power-on reset. In addition, a logical unit shall generate a unit attention condition for each initiator whenever one of the following events occurs:

a) A removable medium may have been changed;
b) The mode parameters in effect for this initiator have been changed by another initiator;
c) The version or level of microcode has been changed;
d) Tasks for this initiator were cleared by another initiator;
e) INQUIRY data has been changed;
f) The logical unit inventory has been changed;
g) The mode parameters in effect for the initiator have been restored from non-volatile memory;
h) A change in the condition of a synchronized spindle; or
i) Any other event requiring the attention of the initiator.

Logical units may queue unit attention conditions. After the first unit attention condition is cleared, another unit attention condition may exist (e.g., a power on condition followed by a microcode change condition).

A unit attention condition shall persist on the logical unit for each initiator until that initiator clears the condition as described in the following paragraphs.

If an INQUIRY command enters the enabled task state, the logical unit shall perform the INQUIRY command and shall neither report nor clear any unit attention condition.

If a REPORT LUNS command enters the enabled task state, the logical unit shall perform the REPORT LUNS command and shall not report any unit attention condition. The logical unit shall clear any unit attention condition established in response to a change in the logical unit inventory for all logical units for the initiator that sent the REPORT LUNS command. The logical unit shall not clear any other unit attention condition.

If a REQUEST SENSE command enters the enabled task state while a unit attention condition exists for the initiator that sent the REQUEST SENSE command, then the logical unit shall either:

a) Report any pending sense data and preserve all unit attention conditions on the logical unit; or,
b) Report a unit attention condition for the initiator that sent the REQUEST SENSE command. The logical unit may discard any pending sense data and shall clear the reported unit attention condition for that initiator.

If the logical unit has already generated the ACA or CA condition for a unit attention condition, the logical unit shall report the unit attention condition (i.e., option b) above).

If a command other than INQUIRY, REPORT LUNS, or REQUEST SENSE enters the enabled task state while a unit attention condition exists for the initiator that sent the command, the logical unit shall terminate the command with a CHECK CONDITION status. The logical unit shall provide sense data that reports a unit attention condition for the initiator that sent the command.

If a logical unit reports a unit attention condition with autosense (see 5.8.4.3) or with an asynchronous event report (see 5.8.4.2) and the UA_INTLCK_CTRL field in the Control mode page contains 00b (see SPC-3), then the logical unit shall clear the reported unit attention condition for that initiator on the logical unit. If the UA_INTLCK_CTRL field in the Control mode page contains 10b or 11b, the logical unit shall not clear unit attention conditions reported with autosense or an asynchronous event report.

### 5.8.6 Hard reset

A hard reset is a target port action in response to a reset event within the service delivery subsystem. A wakeup event (see 3.1.141) is a reset event. The definition of additional reset events is protocol specific. Each SCSI protocol standard that defines reset events shall specify the target port's action in response to reset events.

The target port's response to a hard reset shall include initiating the equivalent of a logical unit reset for all logical units as described in 5.8.7.

While the task manager response to task management requests is subject to the presence of access restrictions, as managed by ACCESS CONTROL OUT commands (see SPC-3), a hard reset in response to a reset event within the service delivery subsystem shall be unaffected by access controls.

### 5.8.7 Logical unit reset

A logical unit reset is:

a) An action in response to a LOGICAL UNIT RESET task management request (see 6.6) or some other logical unit reset event; or
b) Part of an action in response to a TARGET RESET task management function (see 6.7) or a hard reset (see 5.8.6).

The definition of logical unit reset events is dependent on the SCSI protocol.

To process a logical unit reset the logical unit shall:

a) Abort all tasks as described in 5.6;
b) Clear a CA (see 5.8.1.6) or ACA (see 5.8.1.7) condition, if one is present;
c) Release all reservations established using the reserve/release management method (persistent reservations shall not be affected);
d) Return the logical unit's operating mode to the appropriate initial conditions, similar to those conditions that would be found following device power-on. The MODE SELECT parameters (see SPC-2) shall be restored to their last saved values if saved values have been established. MODE SELECT parameters for which no saved values have been established shall be returned to their default values;
e) Set a unit attention condition (see 5.8.5); and
f) Initiate a logical unit reset for all dependent logical units (see 4.12).

In addition to the above, the logical unit shall perform any additional functions required by the applicable standards.

# 6 Task Management Functions

## 6.1 Introduction

Task management functions control the processing of one or more tasks. An application client requests a task management function by means of a procedure call having the following format:

**Service Response = Function name (IN (nexus) )**

Service Response:

One of the following SCSI protocol specific responses shall be returned:

**FUNCTION COMPLETE:** A task manager response indicating that the requested function is complete. The task manager shall unconditionally return this response upon completion of a task management request supported by the logical unit or SCSI target device to which the request was directed. Upon receiving a request to process an unsupported function, the task manager may return this response or the function rejected response described below.

**FUNCTION REJECTED:** An optional task manager response indicating that the operation is not supported by the object to which the function was directed (e.g., the logical unit).

**SERVICE DELIVERY OR TARGET FAILURE:** The request was terminated due to a service delivery failure (see 3.1.111) or target malfunction. The task manager may or may not have successfully performed the specified function.

Each SCSI protocol standard shall define the actual events comprising each of the above service responses.

The task management functions are summarized in table 30.

**Table 30 — Task Management Functions**

| Task Management Function | Nexus | Reference |
|---|---|---|
| ABORT TASK | I_T_L_Q | 6.2 |
| ABORT TASK SET | I_T_L | 6.3 |
| CLEAR ACA | I_T_L | 6.4 |
| CLEAR TASK SET | I_T_L | 6.5 |
| LOGICAL UNIT RESET | I_T_L | 6.6 |
| TARGET RESET | I_T | 6.7 |
| WAKEUP | I_T | 6.8 |

Argument descriptions:

**Nexus:** An I_T Nexus, I_T_L Nexus, or I_T_L_Q Nexus (see 4.10).

**I_T Nexus:** An initiator and target nexus (see 4.10).

**I_T_L Nexus:** An initiator, target, and logical unit nexus (see 4.10).

**I_T_L_Q Nexus:** An initiator, target, logical unit, and tag nexus (see 4.10).

NOTE 11 - The ABORT TASK, ABORT TASK SET, CLEAR TASK SET, LOGICAL UNIT RESET, TARGET RESET, and WAKEUP functions provide a means to abort one or more tasks prior to normal completion.

The task manager response to task management requests is subject to the presence of access restrictions, as managed by ACCESS CONTROL OUT and ACCESS CONTROL IN commands (see SPC-3), as follows:

a) A task management request of ABORT TASK, ABORT TASK SET or CLEAR ACA shall not be affected by the presence of access restrictions;
b) A task management request of CLEAR TASK SET or LOGICAL UNIT RESET received from an initiator that is denied access to the logical unit (either because it has no access rights or because it is in the pending-enrolled state) shall cause no change to the logical unit;
c) A TARGET RESET task management request shall initiate a logical unit reset as described in 5.8.7 for all logical units to which the initiator has access, and shall cause no change to any logical units to which the initiator is denied access; and
d) The task management function Service Response shall not be affected by the presence of access restrictions.

## 6.2 ABORT TASK

Function call:

**Service Response = ABORT TASK (IN (I_T_L_Q Nexus**) )

Description:

This function shall be supported by a logical unit if it supports tagged tasks and may be supported by a logical unit if it does not support tagged tasks.

The task manager shall abort the specified task if it exists. Previously established conditions, including MODE SELECT parameters, reservations, ACA, and CA shall not be changed by the ABORT TASK function.

If the logical unit supports this function, a response of FUNCTION COMPLETE shall indicate that the task was aborted or was not in the task set. In either case, the target shall guarantee that no further responses from the task are sent to the initiator.

All SCSI protocol standards shall provide the functionality needed for a task manager to implement the ABORT TASK task management function.

## 6.3 ABORT TASK SET

Function Call:

**Service Response = ABORT TASK SET (IN (I_T_L Nexus**) )

Description:

This function shall be supported by all logical units.

The task manager shall abort all tasks in the task set that were created by the initiator as described in 5.6.

The task manager shall perform an action equivalent to receiving a series of ABORT TASK requests. All tasks from that initiator in the task set serviced by the logical unit shall be aborted. Tasks from other initiators or in other task sets shall not be aborted. A CA shall be cleared by the ABORT TASK SET function from the faulted initiator (see

5.8.1.6). Other previously established conditions, including MODE SELECT parameters, reservations, and ACA shall not be changed by the ABORT TASK SET function.

All SCSI protocol standards shall provide the functionality needed for a task manager to implement the ABORT TASK SET task management function.

## 6.4 CLEAR ACA

Function Call

**Service response = CLEAR ACA (IN (I_T_L Nexus**) )

Description:

This function shall be supported by a logical unit if it supports ACA (see 5.2.3).

The initiator issues CLEAR ACA to clear an ACA condition from the task set serviced by the logical unit as specified in 5.8.1.7. For tasks with the ACA attribute (see 7.5.4) receipt of an CLEAR ACA function shall have the same effect as receipt of an ABORT TASK function (see 6.2). If successful, this function shall be terminated with a service response of FUNCTION COMPLETE.

If the task manager clears the ACA condition, any task within that task set may be completed subject to the requirements for task set management specified in clause 7.

While a CA is in effect (see 5.8.1), a logical unit that supports the CLEAR ACA task management function shall ignore all CLEAR ACA requests and shall return a service response of FUNCTION COMPLETE.

All SCSI protocol standards shall provide the functionality needed for a task manager to implement the CLEAR ACA task management function.

## 6.5 CLEAR TASK SET

Function Call:

**Service response = CLEAR TASK SET (IN (I_T_L Nexus**) )

Description:

This function shall be supported by all logical units, except in the following cases, when support for this function is optional:

a) The logical unit does not support tagged tasks (see 4.9); or
b) The logical unit supports the basic task management model (see 7.2).

All tasks in the appropriate task set as defined by the TST field in the Control mode page (see SPC-2) shall be aborted as described in 5.6. The medium may have been altered by partially processed commands.

The CA condition (see 5.8.1.6), and all pending status and sense data for the task set defined by the TST field in the Control mode page shall be cleared. Other previously established conditions, including MODE SELECT parameters, reservations, and ACA shall not be changed by the CLEAR TASK SET function.

All SCSI protocol standards shall provide the functionality needed for a task manager to implement the CLEAR TASK SET task management function.

## 6.6 LOGICAL UNIT RESET

Function Call:

**Service Response = LOGICAL UNIT RESET (IN (I_T_L Nexus**) )

Description:

This function shall be supported by all logical units.

Before returning a FUNCTION COMPLETE response, the logical unit shall perform the logical unit reset functions specified in 5.8.7. A unit attention condition for all initiators that have access shall be created on the logical unit and dependent logical unit(s), if any, as specified in 5.8.5.

NOTE 12 - Previous versions of this standard only required LOGICAL UNIT RESET support in logical units that supported hierarchical logical units.

All SCSI protocol standards shall provide the functionality needed for a task manager to implement the LOGICAL UNIT RESET task management function.

## 6.7 TARGET RESET

Function Call:

**Service Response = TARGET RESET (IN (I_T Nexus**) )

Description:

Before returning a FUNCTION COMPLETE response, the target port shall perform logical unit reset functions specified in 5.8.7 for every logical unit. A unit attention condition for all initiators that have access shall be created on each of these logical units as specified in 5.8.5.

An initiator should issue LOGICAL UNIT RESETs only to the logical units it is using rather than issuing a TARGET RESET. This avoids resetting logical units that other initiators may be using.

NOTE 13 - Previous versions of this standard required TARGET RESET support in all targets. SCSI protocols may or may not require that TARGET RESET be supported. SCSI protocols may require additional actions beyond those specified here.

## 6.8 WAKEUP

Function Call:

**Service response = WAKEUP (IN (I_T Nexus**) )

Description:

SCSI protocols may or may not define the WAKEUP function. This function may be supported by SCSI protocols whose interconnects support a shared wakeup signal or individual wakeup signals for each SCSI target port. This function may be supported by SCSI devices on SCSI protocols which support the function.

This function causes a wakeup event (see SPC-3) to be sent to either:

a) The specified SCSI target port, on SCSI protocols supporting individual wakeup signals; or
b) All SCSI target ports connected to the interconnect, on SCSI protocols supporting a shared wakeup signal.

The wakeup function is a reset event and shall cause a hard reset in the recipient target port(s).

## 6.9 Task management protocol services

The protocol services described in this subclause are used by an initiator and target to process a task management remote procedure call. The following arguments are passed:

**Nexus:** An I_T Nexus, I_T_L Nexus, or I_T_L_Q Nexus (see 4.10).

**Function Identifier:** Parameter encoding the task management function to be performed.

All SCSI protocol standards shall define the protocol specific requirements for implementing the Send Task Management Request SCSI protocol service and the Received Function-Executed confirmation described below. Support for the Task Management Request Received indication and Task Management Function Executed SCSI protocol service response by the SCSI protocol standard is optional. All SCSI devices shall implement these protocol services as defined in the applicable SCSI protocol standard.

Request sent by an initiator and application client to a target's task manager:

**Send Task Management Request (IN (Nexus, Function Identifier ) )**

Argument descriptions:

**Nexus:** An I_T Nexus, I_T_L Nexus, or I_T_L_Q Nexus (see 4.10).

**Function Identifier:** Parameter encoding the task management function to be performed.

Indication received by the task manager:

**Task Management Request Received (IN (Nexus, Function Identifier ) )**

Argument descriptions:

**Nexus:** An I_T Nexus, I_T_L Nexus, or I_T_L_Q Nexus (see 4.10).

**Function Identifier:** Parameter encoding the task management function to be performed.

Response from task manager to initiator and application client:

**Task Management Function Executed (IN (Nexus, Service Response ) )**

Argument descriptions:

**Nexus:** An I_T Nexus, I_T_L Nexus, or I_T_L_Q Nexus (see 4.10).

**Service Response:** An encoded value representing one of the following:

FUNCTION COMPLETE: The requested function has been completed.

FUNCTION REJECTED: The task manager does not implement the requested function.

Confirmation received by application client:

**Received Function-Executed (IN (Nexus, Service Response ) )**

Argument descriptions:

**Nexus:** An I_T Nexus, I_T_L Nexus, or I_T_L_Q Nexus (see 4.10).

**Service Response:** An encoded value representing one of the following:

FUNCTION COMPLETE: The requested function has been completed.

FUNCTION REJECTED: The task manager does not implement the requested function.

Since the nexus may not uniquely identify the transaction, there may be no way for an initiator to associate a confirmation with a request. A SCSI protocol that does not provide such an association should not allow an initiator to have more than one pending task management request per I_T_L nexus.

## 6.10 Task management function example

Figure 31 shows the sequence of events associated with a task management function.

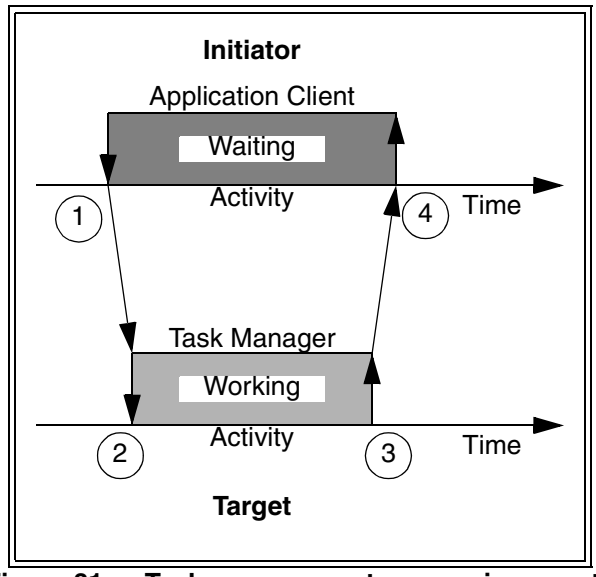



**Figure 31 — Task management processing events**

The numbers in figure 31 identify the events described below.

1. The application client issues a task management request by invoking the **Send Task Management Request** SCSI protocol service.

2. The task manager is notified through a **Task Management Request Received** and begins processing the function.

3. The task manager performs the requested operation and responds by invoking the **Task Management Function Executed** SCSI protocol service to notify the application client. The **Service Response** parameter is set to a value of FUNCTION COMPLETE.

4. A **Received Function-Executed** confirmation is received by the application client.

# 7 Task Set Management

## 7.1 Introduction to task set management

Clause 7 describes some of the controls application clients have over task set management behaviors (see 7.2). Clause 7 also specifies task set management requirements in terms of:

a) Task states (see 7.4),
b) Task attributes (see 7.5),
c) The events that cause transitions between task states (see 7.3 and 7.4), and
d) A map of task state transitions (see 7.6).

Clause 7 concludes with several task set management examples (see 7.7).

Task behavior, as specified in clause 7, refers to the functioning of a task as observed by an application client, including the results of command processing and interactions with other tasks.

The requirements for task set management only apply to a task after it has been entered into a task set. A task shall be entered into a task set unless a condition exists that causes that task to be completed with a status of BUSY, RESERVATION CONFLICT, TASK SET FULL, or ACA ACTIVE. A CHECK CONDITION status caused by the detection of an overlapped command or certain protocol specific errors also should not keep a task from being entered in the task set.

## 7.2 Controlling task set management

The Control mode page (see SPC-2) contains fields that specify particular task set management behaviors. The standard INQUIRY data CmdQue bit (see SPC-2) indicates support for tagged tasks (command queuing). One specific combination of task set management behaviors is identified as the basic task management model. Support for the basic task management model is indicated by values returned in the CMDQUE and BQUE bits in the standard INQUIRY data (see SPC-2). The basic task management model requires the following task set management behaviors:

a) The only task attribute supported shall be SIMPLE;
b) The device server may reorder the actual processing sequence of tasks in any manner. Any data integrity exposures related to task sequence order shall be explicitly handled by the application client using the appropriate commands;
c) All the tasks shall be aborted when an ACA or CA condition is established;
d) It shall not be possible to disable tagged queuing; and
e) Support for the CLEAR TASK SET task management function is optional.

## 7.3 Task management events

The following describe the events that cause changes in task state.

| | |
|---|---|
| All older tasks ended: | If the TST field in the Control mode page (see SPC-2) equals 000b, all tasks have ended that were accepted from all initiators earlier in time than the referenced task. If the TST field in the Control mode page equals 001b, all tasks have ended that were accepted from the referenced initiator earlier in time than the referenced task. |
| All older Head of Queue and older Ordered tasks ended: | If the TST field in the Control mode page equals 000b, all Head of Queue and Ordered tasks have ended that were accepted from all initiators earlier in time than the referenced task. If the TST field in the Control mode page equals 001b, all Head of Queue and Ordered tasks have ended that were accepted from the referenced initiator earlier in time than the referenced task. |
| CA or ACA establishment: | A CA or ACA condition has been established (see 5.8.1). |
| task abort: | A task has been aborted as described in 5.6. |
| task completion: | The device server has sent a service response of TASK COMPLETE for the task (see 5.1 and 5.5). |
| task ended: | A task has completed or aborted. |
| CA cleared: | An CA condition has been cleared (see 5.8.1.6). |
| ACA cleared: | An ACA condition has been cleared (see 5.8.1.7). |

## 7.4 Task states

### 7.4.1 Overview

The model employs four tasks states, described in 7.4.2, 7.4.3, 7.4.4, and 7.4.5.

To simplify the discussion in clause 7:

a) "Enabled task" may be used to refer to a task in the enabled task state,
b) "Blocked task" may be used to refer to a task in the blocked task state,
c) "Dormant task" may be used to refer to a task in the dormant task state, and
d) "Ended task" may be used to refer to a task in the ended task state.

### 7.4.2 Enabled task state

A task in the enabled task state may become a current task and may complete at any time, subject to the task completion constraints specified in the Control mode page (see SPC-2). A task that has been accepted into the task set shall not complete or become a current task unless it is in the enabled task state.

Except for the use of resources required to preserve task state, a task shall produce no effects detectable by the application client before the task's first transition to the enabled task state. Although, before entering this state for the first time, the task may perform other activities visible to lower layers – such as pre-fetching data to be written to the media – this activity shall not result in a detectable change in state as perceived by an application client. In addition, the behavior of a completed task, as defined by the commands it has processed, shall not be affected by the task's states before it enters the enabled task state.

### 7.4.3 Blocked task state

A task in the blocked task state is prevented from completing due to an ACA or CA condition. A task in this state shall not become a current task. While a task is in the blocked task state, any information the logical unit has or accepts for the task shall be suspended. If the TST field in the Control mode page (see SPC-2) equals 000b the blocked task state is independent of the initiator. If the TST field equals 001b the blocked task state applies only to the faulted initiator.

### 7.4.4 Dormant task state

A task in the dormant task state is prevented from completing due to the presence of certain other tasks in the task set. A task in this state shall not become a current task. While a task is in the dormant task state, any information the logical unit has or accepts for the task shall be suspended.

### 7.4.5 Ended task state

A task in the ended task state is removed from the task set.

### 7.4.6 Task states and task lifetimes

Figure 32 shows the events corresponding to two task processing sequences. Except for the dormant task state between times A and B in case 1, logical unit conditions and the commands processed by the task are identical. Assuming in each case the task completes with a status of GOOD at time C, the state observed by the application client for case 1 shall be indistinguishable from the state observed for case 2.

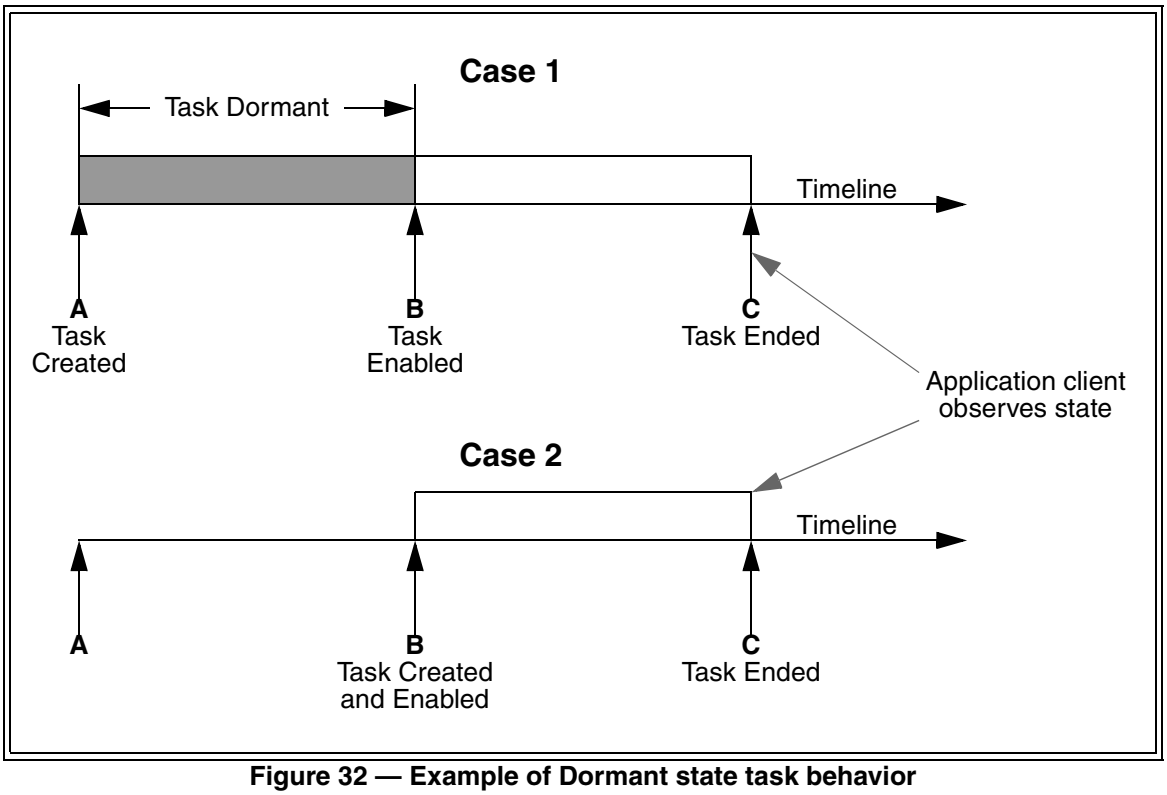



**Figure 32 — Example of Dormant state task behavior**

## 7.5 Task Attributes

### 7.5.1 SIMPLE Task

A task having the Simple attribute shall be accepted into the task set in the dormant task state. The task shall not enter the enabled task state until all older Head of Queue and older Ordered tasks in the task set have ended (see 7.3).

### 7.5.2 ORDERED Task

A task having the Ordered attribute shall be accepted into the task set in the dormant task state. The task shall not enter the enabled task state until all older tasks in the task set have ended (see 7.3).

### 7.5.3 HEAD OF QUEUE Task

A task having the Head of Queue attribute shall be accepted into the task set in the enabled task state.

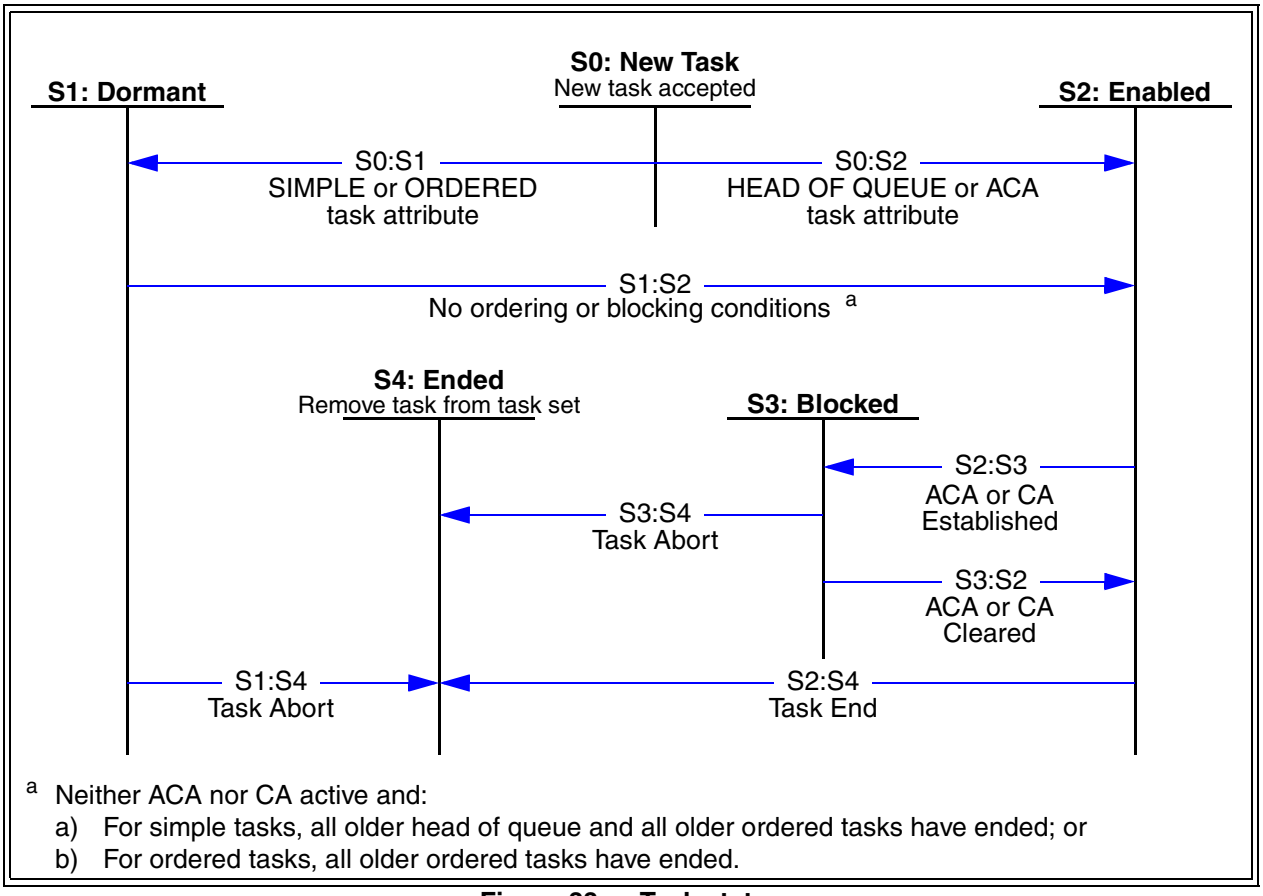
### 7.5.4 ACA Task

A task having the ACA attribute shall be accepted into the task set in the enabled task state. There shall be no more than one ACA task per task set (see 5.8.1.2).

## 7.6 Task state transitions

This subclause describes task state transitions, actions and associated triggering events as they appear to an application client. The logical unit response to events affecting multiple tasks (e.g., a CLEAR TASK SET) may be different from the response to an event affecting a single task. To the application client, the collective behavior appears as a series of state changes occurring to individual tasks.

The task state diagram of figure 33 shows the behavior of a single task in response to an external event.




a Neither ACA nor CA active and:
a) For simple tasks, all older head of queue and all older ordered tasks have ended; or
b) For ordered tasks, all older ordered tasks have ended.

**Figure 33 — Task states**

**Transition S0:S1:** If a newly accepted task has the SIMPLE or ORDERED task attribute, it shall transition to the dormant task state.

**Transition S0:S2:** If a newly accepted task has the HEAD OF QUEUE or ACA task attribute, it shall transition to the enabled task state.

**Transition S1:S2:** The task attribute of a dormant task shall affect the transition to the enabled task state as follows:

a) A dormant task having the SIMPLE task attribute shall enter the enabled task state when all older Head of Queue and older Ordered tasks (see 7.3) have ended; or
b) A dormant task having the ORDERED task attribute shall enter the enabled task state when all older tasks (see 7.3) have ended.

If the TST field in the Control mode page (see SPC-2) contains 000b, then the transition from dormant task to enabled task shall not occur while a CA or ACA is in effect for any initiator (see 5.8.1.4 and 5.8.1.5). If the TST field in the Control mode page contains 001b, then dormant tasks from the faulted initiator shall not transition to the enabled task state while a CA or ACA is in effect for that initiator (see 5.8.1.4).

**Transition S2:S3:** The establishment of a CA or ACA condition (see 7.3) shall cause zero or more enabled tasks to enter the blocked task state as described in 5.8.1.2.

**Transition S3:S2:** When a CA or ACA condition is cleared (see 7.3), tasks that entered the blocked state the CA or ACA condition was established (see 5.8.1.2) shall re-enter the enabled task state.

**Transition S2:S4:** A task that has completed (see 7.3) or aborted (see 7.3 and 5.6) shall enter the ended task state. This is the only state transition that applies to an ACA task.

**Transitions S1:S4, S3:S4:** A task abort event (see 7.3 and 5.6) shall cause the task to unconditionally enter the ended task state.

## 7.7 Task set management examples

### 7.7.1 Introduction

Several task set management scenarios are shown in 7.7.2, 7.7.3, and 7.7.4. The examples are valid for single or multi-initiator cases, when the TST field contains 000b (i.e., the interaction among tasks in a task set is independent of the initiator originating a task). The examples are also valid for a single initiator, when the TST field contains 001b (i.e., task set management proceeds independently for each initiator and the events and transitions in one initiator's task set do not affect the task set management for another initiator's task set. Throughout these examples, the scope of the task set box drawn in each snapshot depends on the setting of the TST field in the Control mode page (see SPC-2).

The figure accompanying each example shows successive snapshots of a task set after various events, such as task creation or completion. In all cases, the constraints on task completion order established using the QUEUE ALGORITHM MODIFIER field and DQUE bit in the Control mode page (see SPC-2) are not in effect.

A task set is shown as an ordered list or queue of tasks with the head of the queue towards the top of the figure. A new Head of Queue task always enters the task set at the head, displacing older Head of Queue tasks. Simple, Ordered and ACA tasks always enter the task set at the end of the queue.

Tasks, denoted by rectangles, are numbered in ascending order from oldest to most recent. Fill, shape and line weight are used to distinguish task states and attributes are shown in table 31.

**Table 31 — Task attribute and state indications in examples**

| Task Attribute | Box Shape | Line Weight | Task State | Box Fill |
|---|---|---|---|---|
| SIMPLE | Rounded Corners | Thin | Enabled | White |
| ORDERED | Square Corners | Thin | Dormant | Grey |
| HEAD OF QUEUE | Square Corners | Thick | Blocked | Black |
| ACA | Square Corners | Thin Dashed | | |

The conditions preventing a dormant task from entering enabled task state (except for ACA and CA conditions) are shown by means of “blocking boundaries”. Such boundaries appear as horizontal lines with an arrow on both ends. The tasks causing the barrier condition are described as part of each example. A task is impeded by the barrier if it is between the boundary and the end of the queue. When no ACA or CA is in effect, a task enters the enabled task state after all intervening barriers have been removed.

### 7.7.2 Head of Queue tasks

Figure 34 shows task set conditions when several Head of Queue tasks are processed.

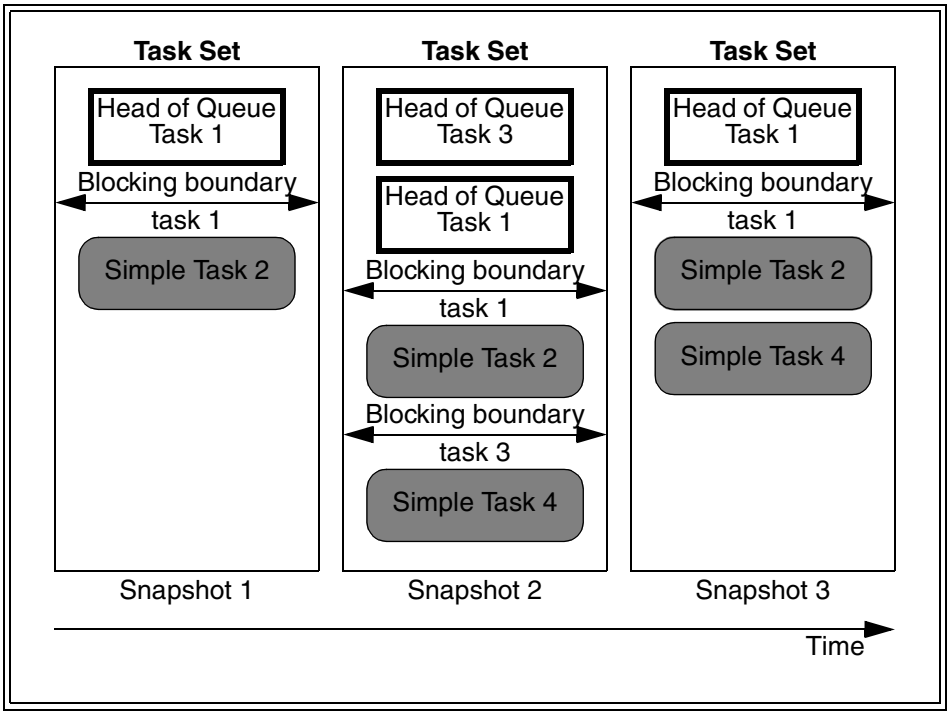



**Figure 34 — Head of Queue tasks and blocking boundaries (example 1)**

In snapshot 1 the task set initially contains one Head of Queue and one Simple task. As shown by the blocking boundary, simple task 2 is in the dormant task state because of the older Head of Queue task. Snapshot 2 shows the task set after Head of Queue task 3 and Simple task 4 are created. The new Head of Queue task is placed at the front of the queue in the enabled task state, displacing task 1. Snapshot 3 shows the task set after task 3 completes. Since the conditions indicated by the task 1 blocking boundary are still in effect, tasks 2 and 4 remain in the dormant task state.

Figure 35 is the same as the previous example, except that task 1 completes instead of task 3.

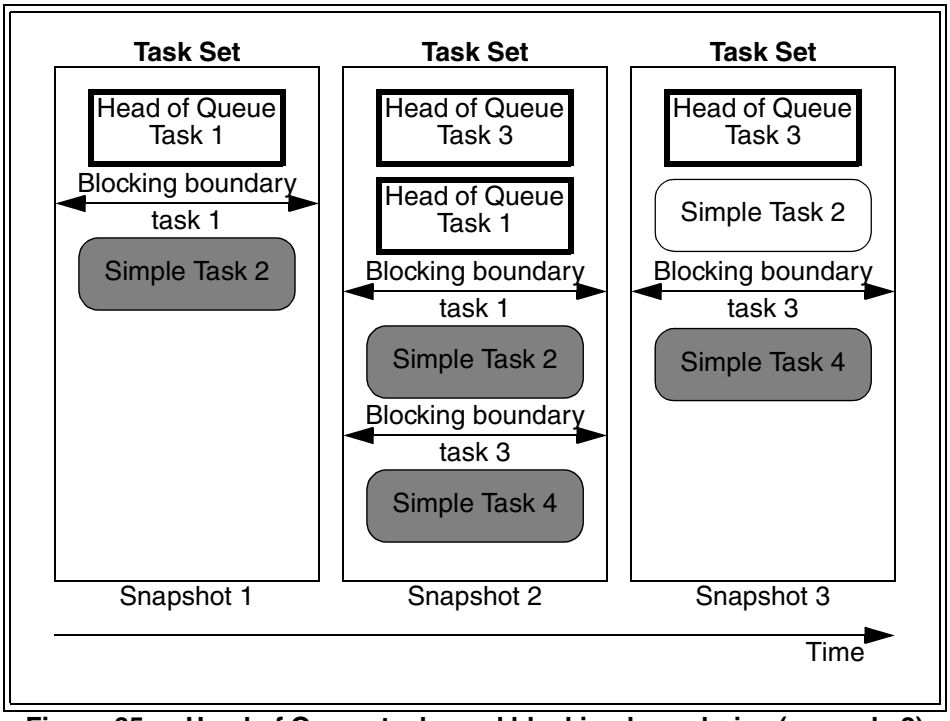



**Figure 35 — Head of Queue tasks and blocking boundaries (example 2)**

The completion of task 1 allows task 2 to enter the enabled task state. Simple task 4 is placed in the dormant task state until task 3 completes.

### 7.7.3 Ordered tasks

An example of Ordered and Simple task interaction is shown in figure 36.

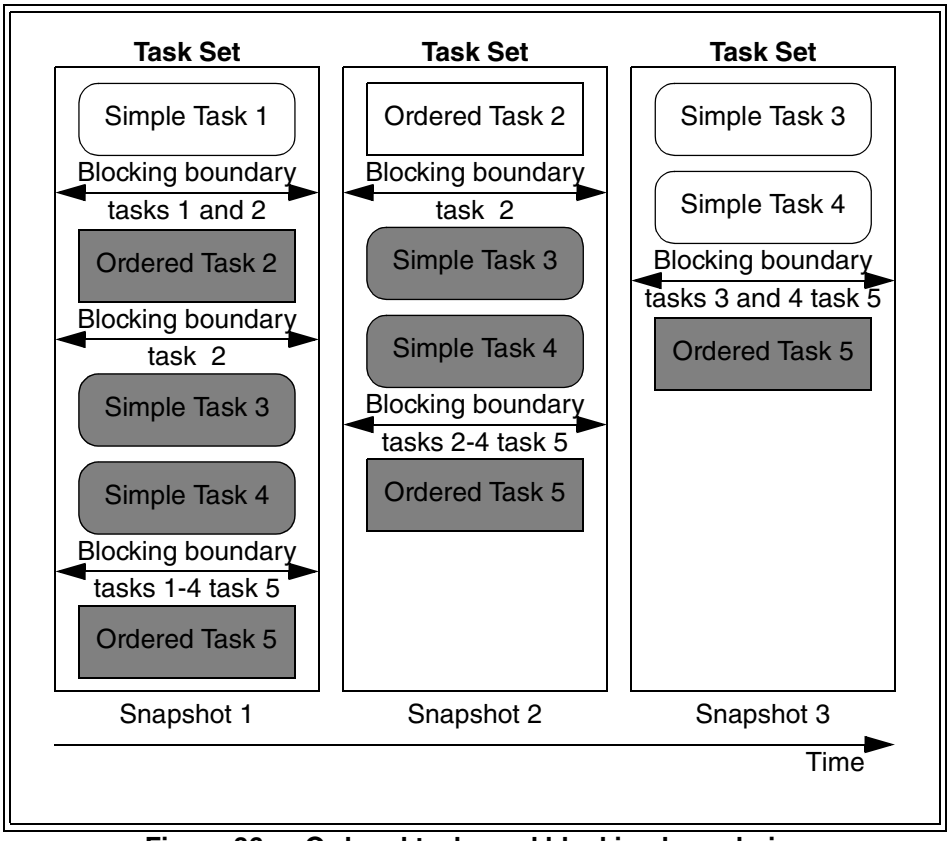



**Figure 36 — Ordered tasks and blocking boundaries**

The state of dormant tasks 2 through 5 is determined by the requirements shown in table 32.

**Table 32 — Dormant task blocking boundary requirements**

| Task | Reason for blocking boundary |
|---|---|
| 2 | An Ordered task is not allowed to enter the enabled task state until all older tasks have ended. |
| 5 | |
| 3 | A Simple task is not allowed to enter the enabled task state until all older Head of Queue and older Ordered tasks have ended. |
| 4 | |

The table 32 constraints are shown by the blocking boundaries in snapshot 1.

In snapshot 2, the completion of task 1 allows ordered task 2 to enter the enabled task state. Since the initial constraints on tasks 3, 4 and 5 are still in effect, these tasks must remain in the dormant task state. As shown in snapshot 3, the completion of task 2 triggers two state changes: the transitions of task 3 and task 4 to the enabled task state. Task 5 must remain in the dormant task state until these tasks end.

### 7.7.4 ACA task

Figure 37 shows the effects of an ACA condition on the task set. This example assumes the QERR field contains 00b in the Control mode page (see SPC-2). Consequently, clearing an ACA condition does not cause tasks to be aborted.

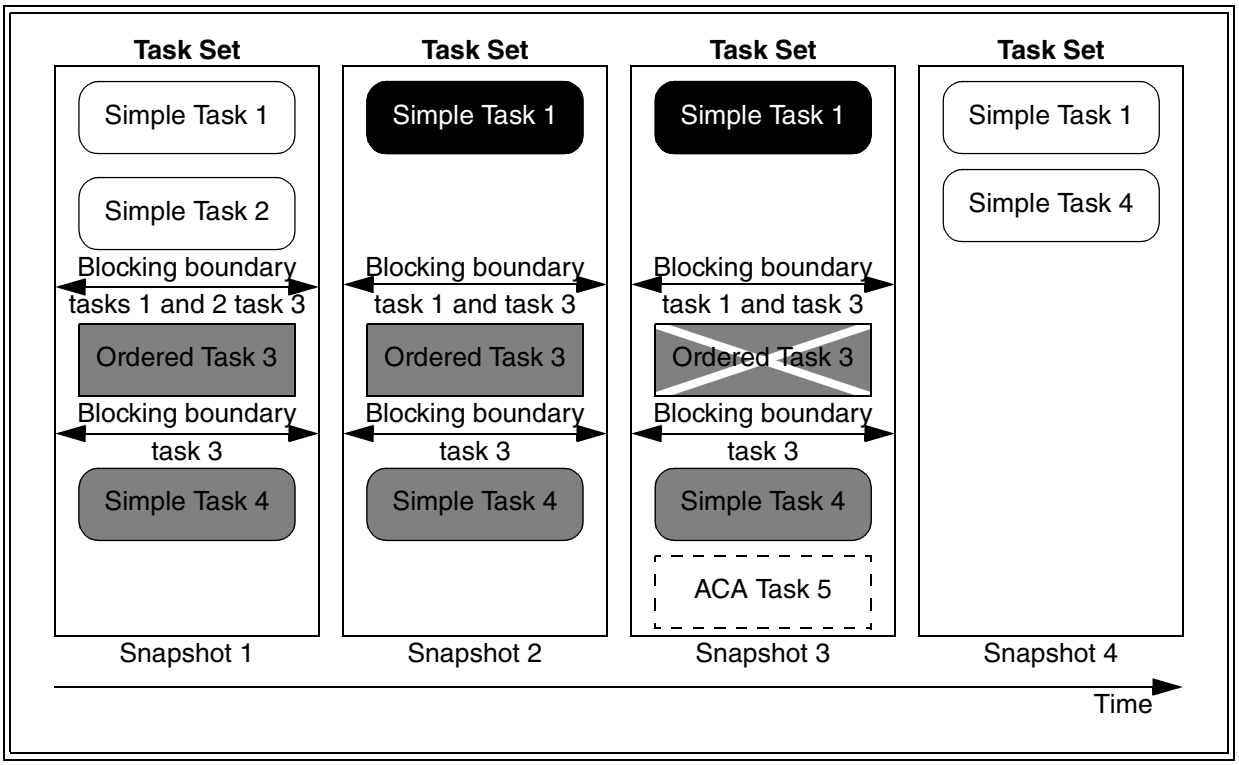



**Figure 37 — ACA task example**

The completion of task 2 with CHECK CONDITION status causes task 1 to enter the blocked task state shown in snapshot 2. In snapshot 3, Ordered task 3 is aborted using the ABORT TASK task management function and ACA task 5 is created to perform additional handling for the exception. Once the ACA condition is cleared, (snapshot 4) Simple task 1 is allowed to reenter the enabled task state. Since there are no Head of Queue or Ordered tasks older than task 4, it too is allowed enter the enabled task state.

# Annex A
(informative)

# Identifiers and names for objects

## A.1 Identifiers and names overview

There needs to be a clear understanding of what SCSI identifiers and names are and how those relate to the objects defined in this standard and SCSI protocol standards. This annex summarizes SCSI identifiers and names.

The following SCSI architecture model objects (i.e., objects) have identifiers and names summarized in this annex:

a) SCSI initiator port (see 3.1.93)
b) SCSI target port (see 3.1.103)
c) Logical unit (see 3.1.59)
d) SCSI initiator device (see 3.1.92)
e) SCSI target device (see 3.1.102)

## A.2 SCSI object and nexus relationship

The I_T_L_Q nexus defines the routing for tasks and task management functions and the identification of tasks. The relationship between the nexus elements and the objects is defined in table A.1.

**Table A.1 — Nexus element to object relationship**

| Nexus Element | Object | Use |
|---|---|---|
| I | Initiator port | Routing and task identification information |
| T | Target port | Routing and task identification information |
| L | Logical unit | Routing and task identification information |
| Q | Task | Task identification information |

## A.3 Identifiers and names

This standard defines the identifiers and name for the objects listed in A.1. The size requirements placed on identifiers and names by this standard are as shown in table A.2. Table A.2 also lists whether this standard or SPC-2 requires SCSI protocols and logical units to support identifiers and names for an object.

**Table A.2 — Object size and support requirements**

| Object | Identifier [a] | | Name [b] | |
|---|---|---|---|---|
| | Size | Support Requirements | Size | Support Requirements |
| Initiator device | n/a | n/a | not specified | optional |
| Target device | n/a | n/a | not specified | optional |
| Initiator port | not specified | mandatory | not specified | optional |
| Target port | not specified | mandatory | not specified [c] | optional |
| Logical unit | 8 bytes (max) | mandatory | not specified [c] | mandatory |

[a] As defined in this standard.
[b] There are no names currently defined in this standard.
[c] Reported in the Device Identification VPD page (see SPC-2).

Each SCSI protocol defines the size and format of identifiers and names for each object.

See table A.3 for a list of the size of the identifiers for each SCSI protocol. See table A.4 for a list of the format of the identifiers for each SCSI protocol.

**Table A.3 — Object identifier size for each protocol**

| Object | Identifier size | | | | |
|---|---|---|---|---|---|
| | SPI-4 | FCP-2 | SRP | iSCSI | SBP-3 |
| Initiator port | 4 bits [a] | 3 bytes | 16 bytes | 262 bytes | 2 bytes |
| Target port | 4 bits [a] | 3 bytes | 16 bytes | 258 bytes | 11 bytes |
| Logical unit | 6 bits (data group transfers) 8 bytes (packetized transfers) | 8 bytes | 8 bytes | 8 bytes | 2 bytes |

[a] SPI-4 uses a bit significant representation of the SCSI port identifier, therefore, the maximum number of SCSI ports is 16.

**Table A.4 — Object identifier format for each protocol**

| Object | Identifier format | | | | |
|---|---|---|---|---|---|
| | SPI-4 | FCP-2 | SRP | iSCSI | SBP-3 |
| Initiator port | bit significant (a max of 16 ports; one for each bit) | binary value | EUI 64 + 8 byte extension [a] | iSCSI name + Initiator Session Identifier (ISID) [b] [c] | binary value |
| Target port | bit significant (a max of 16 ports; one for each bit) | binary value | EUI 64 + 8 byte extension [a] | iSCSI name + Target Portal Group Tag [b] [d] | EUI 64 + Discovery ID [e] |
| Logical unit | binary value (6 bit) or As specified in this standard (8 byte) | As specified in this standard | As specified in this standard | As specified in this standard | As specified in this standard |

[a] Required to be worldwide unique and recommend to be EUI 64 + 8 byte extension.
[b] The iSCSI name should be worldwide unique, 255 bytes maximum in UTF-8 format with null termination.
[c] The Initiator Session Identifier (ISID) is a non-zero six byte integer.
[d] The Target Portal Group Tag is a non-zero two byte integer.
[e] See IEEE Std P1212 for more information on the Discovery ID.

See table A.5 for a list of the size of the names for each SCSI protocol. See table A.6 for a list of the formation of the names for each SCSI protocol.

**Table A.5 — Object name size for each protocol**

| Object | Name size | | | | |
|---|---|---|---|---|---|
| | SPI-4 | FCP-2 | SRP | iSCSI | SBP-3 |
| Initiator name | not specified | not specified | not specified | 256 bytes | not specified |
| Target name | not specified | not specified | not specified | 256 bytes | not specified |
| Initiator port | not specified | 8 bytes | 16 bytes | 262 bytes | 8 bytes |
| Target port | not specified | 8 bytes | 16 bytes | 258 bytes | 11 bytes |
| Logical unit | not specified [a] | 8 or 16 bytes [a] | not specified [a] | not specified [a] | not specified [a] |

[a] Reported in the Device Identification VPD page (see SPC-2).

**Table A.6 — Object name format for each protocol**

| Object | Name format | | | | |
|---|---|---|---|---|---|
| | SPI-4 | FCP-2 | SRP | iSCSI | SBP-3 |
| Initiator name | not specified | not specified | not specified | iSCSI name [a] | not specified |
| Target name | not specified | not specified | not specified | iSCSI name [a] | not specified |
| Initiator port | not specified | Fibre Channel name_identifier | EUI 64 + 8 byte extension [b] | iSCSI name + Initiator Session Identifier (ISID) [a] [c] | EUI 64 |
| Target port | not specified | Fibre Channel name_identifier | EUI 64 + 8 byte extension [b] | iSCSI name + Target Portal Group Tag [a] [d] | EUI 64 + Discovery ID [e] |
| Logical unit | Device Identification VPD page name (see SPC-2) | 8 or 16 byte Fibre Channel name_identifier | not specified | not specified | As specified in this standard |

[a] The iSCSI name should be worldwide unique, 255 bytes maximum in UTF-8 format with null termination.
[b] Required to be worldwide unique and recommend to be EUI 64 + 8 byte extension.
[c] The Initiator Session Identifier (ISID) is a non-zero six byte integer.
[d] The Target Portal Group Tag is a non-zero two byte integer.
[e] See IEEE Std P1212 for more information on the Discovery ID.

## A.4 SCSI protocol acronyms and bibliography

**A.4.1 EUI-64 (Extended Unique Identifier, a 64-bit globally unique identifier):** The IEEE maintains a tutorial describing EUI-64 at http://standards.ieee.org/regauth/oui/tutorials/EUI64.html.

**A.4.2 FCP-2:** SCSI Fibre Channel Protocol -2 (see 1.3).

**A.4.3 IEEE Std P1212:** Standard for a Control and Status Register (CSR) Architecture for Microcomputer Buses. See http://www.ieee.org/.

**A.4.4 iSCSI:** As of this writing, the most recently published iSCSI internet draft is: http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-10.txt. Newer drafts may be identified at http://http://www.ietf.org/html.charters/ips-charter.html.

**A.4.5 SBP-3:** Serial Bus Protocol -3 (see 1.3).

**A.4.6 SPI-4:** SCSI Parallel Interface -4 (see 1.3).

**A.4.7 SRP:** SCSI RDMA Protocol (see 1.3).

**A.4.8 UTF-8:** See ISO/IEC 10646-1:2000, Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane. See http://www.iso.org/.