INTERNATIONAL STANDARD

ISO/IEC 14776-411

First edition
1999-09

Information technology – SCSI-3 Architecture Model (SCSI-3 SAM)

Technologies de l'information –
Modèle d'architecture SCSI-3 (SCSI-3 SAM)

Reference number
ISO/IEC 14776-411:1999(E)

Numéros des publications

Depuis le 1er janvier 1997, les publications de la CEI sont numérotées à partir de 60000.

Publications consolidées

Les versions consolidées de certaines publications de la CEI incorporant les amendements sont disponibles. Par exemple, les numéros d'édition 1.0, 1.1 et 1.2 indiquent respectivement la publication de base, la publication de base incorporant l'amendement 1, et la publication de base incorporant les amendements 1 et 2.

Validité de la présente publication

Le contenu technique des publications de la CEI est constamment revu par la CEI afin qu'il reflète l'état actuel de la technique.

Des renseignements relatifs à la date de reconfirmation de la publication sont disponibles dans le Catalogue de la CEI.

Les renseignements relatifs à des questions à l'étude et des travaux en cours entrepris par le comité technique qui a établi cette publication, ainsi que la liste des publications établies, se trouvent dans les documents ci-dessous:

- **«Site web» de la CEI***
- **Catalogue des publications de la CEI**
  Publié annuellement et mis à jour régulièrement (Catalogue en ligne)*
- **Bulletin de la CEI**
  Disponible à la fois au «site web» de la CEI* et comme périodique imprimé

Terminologie, symboles graphiques et littéraux

En ce qui concerne la terminologie générale, le lecteur se reportera à la CEI 60050: *Vocabulaire Electrotechnique International* (VEI).

Pour les symboles graphiques, les symboles littéraux et les signes d'usage général approuvés par la CEI, le lecteur consultera la CEI 60027: *Symboles littéraux à utiliser en électrotechnique,* la CEI 60417: *Symboles graphiques utilisables sur le matériel. Index, relevé et compilation des feuilles individuelles,* et la CEI 60617: *Symboles graphiques pour schémas.*

* Voir adresse «site web» sur la page de titre.

Numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series.

Consolidated publications

Consolidated versions of some IEC publications including amendments are available. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Validity of this publication

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology.

Information relating to the date of the reconfirmation of the publication is available in the IEC catalogue.

Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is to be found at the following IEC sources:

- **IEC web site***
- **Catalogue of IEC publications**
  Published yearly with regular updates (On-line catalogue)*
- **IEC Bulletin**
  Available both at the IEC web site* and as a printed periodical

Terminology, graphical and letter symbols

For general terminology, readers are referred to IEC 60050: *International Electrotechnical Vocabulary* (IEV).

For graphical symbols, and letter symbols and signs approved by the IEC for general use, readers are referred to publications IEC 60027: *Letter symbols to be used in electrical technology*, IEC 60417: *Graphical symbols for use on equipment. Index, survey and compilation of the single sheets* and IEC 60617: *Graphical symbols for diagrams.*

* See web site address on title page.

INTERNATIONAL STANDARD

ISO/IEC 14776-411

First edition
1999-09

Information technology – SCSI-3 Architecture Model (SCSI-3 SAM)

Technologies de l'information –
Modèle d'architecture SCSI-3 (SCSI-3 SAM)

PRICE CODE W

For price, see current catalogue

CONTENTS

Object Definitions

Figures

Tables

INFORMATION TECHNOLOGY – SCSI-3 Architecture Model (SCSI-3 SAM)

Foreword

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

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

International Standard ISO/IEC 14776-411 was prepared by Joint Technical Committee ISO/IEC JTC 1, Subcommittee SC 25, *Interconnection of information technology equipment*.

Introduction

This International Standard defines generic requirements, which govern SCSI-3 implementation standards, and implementation requirements that apply to all SCSI-3 devices. Implementation requirements specify behavior in terms of measurable or observable parameters pertaining to an implementation. Generic requirements are transformed to implementation requirements by an implementation standard.

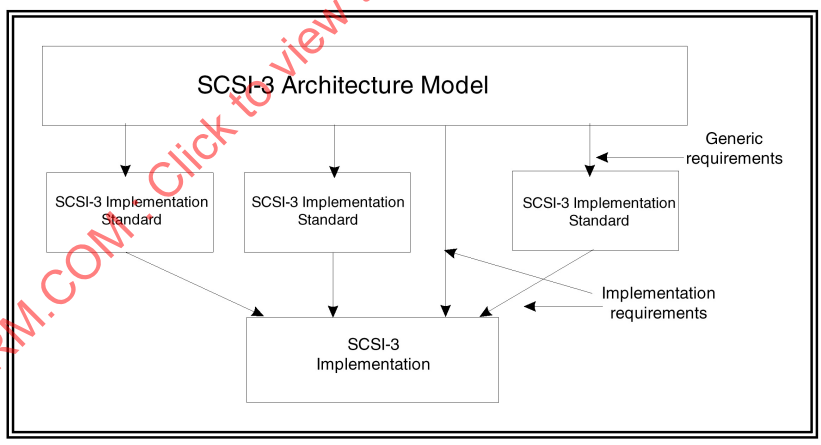

Figure 1 – Requirements Precedence

As shown in figure 1, all SCSI-3 implementation standards shall reflect the generic requirements defined herein. In addition, an implementation claiming SCSI-3 compliance shall conform to the applicable implementation requirements defined in this standard and the appropriate SCSI-3 implementation standards. In the event of a conflict between this document and other SCSI-3 standards the requirements of this standard shall apply.

Table 1 lists the set of ISO/IEC SCSI-3 standards and draft standards in existence at the time of publication. The SCSI-3 implementation standards governed by this standard are those in the Physical I/O, Protocol and Commands categories.

Table 1 – SCSI-3 Document Roadmap

Document Title	Document Categories			
	General	Physical I/O	Protocol	Commands
SCSI Common Access Method Standard (CAM)	9316-421			
SCSI Primary Commands Standard (SPC)				14776-311
SCSI Interlocked Protocol Standard (SIP)			14776-211	
SCSI Parallel Interconnect Standard (SPI)		14776-111		
SCSI Serial Bus Protocol Standard (SBP)			14776-231	
SCSI Architecture Model (SAM)	14776-411			
SCSI Common Access Method -3 Standard (CAM -3)	14776-423			
SCSI Block Commands Standard (SBC)				14776-321
SCSI Stream Commands Standard (SSC)				14776-331
SCSI Graphics Command Standard(SGC)				14776-391
SCSI Medium Changer Commands Standard (SMC)				14776-351
SCSI Fast 20		14776-121		
SCSI Controller Commands Standard (SCC)				14776-341
SCSI Optical Memory Card Reader/Writer (SOMC R/W)				14776-381
SCSI Fibre Channel Protocol (FCP)			14776-221	
SCSI Multi Media Commands Standard (MMC)				14776-361
SCSI Fibre Channel Protocol -2 (FCP -2)			14776-222	
SCSI Parallel Interface -2 (SP I-2)		14776-112		

This document consists of the following seven clauses:

Clause 1 describes the scope of this International Standard.
Clause 2 lists the normative references that apply to this International Standard.
Clause 3 specifies the definitions and notational conventions used by this International Standard.
Clause 4 defines the reference model for an SCSI-3 device.
Clause 5 specifies the model for processing SCSI-3 commands
Clause 6 specifies the task management functions supported by an SCSI-3 device.
Clause 7 specifies the rules for task set management.

INFORMATION TECHNOLOGY – SCSI-3 Architecture Model (SCSI-3 SAM)

1 Scope

This International Standard describes a reference model for the coordination of standards applicable to SCSI-3 I/O systems and a set of common behavioral requirements that are essential for the development of host software and device firmware that can interoperate with any SCSI-3 interconnect or protocol.

2 Normative References

The following standard contains provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the edition indicated was valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent edition of the standard indicated below. Members of IEC and ISO maintain registers of currently valid standards.

ISO/IEC 9316:1995, Information technology – Small Computer System Interface-2.

3 Definitions and Conventions

For purposes of this standard, the following definitions apply.

3.1 Definitions

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

3.1.2 ACA command**:** A command performed by a task with the ACA attribute (see 3.3 and object definition 6).

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

3.1.4 auto contingent allegiance**:** The condition of a task set following the return of a CHECK CONDITION or COMMAND TERMINATED status.

3.1.5 blocked (task state)**:** The state of a task that is prevented from completing due to an ACA condition.

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

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

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

3.1.9 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.

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

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

3.1.12 command descriptor block**:** A structure up to 16 bytes in length used to communicate a command from an application client to a device server.

3.1.13 completed command**:** A command that has ended by returning a status and service response of TASK COMPLETE, LINKED COMMAND COMPLETE, or LINKED COMMAND COMPLETE (WITH FLAG).

3.1.14 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.

3.1.15 confirmation**:** A response returned to an object, which signals the completion of a service request.

3.1.16 confirmed protocol service**:** A service available at the protocol service interface, which requires confirmation of completion.

3.1.17 current task**:** A task that is in the process of sending status or transferring command data to or from the initiator.

3.1.18 destination device**:** The SCSI device to which a service delivery transaction is addressed. See source device.

3.1.19 device server**:** An object, within the logical unit, that executes SCSI tasks according to the rules for task set management described in clause 7.

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

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

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

3.1.23 dormant (task state)**:** The state of a task that is prevented from starting execution due to the presence of certain other tasks in the task set.

3.1.24 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.

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

3.1.26 faulted initiator**:** The initiator to which a COMMAND TERMINATED or CHECK CONDITION status was returned.

3.1.27 faulted task set**:** A task set that contains a faulting task.

3.1.28 faulting command**:** A command that completed with a status of CHECK CONDITION or COMMAND TERMINATED.

3.1.29 faulting task**:** A task that has completed with a status of CHECK CONDITION or COMMAND TERMINATED.

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

3.1.31 hard reset**:** A target response to a reset event or a TARGET RESET in which the target performs the operations described in subclause 5.7.6.

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

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

3.1.34 implementation-specific**:** A requirement or feature that is defined in an SCSI-3 standard but whose implementation may be specified by the system integrator or vendor.

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

3.1.36 initiator**:** An SCSI device containing application clients which originate device service and task management requests to be processed by a target SCSI device.

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

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

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

3.1.40 linked CDB**:** A CDB with the link bit in the control byte set to one.

3.1.41 linked command**:** One in a series of SCSI commands executed by a single task, which collectively make up a discrete I/O operation. In such a series, each command has the same task identifier, and all, except the last, have the link bit in the CDB control byte set to one.

3.1.42 logical unit**:** A target-resident entity which implements a device model and executes SCSI commands sent by an application client.

3.1.43 logical unit number**:** A 64-bit identifier for a logical unit.

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

3.1.45 lower level protocol**:** A protocol used to carry the information representing upper level protocol transactions.

3.1.46 mandatory**:** Implementation of the referenced item is required to claim compliance with a standard.

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

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

3.1.49 option, optional**:** Implementation of the referenced item is not required to claim compliance with an SCSI-3 standard. The implementation of an optional item shall comply with the standard.

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

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

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

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

3.1.54 port**:** Synonymous with "service delivery port".

3.1.55 procedure**:** An operation that can be invoked through an external calling interface.

3.1.56 protocol-specific**:** Implementation of the referenced item is defined by an SCSI-3 protocol standard (see 4.8).

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

3.1.58 protocol option**:** An option whose definition within an SCSI-3 protocol standard is discretionary.

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

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

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

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

3.1.63 queue**:** The arrangement of tasks within a task set, usually according to the temporal order in which they were created. See "task set".

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

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

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

3.1.67 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.

3.1.68 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.

3.1.69 reserved**:** The term used for bits, fields, signals, and code values that are set aside for future standardization.

3.1.70 reset event**:** A protocol-specific event which may trigger a hard reset response from an SCSI device as described in subclause 5.7.6.

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

3.1.72 SCSI application layer**:** The protocols and procedures that implement SCSI commands and task management functions by invoking services provided by an SCSI protocol layer.

3.1.73 SCSI Device**:** A device that is connected to a service delivery subsystem and supports an SCSI application protocol.

3.1.74 SCSI device identifier**:** An address by which an SCSI device is referenced within a domain.

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

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

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

3.1.78 server**:** An SCSI object that performs a service on behalf of a client.

3.1.79 service**:** Any operation or function performed by an SCSI-3 object, which can be invoked by other SCSI-3 objects.

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

3.1.81 service delivery port**:** a device-resident interface used by the application client, device server or task manager to enter and retrieve requests and responses from the service delivery subsystem. Synonymous with "port".

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

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

3.1.84 signal**:** When used as a noun, denotes a detectable asynchronous event possibly accompanied by descriptive data and parameters. When used as a verb, denotes the act of generating such an event.

3.1.85 source device**:** The SCSI device from which a service delivery transaction originates. See destination device.

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

3.1.87 suspended information**:** Information stored within a logical unit that is not available to any pending tasks.

3.1.88 target**:** An SCSI device which receives SCSI commands and directs such commands to one or more logical units for execution.

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

3.1.90 task abort event**:** An event or condition indicating that the task has been aborted by means of a task management function.

3.1.91 task completion event**:** An event or condition indicating that the task has ended with a service response of TASK COMPLETE.

3.1.92 task ended event**:** An event or condition indicating that the task has completed or aborted.

3.1.93 task management function**:** A task manager service which can be invoked by an application client to effect the execution of one or more tasks.

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

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

3.1.96 task manager**:** A server within the target which executes task management functions.

3.1.97 task set**:** A group of tasks within a target device, whose interaction is dependent on the queuing and auto contingent allegiance rules of clause 7.

3.1.98 task slot**:** Resources within the logical unit that may be used to contain a task.

3.1.99 third-party command**:** An SCSI command which requires a logical unit within the target device to assume the initiator role and send an SCSI command to a target device.

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

3.1.101 unconfirmed protocol service**:** A service available at the protocol service interface, which does not result in a completion confirmation.

3.1.102 unlinked command**:** An SCSI-3 command having the link bit set to zero in the CDB control byte.

3.1.103 upper level protocol**:** An application-specific protocol executed through services provided by a lower level protocol.

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

3.2 References to SCSI Standards

The original Small Computer System Interface Standard (ISO 9316:1989), is referred to herein as SCSI-1. SCSI-1 was revised resulting in the Small Computer System Interface -2 (ISO/IEC 9316:1995), referred to herein as SCSI-2. The set of SCSI-3 standards are collectively referred to as SCSI-3. The term SCSI is used wherever it is not necessary to distinguish between the versions of SCSI.

References within this document to individual SCSI-3 standards use one of the document acronyms given in table 1.

3.3 Acronyms and Abbreviations

ACA: Auto contingent allegiance.

AER: Asynchronous event report.

CAM: Common Access Method.

CDB: Command descriptor block.

LLP: Lower level protocol.

LUN: Logical unit number.

SDP: Service delivery port.

SDS: Service Delivery Subsystem.

ULP: Upper level protocol.

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.

Named quantities having a numeric value defined in an SCSI-3 standard are specified using upper case. For example, CHECK CONDITION refers to the status value specified in table 5.

Callable procedures are identified by a name in bold type, such as **Execute Command** (see 5.1). Names of procedural arguments are denoted by capitalizing each word in the name. For instance, Task Identifer is the name of an argument in the **Execute Command** procedure call.

Quantities having a discrete but unspecified value are identified using small capital letters. As an example, LINKED COMMAND COMPLETE (WITH FLAG), indicates a quantity returned by the **Execute Command** procedure call. Such quantities are usually associated with an event or indication whose observable behavior is specific to a given implementation standard.

3.5 Numeric Conventions

Digits 0 to 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 to 9 and the upper case letters "A" to "F" immediately followed by lower-case "h" are hexadecimal values.

Large numbers are not separated by commas or spaces (e.g., 12345; not 12,345 or 12 345).

3.6 Reserved Fields and Code Values

Reserved fields and code values within a data structure specified by this or any other SCSI-3 standard are set aside for future standardization. Their use and interpretation may be specified by future extensions to these standards. A reserved field shall be set to zero, or in accordance with a future extension to any SCSI-3 standard.

3.7 Objects and Object Notation

The SCSI architecture is defined in terms of objects. As specified in this standard, objects are abstractions encapsulating a set of related functions, data types and other objects. Certain objects, such as an interconnect, may correspond to a physical entity while others, such as a task, may only exist conceptually. That is, although such objects exhibit a well-defined, observable set of behaviors, they do not exist as separate physical elements.

An object is a container that may enclose single entities and other objects. For example, an SCSI device may contain logical units. A logical unit may have tasks, a task set and so forth. The following clauses describe notational and graphical conventions for specifying objects.

3.7.1 Notation for Objects

The following symbols are used to define the composition of an object.

=	"is composed of"	This symbol indicates that the object named on the left is composed of the objects named of the right.
+	"together with"	This symbol collects objects into a group. No ordering is implied. In the expression: **A = B + C** object **A** is composed of **B** together with **C**.
[|]	"select one of"	This is equivalent to an "exclusive or" operation. In the expression: **A = [B|C|D]** object **A** is composed of one object selected from **B**, **C** or **D**.

()	"optional"	The objects enclosed in parenthesis are optional. In the expression: **A = B + (C)** object **A** includes **B** and, optionally, **C**.
{ }	"instances of"	A set of objects enclosed within curly brackets may occur any number of times in a given instance. No physical ordering is implied. The brackets may be indexed. For example, M{...}N indicates any number of instances from M to N. Thus: {...}3 implies 0, 1, 2 or 3 instances. 3{...} implies 3 or more instances. The upper limit is implementation or protocol specific. 3{...}3 implies exactly 3 instances. 3{...}5 implies from 3 to 5 instances.
"xxx"	ASCII character string	Object consists of the ASCII encodings for the characters enclosed in quotation marks.
nn	binary encoded value	Object consists of the binary encoding representing the specified value. For example **A = 54** defines object **A** as the binary encoding of the decimal value 54.
...	range	Denotes a sequential set of discrete values. Thus: [1|...|100] implies one out of a set of binary encoded integers between 1 and 100. ["A"|...|"Z"] implies one out of a set of ASCII-encoded alphabetic characters between "A" and "Z". Unless stated otherwise, literals outside the range of specified values are reserved for future standardization.
←	"physically contains"	As in **A** ← **B**. Object **A** physically contains object **B**. Use of this notation is restricted to the specification of object addresses and identifiers as described below.

<nn>	vector of objects	Denotes "nn" physically contiguous instances of the object. Thus, for example: **byte<10>** defines a physically contiguous vector of ten bytes.
/*...*/	remark	Encloses a comment.

There is no physical ordering implied by the sequence in which objects are specified in a grouping. Thus, the term "8{byte}8" or the expression "**A** + **B** + **C**" in an object definition says nothing about the physical ordering of these objects. A physically contiguous vector of items is denoted by means of the array notation specified above.

3.7.2 Object Addresses and Identifiers

This standard defines certain objects that are referenced externally by an address. By convention, such an object is made up of two components: a storage object specifying the maximum size of the address field and a set of allowable identifier values. The following is an example of an identifier object definition.

Ident_a = **byte** ← [0 | ... | 243]

The object "Ident_a" is an identifier composed of an 8-bit "container" (the 'byte' object) and the binary encoding of a single value from 0 to 243. Values not in the range are reserved. The left arrow operator ("←") indicates that the storage object physically contains the encoding for one of the allowed values.

3.7.3 Predefined Objects

The following predefined objects are used throughout this standard:

Constant:	Object containing a fixed value. The container size and contents are implementation-specific.
Buffer = Byte<nn>:	Byte array of size nn.
Value:	Numeric quantity.
Flag = bit ← [0|1]:	A two-valued quantity as shown.

3.7.4 Hierarchy Diagrams

Hierarchy diagrams show how objects are related to each other. The hierarchy diagram of figure 2, for example, shows the relationships among the objects comprising the "Book" object described in the following definition.

Book = 1{Chapter} + (Index) + Table of Contents + (Preface)

Chapter = 1{Section} + 0{Figure}

Preface = 1{Introductory Text} + 0{Figure}

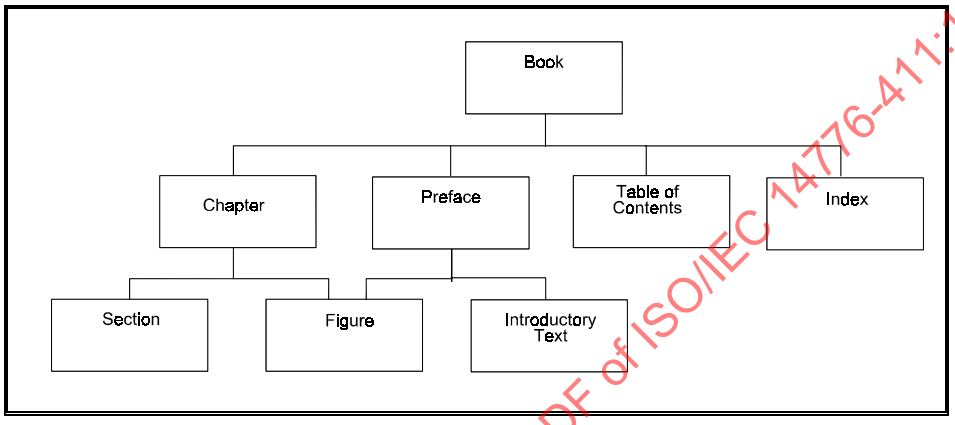

Figure 2 – Example of Hierarchy Diagram

As given in the object definition, a Book object consists of one or more Chapters, a Table of Contents, an optional Preface and optional Index. 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 "Book" and its constituent objects "Chapter", "Index", "Table of Contents" and “Preface”. Similarly, connecting lines show that "Chapter" contains objects "Section" and "Figure". Note that the Figure object may also be a component of Preface.

The hierarchy diagram does not show multiple instances of an object or the fact that certain objects are optional. In this example, the Figure object is shown only once, even though a chapter or preface may have several (or no) instances of this object. Similarly, the Index object is shown even though it too is optional.

3.8 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 ([input-1] [,input-2] ...] || [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.
"(...)":	Parentheses enclosing the lists of input and output arguments.
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.
"||":	A separator providing the demarcation between inputs and outputs. Inputs are listed to the left of the separator; outputs are listed to the right.
"[...]":	Brackets enclosing optional or conditional parameters and arguments.

The data objects are specified using the notation of 3.7.1. This notation allows any data objects to be specified as inputs and outputs. The following is an example of a procedure specification:

Found = Search (Pattern, Item List || [Item Found])

Where:

Found = **Flag**

Flag, which, 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>** /* 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.

Predefined objects commonly used as arguments are defined in 3.7.3.

3.9 State Diagram

All state diagrams use the notation shown in figure 3.

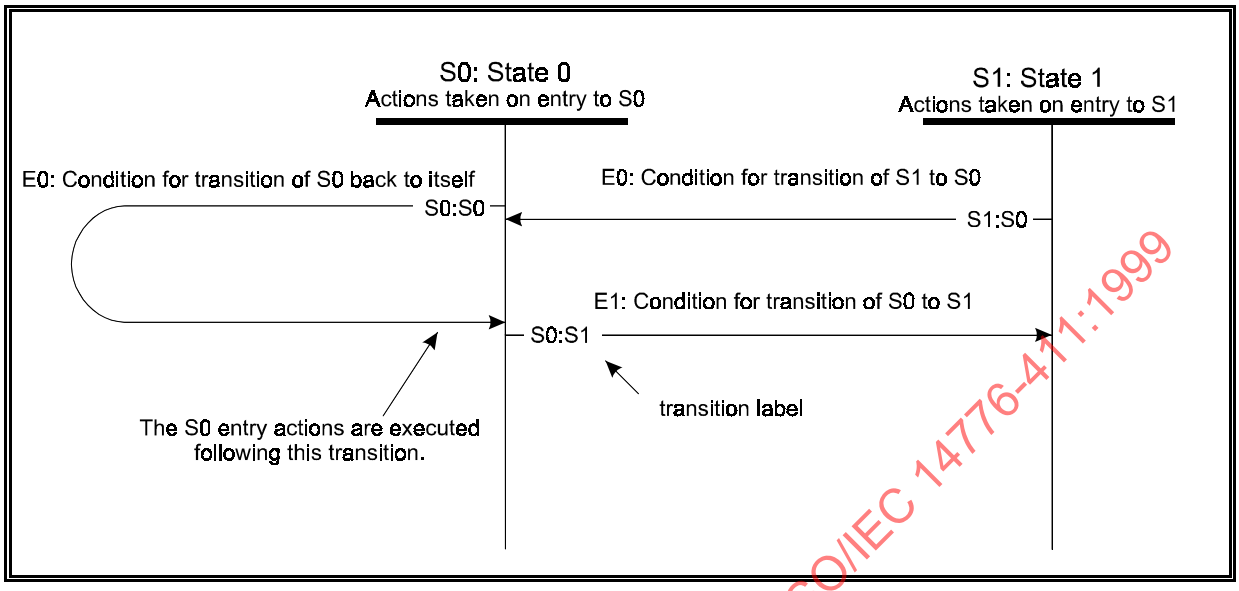

Figure 3 – State Diagram

A system specified in this manner has the following properties:

a) Time elapses only within discrete states.

b) State transitions are logically instantaneous.

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 will restart the actions from the beginning.

4 SCSI-3 Architecture Model

4.1 Introduction

The purpose of the SCSI-3 architecture model is to:

d) Provide a basis for the coordination of SCSI-3 standards development which allows each standard to be placed into perspective within the overall SCSI-3 Architecture model.

e) 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.

f) Provide the foundation for application compatibility across all SCSI-3 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 real SCSI-3 system, elements in a real 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-3 devices in any SCSI-3 interconnect and protocol environment. As such, the model does not address other requirements which 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-3 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 reader not familiar with the concept of abstract modeling is cautioned that concepts introduced in the description of an SCSI-3 I/O system constitute an abstraction despite a similar appearance to concepts possibly found in real systems. Therefore, a real SCSI-3 I/O system need not be implemented as described by the model. Such a system, regardless of how it is implemented, shall, however, comply with the requirements of this and all other applicable standards.

The SCSI-3 architecture model is described in terms of objects, protocol layers and service interfaces between objects. As discussed in 3.7, 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. 4.4 specifies the structure of an SCSI I/O system by defining the relationship among objects. The set of distributed services to be provided are specified in 5.1 and 6.1.

Requirements that apply to each SCSI-3 protocol standard are specified in the protocol service model described in subclauses 5.4 and 6.8. The model describes required behavior in terms of layers, objects within layers and protocol service transactions between layers.

4.2 The SCSI-3 Distributed Service Model

Service interfaces between distributed objects are represented by the client-server model shown in figure 4. Dashed horizontal arrows denote a single request-response transaction as it appears to the client and server. The solid arrows indicate the actual transaction path through the service delivery subsystem. In such a model, each client or server is a single thread of execution which runs concurrently with all other clients or servers.

A client-server transaction is represented as a remote procedure call with inputs supplied by the caller (the client). The procedure executed 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, which identifies the server and the service to be performed, 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 returned. 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 will usually be a time skew between the server and client's perception of request status and logical unit state. All allusions to a pending command or task management function in this standard are in the application client's frame of reference.

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 standpoint, 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 will be detected by the client's service delivery port.

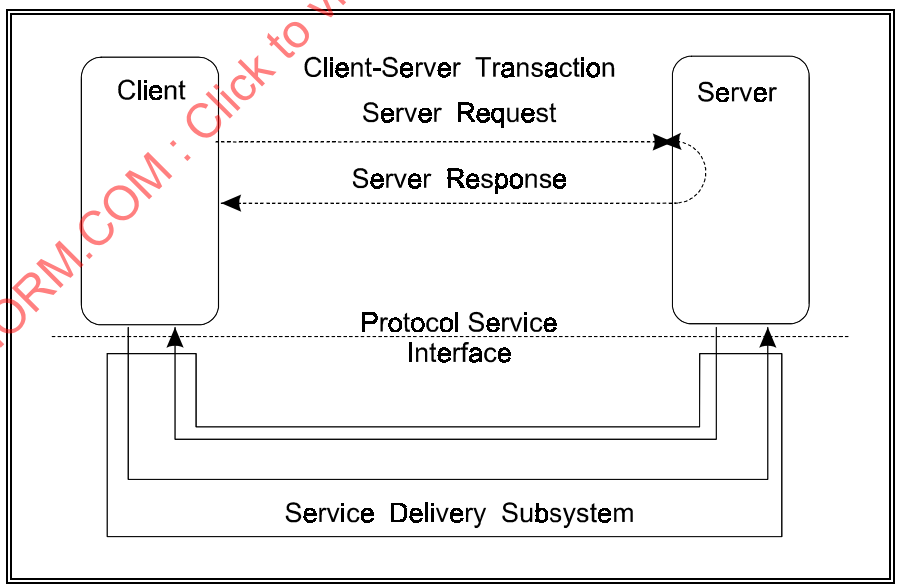

Figure 4 – Client-Server Model

4.3 The SCSI-3 Client-Server Model

As shown in figure 5, each SCSI-3 target device provides two classes of service, device services executed 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-3 command standards and executes SCSI-3 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. As described below, each unit of work is represented within the target by a task which can be externally referenced and controlled through requests issued to the task manager.

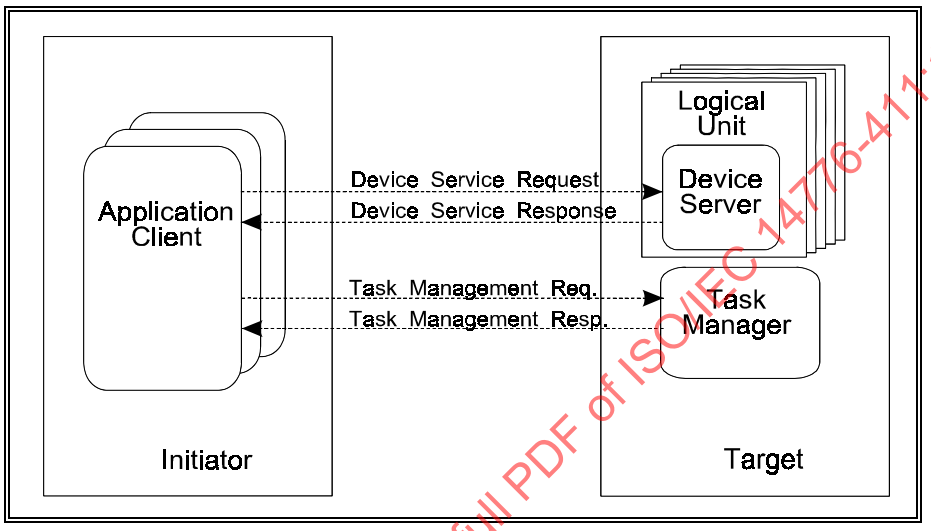

Figure 5 – SCSI Client-Server Model

All requests originate from application clients residing within an initiator device. 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. In the architecture model, an application client is created to issue a single SCSI-3 command or task management function; it 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 SCSI-3 command is issued as a request for device service directed to a device server within a logical unit. Each device service request contains a command descriptor block, 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. 5.6.1 gives an example of the processing for a single command. 5.6.2 gives an example of linked command processing.

An application client may request execution of a task management function through a request directed to the task manager. 6.9 shows the interactions between the task manager and application client when a task management request is processed.

4.4 The SCSI-3 Structural Model

This clause uses the notation for hierarchy diagrams of 3.7.4 and the object notation specified in 3.7.1 to formally define the structure of an SCSI-3 I/O system as seen by an application client. Certain object definitions may include one or more numeric parameters defining an allowable range for addresses or identifiers. The range of addresses or identifiers that shall be supported by an SCSI-3 protocol implementation shall be defined in the SCSI-3 protocol standard that applies to that implementation. Such objects, however, shall not exceed the values specified in this standard. In addition, unless specified otherwise in this standard, an address or identifier supported by an SCSI-3 protocol may be less than the maximum defined herein. To ensure compatibility with any SCSI-3 protocol, the protocol-independent portions of a system implementation should be designed to use the address or identifier specifications as they appear in this standard.

The SCSI-3 structural model represents a view of the elements comprising an SCSI-3 I/O system as seen by application clients interacting with the system through the service delivery port. In an implementation, this view is similar to that seen by a CAM device driver interacting with the system through the CAM SIM layer. This model is defined as a hierarchy of objects. As shown in figure 6, the fundamental object is the SCSI domain, which represents an I/O system. A domain is made up of SCSI devices and a service delivery subsystem, which transports commands and data. An SCSI device, in turn, may consist of logical units and so forth.

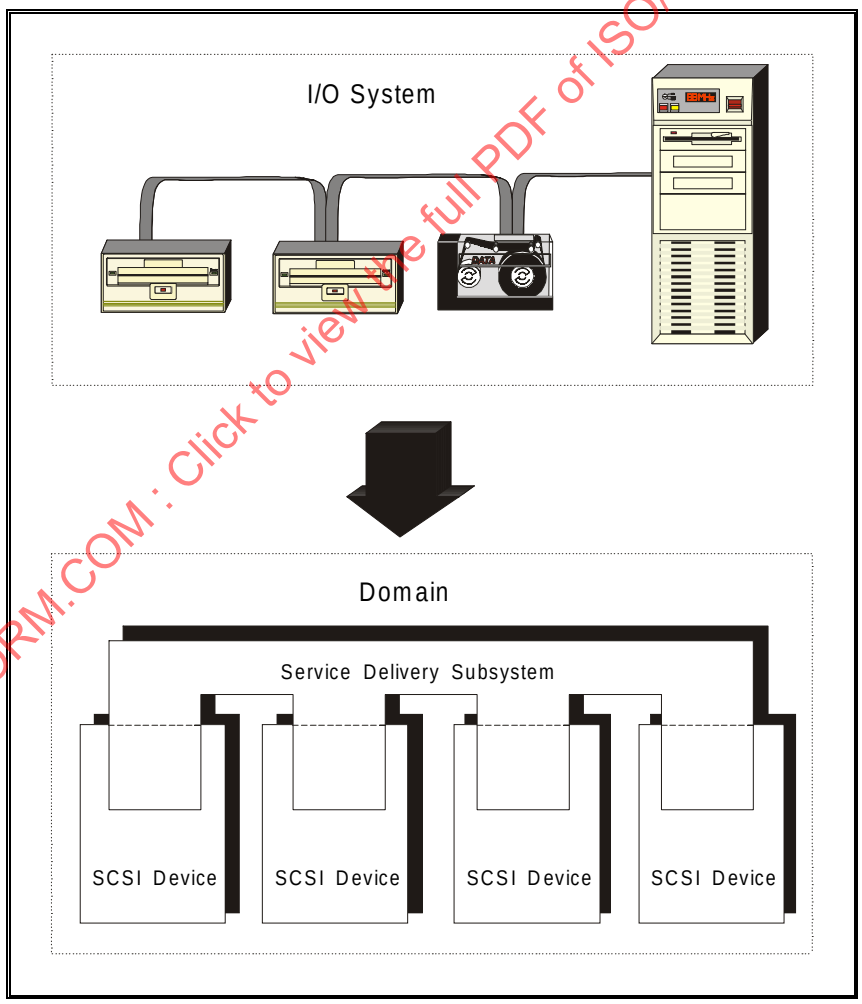

Figure 6 – SCSI I/O System and Domain Model

Figure 7 shows the main functional objects making up the SCSI hierarchy. The following clauses define these objects in greater detail using the conventions of 3.7. Each clause gives the following information:

a) Where appropriate, a functional model indicating the physical relationship among the principle objects comprising the entity;
b) A hierarchy diagram showing the logical relationship among the principle objects comprising the entity;
c) A formal description of the object using the notation defined in 3.7.1.

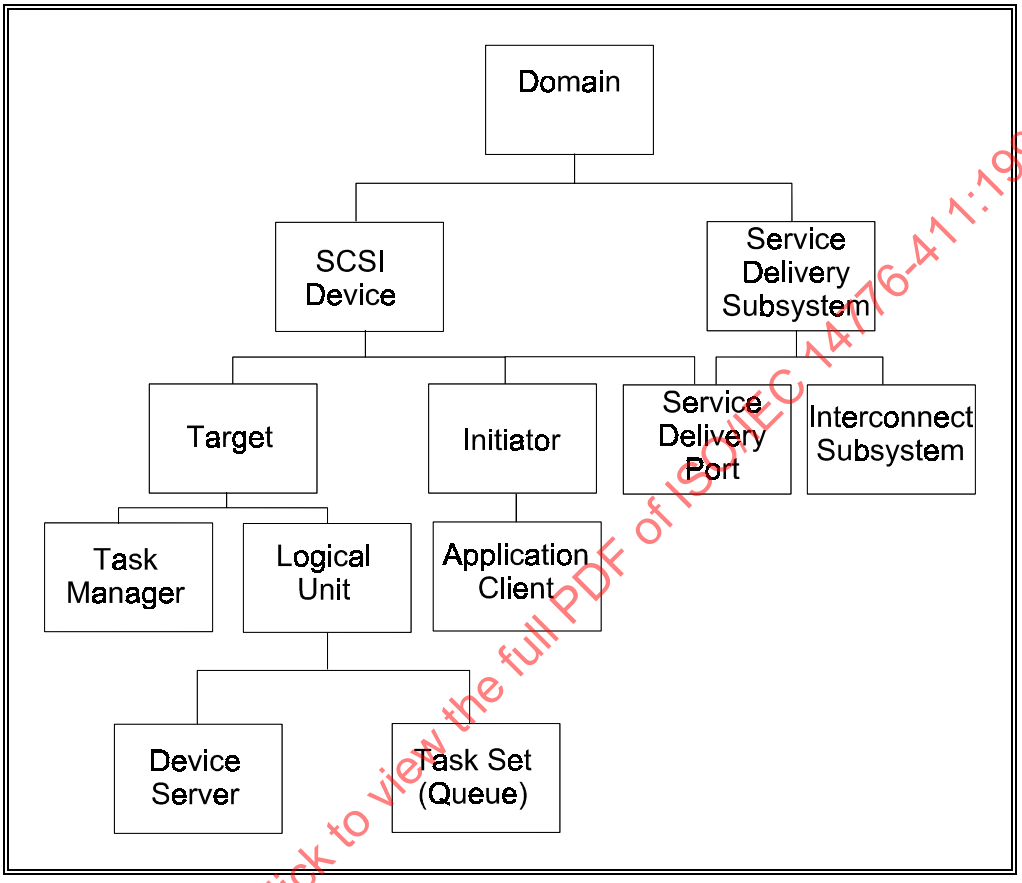

Figure 7 – SCSI Hierarchy

4.5 SCSI Domain

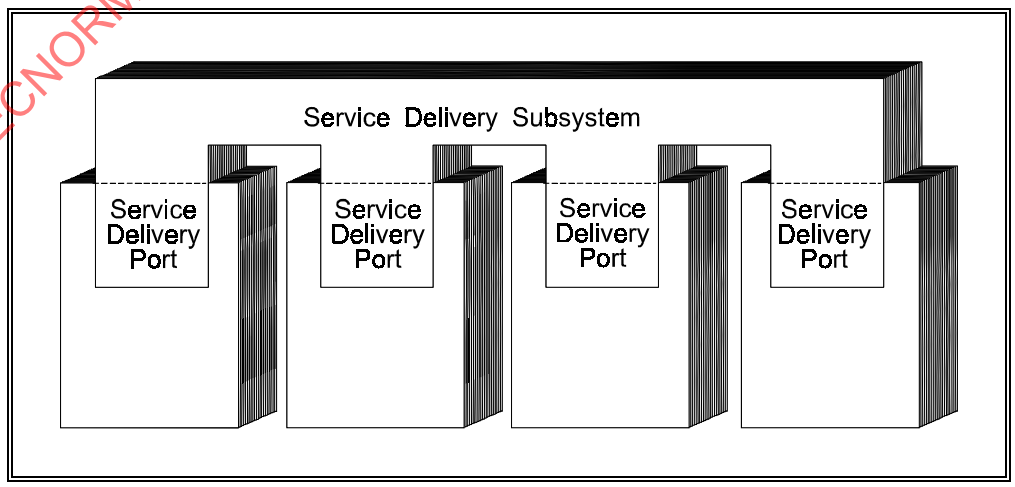

Figure 8 – Domain Functional Model

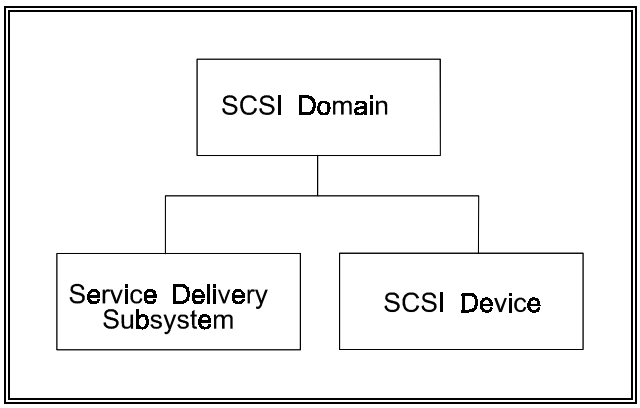

Figure 9 – Domain Hierarchy

Object Definition 1: SCSI Domain (figure 9)

SCSI Domain = 2{SCSI Device} + Service Delivery Subsystem

Object Descriptions:

SCSI Device:	A device that originates or services SCSI-3 commands. As described in 4.7, an SCSI-3 device originating a command is called an initiator; a device containing logical units that service commands is called a target.
Service Delivery Subsystem:	Subsystem through which clients and servers communicate (see 4.6).

The domain boundaries are established by the system implementor, within the constraints of a specific SCSI-3 protocol and interconnect standard.

The Service Delivery Subsystem

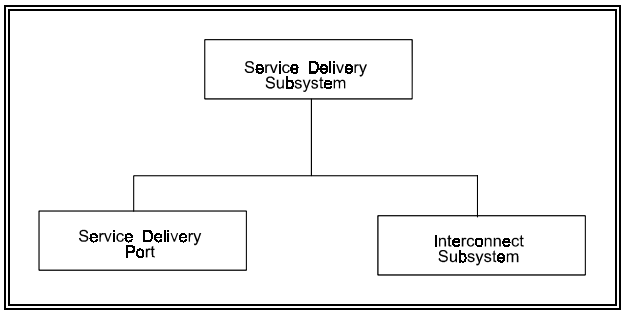

Figure 10 – Service Delivery Subsystem Hierarchy

Object Definition 2: Service Delivery Subsystem (figure 10)

Service Delivery Subsystem = 2{Service Delivery Port} + Interconnect Subsystem

Object Descriptions:

Service Delivery Port:	Device-resident component of the service delivery subsystem (see object definition 3). This object may contain hardware and software that implements the protocols and interface to the interconnect subsystem.
Interconnect Subsystem:	A set of one or more physical interconnects that appear to a client or server as a single path for the transfer of 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 an SCSI-3 implementation may perform these transfers through several interactions with its SCSI-3 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 execute 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 service delivery port for transmission; it is "in transit" until delivered and "received" when it has been forwarded to the receiver via the destination device's service delivery port.

4.5.1 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 initiator. Some SCSI-3 protocols, however, may require the target 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 target and the initiator's awareness of that change.

The model assumes that state synchronization, if required by an SCSI-3 protocol standard, is enforced by the service delivery subsystem transparently to the server. That is, whenever the server invokes a protocol service to return a response as described in 6.8 and 5.4, it is assumed that the service delivery port for such a protocol will not return control to the server until the response has been successfully delivered to the initiator.

4.5.2 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 devices, 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. For example, the sequence in which requests are received is often important whenever an initiator issues a series of SCSI-3 commands with the ORDERED attribute to a logical unit as described in 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 initiator acquires knowledge about the state of pending commands and task management functions and may subsequently take action based on the nature and sequence of target responses. For example, if the initiator aborts a command whose completion response is in transit and the abort response is received out of order, the initiator could incorrectly conclude that no further responses are expected from that command.

The manner in which ordering constraints are established is implementation-specific. An implementation may choose to delegate this responsibility to the application client (e.g., the device driver) or the service delivery port. In some cases, in-order delivery may be an intrinsic property of the transport subsystem or a requirement established by the SCSI-3 protocol standard.

For convenience, the SCSI-3 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 one sending device and several receiving devices.

4.6 SCSI Device Models

Figure 11 shows the functional models for SCSI devices that can perform only target or initiator functions or are capable of supporting both functions. The definition and hierarchy are shown in object definition 3 and figure 12.

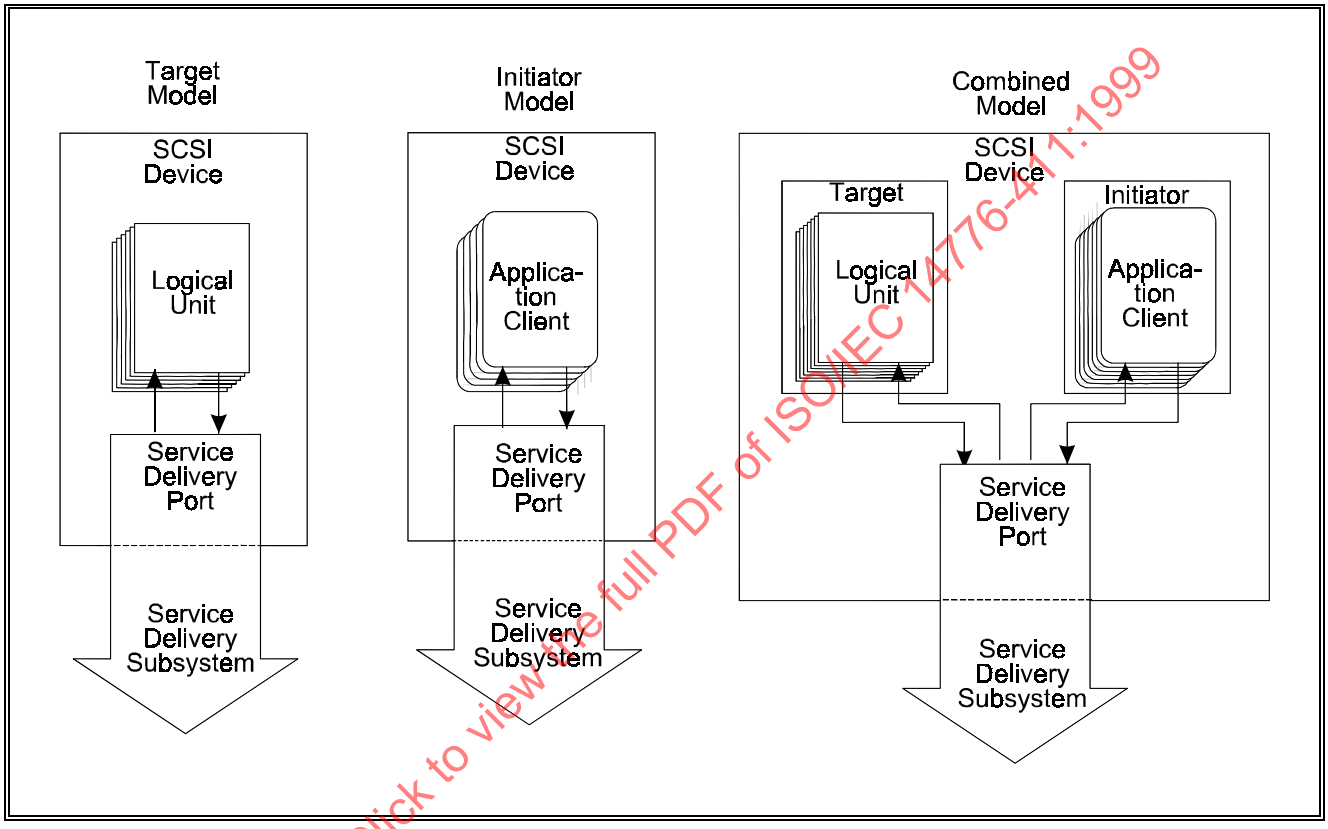

Figure 11 – SCSI Device Functional Models

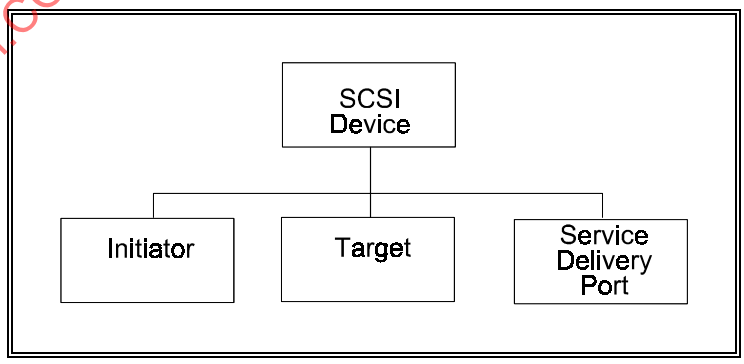

Figure 12 – SCSI Device Hierarchy Diagram

Object Definition 3: SCSI Device (figure 12)

SCSI Device = [Initiator | Target | Target + Initiator] + 1{Service Delivery Port}

Service Delivery Port = Implementation-specific hardware and software

Object Descriptions:

Initiator:	An SCSI-3 device which is capable of originating SCSI-3 commands and task management requests (see 4.7.1).
Target:	An SCSI-3 device which is capable of executing SCSI-3 commands and task management requests (see 4.7.2).
Service Delivery Port:	Device-resident component of the Service Delivery Subsystem containing the hardware and software needed to implement an SCSI-3 protocol and an interface to the interconnect subsystem (see object definition 2 in 4.6).

A device is referred to by its role when it participates in an I/O operation. That is, such a device is called a target when it executes an SCSI-3 command or task management function and an initiator when it issues an SCSI-3 command or task management request.

4.7.1 and 4.7.2 formally define the target and initiator device models.

4.6.1 SCSI Initiator Model

Object Definition 4: Initiator

Initiator = 0{Application Client}

Object Descriptions:

Application Client:	Source of commands and task management functions. There is one application client for each pending command or task management function.

4.6.2 SCSI Target

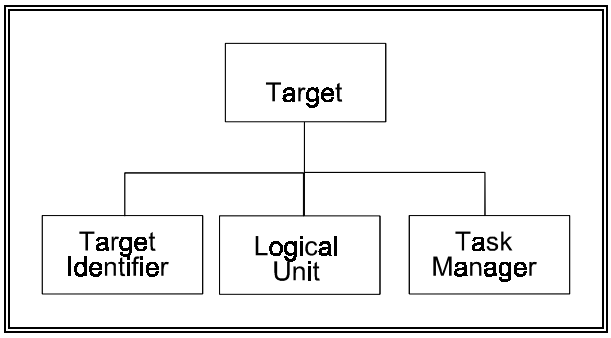

Figure 13 – Target Object Hierarchy

Object Definition 5: Target (figure 13)

Target = 0{Logical Unit} + Logical Unit 0 + 1{Target Identifier} + Task Manager

Target Identifier = bit<64> ← [0|...|2^{64}-1]

Object Descriptions:

Target Identifier: 64 bits identifying the target device.

As implied by object definition 5 above, a target device may respond to more than one target identifier. Each target identifier shall be unique within the scope of the domain. The set of identifiers by which a target device is referenced shall be the same for every initiator in the domain.

Task Manager: Server that controls one or more tasks in response to task management requests.

Logical Unit: Object to which SCSI-3 device commands are directed.

Logical Unit 0: A logical unit whose logical unit number is zero (see 4.7.4).

4.6.3 The Task Manager

The task manager controls the execution of one or more tasks by servicing the task management functions specified in 6.1. Its external address is equal to the target identifier. As specified in object definition 5, there is one task manager per target device.

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

4.6.4 Logical Unit

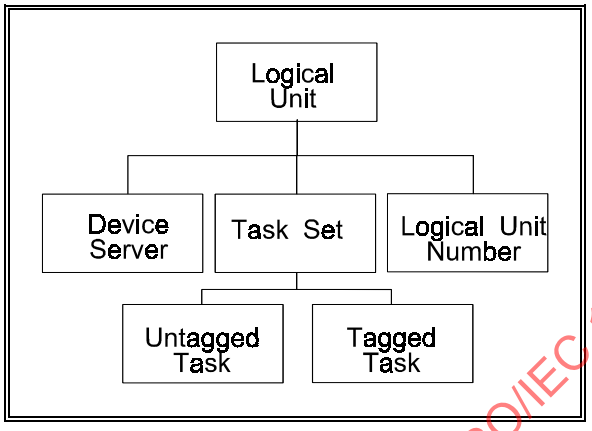

Figure 14 – Logical Unit Object Hierarchy

Object Definition 6: Logical Unit (figure 14)

Logical Unit = Device server + Logical Unit Number + Task Set

Logical Unit Number = bit<64> ← [0|...|2^{64}-1]

Logical Unit Identifier = Target Identifier + Logical Unit Number

Task Set = [0{Tagged Task} + 0{Untagged Task} | 0{Untagged Task}]

Object Descriptions:

Device Server:	Object that executes SCSI commands and manages the task set according to the rules defined in 7.
Task Set:	A set of tasks whose interaction is determined by the rules for task set management specified in 7 and the auto contingent allegiance rules specified in 5.7.1.
Tagged task:	A task whose identifier includes an initiator-specified component (tag) and one of the task attributes specified in object definition 7.
Untagged task:	A task whose identifier does not include a tag component (see object definition 7).

Logical Unit Number: An encoded identifier for the logical unit.

Logical Unit Identifier: External identifier used by an initiator to reference the logical unit.

Object Definition 7: Task

Task = [Tagged Task | Untagged Task]

Tagged Task = Tagged Task Identifier + Task Attribute

Untagged Task = Untagged Task Identifier + SIMPLE

Tag = bit<64> ← [0|...|2^{64}-1]

Task Attribute = [SIMPLE | ORDERED | HEAD OF QUEUE | ACA]

Task Identifier = [Untagged Task Identifier | Tagged Task Identifier]

Tagged Task Identifier = Initiator Identifier + Logical Unit Identifier + Tag

Untagged Task Identifier = Initiator Identifier + Logical Unit Identifier

Task Address = [Untagged Task Address | Tagged Task Address]

Tagged Task Address = Logical Unit Identifier + Tag

Untagged Task Address = Logical Unit Identifier

Initiator Identifier = bit<64> ← [0|...|2^{64} -1]

Object Descriptions:

Tag: 64-bit identifier assigned by the initiator.

Initiator Identifier: Protocol-specific identifier of the initiator from which the command originated (see 4.7.1).

Logical Unit Identifier: Logical unit identifier as defined in object definition 6.

Task Attribute: One of the attributes described in 7.5

Task Address: The address used by an application client to reference a task.

Tagged Task Address: The address used by an application client to reference a tagged task. When used as an argument in a device server or task manager request, the service delivery subsystem will convert this parameter to a tagged task identifier before passing it to the server.

Untagged Task Address:	The address used by an application client to reference an untagged task. When used as an argument in a device server or task manager request, the service delivery subsystem will convert this parameter to an untagged task identifier before passing it to the server.

Every SCSI-3 protocol shall support tagged and untagged tasks. Support for the creation of tagged tasks by a logical unit, however, is a logical unit implementation option.

A task identifier that is in use shall be unique as seen by the initiator originating the command and the target to which the command was addressed. (A task identifier is in use over the interval bounded by the events specified in 5.5). A task identifier is unique if one or more of its components is unique within the scope specified above. By implication, therefore, an initiator shall not cause the creation of more than one untagged task having identical values for the target and logical unit identifiers. Conversely, an initiator may create more than one task with the same tag value, provided at least one of the remaining identifier components is unique.

4.7 The SCSI-3 Model for Distributed Communications

The SCSI-3 model for communications between distributed objects is based on the technique of layering. According to this 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 15.

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 which 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:

a) SCSI-3 application layer: Contains the clients and servers that originate and execute SCSI-3 I/O operations by means of an SCSI-3 application protocol;

d) SCSI-3 protocol layer: Consists of the services and protocols through which clients and servers communicate;

e) Physical interconnect layer: Comprised of the services, signaling mechanism and interconnect subsystem needed for the physical transfer of data from sender to receiver.

The subsystems that make up the protocol and interconnect layers are collectively referred to as the service delivery subsystem. The service delivery port is the device-resident portion of this system.

The set of protocol services implemented by the service delivery subsystem are intended to identify external behavioral requirements that apply to SCSI-3 protocol specifications. While these protocol services may serve as a guide for designing reusable software or firmware that can be adapted to different SCSI-3 protocols, there is no requirement for an implementation to provide the service interfaces specified in this standard.

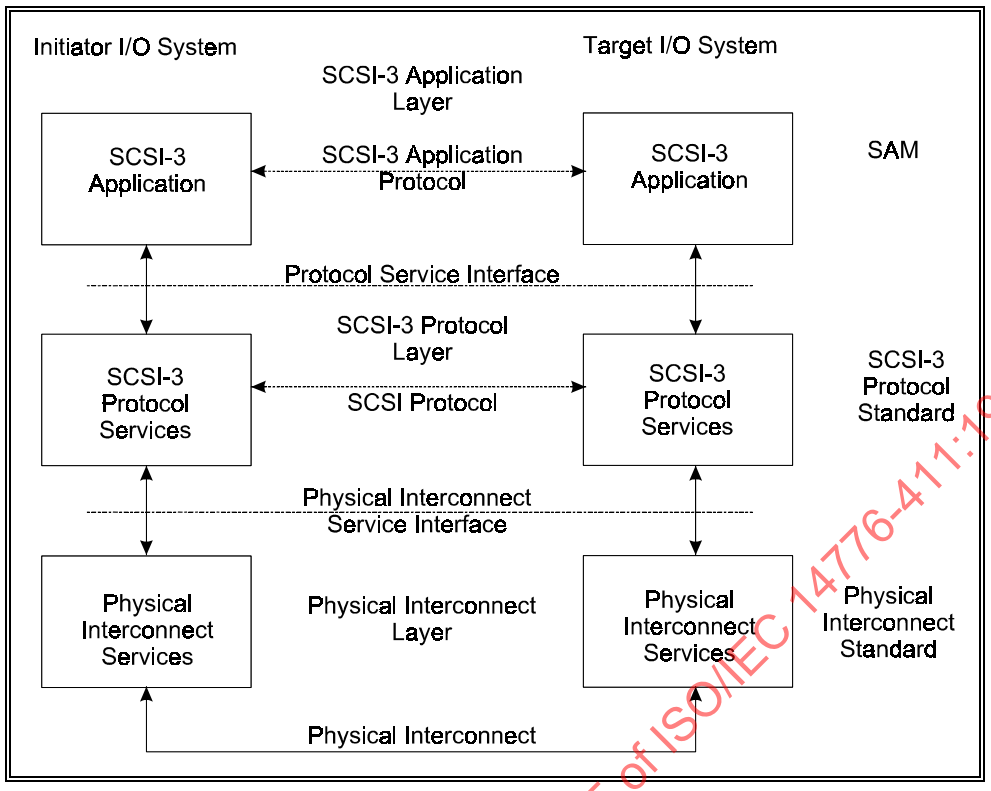

Figure 15 – Protocol Service Reference Model

An interaction between layers can originate from an entity within the LLP or ULP layer. Such interactions are defined with respect to the ULP layer as outgoing or incoming interactions. An outgoing interaction takes the form of a procedure call invoking an LLP service. An incoming interaction appears as a signal sent by the LLP layer, which may be accompanied by parameters and data. Both types of interaction are described using the notation for procedures specified in 3.8. In this model, input arguments are defined relative to the layer receiving an interaction. That is, 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:

a) Protocol service request: A request from the ULP layer invoking some service provided by the LLP layer;

b) Protocol service indication: A signal from the LLP layer informing the ULP layer that an asynchronous event has occurred, such as a reset or the receipt of a peer-to-peer protocol transaction;

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

d) Protocol service confirmation: A signal from the LLP layer notifying the ULP layer that a protocol service request has completed. A confirmation may communicate parameters that indicate the completion status of the protocol service request or any other status. A 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.

All four protocol service types are related as shown in figure 16.

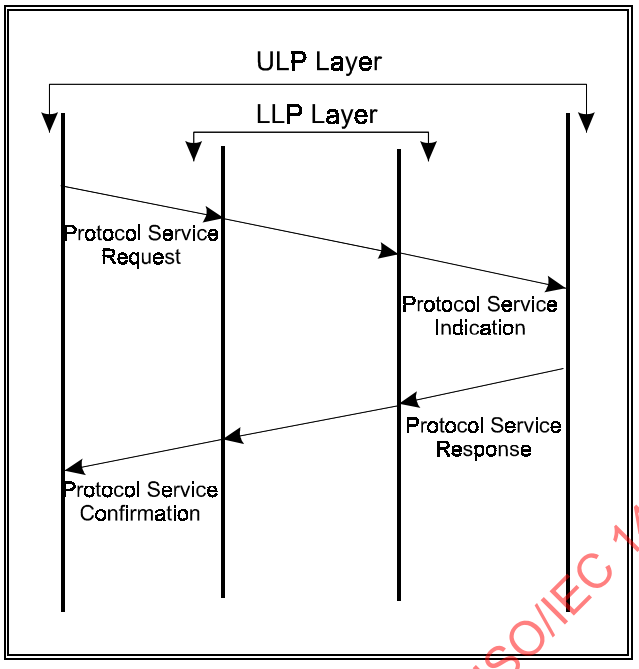

Figure 16 – Protocol Service Model

Figure 17 shows how protocol services may be used to execute a client-server request-response transaction at the SCSI application layer.

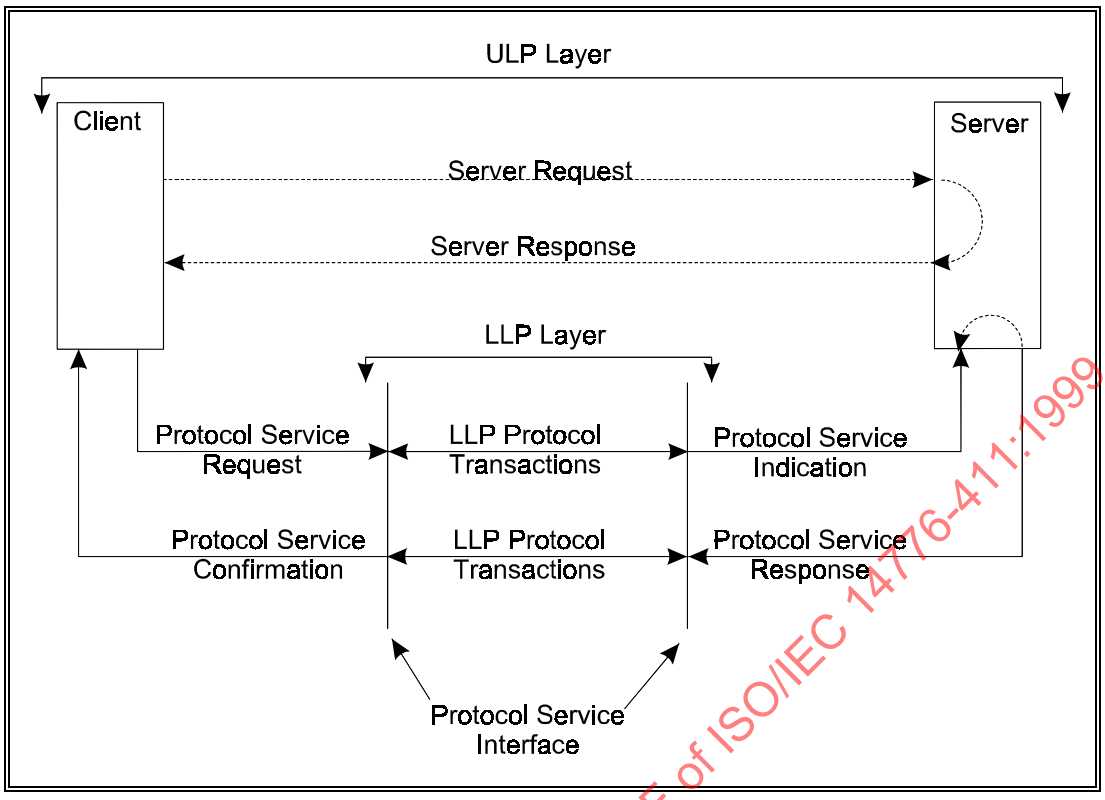

Figure 17 – Request-Response ULP Transaction and Related LLP Services

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

5 SCSI Command Model

This clause describes the reference model for the execution of SCSI commands.

5.1 Remote Procedure for SCSI Command Execution

An application client invokes the following remote procedure to execute an SCSI command:

Service response = Execute Command (Task Address, CDB, [Task Attribute], [Data-Out Buffer], [Command Byte Count], [Autosense Request] || [Data-In Buffer], [Sense Data], Status)

Input Arguments:

Task Address:	See object definition 7 in 4.7.4.
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 next command in a sequence of linked commands. (Untagged tasks shall implicitly have the SIMPLE attribute. The attribute of a task that executes linked commands shall be set according to the Task Attribute argument specified for the first command in the sequence.)
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.
Command Byte Count:	The maximum number of bytes to be transferred by the command.
Autosense Request:	An argument requesting the automatic return of sense data by means of the autosense mechanism specified in 5.7.4.2. It is not an error for the application client to provide this argument when autosense is not supported by the SCSI-3 protocol or logical unit.

Output Arguments:

Data-In buffer:	A buffer containing 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 will usually have to obtain additional information from the logical unit, such as sense data, to determine the state of the buffer contents.

Sense Data: A buffer containing sense data returned by means of the autosense mechanism (see 5.7.4.2).

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.

An SCSI-3 command shall not allow both the Data-In Buffer and the Data-Out Buffer arguments.

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.

LINKED COMMAND COMPLETE (WITH FLAG): A value of LINKED COMMAND COMPLETE (WITH FLAG) indicates that a linked command with the flag bit set to one in the CDB control byte has completed.

SERVICE DELIVERY OR TARGET FAILURE: The command has been ended due to a service delivery failure or target device malfunction. All output parameters may be invalid.

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

An application client requests execution of a linked command by setting the link bit to one in the CDB control byte as specified in 5.2.2. 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 or LINKED COMMAND COMPLETE (WITH FLAG), an application client may issue the next command in the series through an **Execute Command** remote procedure call having the same task identifier. The Task Attribute argument shall be omitted. If the application client issues the next command without waiting for one of the linked command complete responses, the overlapped command condition described in 5.7.2 may result.

5.2 Command Descriptor Block

The command descriptor block defines the operation to be performed by the device server. For some commands, the command descriptor block is accompanied by a list of command parameters contained in the Data-Out buffer defined in subclause 5.1. The parameters required for each command are specified in the applicable SCSI-3 command standards.

Validation of reserved fields in a CDB is a logical unit option. 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 the SPC standard). It shall also be acceptable for a logical unit to interpret a field or code value in accordance with a future revision to an SCSI-3 standard.

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

As shown in table 2, all command descriptor blocks shall have an operation code (see table 3) as the first byte and a control byte (see table 4) as the last byte. The remaining parameters depend on the command to be executed. All SCSI protocol specifications shall accept command descriptor blocks less than or equal to 16 bytes in length. Command descriptor blocks shall not exceed sixteen bytes in length.

Table 2 – Format of Command Descriptor Block

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code							
1	Command-Specific Parameters							
n -1								
n	Control							

5.2.1 Operation Code

The first byte of an SCSI command descriptor block shall contain an operation code. The operation code (table 3) of the command descriptor block 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 for CDBs specified therein shall correspond to the length of the command descriptor as set forth in the following list.

The group code specifies one of the following groups:
Group 0 - six-byte commands
Group 1 - ten-byte commands
Group 2 - ten-byte commands
Group 3 - reserved
Group 4 - sixteen-byte commands
Group 5 - twelve-byte commands
Group 6 - vendor specific
Group 7 - vendor specific

Table 3 – Operation Code

Bit	7	6	5	4	3	2	1	0
	Group Code			Command Code				

5.2.2 Control Field

The control field is the last byte of every command descriptor block. The control field is defined in table 4.

Table 4 – Control Field

Bit	7	6	5	4	3	2	1	0
	Vendor Specific		Reserved			NACA	Flag	Link

All SCSI-3 protocol specifications and protocol implementations shall provide the functionality needed for a logical unit to implement the NACA bit, link bit and flag bit as described herein.

The NACA (Normal ACA) bit is used to control the rules for handling an ACA condition caused by the command. 5.7.1.1 specifies the actions to be taken by a logical unit in response to an auto contingent allegiance condition for NACA bit values of one or zero. All logical units shall implement support for the Normal ACA value of zero and may support the Normal ACA value of one. The ability to support a Normal ACA value of one is indicated in standard INQUIRY data.

If the NACA bit is set to a value which is not supported, the logical unit shall complete the command with a status of CHECK CONDITION and a sense key of ILLEGAL REQUEST. The rules for handling the resulting auto contingent allegiance condition shall be in accordance with the supported bit value.

The link bit is used to continue the task across multiple commands. The flag bit may be used, in conjunction with the link bit, to notify the initiator in an expedited manner that the command has completed.

Support for the link bit is a logical unit option. A link bit of one indicates that the initiator requests continuation of the task across two or more SCSI commands. If the link bit is one and the flag bit is zero and if 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).

Support for the flag bit is a logical unit option. If the link bit and flag bit are both set to one and if the command completes with a status of INTERMEDIATE or INTERMEDIATE-CONDITION MET a logical unit that supports the flag bit shall return a service response of LINKED COMMAND COMPLETE (WITH FLAG).

The logical unit shall complete the command with a status of CHECK CONDITION and a sense key of ILLEGAL REQUEST if:

a) The link bit is set to one and the logical unit does not support linked commands or,
b) The flag bit is set to one and the logical unit does not support the flag bit or,
c) The flag bit is set to one and the link bit is set to zero.

5.3 Status

The status codes are specified in table 5. Status shall be sent from the logical unit to the application client whenever a command ends with a service response of TASK COMPLETE, LINKED COMMAND COMPLETE, or LINKED COMMAND COMPLETE (WITH FLAG). The receipt of any status, except INTERMEDIATE or INTERMEDIATE-CONDITION MET, shall indicate that the associated task has ended.

Table 5 – Status Code

Status byte codes	Status
0h	GOOD
2h	CHECK CONDITION
4h	CONDITION MET
8h	BUSY
10h	INTERMEDIATE
14h	INTERMEDIATE-CONDITION MET
18h	RESERVATION CONFLICT
22h	COMMAND TERMINATED
28h	TASK SET FULL
30h	ACA ACTIVE
All other codes	Reserved

Definitions for each status byte code are given below.

GOOD. This status indicates that the Device Server has successfully completed the task.

CHECK CONDITION. This status indicates that an auto contingent allegiance condition has occurred (see 5.7.1).

CONDITION MET. This status is returned whenever the requested operation specified by an unlinked command is satisfied (see the SEARCH DATA (SBC) and PRE-FETCH (SBC) commands).

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.

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 or COMMAND TERMINATED 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 SEARCH DATA (SBC) and PRE-FETCH (SBC) commands), unless the command is terminated with CHECK CONDITION, RESERVATION CONFLICT, TASK SET FULL, BUSY or COMMAND TERMINATED status. If INTERMEDIATE or INTERMEDIATECONDITION 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 extent within a logical unit that is reserved with a conflicting reservation type for another SCSI device (see the Reserve commands in the SPC standard). The recommended initiator recovery action is to issue the command again at a later time.

COMMAND TERMINATED. This status shall be returned whenever the logical unit terminates a task in response to a TERMINATE TASK task management request (see 6.7). This status also indicates that an auto contingent allegiance has occurred (see 5.7.1)

TASK SET FULL. This status shall be implemented if the logical unit supports the creation of tagged tasks (see object definition 7). This status shall be returned when the logical unit receives a command and does not have enough resources to enter the associated task in the task set.

ACA ACTIVE. This status shall be returned when an auto contingent allegiance 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 in the task set;
b) The initiator issuing the command did not cause the ACA condition;
c) The task created to execute 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 5.7.1).

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

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 Protocol Services in Support of Execute Command

This section describes the protocol services which support the **Execute Command** remote procedure call. All SCSI-3 protocol specifications shall define the protocol-specific requirements for implementing the Send SCSI Command Protocol service request and the Command Complete Received confirmation described below. Support for the SCSI Command Received indication and Send Command Complete response by an SCSI-3 protocol standard is optional. All SCSI-3 I/O systems shall implement these protocols as defined in the applicable protocol specification.

Unless stated otherwise, argument definitions and the circumstances under which a conditional argument must be present are the same as in clause 5.1.

Protocol Service Request:

Send SCSI Command **(Task Address, CDB, [Task Attribute], [Data-Out Buffer], [Command Byte Count], [Autosense Request] ||)**

Protocol Service Indication:

SCSI Command Received **(Task Identifier, [Task Attribute], CDB, [Autosense Request] ||)**

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-3 protocol and logical unit.

Protocol Service Response (from device server):

Send Command Complete	**(Task Identifier, [Sense Data], Status, Service Response ‖)**

The Sense Data argument, if present, instructs the target's service delivery port to return sense information to the initiator automatically (see 5.7.4.2).

Protocol Service Confirmation:

Command Complete Received	**(Task Address, [Data-In Buffer], [Sense Data], Status, Service Response ‖**)

5.4.1 Data Transfer Protocol Services

The data transfer services described in this section are provided to complete the functional model of target protocol services which support the Execute Command remote procedure call. All SCSI-3 protocol standards shall define the protocols required to implement these services.

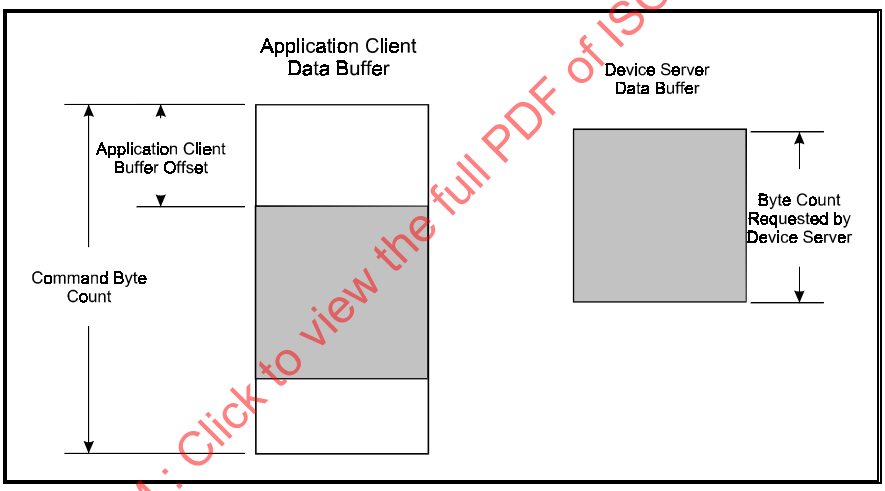

Figure 18 – Model for buffered data transfers

It is assumed that the buffering resources available to the device server are limited and may be much less than the amount of data that can be transferred in one SCSI command. In this case, such data must 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 request is usually a function of the buffering resources available to the logical unit. Figure 18 shows the model for such incremental data transfers.

As shown in figure 18, the application client's buffer is a single, logically contiguous block of memory large enough to hold all the data required by the command. The model requires unidirectional data transfer. That is, the execution of an SCSI-3 command shall not require the transfer of data for that command both to and from the application client.

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

Application client buffer offset:	Offset in bytes from the beginning of the application client's buffer to the first byte of transferred data.
Byte count requested by device server:	Number of bytes to be moved by the data transfer request.
Command byte count:	Upper limit on the extent of the data to be transferred by the SCSI command.

If an SCSI-3 protocol supports random buffer access, as described below, 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 transfer extent and shall not be used by an initiator or target implementation to determine the command byte count.

All SCSI-3 protocol specifications and initiator implementations shall support a resolution of one byte for the above parameters. A target device may support any convenient resolution.

Random buffer access occurs when the device server requests data transfers to or from segments of the application client's buffer which have an arbitrary offset and extent. Buffer access is sequential when successive transfers access a series of monotonically increasing, adjoining buffer segments. Support for random buffer access by an SCSI-3 protocol specification is optional. A device server implementation designed for any protocol implementation should be prepared to use sequential buffer access when necessary.

The following clauses specify the LLP confirmed services used by the device server to request the transfer of command data to or from the application client. The initiator protocol service interactions are unspecified.

5.4.2 Data-In Delivery Service

Request:

Send Data-In (Task Identifier, Device Server Buffer, Application Client Buffer Offset, Request Byte Count ||)

Argument Descriptions:

Task Identifier:	See object definition 7.
Device server buffer:	Buffer from which data is to be transferred.
Application client buffer offset:	Offset in bytes from the start of the buffer to the location within the application client’s buffer to receive the first byte of data.
Request byte count:	Number of bytes to be moved by this request.

Confirmation:

Data Delivered (Task Identifier ||)

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

5.4.3 Data-Out Delivery service

Request:

Receive Data-Out (Task Identifier, Application Client Buffer Offset, Request Byte Count, Device Server Buffer ||)

Argument Descriptions:

See 5.4.2.

Confirmation:

Data-Out Received (Task Identifier ||)

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

5.5 Task and Command Lifetimes

This clause specifies the events delimiting the beginning and end of a task or pending SCSI-3 command from the viewpoint of the device server and application client.

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

The task shall exist until:

a) The device server sends a protocol service response for the task of TASK COMPLETE.

b) A power on condition occurs.

c) The target executes a hard reset operation as described in 5.7.6.

d) The task manager executes an ABORT TASK referencing the specified task.

e) The task manager executes an ABORT TASK SET or CLEAR TASK SET task management function directed to the task set containing the specified task.

An SCSI-3 command is pending when the associated SCSI Command Received indication is passed to the device server. The command ends on the occurrence of one of the conditions described above or when the device server sends a service response for the task of LINKED COMMAND COMPLETE or LINKED COMMAND COMPLETE (WITH FLAG).

The application client assumes that the task exists from the time the **Send SCSI Command** 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) A unit attention condition with one of the following additional sense codes:

COMMANDS CLEARED BY ANOTHER INITIATOR (if in reference to the task set containing the task),
POWER ON,
RESET,
TARGET RESET.

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 CLEAR TASK SET task management function directed to the task set containing the specified task.

f) A service response of FUNCTION COMPLETE in response to a TARGET RESET.

The application client assumes the command is pending from the time it calls the **Send SCSI Command** protocol service until one of the above responses or a service response of LINKED COMMAND COMPLETE or LINKED COMMAND COMPLETE (WITH FLAG) is received.

As discussed in 4.6.1, when an SCSI-3 protocol does not require state synchronization, there will usually 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 will usually be different from the lifetime observed by the device server.

5.6 Command Processing Examples

The following clauses give examples of the interactions for linked and unlinked commands.

5.6.1 Unlinked Command Example

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

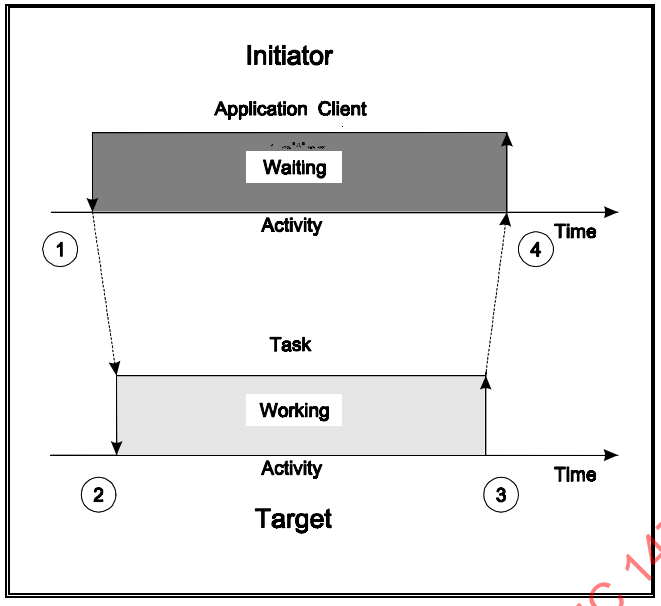

Figure 19 – Command processing events

The numbers in figure 19 identify the events described below.

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

2. The Device Server is notified through an **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 execution.

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

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

5.6.2 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 following example shows the events in a sequence of two linked commands.

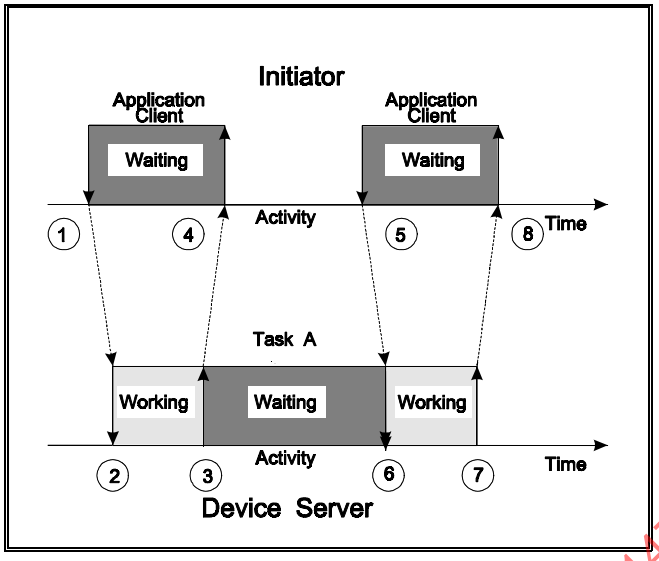

Figure 20 – Linked Command Processing Events

The numbers in figure 20 Identify the events described below.

1. The application client performs an **Execute Command** remote procedure call by invoking the **Send SCSI Command** 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 5.2.2).

2. The target's service delivery port issues an **SCSI Command Received** indication 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** 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 initiator's service delivery port 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** protocol service as described in step 1. The Task Attribute argument is omitted. The link bit in the CDB control byte is clear.

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

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

8. The LLP delivers a **Command Complete Received** confirmation to the application client, which contains the service response and status.

5.7 Command Processing Considerations and Exception Conditions

The following clauses describe some exception conditions and errors associated with command processing and the sequencing of commands.

5.7.1 Auto Contingent Allegiance

The auto contingent allegiance condition shall exist within the task set when the logical unit completes a command by returning a COMMAND TERMINATED or CHECK CONDITION status (see 5.3).

In the following discussion, the term "faulting command" refers to the command that completed with a CHECK CONDITION or COMMAND TERMINATED status. The term "faulted initiator" refers to the initiator receiving the COMMAND TERMINATED or CHECK CONDITION status. The term "faulted task set" refers to the task set having the auto contingent allegiance condition.

5.7.1.1 Logical Unit Response to Auto Contingent Allegiance

The auto contingent allegiance condition shall not cross task set boundaries and shall be preserved until it is cleared as described in 5.7.1.2. If requested by the application client and supported by the protocol and logical unit, sense data shall be returned as described in 5.7.4.2.

NOTE 1 The SCSI-2 contingent allegiance condition and extended auto contingent allegiance condition have been replaced in SCSI-3 by auto contingent allegiance.

NOTE 2 If the SCSI-3 protocol does not enforce state synchronization as described in 4.6.1, there may be a time delay between the occurrence of the auto contingent allegiance condition and the point at which the initiator becomes aware of the condition.

After sending status and a service response of TASK COMPLETE, the logical unit shall modify the state of all tasks in the faulted task set as described in clause 7.

A task created by the faulted initiator while the auto contingent allegiance condition is in effect may be entered into the faulted task set under the conditions described below. Tasks created by other initiators while the ACA condition is in effect shall not be entered into the faulted task set and shall be completed with a status of ACA ACTIVE.

As described in 5.7.1.2, the setting of the NACA bit in the control byte of the faulting command determines the rules that apply to an ACA condition caused by that command. If the NACA bit was set to zero the SCSI-2 contingent allegiance rules shall apply. In that case, the completion of a subsequent command from the faulted initiator with a status of CHECK CONDITION or COMMAND TERMINATED shall cause a new auto contingent allegiance condition to exist. The rules for responding to the new auto contingent allegiance condition shall be determined by the state of the NACA bit in the new faulted command.

If the NACA bit was set to one in the CDB control byte of the faulting command, then a new task created while the ACA condition is in effect shall be entered into the faulted task set provided:

a) The command was originated by the faulted initiator,
b) The task has the ACA attribute,
c) No other task having the ACA attribute is in the task set.

The auto contingent allegiance condition shall not be cleared. If the conditions listed above are not met, the newly created task shall not be entered into the task set and shall be completed with a status of ACA ACTIVE.

If a task having the ACA attribute is received and no auto contingent allegiance condition is in effect for the task set or if the NACA bit was set to zero in the CDB for the faulting command, then the ACA task shall be completed with a status of CHECK CONDITION. The sense key shall be set to ILLEGAL REQUEST with an additional sense code of INVALID MESSAGE ERROR. As noted in 5.7.1.2, a new auto contingent allegiance condition shall be established.

5.7.1.2 Clearing an Auto Contingent Allegiance Condition

An auto contingent allegiance condition shall always be cleared after a power on condition or a hard reset (see 5.7.6).

If the logical unit accepts a value of one for the NACA bit and this bit was set to one in the CDB control byte of the faulting command, then the auto contingent allegiance condition may also be cleared by means of a CLEAR ACA task management function issued by the faulting initiator as described in 6.4.

If the NACA bit is set to zero in the CDB control byte of the faulting command, then the SCSI-2 rules for clearing contingent allegiance shall apply. In this case, the logical unit shall also clear the associated contingent allegiance condition upon the return of sense data by means of the autosense mechanism described in 5.7.4.2.

While the SCSI-2 rules for clearing the ACA condition are in effect, 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 (see 6.4).

The state of all tasks in the task set when an auto contingent allegiance condition is cleared shall be modified as described in clause 7.

5.7.2 Overlapped Commands

An overlapped command occurs when an application client reuses a task address in a new command while a previous command to which that address was assigned is still pending as specified in 5.5. (The format of a task address is described in 4.7.4, object definition 7.)

Each SCSI-3 protocol standard shall specify whether or not a logical unit is required to detect overlapped commands. A logical unit that detects an overlapped command shall abort all tasks for the initiator in the task set and shall return CHECK CONDITION status for that command. If the overlapped command condition was caused by an untagged task or a tagged task with a tag value exceeding FFh, then the sense key shall be set to ABORTED COMMAND and the additional sense code shall be set to OVERLAPPED COMMANDS ATTEMPTED. Otherwise, an additional sense code of TAGGED OVERLAPPED TASKS shall be returned with the additional sense code qualifier byte set to the value of the duplicate tag.

NOTE 1 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).

NOTE 2 Some logical units may not detect an overlapped command until after the command descriptor block has been received.

5.7.3 Incorrect Logical Unit Selection

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

The logical unit identifier 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 data shall be set to the values specified for the REQUEST SENSE command above;

d) 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 the SPC standard. 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 data shall be set to the values specified for the REQUEST SENSE command above;

e) 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 the SPC standard. 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;

f) 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 the SPC standard. In response to a REQUEST SENSE command the target shall return the REQUEST SENSE data with a sense key of NO SENSE unless an auto contingent allegiance exists. The target's response to any other command is vendor specific.

5.7.4 Sense Data

Sense data shall be made available by the logical unit in the event a command completes with a CHECK CONDITION status, COMMAND TERMINATED 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, the SPC standard, the applicable device command standard and applicable SCSI-3 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.

This information may be obtained by the initiator through:

a) The REQUEST SENSE command specified in the SPC standard;
b) An asynchronous event report;
c) Autosense delivery.

The following subclauses describe the last two delivery methods.

5.7.4.1 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 specification shall provide a mechanism for asynchronous event reporting, including a procedure whereby an SCSI device can selectively enable or disable asynchronous event reports from being sent to it by a specific target. (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.

> IMPLEMENTORS NOTE: An SCSI device which can produce asynchronous event reports at initialization time should provide means to defeat these reports. This can be done with a switch or jumper wire. Devices which 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 affecting the use of asynchronous event reporting are contained in the control mode page (see the SPC standard).

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

a) an error condition was encountered after command completion;
b) a newly initialized device is available;
c) some other type of unit attention condition has occurred;
d) an asynchronous event has occurred.

An example of the first case above is 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 the second case above 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 the third case above is 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 the fourth case above is a sequential-access device performing a REWIND command with the immediate bit set to one. 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 started in the immediate mode is another example of this case.

Sense data accompanying the report identifies the condition (see 5.7.4).

An error condition or unit attention condition 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 command-related error conditions shall be sent only to the device that initiated the affected task.

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.

5.7.4.2 Autosense

Autosense is the automatic return of sense data to the application client coincident with the completion of an SCSI-3 command under the conditions described below. The return of sense data in this way is equivalent to an explicit command from the application client requesting sense data immediately after being notified that an ACA condition has occurred. Inclusion of autosense support in an SCSI-3 protocol standard is optional.

As specified in clause 5.1, the application client may request autosense service for any SCSI command. If supported by the protocol and logical unit and requested by the application client, the device server shall only return sense data in this manner coincident with the completion of a command with a status of CHECK CONDITION or COMMAND TERMINATED. The sense data shall then be cleared.

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-3 protocol or logical unit implementation. If the application client requested the return of sense data through the autosense facility and the protocol service layer does not support this feature, then the confirmation returned by the initiator's service delivery port should indicate that no sense data was returned. If the protocol service 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.

5.7.5 Unit Attention Condition

Each logical unit shall generate a unit attention condition whenever the logical unit has been reset as described in 5.7.6 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 mode parameters in effect for the initiator have been restored from non-volatile memory;
g) A change in the condition of a synchronized spindle;
h) 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 is received from an initiator to a logical unit with a pending unit attention condition (before the logical unit generates the auto contingent allegiance condition), the logical unit shall perform the INQUIRY command and shall not clear the unit attention condition.

If a request for sense data is received from an initiator with a pending unit attention condition (before the logical unit establishes the automatic contingent allegiance condition), then the logical unit shall either:

a) report any pending sense data and preserve the unit attention condition on the logical unit, or,
b) report the unit attention condition.

If the second option is chosen (reporting the unit attention condition), the logical unit may discard any pending sense data and may clear the unit attention condition for that initiator.

If the logical unit has already generated the auto contingent allegiance condition for the unit attention condition, the logical unit shall perform the second action listed above.

If an initiator issues a command other than INQUIRY or REQUEST SENSE while a unit attention condition exists for that initiator (prior to generating the auto contingent allegiance condition for the unit attention condition), the logical unit shall not perform the command and shall report CHECK CONDITION status unless a higher priority status as defined by the logical unit is also pending (see 5.3).

If a logical unit successfully sends an asynchronous event report informing the initiator of the unit attention condition, then the logical unit shall clear the unit attention condition for that initiator on the logical unit (see 5.7.4.1).

5.7.6 Hard Reset

Hard reset is a target response to a TARGET RESET task management request, (see 6.6) or a reset event within the service delivery subsystem. The definition of reset events is protocol and interconnect specific. Each SCSI-3 protocol standard shall specify the target response to a reset event including the conditions under which a hard reset shall be executed.

To execute a hard reset a target shall:

a) Abort all tasks in all task sets;
b) Clear all auto contingent allegiance conditions;
c) Release all SCSI device reservations;
d) Return any device operating modes to their appropriate initial conditions, similar to those conditions that would be found following device power-on. The MODE SELECT conditions (see the SPC standard) shall be restored to their last saved values if saved values have been established. MODE SELECT conditions for which no saved values have been established shall be returned to their default values;
e) Unit Attention condition shall be set (see 5.7.5).

In addition to the above, the target shall execute any additional functions required by the applicable protocol or interconnect specifications.

6 Task Management Functions

This clause specifies the set of SCSI-3 task management functions and the the reference model for their execution.

6.1 Remote Procedure for Task Management Functions

Task management functions provide an initiator with a way to explicitly control the execution of one or more tasks. An application client invokes a task management function by means of a procedure call having the following format:

Service Response = Function name (Object Identifier [,Input-1] [,Input-2]... ||
[Output-1] [,Output-2]...)

Service Response:

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

FUNCTION COMPLETE: A logical unit 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 target device to which the request was directed. Upon receiving a request to execute 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 or target device).

SERVICE DELIVERY OR TARGET FAILURE: The request was terminated due to a service delivery failure or target malfunction. The target 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 following task management functions are defined:

ABORT TASK (Task Address ||) – Abort the specified task. This function shall be supported if the logical unit supports tagged tasks and may be supported if the logical unit does not support tagged tasks (see object definition 7 in 4.7.4).

ABORT TASK SET (Logical Unit Identifier ||) – Abort all tasks in the task set for the requesting initiator. This function shall be supported by all logical units.

CLEAR ACA (Logical Unit Identifier ||) – Clear auto contingent allegiance condition. This function shall be supported if the logical unit accepts an NACA bit value of one in the CDB control byte and may be supported if the logical unit does not accept an NACA bit value of one in the CDB control byte (see 5.2.2).

CLEAR TASK SET (Logical Unit Identifier ||) - Abort all tasks in the specified task set. This function shall be supported by all logical units that support tagged tasks (see object definition 7) and may be supported by logical units that do not support tagged tasks.

TARGET RESET (Target Identifier ||) - Reset the target device and terminate all tasks in all task sets. All target devices shall support this function.

TERMINATE TASK (Task Address ||) - Terminate the specified task. Implementation of this function is a logical unit option.

Argument descriptions:

Target Identifier:	Target device identifier defined in 4.7.2, object definition 5.
Logical Unit Identifier:	Logical Unit identifier defined in 4.7.4, object definition 6.
Task Address:	Task address defined in 4.7.4, object definition 7.

NOTE The TARGET RESET, CLEAR TASK SET, ABORT TASK and ABORT TASK SET functions provide a means to terminate one or more tasks prior to normal completion. The TARGET RESET command clears all tasks for all initiators on all task sets of the target. The CLEAR TASK SET function terminates all tasks for all initiators on the specified task set of the target. An ABORT TASK SET function terminates all tasks for the initiator on the specified task set of the target. An ABORT TASK function terminates only the specified task.

All SCSI-3 protocol specifications shall provide the functionality needed for a task manager to implement all of the task management functions defined above.

6.2 ABORT TASK

Function Call:

Service Response = ABORT TASK (Task Address **||**)

Description:

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

The task manager shall abort the specified task if it exists. Previously established conditions, including MODE SELECT parameters, reservations, and auto contingent allegiance 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.

6.3 ABORT TASK SET

Function Call:

Service Response = ABORT TASK SET (Logical Unit Identifier **||**)

Description:

This function shall be supported by all logical units.

The task manager shall terminate all tasks in the task set that were created by the initiator.

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 terminated. Previously established conditions, including MODE SELECT parameters, reservations, and auto contingent allegiance shall not be changed by the ABORT TASK SET function.

6.4 CLEAR ACA

Function Call:

Service response = CLEAR ACA (Logical Unit Identifier **||**)

Description:

This function shall only be implemented by a logical unit that accepts an NACA bit value of one in the CDB control byte (see 5.2.2).

The initiator invokes CLEAR ACA to clear an auto contingent allegiance condition from the task set serviced by the logical unit according to the rules specified in 5.7.1.2. The function shall always be terminated with a service response of FUNCTION COMPLETE.

If the task manager clears the auto contingent allegiance condition, any task within that task set may be completed subject to the rules for task set management specified in clause 7.

6.5 CLEAR TASK SET

Function Call:

Service response = CLEAR TASK SET (Logical Unit Identifier **||**)

Description:

This function shall be supported by all logical units that support tagged tasks (see object definition 7) and may be supported by logical units that do not support tagged tasks.

The target shall perform an action equivalent to receiving a series of ABORT TASK requests from each initiator. All tasks, from all initiators, in the specified task set shall be aborted. The medium may have been altered by partially executed commands. All pending status and data for that logical unit for all initiators shall be cleared. No status shall be sent for any task. A unit attention condition shall be generated for all other initiators with tasks in that task set. When reporting the unit attention condition the additional sense code shall be set to COMMANDS CLEARED BY ANOTHER INITIATOR.

Previously established conditions, including MODE SELECT parameters (see the SPC standard), reservations, and auto contingent allegiance shall not be changed by the CLEAR TASK SET function.

6.6 TARGET RESET

Function Call:

Service Response = TARGET RESET (Target Identifier **||**)

Description:

This function shall be supported by all target devices.

Before returning a **Function Complete** response the target shall perform the hard reset functions specified in 5.7.6 and shall create a unit attention condition for all initiators as specified in 5.7.5.

6.7 TERMINATE TASK

Function Call:

Service Response = TERMINATE TASK (Task Address **||**)

Description:

Support for this function is a logical unit option.

The TERMINATE TASK function is invoked by the initiator to request task completion. A response of FUNCTION COMPLETE indicates that the request has been accepted and does not imply that the referenced task has ended. Assuming the task existed when the TERMINATE TASK function was invoked, the initiator shall consider the task to continue in existence until one of the events specified in 5.5 is detected.

With the following exceptions, the logical unit shall complete the specified task and return COMMAND TERMINATED status. The sense key shall be set to NO SENSE. The additional sense code and qualifier are set to TASK TERMINATED.

If the work performed by the terminated task involves the transfer of data, the logical unit shall set the valid bit in the sense data to one and set the information field as follows:

a) If the command descriptor block specifies an allocation length or parameter list length, the information field shall be set to the difference (residue) between the number of bytes successfully transferred and the requested length;

b) If the command descriptor block specifies a transfer length field, the information field shall be set as defined in the REQUEST SENSE command (see the SPC standard).

If an error is detected for the specified task, the logical unit shall ignore the TERMINATE TASK request and return a service response of FUNCTION COMPLETE.

If the operation requested for the specified task has been completed but status has not been returned, the logical unit shall ignore the TERMINATE TASK request and return a service response of FUNCTION COMPLETE.

If the target does not support this function or is unable to stop the task, the target shall return a service response of FUNCTION REJECTED to the initiator and continue the task in a normal manner.

The effect of a TERMINATE TASK request on the task set depends on the task set error recovery option specified in the control mode page (see the SPC standard) and on whether or not an auto contingent allegiance condition is generated.

IMPLEMENTORS NOTE: The TERMINATE TASK function provides a means for the initiator to request the logical unit to reduce the transfer length of the referenced command to the amount that has already been transferred. The initiator can use the sense data to determine the actual number of bytes or blocks that have been transferred. This function is normally used by the initiator to stop a lengthy read, write, or verify operation when a higher-priority command is available to be executed. It is up to the initiator to complete the terminated command at a later time, if required.

6.8 Task Management Protocol Services

The confirmed service described in this subclause is used by an application client to issue a task management remote procedure call. The following arguments are passed:

Object Address:	A Task Address, Logical Unit Identifier or Target Identifier supplied by the application client to identify the object to be operated upon. The initiator's service delivery port will convert a Task Address to a Task Identifier before forwarding the request to the target.
Object Identifier:	A Task Identifier, Logical Unit Identifier or Target Identifier passed to the task manager by the protocol service indication.
Function Identifier:	Parameter encoding the task management function to be performed.

All SCSI-3 protocol specifications shall define the protocol-specific requirements for implementing the Send Task Management Request protocol service and the Received Function-Executed confirmation described below. Support for the Task Management Request Received indication and Task Management Function Executed protocol service response by the SCSI-3 protocol standard is optional. All SCSI-3 I/O systems shall implement these protocols as defined in the applicable protocol specification.

The argument definitions correspond to those of clause 6.1.

Request:

Send Task Management Request (Object Address, Function Identifier ||)

Indication received by task manager:

Task Management Request Received (Object Identifier, Function Identifier ||)

Response from task manager:

Task Management Function Executed (Object Identifier, Service Response ||)

The **Service Response** parameter encodes a value representing one of the following (see 6.1):

FUNCTION REJECTED:	The task manager does not implement the requested function.
FUNCTION COMPLETE:	The requested function has been completed.

Confirmation:

Received Function-Executed (Object Address, Service Response ||)

Since the object identifier does not uniquely identify the transaction, there may be no way for an initiator to associate a confirmation with a request. An SCSI protocol that does not provide such an association should not allow an initiator to have more than one pending task management request per logical unit.

6.9 Task Management Function Example

Figure 21 shows the sequence of events associated with a task management function.

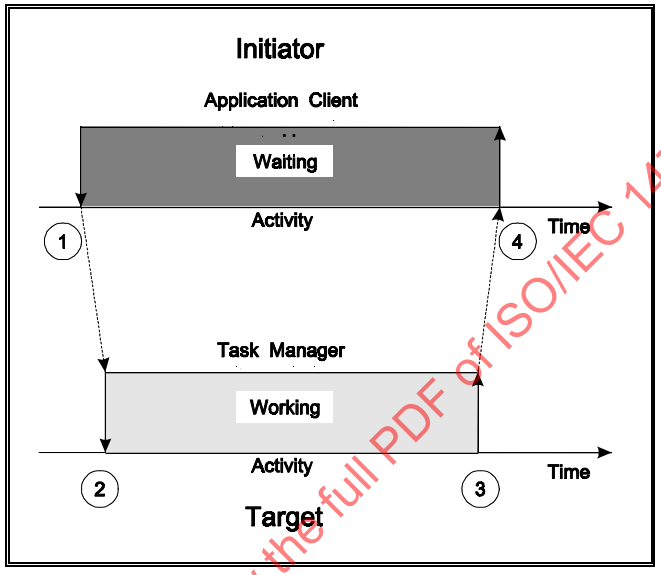

Figure 21 – Task Management Request processing

The numbers in figure 21 Identify the events described below.

1. The application client issues a task management request by invoking the **Send Task Management Request** protocol service.

2. The task manager is notified through a **Task Management Request Received** indication and begins executing the function.

3. The task manager performs the requested operation and responds by invoking the **Task Management Function Executed** protocol service to notify the application client. The **Service Response** parameter is set to a value of **Function Complete**

4. A confirmation of **Received Function-Executed** is returned to the application client.