Information technology -
SCSI Architecture Model - 2 (SAM-2)

T10 Technical Editor: Ralph O. Weber
                             ENDL Texas
                             18484 Preston Road
                             Suite 102#178
                             Dallas, TX 75252
                             USA

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

Points of Contact:

T10 Chair
John B. Lohmeyer
LSI Logic
4420 Arrows West Drive
Colorado Springs, CO 80907-3444
Tel: (719) 533-7560
Fax: (719) 533-7036
Email: lohmeyer@ix.netcom.com

T10 Vice-Chair
George O. Penokie
IBM
Deptartment 2B7
3605 Highway 52 North
Rochester, MN 55901
Tel: (507) 253-5208
Fax: (507) 253-2880
Email: gop@us.ibm.com

NCITS Secretariat
NCITS Secretariat   Telephone: 202-737-8888
1250 Eye Street, NW   Suite 200   Facsimile: 202-638-4922
Washington, DC   20005   Email: ncits@itic.nw.dc.us

T10 Web Site   www.t10.org

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

Document Distribution
Global Engineering   Telephone: 303-792-2181 or
15 Inverness Way East   800-854-7179
Englewood, CO 80112-5704   Facsimile: 303-792-2192

Revision Information

1 Approved Documents Included

The following T10 approved proposals have been incorporated SAM-2 up to and including this revision:

94-236r3	Addressability of Logical Unit For Resets
95-229r2	Proposal for Persistent Reserve
96-169r0	Proposed Changes for SAM-2
96-198r4	New Task Management Models for SAM-2
96-277r2	Proposed Change in QErr for SPC-2
97-122r4	Addressing Model for SAM -2
97-219r2	LUN discovery
97-225r4	Proposal for Contingent Allegiance / Auto Contingent Allegiance Handling
98-148	Minutes of T10 Meeting #26 — Make Terminate Task task management function obsolete
98-202	Minutes of T10 Meeting #27 — Make usage of hierarchical LUN format mandatory
98-202	Minutes of T10 Meeting #27 — Make extent reservations obsolete
98-188r2	Method for defining very long command blocks
98-241r1	Obsoleting the flag bit in the control byte
98-246r1	"May not" clarifications in SPI-3
99-144r1	Suggested changes to REPORT LUNS command for SPC-2
99-232r1	SAM-2 Tasks and Work
99-343r1	Proposal for QUEUE FULL Clarification

To the best of the technical editor's knowledge, all proposals for SAM-2 approved by T10 have been included in this revision.

2 Revision History

2.1 Revision 1 (1 September 1996, Charles Monia)

Revision 1 incorporates the following T10 approved proposals:

95-229r2	Proposal for Persistent Reserve
96-169r0	Proposed Changes for SAM-2

In addition, modify the clauses below to clarify the wording as indicated.

Clause 5.6.1.1, seventh paragraph:

Previous wording:

"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:"

Revised wording:

"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 not be entered into the faulted task set unless all of the following conditions are true:"

Clause 5.6.1.1, paragraph following list

Previous wording:

> "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."

Revised wording:

> "In any of 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. The auto contingent allegiance condition shall not be cleared."

Clause 5.2, change the wording as noted below.

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

2.2 Revision 2 (28 March 1997, Charles Monia)

Modified clause 3.7.2 to simplify the notation for objects having a numerical value.

Modified clause 3.5 to fully describe typographical conventions.

As instructed by the September 11, 1996 working group, backed out rev 01 changes in service and remote procedure call names.

Revised object definition 6 (logical unit), to include the following supplemental wording in the Task Set object description:

> "There shall be one task set per logical unit."

2.3 Revision 3 (5 May 1997, Charles Monia)

Revision 3 incorporates the following T10 approved proposals:

    94-236r3   Addressability of Logical Unit For Resets
    97-122r0   Addressing Model for SAM -2 *(not T10 approved)*

It must be noted that 97-122r0 was further revised by T10 before being approved as 97-122r4.

2.4 Revision 4 (January 1998)

Revision 4 incorporates the following T10 approved proposals:

    97-122r4   Addressing Model for SAM -2

The document has been converted to FrameMaker. The source for the conversion was the revision 3 PDF file, as taken from the T10 web site.

To facilitate the conversion, some of the boiler-plate information was taken from SPC revision 1 and revised with text from the SAM-2 PDF file to match the needs of the SAM-2 document. The Scope clause has been restructured slightly so that the new format documents roadmap can be used.

The glossary definition for "mandatory" has been removed since "mandatory" is defined as a keyword. Definitions for 'sense data", 'sense key", and 'additional sense codes" have been added.

The acronyms "SDP (Service Delivery Port)" and "SDS (Service Delivery Subsystem)" have been removed, since they are not used in the body of the working draft. Acronym definitions have been added for SCSI, SCSI-2, SCSI-3, SPC-2, and SBC. The conventions clause "References to SCSI Standards", containing similar acronym information, has been removed. The newer SPC keyword definitions have been used as the basis for keyword definitions here. A keyword definition for "invalid" has been added and edited slightly to accommodate the usage of "invalid" in clause 5 (SCSI command model).

In the state diagram example, the "E*x*:" labels have been removed and descriptions based on the transition labels have been added after the figure. This makes the example more consistent with the one and only state diagram in the working draft.

Graphical aids, such as shading, were added to several figures to make their content more clear. The SCC-2 convention of putting field names in small caps has been adopted. This presents some conflicts with the existing usage of small caps to represent "undefined" names. Every effort has been made to follow the SCC-2 convention rigorously, and several field names have been changed from nondescript lower case to small caps.

In revision 3, an attempt was made to incorporate a pre-approval draft of a proposal describing hierarchical addressing in the logical unit number value (97-122r0). Four more revisions of the proposal were generated before T10 approved 97-122r4 for inclusion in SAM-2. 97-122r0 lacked clarity in several areas, which motivated substantial embellishment on the content of the proposal text in SAM-2 revision 3. After SAM-2 revision 3 was distributed, T10 corrected the omissions and added the needed clarity. However, several of the T10 approved changes conflict with the embellishments found in SAM-2 revision 3. In order to avoid replicating misleading work, 97-122r4 has been incorporated in revision 4. This violates the principle of a "conversion only" working draft revision for this change of technical editors and document processing software. A "conversion only" revision would have been much preferred, however the duplication of clearly incorrect information and the wasted work argued against a pure conversion working draft. The technical editor makes his apologies here.

In response to a decision of the March 1998 SCSI General Working Group meeting (minutes in 98-126), the definition of the 'reserved' keyword has been changed to match the definition found in SBC.

2.5 Revision 5 (13 April 1998)

Rather than starting any of the work described in the "Plans for Future Revisions", the technical editor elected to incorporate all approved documents in revision 5. Also, housekeeping was performed on the revision 4 material, specifically: change bars were removed, text intended for deletion (shown by strike throughs) was removed, and any editor's notes describing changes made between revision 3 and revision 4 were removed.

Revision 5 incorporates the following T10 approved proposals:

    96-198r4   New Task Management Models for SAM-2
    97-219r2   LUN discovery

During incorporation of 97-219r2 it was discovered that the object descriptions for object definition 5 (Target) did not contain the same rigorous cross references as the other object definitions, and suitable cross references were added. It must also be noted that the requirement that Logical Unit 0 be present is not well articulated in this revision of SAM-2. The object notation does not include a way of expressing the thought that one instance of an object must take on a specific object property (which is the notation needed in this case). The technical editor does not wish to invent the needed notation because the goal is to eliminate the object notation completely from SAM-2.

In revision 4, editor's notes 9 and 10 were incorrect in their description of the problem, and thus the chosen resolution also was incorrect. There is a definition for Task Address, it occurs in object definition 10 not in object definition 7. Thus the problem should be resolved by correcting the object definition cross reference, not by changing Task Address to Tag. Revision 5 corrects the problems introduced in revision 4.

The definitions (clause 3 additions and changes) 97-225r2 proposed for SPC-2 are more complete than those proposed for SAM-2, so the proposed SPC-2 definitions have been incorporated with slight modifications. The non-proposal questions raised by 97-225r2 have either been addressed editorially or included as editor's notes.

Because of the increasing number of references to Control mode page and standard inquiry data fields, glossary definitions have been added to help the first-time reader comprehend the terminology associated with such references.

2.6 Revision 5a (30 April 1998)

Revision 5a incorporates the following T10 approved proposals:

    96-277r2   Proposed Change in QErr for SPC-2

An error was reported in the changed Object Definition 5 and expeditious correction seemed appropriate. It was also noted that 96-277 changes QErr from a bit to a field. While T10 did not view 96-227 as affecting SAM-2, it actually does and the necessary changes appear in revision 5a. It also is interesting to note that 96-277r2 requires the removal of QErr where ever it appears in 96-184r4. While that change might have been thought of as affecting SAM-2, it does not.

Revision 5a contains all change bars found in revision 5 and any necessary new change bars.

2.7 Revision 6 (14 May 1998)

Revision 6 incorporates the following T10 approved proposals:

    98-148   Minutes of T10 Meeting #26 — Make Terminate Task task management function obsolete

Incorporation of the decision to make the Terminate Task task management function obsolete involves mostly deletions. To highlight the deletions, they are shown with strike-throughs in revision 6 and the actual removal will occur in revision 7. The only exception concerns the status code value definition for COMMAND TERMINATED, where the code is shown as obsolete with no strike-through text.

Also, the May 1998 SCSI Working Group held discussed all the editor's notes found in revisions 4, 5, and 5a (minutes in 98-147). Changes agreed during those discussion have been added to revision 6. In a few cases, wording was left to the editor's discretion. Document 98-173r0 has been produced to present the details of such changes to the T10 membership.

2.8 Revision 7 (3 July 1998)

Revision 7 is devoted exclusively to converting the object notation definitions to English text. To accomplish the change to English text, the clause containing the model for logical units was split to produce separate clauses for the logical unit model, the task model, and separately identified clauses for the task identification models. The model definition of the Initiator Identifier object was moved to the initiator model clause. The editor tried to avoid gratuitous text additions. However, that temptation could not be resisted for statements that the multiple identifiers for one initiator or target cannot be associated based solely on value. Text was added to clarify the ways that Task Identifiers are made unique as required in clause 4.9. Text was added describing the differences between tagged

and untagged tasks. All changes are marked with change bars. No T10 approved proposals or other changes have been incorporated in revision 7.

2.9 Revision 8 (23 July 1998)

Revision 8 incorporates the following T10 approved proposals:

98-202   Minutes of T10 Meeting #27 — Make usage of hierarchical LUN format mandatory
98-202   Minutes of T10 Meeting #27 — Make extent reservations obsolete

In clauses 3.6 and 4.1 through 4.9, make changes agreed by the SAM-2 review meeting. These clauses were the ones most affected by the conversion from object notation to English. The review meeting noted and corrected several errors and problems in these clauses. All corrections appear in revision 8, with change bars.

Removed clause 7.8.5 (Deferred task completion) as discussed in the SCSI Working Group (minutes in 98-201), since a similar example cannot be constructed in the absence of the Terminate Task task management function. Incorporated editorial comments on revision 6 received from Gene Milligan via the T10 Reflector. Editorial corrections received from Arlan Stone were made in clause 1.2 (marked with change bars).

2.10 Revision 9 (10 September 1998)

During the review of revision 7 changes at the July 1998 meetings, many committee members expressed concerns about the Clause 4 representation of SCSI Devices with multiple Service Delivery Ports. It was widely felt that 4.4 through 4.7 failed to convey the very important idea that each port is a unique SCSI device. Revision 9 is a first attempt to address these concerns. With the exception of some editorial change in 3.6.1 (notational conventions for model hierarchy diagrams) and the addition of a definition for SCSI Multi-port Device, all the revision 9 changes can be found in 4.4 through 4.11.

2.11 Revision 10 (13 March 1999)

Revision 10 incorporates the following T10 approved proposals:

98-188r2   Method for defining very long command blocks
98-241r1   Obsoleting the flag bit in the control byte
98-246r1   "May not" clarifications in SPI-3

In addition to incorporation of the above documents, revision 10 includes changes noted during the September T10 meetings not related to multi-port issues. Most uses of 'SCSI-3' have been replaced with 'SCSI', and 'SPC' with 'SPC-2'. Following an editorial change in SPC-2, 'HiSupport' has been changed to 'HiSup'. HiSup is the name of a bit in the standard Inquiry data. References to Search Data commands have been marked for deletion because those commands have been made obsolete in SBC.

The ANSI patent statements have been updated based on comments received regarding equivalent paragraphs in SPC-2. The definition of the 'reserved' keyword has been changed to match SPI-3 and SPC-2. The list of standards in the SCSI Family has been updated. This will be the last revision in which changes in the list of standards are marked with change bars. In the 'Hierarchies of Dependent Logical Units' clause, clarified the phrase, "… or may be defined by an application client by use of configuration commands" by changing it to "… or may be defined by an application client (e.g., by the use of SCC-2 configuration commands)."

Most uses of 'SCSI-3' have been replaced with 'SCSI'. The SCSI working group requested this change in SPC-2 with the intent that it would be applied in SAM-2 too. The following note from the 'Substantial Changes' clause

applies to this action. The proposed SCSI definition change has been implemented in SAM-2 and SPC-2 revision 9.

> The use of "SCSI-3" might be seen as conflicting with the document's title "SCSI Architecture Model -2". One solution would be to change to "SCSI" in the Foreword, Introduction, and Scope clauses. However, making such a change also requires that the definition of SCSI be changed to exclude SCSI-2 and SCSI-1, a change that may not conform with the wishes of T10.

All changes are marked with change bars, except editorial changes in figures.

Revision 10 makes no effort to address the multi-port device issues raised by T10. On a best-effort basis, editor's notes have been added to indicated where information is known to be subject to revision based on unprocessed multi-port comments.

2.12 Revision 11 (16 July 1999)

Revision 11 incorporates the following T10 approved proposals:

> 99-144r1   Suggested changes to REPORT LUNS command for SPC-2

Revision 11 also includes the technical editor's attempts to address the multi-port device issues raised during the September 1998 General SCSI Working Group meeting. All notes taken at that meeting have been reviewed and addressed. The object name assigned to a multi-port device is SCSI Multi-port Unit (or SMU). Glossary and abbreviations definitions have been added for SMU. With only one or two exceptions, the changes made to address the multi-port device issues appear in 4.10 and include the addition of several sub-clauses under 4.10. These pages were distributed at the July 1999 T10 meetings marked as revision 10a. The results of the review of the distributed material by the July 1999 T10 meetings has been incorporated in revision 11.

Revision 11 also includes several grammatical changes from old notes. Changes appear in the definitions of 'confirmed protocol service', 'task', 'third-party command', and 'unconfirmed protocol service'. The object definition for 'Interconnect Subsystem' was enhanced to include the transfer of 'requests and responses' in addition to the previously defined 'data'.

A few corrections were made in revision 11 as a result of e-mail messages received from people reading SAM-2. The LINK bit was returned to bit 0 of the control byte. The error locating the LINK bit in bit 1 was introduced in conversion from WordPerfect to FrameMaker in revision 4. A clause cross reference for Target Identifier was corrected in clause 6.

2.13 Revision 12 (17 September 1999)

Revision 12 incorporates the following T10 approved proposals:

> 99-232r1   SAM-2 Tasks and Work

In response to comments from the working group during the discussion of 99-232r1, the phrase "without regard for …" has been removed throughout. At the request of the working group, editorial changes the Logical Unit Reset clause was modified to clarify that persistent reservations are not cleared by a logical unit reset.

Revision 12 also corrected the reference to the T10 web site and removes the SCSI BBS from the cover page. Several uses of 'multi-port SCSI unit' were corrected to 'SCSI multi-port unit'. T10 approved 98-188r2 (incorporated in revision 10) provided for CDBs that are longer than 16 bytes, but failed to identify three locations in SAM-2 that need to except the very long CDB format. These are corrected in revision 12. The corrections to the link bit

location (bit 0 not bit 1 in the control byte) made in revision 11 were not complete. A paragraph later in the clause describes the obsolete bit as 'bit 0', which revision 12 corrects to read 'bit 1'.

2.14 Revision 13 (22 March 2000)

Revision 13 incorporates the following T10 approved proposals:

99-343r1   Proposal for QUEUE FULL Clarification

Duplicated the following sentence: "A contingent allegiance (NACA=0) shall be cleared by the xxx function" from the CLEAR TASK SET description to the ABORT TASK SET description as an editorial change according to the recommendation of the SCSI Protocol Working Group (minutes in 99-309r0). Swapped the locations of the paragraphs before and after the first figure in 5.3.1 (Figure 29 - Model for buffered data transfers) to improve the readability of the clause. Updated the editor's contact information on the cover page.

3 Plans for Future Revisions

This is a list of the work the technical editor considers required in future revisions of SAM-x. The list is not complete as of revision 12. Based on schedule considerations for SAM-2, these changes probably will be deferred to SAM-3.

3.1 Minor Changes

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

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

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

The technical editor wishes to remove the definition of "ended command". Strictly speaking, the term "ended command" is not used anywhere in the working draft, thus allowing removal of the definition as a strictly editorial change. However, some consideration of the change appears prudent. The word "ended" is used frequently in the working draft, all uses appear to have the standard English meaning, but this thesis needs additional verification. Also, there is an "ended (task state)" that lacks a glossary definition and perhaps should have one.

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

Should the glossary definition for "pending task" really be for "pending (task state)"?

Is the term "protocol service request" really so general as to require its being defined in terms of "call" (an architectural abstraction)? Also, the technical editor believes that "protocol service response" should be defined as "A reply

to the upper level protocol …", not "A reply from the upper level protocol …" Finally, would it be possible to cast both definitions in terms of specific entities from the SCSI roadmap, instead of "lower level protocol" and "upper level protocol" (see 3.2)?

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

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

It certainly would be nice to avoid defining "task" in terms of architectural abstractions (e.g., "object"). Perhaps, the word "entity" could be used in its English meaning?

The technical editor cannot help wondering if there is a way to eliminate the "task slot" definition. It seems to overly restrict (or define) a target implementation.

In the definition of "third-party command", "an SCSI command" should be replaced by the more nebulous "SCSI commands". There is not a one-to-one relationship between an SCSI command sent to a logical unit and the number of third-party commands the logical unit issues to complete it. (Done in revision 11.)

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

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

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

Are the requirements on protocols really contained only in 5.3 and 6.7, as is stated in 4.1?

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

Clause 5.3 is a mess. The data delivery services are given individual 5.3.x clauses but the command protocol services are not. 5.3.1 fails to identify the party responsible for establishing the parameters for the transfer of a buffer segment. The rule prohibiting input and output transfers by a single command is buried in a paragraph that starts with a discussion of buffer segmentation. The title on figure 29 "Model for buffered data transfers" suggests buffered data versus programed I/O data operation, which is not the intent. The technical editor was very tempted to rewrite the whole clause during the revision 4 conversion, but wrote this reminder to himself instead.

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

The technical editor believes that persistent reservations should be excluded from the statement in 5.6.7 bullet c. Comments from T10?

3.2 Substantial Changes

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

  a)   The Foreword and Introduction clauses need to be modified to remove the work "requirements"; at the time of this writing, "capabilities" is the preferred replacement;

    b)   Most of 1.1 probably would be obsolete; and

    c)   A careful audit of the requirements statements will be needed to adjust those placing requirements on other standards.

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

    a)   Overly general layering terms and discussions; and

    b)   Discussion of a new application client for each new request or task management function.

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

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

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

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

It is almost impossible for the average reader to absorb and apply the following 4.1 statement:

> "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."

This statement should go and the places where abstraction is not related to implementation should be clearly identified at the sites where they occur. A revision of SAM-2 should be devoted solely to this change.

Is the following 4.2 statement rigorously true?

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

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

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

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

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

The technical editor suspects that "confirmed service" has multiple definitions.

Draft

**American National Standards
for Information Systems -**

SCSI Architecture Model - 2 (SAM-2)

Secretariat
National Committee for Information Technology Standards

Approved mm dd yy

American National Standards Institute, Inc.

Abstract

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

Draft

American National Standard

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

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

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

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

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

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

Printed in the United States of America

Draft

Contents

Tables

Figures

Foreword

This foreword is not part of American National Standard NCITS.***:199x.

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

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

These Bulletins, while reflecting the opinion of the Technical Committee that developed the standard, are intended solely as supplementary information to other users of the standard. This standard, ANSI NCITS.***:199x, as approved through the publication and voting procedures of the American National Standards Institute, is not altered by these bulletins. Any subsequent revision to this standard may or may not reflect the contents of these Technical Information Bulletins.

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

Global Engineering   Telephone: 303-792-2181 or
15 Inverness Way East   800-854-7179
Englewood, CO   80112-5704   Facsimile: 303-792-2192

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

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

<<Insert NCITS member list>>

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

<<Insert T10 member list>>

Introduction

The SCSI Architecture Model (SAM-2) standard is divided into seven clauses:

Clause 1 is the scope.

Clause 2 enumerates the normative references that apply to this standard.

Clause 3 describes the definitions, symbols, and abbreviations used in this standard.

Clause 4 describes the overall SCSI architectural model

Clause 5 describes the SCSI command model element of the SCSI architecture

Clause 6 describes the task management functions common to SCSI devices

Clause 7 describes the task set management capabilities common to SCSI devices

American National Standard   **NCITS.***:199x**

American National Standard for Information Systems -
Information Technology -
SCSI Architecture Model - 2 (SAM-2)

1 Scope

The set of SCSI standards consists of the SCSI Architecture Model - 2 (this standard) and the SCSI implementation standards described in 1.1. This standard defines a reference model that specifies common behaviors for SCSI devices, and an abstract structure that is generic to all SCSI I/O system implementations.

1.1 Requirements precedence

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

Generic requirements are transformed to implementation requirements by an implementation standard. An example of a generic requirement is the target hard reset behavior specified in 5.6.6.
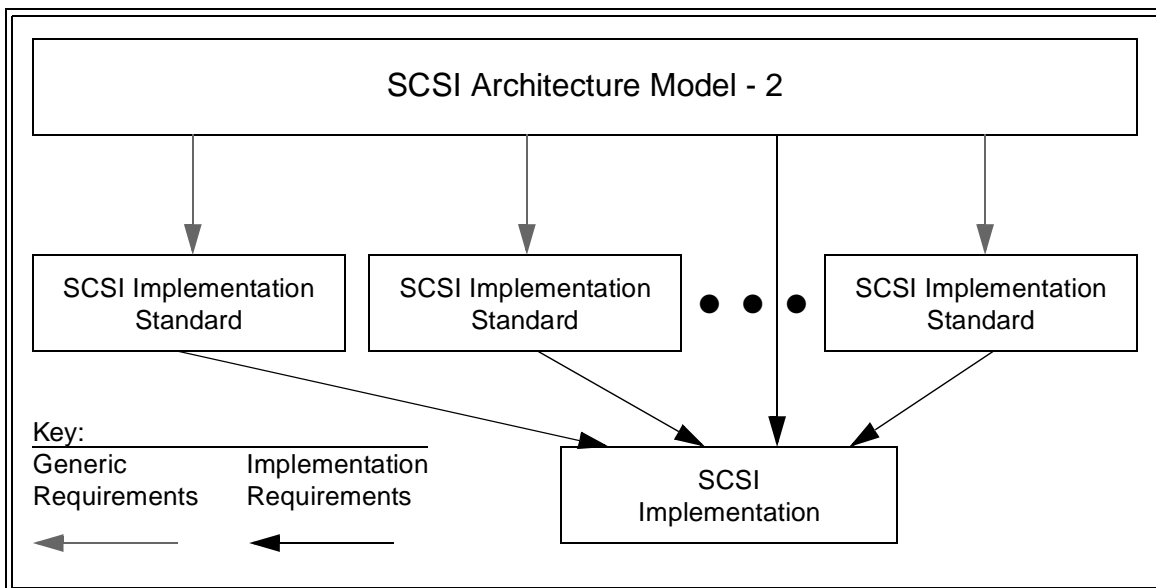
Figure 1 — Requirements precedence

As shown in figure 1, all SCSI implementation standards shall reflect the generic requirements defined herein. In addition, an implementation claiming SCSI compliance shall conform to the applicable implementation requirements defined in this standard and the appropriate SCSI implementation standards. In the event of a conflict between this document and other SCSI standards under the jurisdiction of technical committee T10, the requirements of this standard shall apply.

1.2 SCSI standards family

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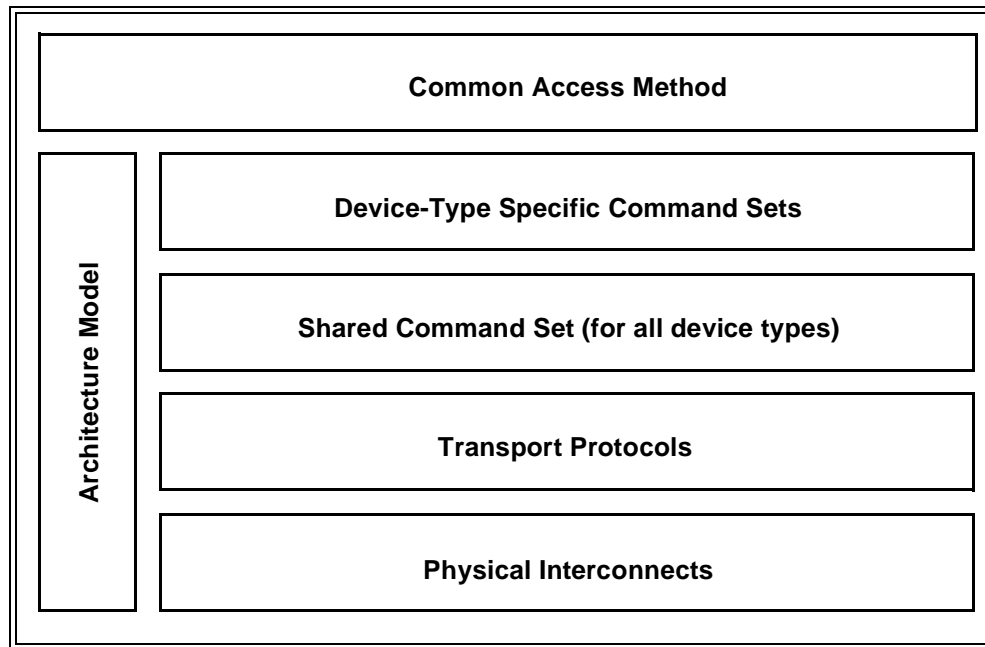
Figure 2 — SCSI document roadmap

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

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

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

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

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

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

Transport Protocols: Implementation standards that define the rules for exchanging information so that different SCSI devices can communicate.

Physical Interconnects: Implementation standards that define the electrical and signaling rules essential for devices to interoperate over a given physical interconnect.

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

Physical Interconnects:
Fibre Channel Arbitrated Loop	FC-AL	[ANSI X3.272:1996]
Fibre Channel Physical and Signalling Interface	FC-PH	[ANSI X3.230:1994]
Fiber Channel Physical Amendment 1		[ANSI X3.230/AM1:1996]
Fibre Channel 3rd Generation Physical Interface	FC-PH-3	[ANSI X3.303-1998]
Fibre Channel Framing and Signaling Interface	FC-FS	[T11/1331-D]
High Performance Serial Bus		[ANSI IEEE 1394:1995]
SCSI Parallel Interface - 2	SPI-2	[ANSI X3.302:1999]
SCSI Parallel Interface - 3	SPI-3	[ANSI NCITS.336:200x]
SCSI Parallel Interface - 4	SPI-4	[T10/1365-D]
Serial Storage Architecture Physical Layer 1	SSA-PH	[ANSI X3.293:1996]
Serial Storage Architecture Physical Layer 2	SSA-PH-2	[ANSI NCITS.307:1998]

Transport Protocols:
Serial Storage Architecture Transport Layer 1	SSA-TL-1	[ANSI X3.295:1996]
Serial Storage Architecture Transport Layer 2	SSA-TL-2	[ANSI NCITS.308:1998]
SCSI-3 Interlocked Protocol	SIP	[ANSI X3.292:1997]
SCSI-3 Fibre Channel Protocol	FCP	[ANSI X3.269:1996]
SCSI-3 Fibre Channel Protocol - 2	FCP-2	[T10/1144-D]
Serial Bus Protocol - 2	SBP-2	[ANSI NCITS.325:1999]
Serial Storage Architecture SCSI-2 Protocol	SSA-S2P	[ANSI X3.294:1996]
Serial Storage Architecture SCSI-3 Protocol	SSA-S3P	[ANSI NCITS.309:1998]
SCSI on Scheduled Transfer	SST	[T10/1380-D]
SCSI VI Protocol	SVP	[T10/1415-D]

Shared Command Sets:
SCSI-3 Primary Commands	SPC	[ANSI X3.301:1997]
SCSI Primary Commands - 2	SPC-2	[T10/1236-D]
SCSI Primary Commands - 3	SPC-3	[T10/1416-D]

Device-Type Specific Command Sets:
SCSI-3 Block Commands	SBC	[ANSI NCITS.306:1998]
SCSI Block Commands - 2	SBC-2	[T10/1417-D]
SCSI-3 Stream Commands	SSC	[ANSI NCITS.335:200x]
SCSI-3 Medium Changer Commands	SMC	[ANSI NCITS.314:1998]
SCSI Medium Changer Commands - 2	SMC-2	[T10/1383-D]
SCSI-3 Multimedia Command Set	MMC	[ANSI X3.304:1997]
SCSI Multimedia Command Set - 2	MMC-2	[ANSI NCITS.333:200x]
SCSI Multimedia Command Set - 3	MMC-3	[T10/1363-D]
SCSI-3 Controller Commands	SCC	[ANSI X3.276:1997]
SCSI Controller Commands - 2	SCC-2	[ANSI NCITS.318:1998]
SCSI Reduced Block Commands	RBC	[ANSI NCITS.330:200x]
SCSI Reduced MultiMedia Commands	RMC	[T10/1364-D]
SCSI-3 Enclosure Services Commands	SES	[ANSI NCITS.305:1998]
SCSI Specification for Optical Card Reader/Writer	OCRW	[ISO/IEC 14776-381]

Architecture Model:
SCSI-3 Architecture Model	SAM	[ANSI X3.270:1996]
SCSI Architecture Model - 2	SAM-2	[T10/1157-D]

Common Access Method:

SCSI Common Access Method	CAM	[ANSI X3.232:1996]
SCSI Common Access Method - 3	CAM-3	[T10/990-D]

The term SCSI is used to refer to the family of standards described in this clause. The Small Computer System Interface - 2 standard (ANSI X3.131-1994) and the architecture that it describes are referred to herein as SCSI-2.

2 Normative references

2.1 Document and draft document availability information

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

At the time of publication, X3 practice was to make working draft standards and draft proposed American National Standards available through Global Engineering at 800-854-7179 (toll free phone), 303-792-2181 (phone) or 303-792-2192 (fax).

2.2 Normative approved references for mandatory features

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

- SCSI-2 Small Computer System Interface   SCSI-2   ANSI X3.131 - 1994

2.3 Normative approved references for optional features

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

- SCSI-2 Small Computer System Interface   SCSI-2   ANSI X3.131 - 1994

3 Definitions, symbols, abbreviations, and conventions

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, 3.1.5 and 4.9).

3.1.3 additional sense code: A field in the sense data. See 3.1.84 and SPC-2.

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

3.1.5 auto contingent allegiance: One of the conditions of a task set following the return of a CHECK CONDITION status. (See 5.6.1.)

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

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

3.1.8 byte: An 8-bit construct.

3.1.9 call: The act of invoking a procedure.

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

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

3.1.12 code value: A one or a series of defined numeric values each representing an identified and described instance or condition. Code values are defined to be used in a specific field (see 3.1.36), in a procedure input data object (see 3.6.2), in a procedure output data object, or in a procedure result.

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

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

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

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

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

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

3.1.19 contingent allegiance: One of the conditions of a task set following the return of a CHECK CONDITION status. A detailed definition of CA may be found in SCSI-2.

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

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

3.1.22 dependent logical unit: a logical unit that is addressed via some other logical unit(s) in a hierarchical logical unit structure (see 3.1.39), also a logical unit that is at a higher numbered level in the hierarchy than the referenced logical unit (see 4.10.4).

3.1.23 destination device: The SCSI device to which a service delivery transaction is addressed. See source device (3.1.93).

3.1.24 Device identifier: Equivalent to SCSI device identifier (see 3.1.79).

3.1.25 device server: An object within the logical unit which executes SCSI tasks according to the rules for task management described in clause 7.

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

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

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

3.1.29 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.

3.1.30 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.

3.1.31 ended command: A command that has completed or aborted.

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

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

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

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

3.1.36 field: A group of one or more contiguous bits, part of a larger structure such as a CDB (see 3.1.14) or sense data (see 3.1.84).

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

3.1.38 hard reset: A target response to a reset event or a TARGET RESET task management function in which the target performs the operations described in 5.6.6.

3.1.39 hierarchical logical unit: an inverted tree structure for forming and parsing logical unit numbers (see 3.1.52) containing up to four addressable levels (see 4.10.4).

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

3.1.41 implementation: The physical realization of an object.

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

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

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

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

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

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

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

3.1.49 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.

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

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

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

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

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

3.1.55 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.

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

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

3.1.58 peer entities: Entities within the same layer.

3.1.59 pending task: A task that is not a current task.

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

3.1.61 port: Synonymous with "service delivery port" (see 3.1.89).

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

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

3.1.64 protocol option: An option whose definition within an SCSI protocol standard is discretionary.

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

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

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

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

3.1.69 queue: The arrangement of tasks within a task set, usually according to the temporal order in which they were created. See "task set" (3.1.106).

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

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

3.1.72 request: A transaction invoking a service.

3.1.73 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.

3.1.74 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.

3.1.75 reset event: A protocol-specific event which may trigger a hard reset response from an SCSI device as described in 5.6.6.

3.1.76 response: A transaction conveying the result of a request.

3.1.77 SCSI application layer: The protocols and procedures that implement or invoke SCSI commands and task management functions by using services provided by an SCSI protocol layer.

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

3.1.79 SCSI device identifier: An address by which an SCSI device is referenced within a domain. Depending on the device model is use, the SCSI device identifier is either an Initiator Identifier or a Target Identifier (see 4.7.3).

3.1.80 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.

3.1.81 SCSI multi-port unit: A device that has multiple service delivery ports (see 3.1.89) or responds to multiple SCSI device identifiers (see 3.1.79). A description of SCSI multi-port units can be found in 4.10.

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

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

3.1.84 sense data: Data returned to an application client as a result of an autonsense operation, asynchronous event report, or REQUEST SENSE command (see 5.6.4). Fields in the sense data are referenced by name in this standard and the complete sense data format definition can be found in SPC-2.

3.1.85 sense key: A field in the sense data. See 3.1.84 and SPC-2.

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

3.1.87 service: Any operation or function performed by an SCSI object, which can be invoked by other SCSI objects.

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

3.1.89 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" (3.1.61).

3.1.90 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.

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

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

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

3.1.94 standard inquiry data: Data returned to an application client as a result of an INQUIRY command. Fields in the standard inquiry data are referenced by name in this standard and the complete standard inquiry data format definition can be found in SPC-2.

3.1.95 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.

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

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

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

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

Editors Note 1 - ROW: Are the next two definitions used?

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

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

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

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

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

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

3.1.106 task set: A group of tasks within a logical unit, whose interaction is dependent on the task management (queuing), CA, and ACA rules. (See 4.8.)

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

3.1.108 third-party command: An SCSI command which requires a logical unit within the target device to assume the initiator role and send SCSI command(s) to a target device.

3.1.109 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.

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

3.1.111 unlinked command: An SCSI command having the LINK bit set to zero in the CDB CONTROL byte.

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

3.2 Acronyms

ACA   Auto Contingent Allegiance (see 3.1.5)
AER   Asynchronous Event Reporting

CA   Contingent Allegiance (see 3.1.13 and SCSI-2)
CAM   Common Access Method (see 1.2)
CDB   Command Descriptor Block (see 3.1.14)
LLP   Lower Level Protocol (see 3.1.54)
LUN   Logical Unit Number (see 3.1.52)
SBC   SCSI-3 Block Commands (see 1.2)
SCSI   The architecture defined by the family of standards described in 1.2
SCSI-2   The architecture defined by the Small Computer System Interface - 2 standard (ANSI X3.131-1994)
SIM   SCSI Interface Module (a component of CAM software, see CAM)
SPC-2   SCSI Primary Commands -2 (see 1.2)
SMU   SCSI Multi-port Unit (see 3.1.81)
ULP   Upper Level Protocol (see 3.1.112)

3.3 Keywords

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

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

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

3.3.4 may: A keyword that indicated flexibility of choice with no implied preference (equivalent to "may or may not").

3.3.5 may not: A keyword that indicated flexibility of choice with no implied preference (equivalent to "may or may not").

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

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

3.3.8 protocol-specific: Implementation of the referenced item is defined by a transport protocol standard (see 1.2).

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

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

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

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

3.4 Editorial Conventions

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

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

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

Quantities having a defined numeric value are identified by large capital letters. CHECK CONDITION, for example, refers to the numeric quantity defined in table 12. Quantities having a discrete but unspecified value are identified using small capital letters. As an example, LINKED COMMAND COMPLETE, indicates a quantity returned by the **Execute Command** procedure call (see clause 5). Such quantities are usually associated with an event or indication whose observable behavior or value is specific to a given implementation standard.

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

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

Notes do not constitute any requirements for implementors.

3.5 Numeric Conventions

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

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

3.6 Notation Conventions

3.6.1 Hierarchy diagram conventions

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

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
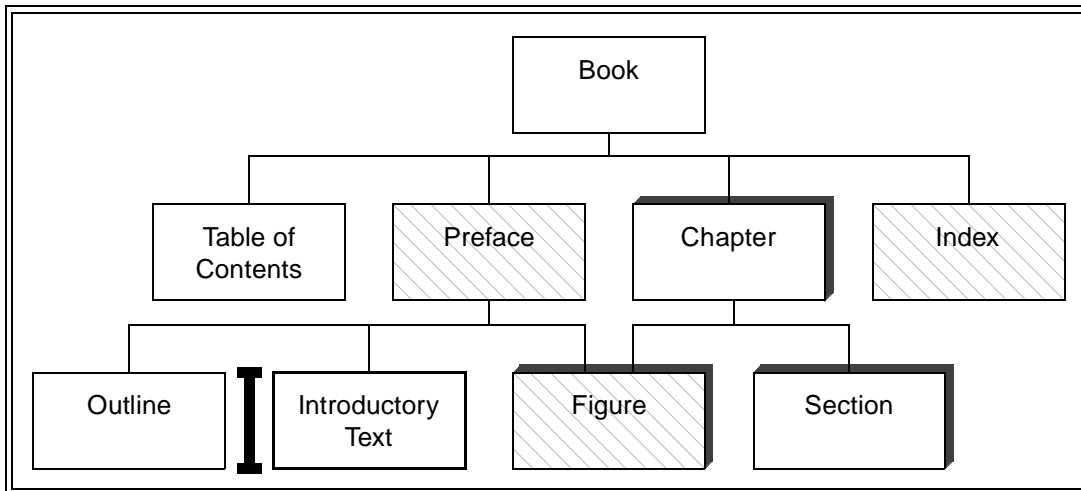
Figure 3 — Example hierarchy diagram

between the "Book" object and its constituent objects "Table of Contents", "Preface", "Chapter" and "Index". Similarly, connecting lines show that a "Chapter" object contains the objects "Section" and "Figure". Note that the "Figure" object also may be a component of the "Preface" object. The I-beam symbol denotes the relationship where an object contains either one or the other or both of a pair of other objects, as in a "Preface" containing either an "Outline" or "Introductory Text" or both an "Outline" and "Introductory Text".

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

3.6.2 Notation for procedures and functions

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

   [Result =] Procedure Name ([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.

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

Output Arguments:

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

3.6.3 Notation for state diagrams

All state diagrams use the notation shown in figure 4.
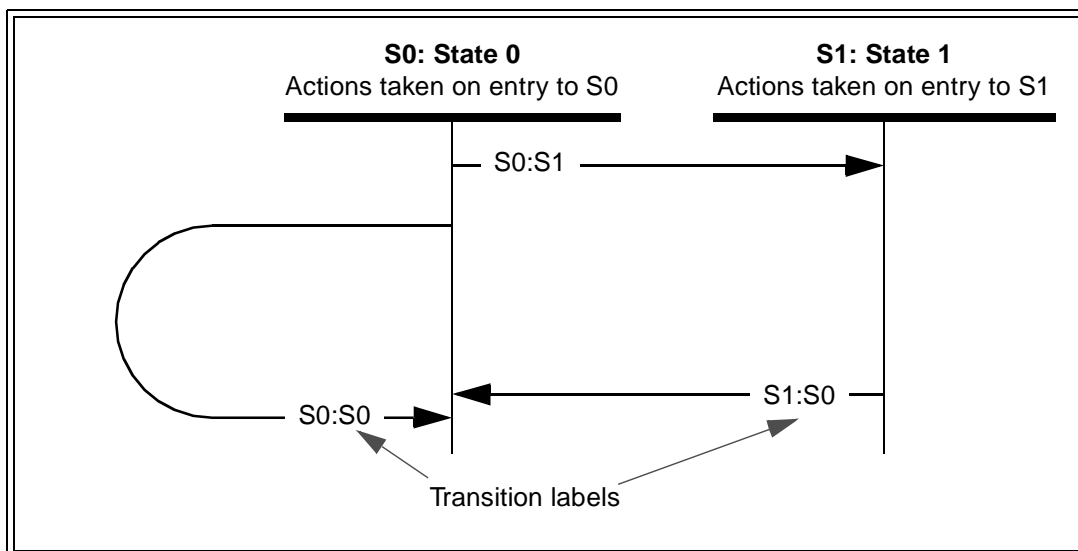
Figure 4 — Example state diagram

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

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

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

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

A system specified in this manner has the following properties:

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

4 SCSI Architecture Model

4.1 Introduction

The purpose of the SCSI architecture model is to:

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

The development of this standard is assisted by the use of an abstract model. To specify the external behavior of a real SCSI 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 devices in any SCSI 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 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 I/O system constitute an abstraction despite a similar appearance to concepts possibly found in real systems. Therefore, a real SCSI 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 architecture model is described in terms of objects (see 3.1.56), protocol layers and service interfaces between objects. As used in this standard, objects are abstractions, encapsulating a set of related functions, data types, and other objects. Certain objects, such as an interconnect, are defined by SCSI, while others, such as a task, are needed to understand the functioning of SCSI but have implementation definitions outside the scope of SCSI. That is, although such objects exhibit well-defined, observable behaviors, they do not exist as separate physical elements. An object may be a single numeric parameter, such as a logical unit number, or a complex entity that performs a set of operations or services on behalf of another object.

Service interfaces are defined between distributed objects and protocol layers. The template for a distributed service interface is the client-server model described in 4.2. Clause 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 clauses 5 and 6.

Requirements that apply to each SCSI protocol standard are specified in the protocol service model described in 5.3 and 6.7. The model describes required behavior in terms of layers, objects within layers and protocol service transactions between layers.

4.2 The SCSI distributed service model

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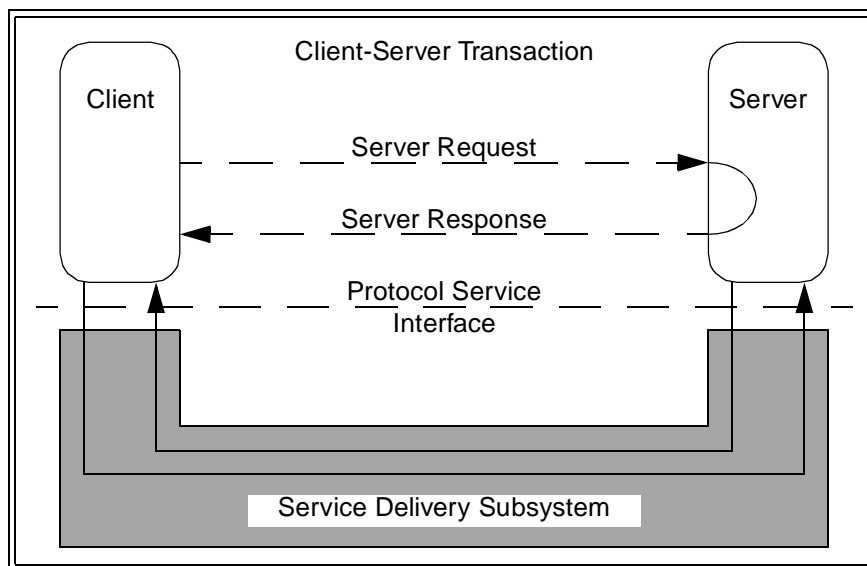
Figure 5 — Client-Server model

A client-server transaction is represented as a remote procedure call with inputs supplied by the caller (the client). The procedure 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 sent. As seen by the server, the request becomes pending upon receipt and completes when the response is passed to its service delivery subsystem for return to the client. As a result there 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.

4.3 The SCSI client-server model

As shown in figure 6, each SCSI 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 command standards and executes SCSI commands such as reading from or writing to the media. Each pending SCSI command or series of linked commands defines a unit of work to be performed by the logical unit. 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.
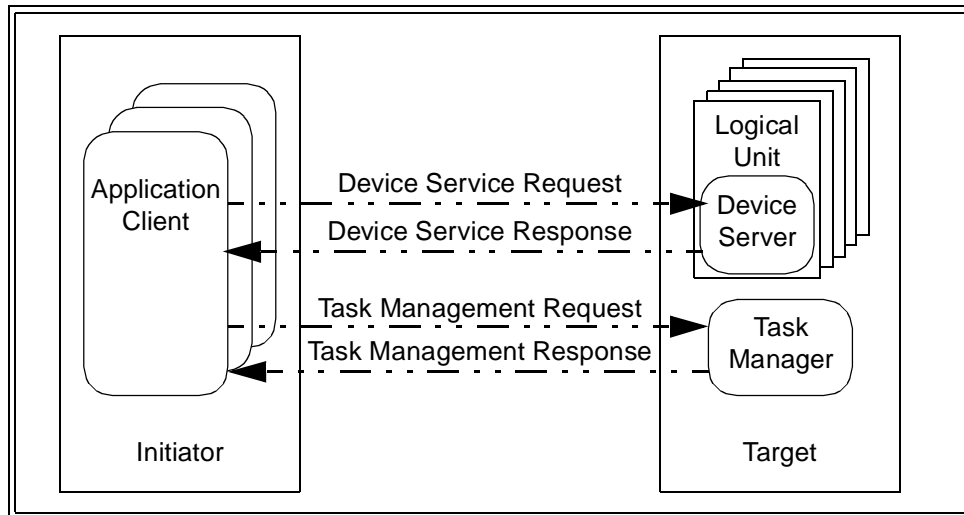
Figure 6 — SCSI client-server model

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 protocol. In an implementation, that thread could correspond to the device driver and any other code within the operating system that is capable of managing I/O requests without requiring knowledge of the interconnect or SCSI protocol. In the architecture model, an application client is created to issue a single SCSI command or task management function; 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 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. Clause 5.5.1 gives an example of the processing for a single command. Clause 5.5.2 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. Clause 6.8 shows the interactions between the task manager and application client when a task management request is processed.

4.4 The SCSI structural model

The SCSI structural model represents a view of the elements comprising an SCSI I/O system as seen by the 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 7, 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.
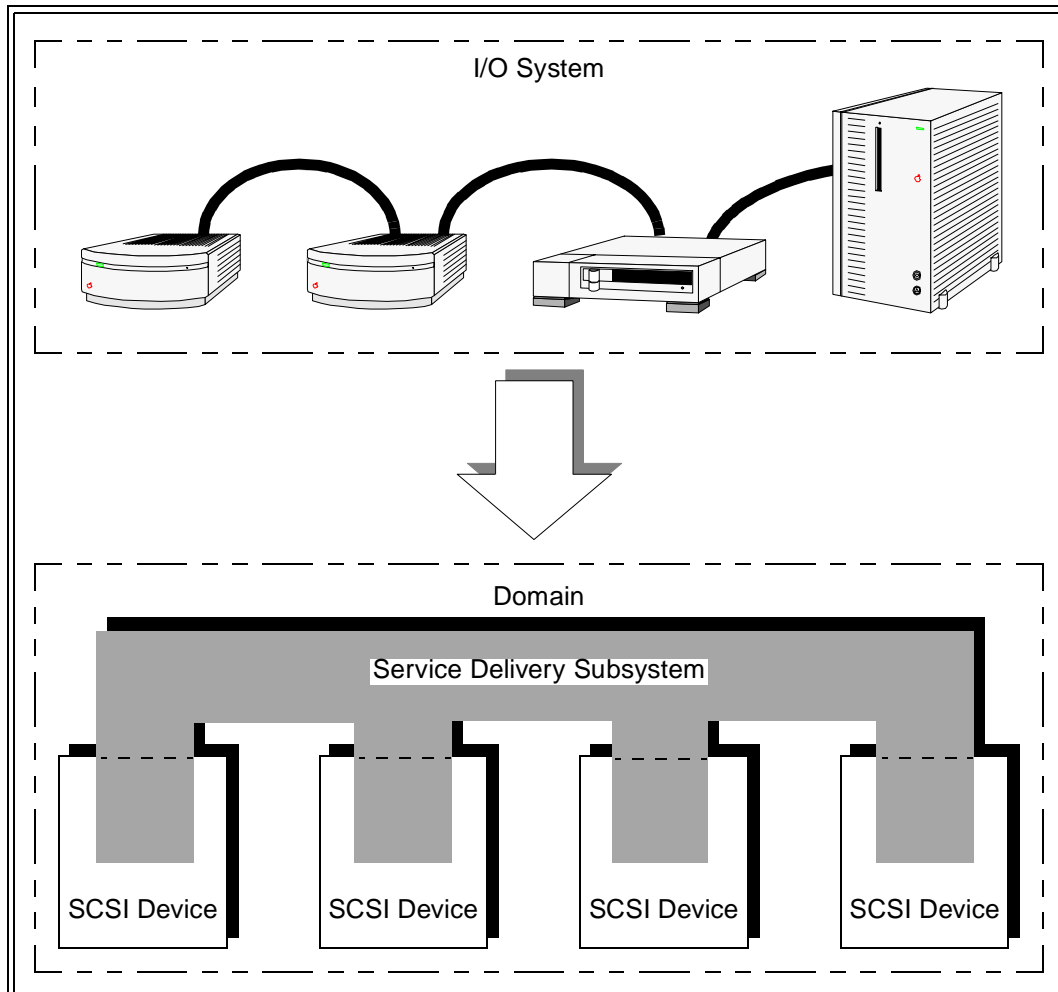
Figure 7 — SCSI I/O system and domain model

The SCSI structural model is organized as follows:

    a)   a basic structural model (see 4.4 through 4.9);
    b)   an enhancement for multiple port devices (see 4.10); and
    c)   an enhancement for logical units with dependent logical units (see 4.11).

The basic structural model applies to all SCSI Devices and Domains. The two model enhancements are available when functional or configuration requirements exceed the capabilities of the basic structural model.

Figure 8 shows the main functional components of the basic SCSI hierarchy. The following clauses define these components in greater detail.
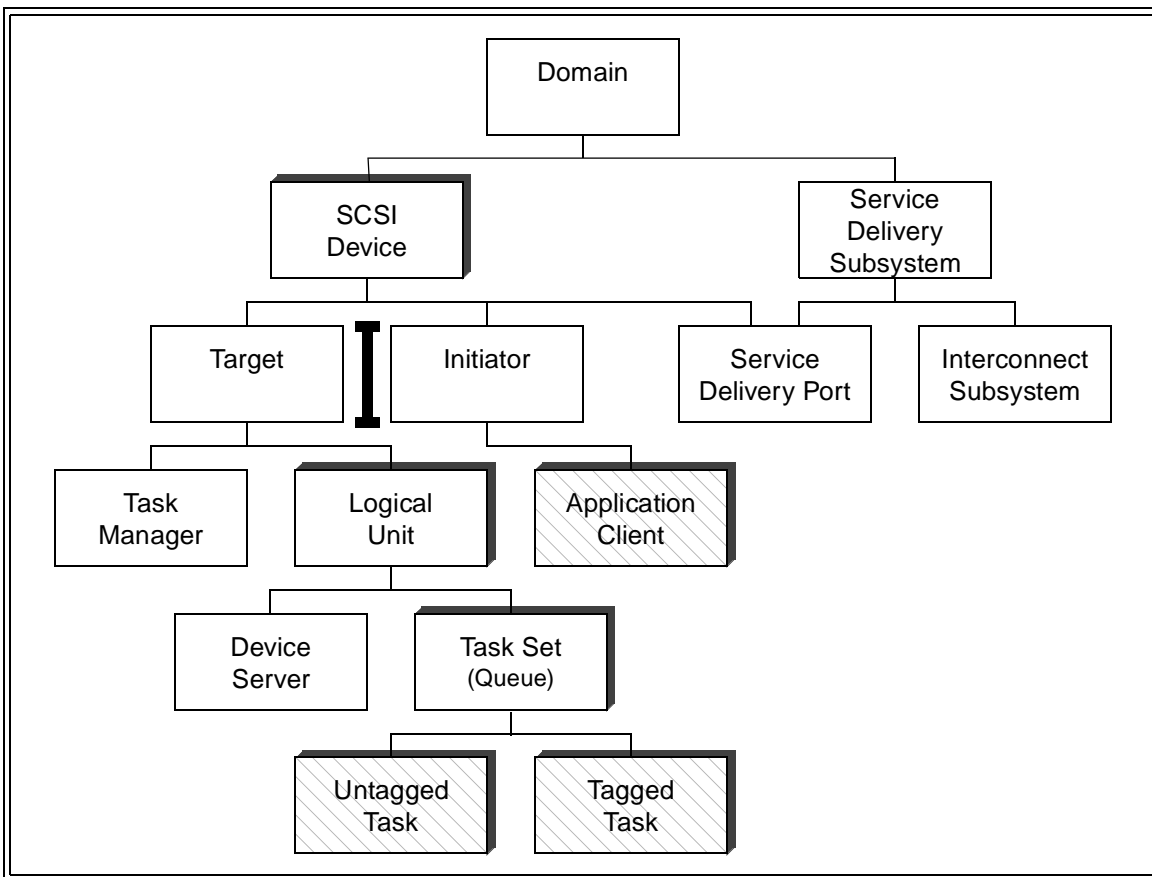
Figure 8 — Basic SCSI hierarchy
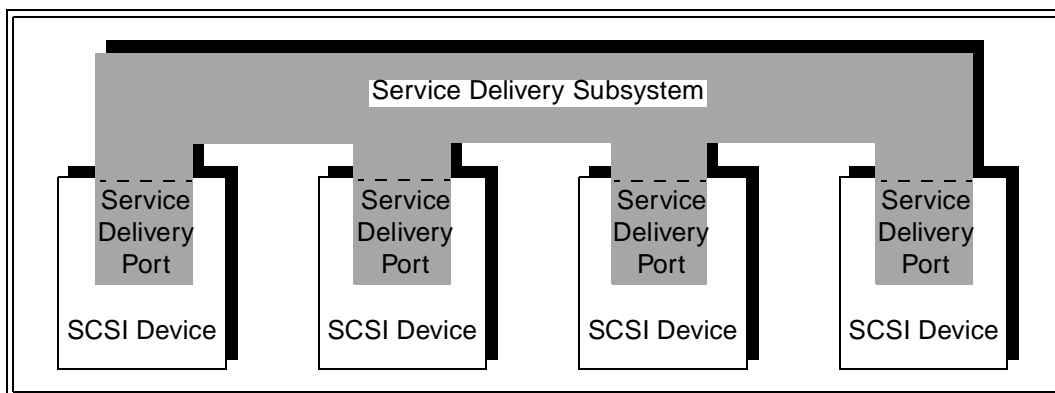
Figure 9 — Domain functional model

4.5 SCSI domain

An SCSI Domain is composed of two or more SCSI Devices and a Service Delivery Subsystem. Figure 9 shows a functional model of a SCSI Domain corresponding to the SCSI I/O system model shown in figure 7. Figure 10 shows the hierarchy of SCSI Domain objects.
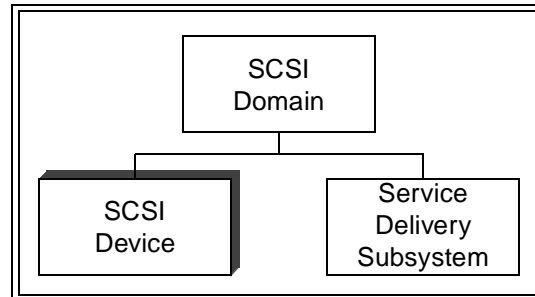
Figure 10 — Domain hierarchy

An SCSI Device is an object that originates or services SCSI commands. As described in 4.7, an SCSI Device originating a command is called an Initiator; an SCSI Device containing logical units that service commands is called a Target. The Service Delivery Subsystem connects all the SCSI Devices in the SCSI Domain, providing a subsystem through which application clients and device servers communicate (see 4.6). The boundaries of a SCSI Domain are established by the system implementor, within the constraints of a specific SCSI protocol and interconnect standards.

4.6 The service delivery subsystem

The Service Delivery Subsystem is composed of two or more Service Delivery Ports and the Interconnect Subsystem (see figure 11). The minimum of two Service Delivery Ports is based on the requirement that an SCSI Domain contain a minimum of two SCSI Devices each of which must have at least one Service Delivery Port (see 4.7).
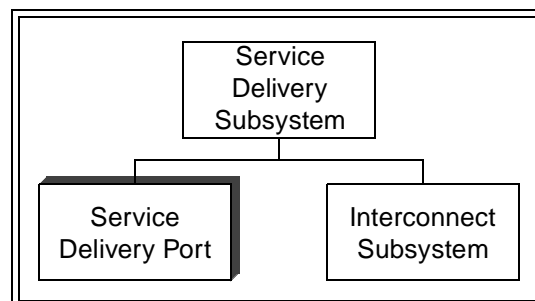
Figure 11 — Service delivery subsystem hierarchy

Each Service Delivery Port resides in one SCSI Device (see 4.7). In this model, the Service Delivery Port object represents the hardware and software that implements the protocols and interfaces between servers or clients in the SCSI Device and the Interconnect Subsystem. The Interconnect Subsystem is a set of one or more physical interconnects that appear to a client or server as a single path for the transfer of requests, responses, and data between SCSI Devices.

The service delivery subsystem is assumed to provide error-free transmission of requests and responses between client and server. Although a device driver in an SCSI implementation may perform these transfers through several interactions with its SCSI protocol layer, the architecture model portrays each operation, from the viewpoint of the

application client, as occurring in one discrete step. In this model, the data comprising an outgoing request is sent in a single "package" containing all the information required to 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.

4.6.1 Synchronizing client and server states

The client is usually informed of changes in server state through the arrival of server responses. In the architecture model such state changes occur after the server has sent the associated response and possibly before the response has been received by the initiator. Some SCSI 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 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 6.7 and 5.3, 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.

4.6.2 Request/Response ordering

In this standard, request or response transactions are said to be in order if, relative to a given pair of sending and receiving 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 commands with the ORDERED attribute to a logical unit as described in clause 7. In this case, the order in which these commands are completed, and hence the final state of the logical unit, may depend on the order in which these commands are received. Similarly, the 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 protocol standard.

For convenience, the SCSI architecture model assumes in-order delivery to be a property of the service delivery subsystem. This assumption is made to simplify the description of behavior and does not constitute a requirement. In addition, this specification makes no assumption about, or places any requirement on the ordering of requests or responses between one sending device and several receiving devices.

4.7 SCSI Device models

In the basic structural model, an SCSI Device (see figure 12) is composed of a Service Delivery Port (see 4.6) combined with an Initiator, or a Target, or both an Initiator and a Target.
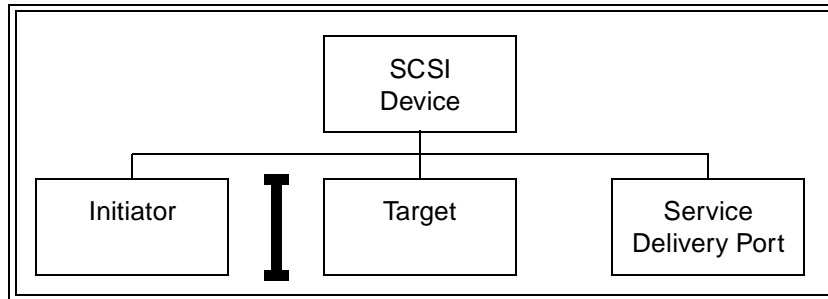
Figure 12 — SCSI Device hierarchy diagram

An Initiator is an SCSI Device which is capable of originating SCSI commands and task management requests (see 4.7.1). A Target is an SCSI Device which is capable of executing SCSI commands and task management requests (see 4.7.2). To be functional, an SCSI Domain needs to contain an SCSI Device that contains a Target and an SCSI Device that contains an Initiator.

There are three models for implementing Initiator and Target capabilities in an SCSI Device, as shown in figure 13. An SCSI Device may perform only Target functions, or only Initiator functions, or it may be capable of performing both functions.
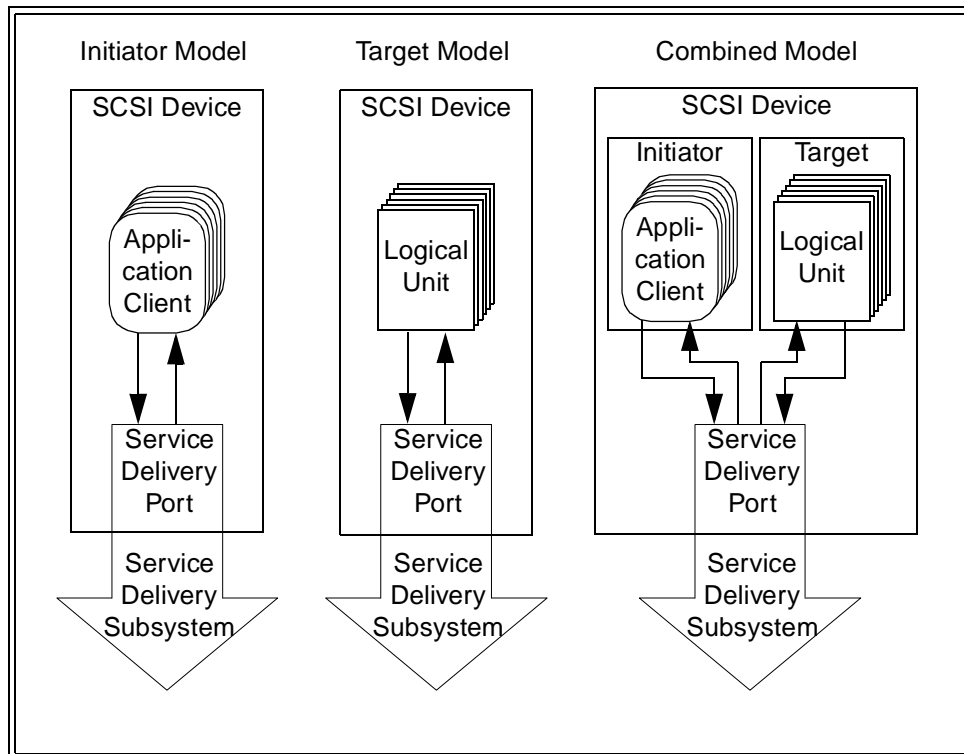
Figure 13 — SCSI device functional models

An SCSI Device is referred to by its role when it participates in an I/O operation. That is, an SCSI Device is called a target when it receives SCSI commands or task management functions, or it is called an initiator when it issues SCSI commands or task management requests.

4.7.1 SCSI initiator model

An Initiator (see figure 14) is composed of an Initiator Identifier and zero or more Application Clients.
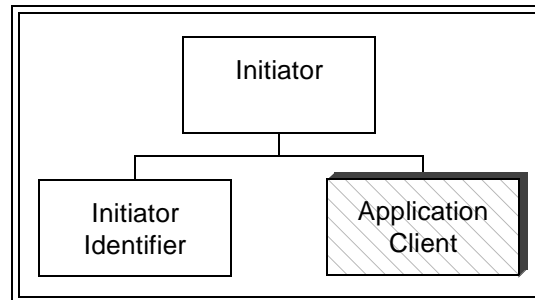
Figure 14 — Initiator hierarchy diagram

An Initiator Identifier is a field containing up to 64 bits that is a SCSI device identifier for the initiator device. An Application Client is the source of commands and task management functions. In this model, there is one application client for each pending command or task management function.

4.7.2 SCSI target model

A Target (see figure 15) is composed of a Target Identifier, a Task Manager, and one or more Logical Units.
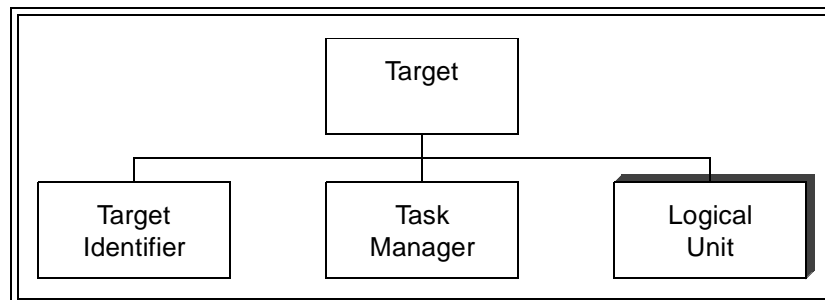
Figure 15 — Target hierarchy diagram

A Target Identifier is a field containing up to 64 bits that is a SCSI device identifier for the target device. A Task Manager is a server that controls one or more tasks in response to task management requests (see 4.7.4). A Logical Unit (see 4.8) is the object to which SCSI commands are addressed. One of the Logical Units composing a Target shall be addressed using the Logical Unit Number zero.

4.7.3 SCSI device identifier

SCSI Device Identifier is the object name used to represent either an Initiator Identifier when the SCSI Device implements the initiator model or a Target Identifier when the SCSI Device implements the target model. SCSI Device Identifier is used when either initiator or target model might be applicable or when other context in the description identifies the initiator or target usage.

4.7.4 The Task Manager

The task manager controls the execution of one or more tasks by servicing the task management functions specified in clause 6. Its external address is equal to the target identifier of the target device. As specified in 4.7.2, 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 referencing a specific logical unit, an initiator should, therefore, not have more than one such request pending to that logical unit.

4.8 Logical Units

A basic Logical Unit (see figure 16) is composed of a Logical Unit Number, a Device Server, and one or more Task Sets.
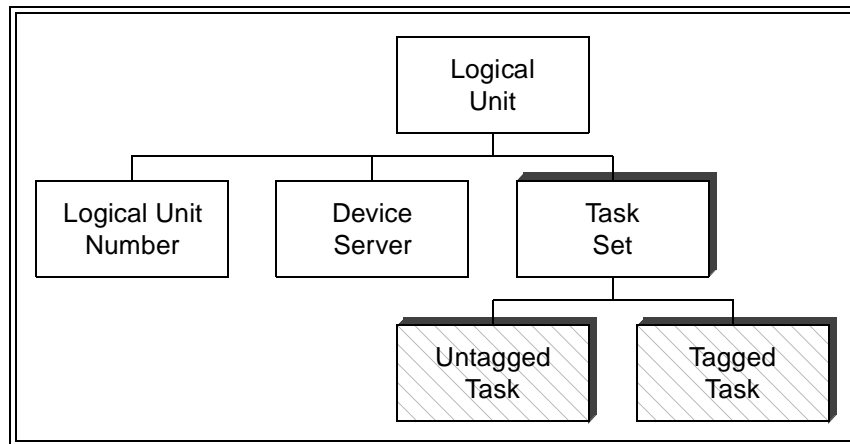
Figure 16 — Logical Unit hierarchy diagram

A Logical Unit Number is a field containing up to 64 bits that identifies the Logical Unit within a target device. If a target device contains 256 or fewer Logical Units none of which are dependent logical units (see 4.10.4), then its Logical Unit Numbers shall have the format shown in table 1, which is a single level subset of the format described in 4.10.4.

Table 1 — Single Level LUN structure

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

In the single level subset format, all LUN structure fields shall be zero except the SINGLE LEVEL LUN field (see table 1). The value in the SINGLE LEVEL LUN field shall be between 0 and 255. The 00b in the ADDRESS METHOD field and the 00h in the BUS IDENTIFIER field indicate addressing for a logical unit at the current level (see 4.11.4). When the single level subset format is used, the HISUP bit shall be set in the standard inquiry data returned by logical unit 0 (see SPC-2).

If any Logical Unit within the scope of a target device includes dependent logical units in its composition, all Logical Unit Numbers within the scope of the target device shall have the format described in 4.10.4.

A Logical Unit Identifier is an external identifier used by an initiator to reference the logical unit and is composed of the Target Identifier for the target device that contains the Logical Unit and the Logical Unit Number that identifies the Logical Unit within the target.

A Device Server is the object that executes SCSI commands and manages the task set according to the rules defined in clause 7.

A Task Set is composed of zero or more Untagged Tasks or a combination of zero or more Tagged Tasks and zero or more Untagged Tasks. See 4.9 for additional restrictions on the Untagged Tasks and Tagged Tasks in a Task Set.

For convenience, Task (see 4.9) refers to either a Tagged Task or an Untagged Task. The interactions among the tasks in a Task Set are determined by the rules for task set management specified in clause 7 and the auto contingent allegiance and contingent allegiance rules specified in 5.6.1 and SCSI-2.   The number of task sets per Logical Unit and the boundaries between task sets are governed by the TST field in the Control mode page (see SPC-2).

4.9 Tasks

The Task object represents either a Tagged Task or an Untagged Task. The composition of a Task includes a definition of the work to be performed by the logical unit in the form of a command or a group of linked commands. A Tagged Task is composed of a definition of the work to be performed by the logical unit, a Tagged Task Identifier (see 4.9.2) and a Task Attribute (see 7.6). An Untagged Task is composed of a definition of the work to be performed by the logical unit, a Untagged Task Identifier (see 4.9.2) and implicitly a SIMPLE task attribute (see 7.6). For convenience, Task Identifier (see 4.9.2) refers to either a Tagged Task Identifier or an Untagged Task Identifier.

A Tagged Task includes a Tag (see 4.9.1) in its Tagged Task Identifier that allows many uniquely identified tagged tasks to be present concurrently in a single task set. A Tagged Task also includes one of the task attributes described in 7.6, which allows the initiator to specify processing relationships between various tagged tasks. An Untagged Task does not include a Tag in any of its component definitions, which restricts the number of concurrent untagged tasks in a single task set to one per initiator. Also, an Untagged Task is assumed to have a SIMPLE task attribute, leaving the initiator no control over its relationship to other tasks in the task set.

Every SCSI protocol shall support tagged 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 5.4). 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.

4.9.1 Task tags

A Tag is a field containing up to 64 bits that is a component of a Tagged Task Identifier. An initiator assigns tag values in each Tagged Task Identifier in a way that ensures that the identifier uniqueness requirements stated in 4.9 are met.

4.9.2 Target identification of tasks

A target identifies a task with a Task Identifier. The Task Identifier object represents either a Tagged Task Identifier or an Untagged Task Identifier. A Tagged Task Identifier is composed of an Initiator Identifier (see 4.7.1), a Logical Unit Identifier (see 4.8) and a Tag (see 4.9.1). An Untagged Task Identifier is composed of an Initiator Identifier and a Logical Unit Identifier.

If a target implements the enhanced model for multiple port devices (see 4.10), then the Task Identifier objects contain one additional object (beyond those mentioned above). For a multiple port device, a Tagged Task Identifier is composed of a Port Identifier (see 4.10.2), an Initiator Identifier (see 4.7.1), a Logical Unit Identifier (see 4.8) and a Tag (see 4.9.1). An Untagged Task Identifier is composed of a Port Identifier, an Initiator Identifier, and a Logical Unit Identifier.

4.9.3 Initiator identification of tasks

An initiator identifies a task to a target using a Task Address. The Task Address object represents either a Tagged Task Address or an Untagged Task Address. A Tagged Task Address is composed of a Logical Unit Identifier (see 4.8) and a Tag (see 4.9.1). An Untagged Task Address is composed of a Logical Unit Identifier.

If an initiator implements the enhanced model for multiple port devices (see 4.10), then it may chose to enhance the definition of a Task Address with port identification information.

4.10 Devices With Multiple Service Delivery Ports

Optionally, a single device may have multiple Service Delivery Ports (see 4.6), however, the model for such a device is not a single SCSI Device (see 4.7) but multiple SCSI Devices, one for each Service Delivery Port. Similarly, a single device may respond to multiple SCSI device identifiers. The model for such a device also is one of multiple SCSI Devices, one for each SCSI device identifier. Devices that have multiple Service Delivery Ports or respond to multiple SCSI device identifiers are called SCSI Multi-port Units.

SCSI Multi-port Units are objects in SCSI architecture model (see figure 17) that do not fit in the two dimensional hierarchy shown in 4.4. Instead, SCSI Multi-port Units combine basic objects from the SCSI device model in a way that produces a different structural plane. Figure 17 also shows that the multiple SCSI devices instantiated by an SMU may be constituents of different SCSI Domains.
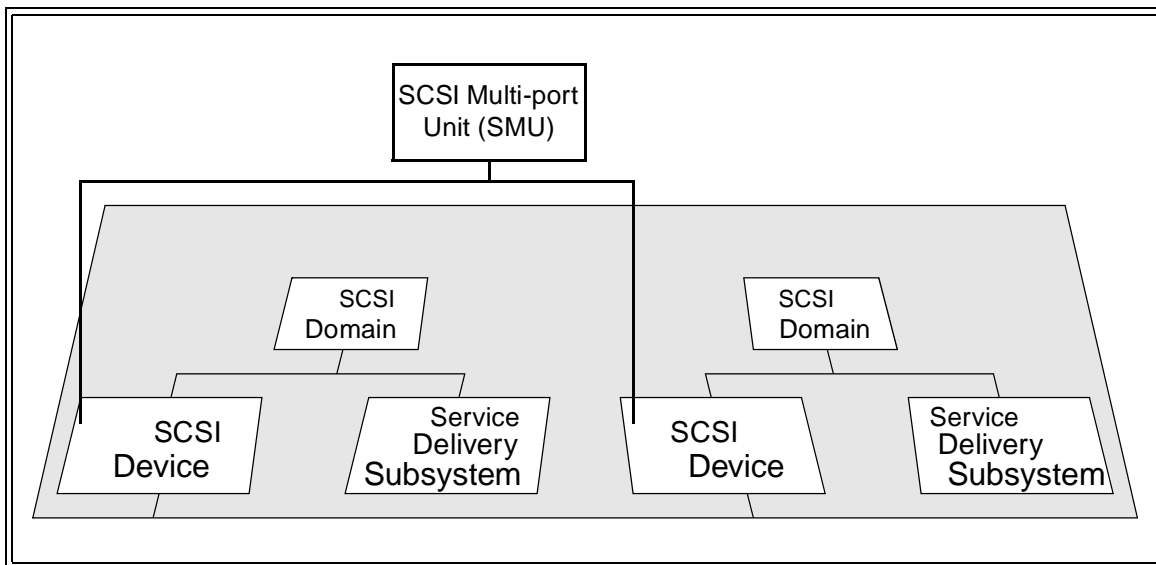
Figure 17 — A SCSI multi-port unit and multiple SCSI domains

The multiple SCSI device identifiers representing an SMU shall meet the requirements for Initiator Identifiers (see 4.7.1) or Target Identifiers (see 4.7.2) or both. The structure of an SMU depends on whether the device is a target or an initiator. SCSI multi-port units that implement both target and initiator models combine the target and initiator multi-port structures in vendor-specific ways that meet product requirements while maintaining the multi-port model for the target and initiator functions performed by the product. How an SMU is viewed by counterpart initiators or targets in the SCSI domain also depends on whether an initiator is examining an SMU target or a target is servicing an SMU initiator. The structures and views of SCSI multi-port units are asymmetric for targets and initiators. The clauses that follow discuss the four principle cases described above.

4.10.1 SMU target structure

Figure 18 shows the structure of an SMU target. A single Task Manager and collection of Logical Units share all the SCSI Devices and Service Delivery ports in the SMU.
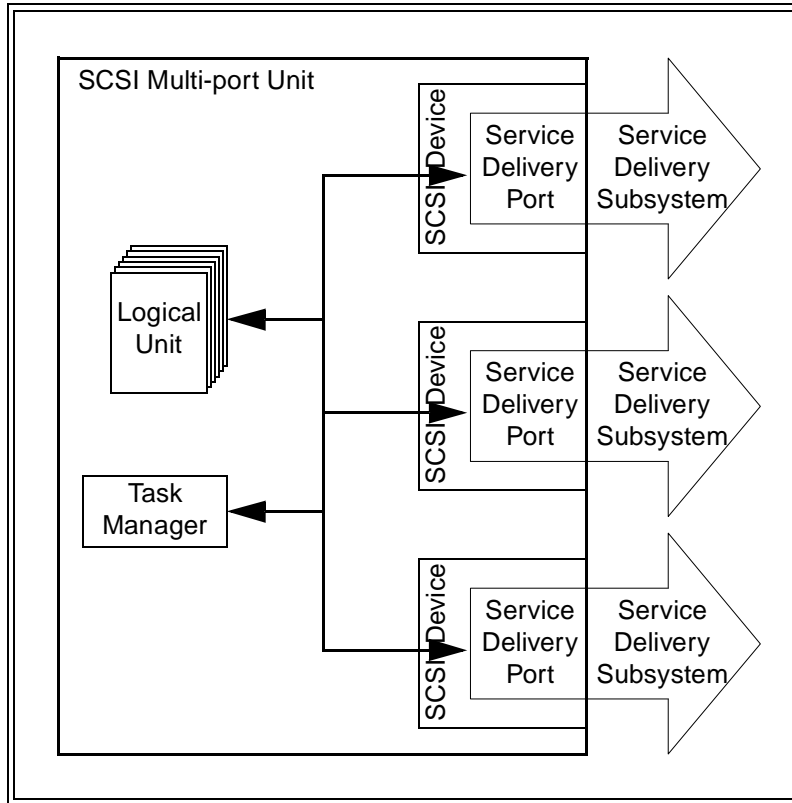
Figure 18 — SMU target structure model

Two-way communications shall be possible between all logical units and all service delivery ports, however, communications between any logical unit and any service delivery port may be inactive sometimes. Two-way communications shall be available between the task manager and all service delivery ports. Each service delivery port shall accept commands addressed to LUN 0 and shall forward them to a device server for processing. The REPORT LUNS commands (see SPC-2) shall be accepted by logical unit 0 from any service delivery port and shall return the logical unit inventory available via that service delivery port. The availability of a the same logical unit through multiple delivery ports is discovered by matching Device Identifier values in the INQUIRY command Vital Product Data page (see SPC-2).

4.10.2 SMU Task Identifiers

In addition to the Task Identifier constituent objects described in 4.9.2, SMU targets shall provide a Port Identifier object to contribute to the construction of Task Identifiers. The exact nature of the Port Identifier is vendor specific. The complete contents of the Task Identifier object, including the Port Identifier, is defined in 4.9.2.

4.10.3 SMU initiator structure

Figure 19 shows the structure of an SMU initiator. A collection of Application Clients share all the SCSI Devices and Service Delivery ports in the SMU.
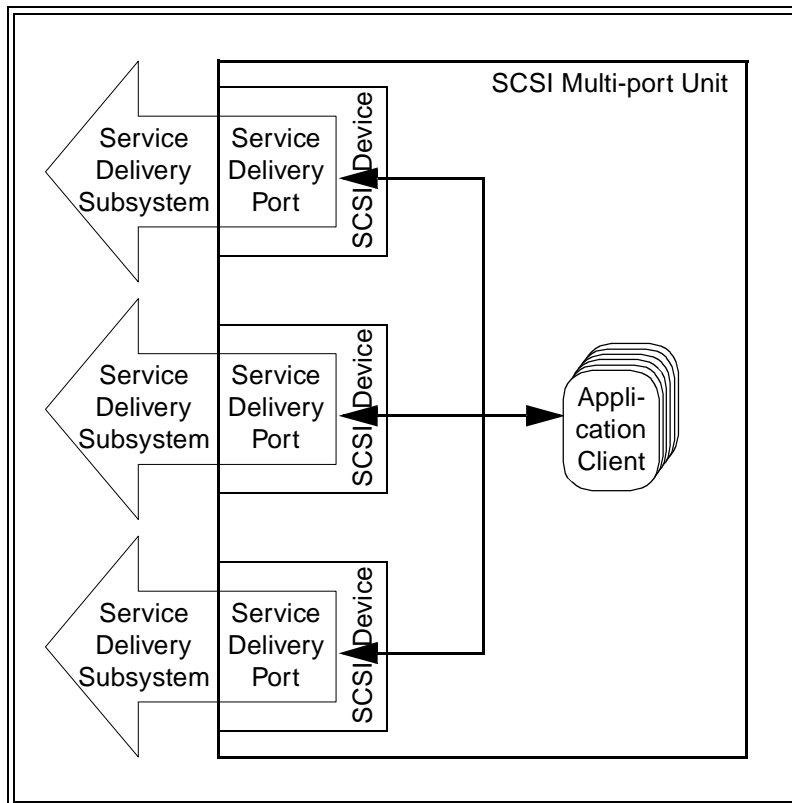
Figure 19 — SMU initiator structure model

Two-way communications shall be possible between all application clients and all service delivery ports, however, communications between any logical and any service delivery port may be inactive sometimes. Mechanisms by which a target would have the ability to discover that it is communicating with an SMU initiator are beyond the scope of any standards in the SCSI family of standards.

4.10.4 Initiator view of an SMU target

A SMU target may be connected to multiple independent service delivery subsystems in such a way that no single initiator can communicate with a logical unit or task manager using two or more of the service delivery ports in the SMU. In this case, the application clients in the initiators cannot distinguish the SMU from a basic SCSI Device.

However, SCSI Multi-port Devices may be configured in many different ways where application clients have the ability to discover that one or more logical units are accessible via multiple service delivery ports. Figure 20 and figure 21 show two examples of such configurations.

Figure 20 shows an SMU target participating in a single SCSI Domain with two SCSI initiator devices. There are four service delivery ports in this configuration, two for the SMU target and one each for the two initiators. There are two Target Identifiers and two Initiator Identifiers in this SCSI Domain. Using the INQUIRY command Vital Product Data page as described in 4.10.1, the application clients in each initiator have the ability to discover that

logical units in the SMU target are accessible via multiple Target Identifiers (service delivery ports) and map the configuration of the SMU target.
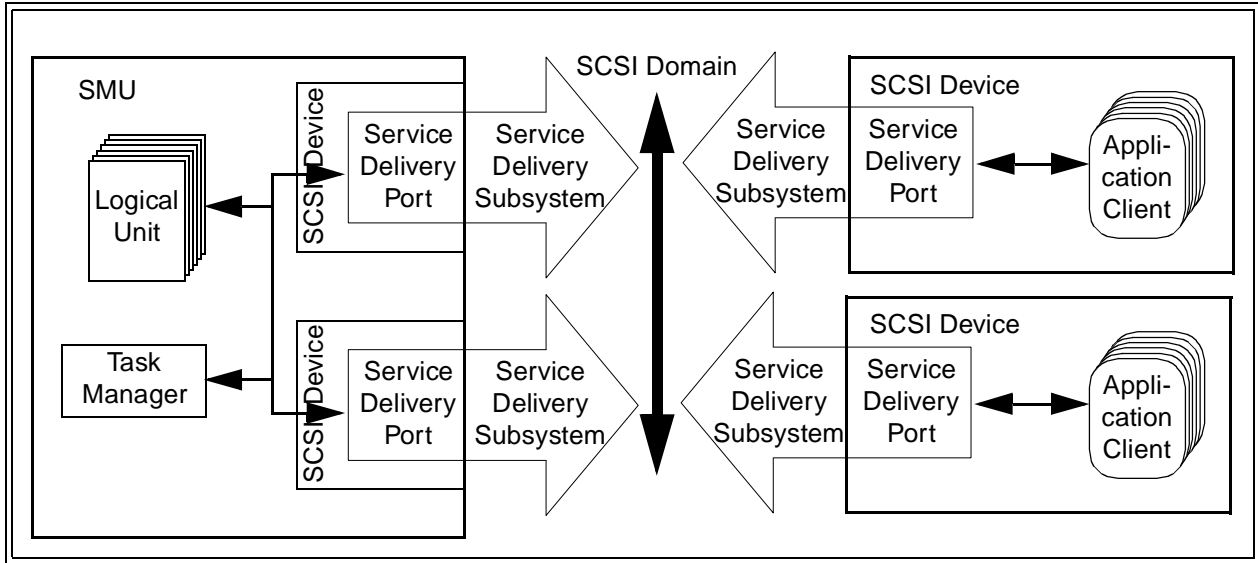
Figure 20 — SMU target configured in a single SCSI Domain

Figure 21 shows an SMU target participating in a two SCSI Domains and an SMU initiator participating in the same two SCSI Domains. There are four service delivery ports in this configuration, two for the SMU target and two for the SMU initiator. There is one Target Identifier and one Initiator Identifier in each of the two SCSI Domains. Using the INQUIRY command Vital Product Data page as described in 4.10.1, the application clients in the SMU initiator have the ability to discover that logical units in the SMU target are accessible via multiple service delivery ports in the SMU initiator and map the configuration. However, the methods available to application clients to distinguish between the configuration shown in figure 21 and the configuration shown in figure 22 may be beyond the scope of the SCSI family of standards.
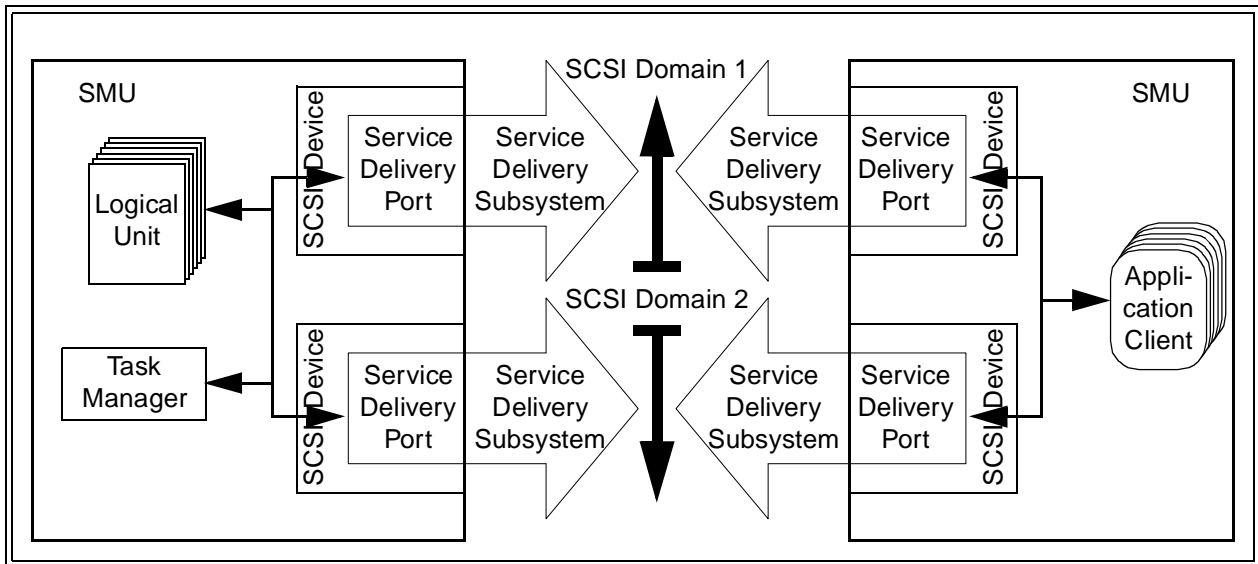
Figure 21 — SMU target configured in multiple SCSI Domains

Figure 22 shows the same configuration as figure 21 except that the two SCSI Domains have been replaced by a single SCSI Domain.
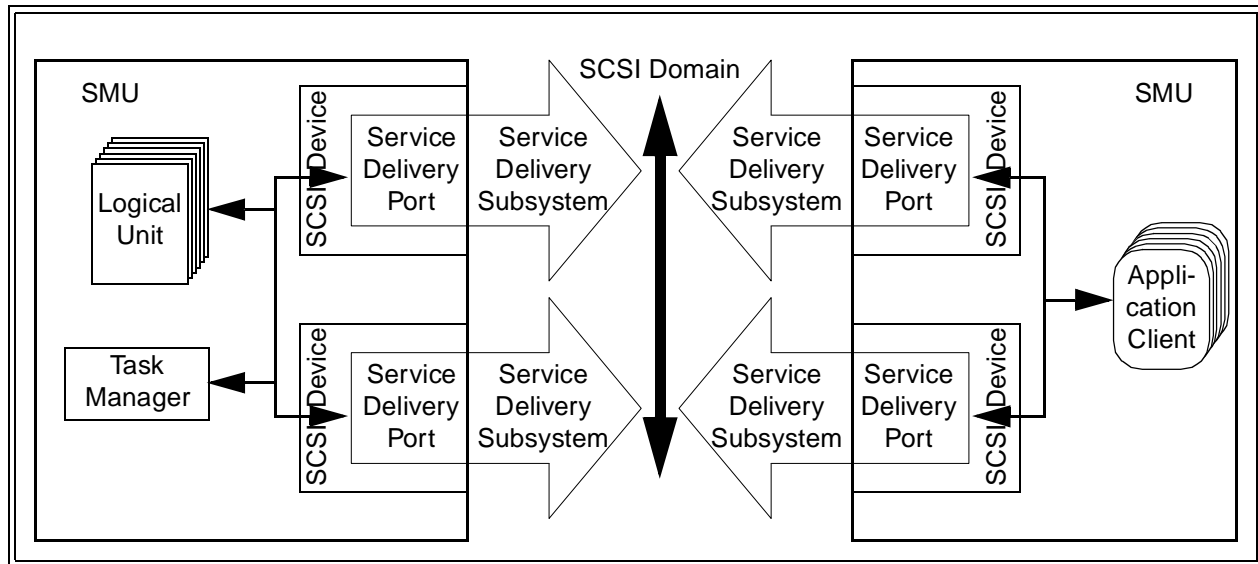
Figure 22 — SMU target and SMU initiator configured in a single SCSI Domain

This model for application client determination of SMU target configurations relies on information that is available only to the application clients via SCSI commands. The service delivery ports in the initiators (figure 20) or initiator (figure 21 and figure 22) are unable to distinguish the multiple service delivery ports in the SMU from individual service delivery ports in two separate SCSI Devices.

4.10.5 Target view of an SMU initiator

An SCSI target device or SMU target does not have the ability to detect the presence of an SMU initiator. Therefore, a target handles an SMU initiator exactly as it would handle multiple separate SCSI initiator devices. For example, an SCSI target handles the configurations shown in figure 21 and figure 22 in exactly the same way it handles the configuration show in figure 20.

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

4.10.6 SMU considerations for task management functions

Although there is only one Task Manager for all logical units and all service delivery ports in an SMU, the Task Manager in an SMU shall observe the requirements described in the following clauses in addition to the requirements placed on a task set manager by the SCSI architecture basic structural model.

> Editors Note 2 - ROW: More needs to be said here, and proposals brought to T10 will define and enhance this portion of the SMU model. Also, the SCSI working group has agreed that requirements arising out of the SMU model will be placed in the affected sections of the task manager requirements (e.g., the SMU reset requirements will appear in the 'target hard reset' clause 5.6.6). The text in these clauses will be informational and point the reader to the proper definitions clauses.

4.10.6.1 Reset conditions

4.10.6.2 Task set management

4.10.7 SMU considerations for command processing

For a single Task (see 4.9), two and only two SCSI Devices shall participate in transmitting and receiving the remote procedure model elements described in clause 5, one as initiator and one as target. A task involving one initiator-target pair shall not specify a third SCSI device to participate in transmitting and receiving the remote procedure model elements for that task. Thus, an SMU initiator shall not create a task using one service delivery port with the expectation that the data transfer or status return for that task would occur via a different service delivery port.

Task Identifiers (see 4.9.2) are unique for all tasks in an SMU target, however, some components of the Task Identifiers (e.g., Tag) may not be unique for all tasks in an SMU target. Task Addresses (see 4.9.3) are not unique for all tasks created by application clients in an SMU initiator. In order to make initiator task identification values, the Task Address must be additionally qualified by an identifier for the service delivery port in a vendor-unique manner.

4.11 Hierarchies of Dependent Logical Units

Optionally, the basic model for a logical unit (see 4.8) may be enhanced to include one or more unique logical units embedded within another logical unit. In such cases, the model hierarchy diagram in 4.8 is enhanced to become the diagram shown in figure 16 and the embedded logical units are called dependent logical units (see 3.1.22).
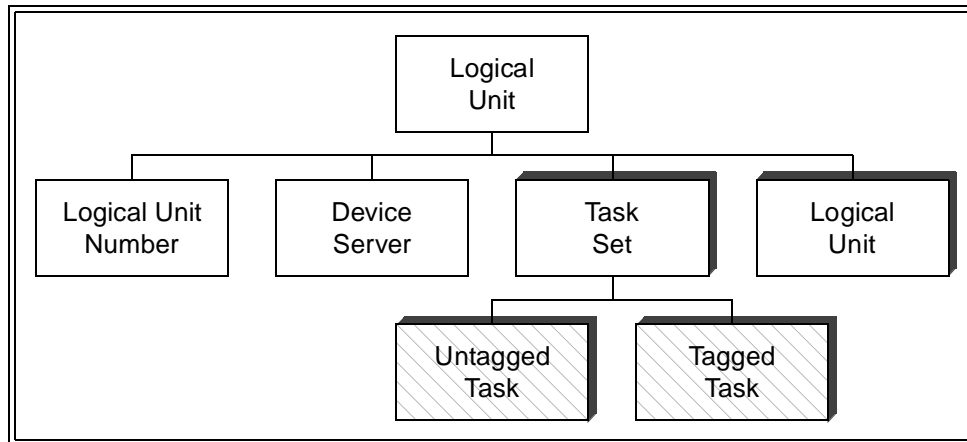
Figure 23 — Dependent Logical Unit hierarchy diagram

When the dependent logical unit model enhancement is utilized, the hierarchical logical unit structure defined here shall be used. If any Logical Unit within the scope of a target device includes dependent logical units, all Logical Unit Numbers within the scope of the target device shall have the format described in this clause. A device server that implements the hierarchical structure for dependent logical units described here shall set the HISUP bit in the standard inquiry data returned by logical unit 0 (see SPC-2). Clause 4.8 defines cases when target devices that do not implement dependent logical units are required to implement a subset of the logical unit structure described in this clause.

As shown in figure 24, the hierarchical logical unit structure is an inverted tree containing up to four addressable levels. The example in figure 24 is a three-level system that consists of:
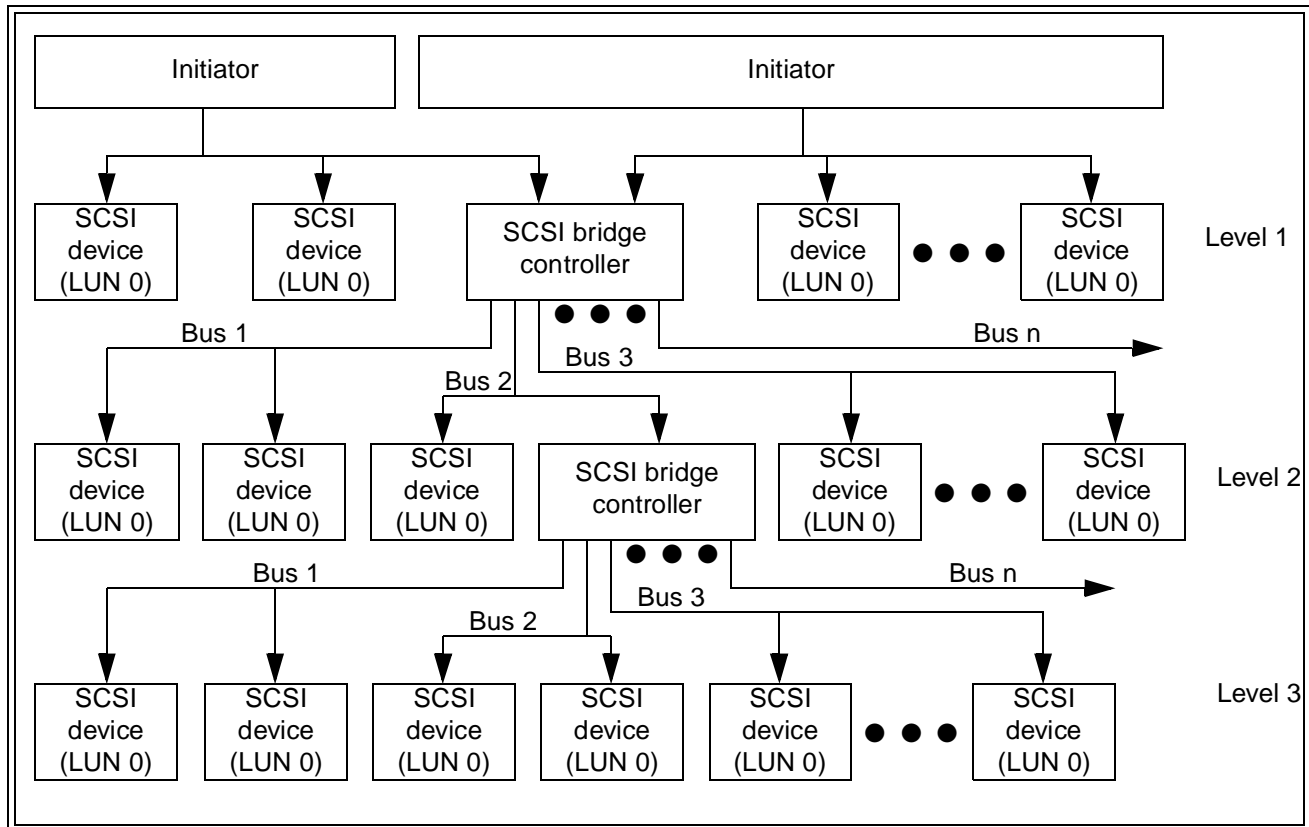
Figure 24 — Example of hierarchical system diagram

   a) One initiator that has three SCSI devices attached on a single SCSI bus that is not expandable. One of the
       SCSI devices is a dual ported SCSI bridge controller.
   b) One initiator has three SCSI devices attached on a single SCSI bus that is expandable. One of the SCSI
       devices contains a dual ported SCSI bridge controller.
   c) The SCSI bridge controller has three SCSI buses with SCSI devices attached and is capable of driving
       more SCSI buses.
       a) Two of the SCSI buses contain two SCSI devices each and these SCSI buses are not expandable.
           One of the SCSI devices contains a SCSI bridge controller.
       b) One of the SCSI buses contains two SCSI devices and is expandable.
       c) The SCSI bridge controller has three SCSI buses with SCSI devices attached and is capable of driving
           more SCSI buses.
           a) Two of the SCSI buses contain two SCSI devices each and these SCSI buses are not expandable.
           b) One of the SCSI buses contains two SCSI devices and is expandable.

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

   a) Logical unit address method (see 4.11.3);
   b) Peripheral device address method (see 4.11.4); and
   c) Virtual device address method (see 4.12).

All peripheral device addresses, except LUN 0 (see 4.11.1), default to vendor specific values. All addressable
entities may default to vendor specific values or may be defined by an application client (e.g., by the use of SCC-2
configuration commands).

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

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

4.11.1 LUN 0 address

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

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

4.11.2 Eight byte LUN structure

The eight byte LUN structure (see table 3) allows up to four levels of devices to be addressed under a single target. Each level shall use bytes 0-1 to define the address and/or location of the SCSI device to be addressed on that level.

If the LUN indicates that the command is to be relayed to the next layer then the current layer shall use bytes 0-1 of the eight byte LUN structure to determine the address of the device to which the command is to be sent. When the command is sent to the target the eight byte LUN structure that was received shall be adjusted to create a new eight byte LUN structure (see table 2 and figure 25).

Devices shall keep track of the addressing information necessary to transmit information back through all intervening layers to the task's originating initiator.
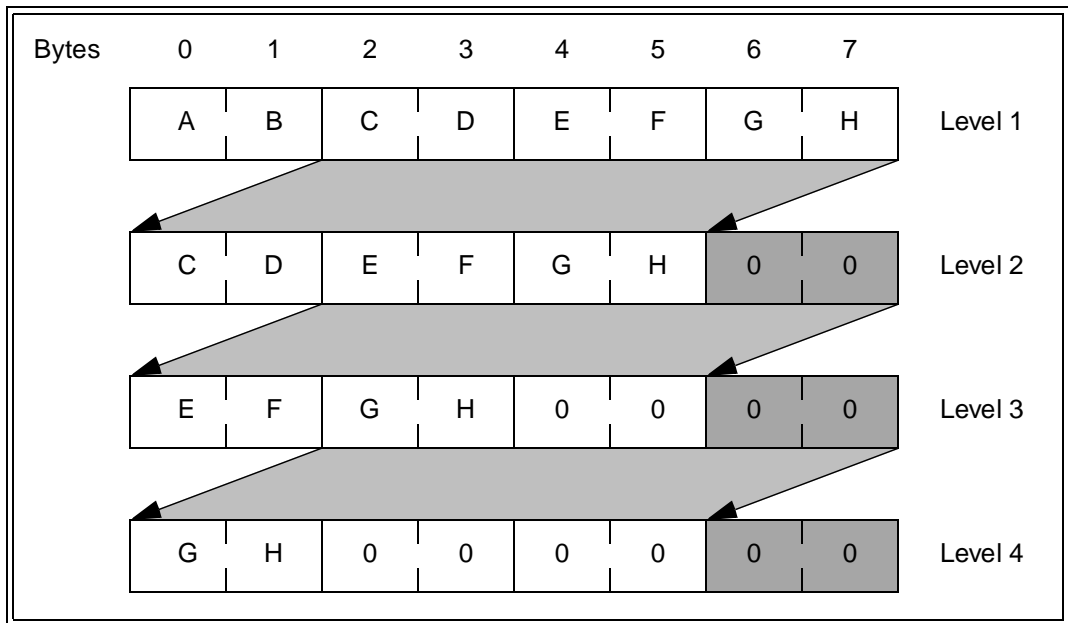
Figure 25 — Eight Byte LUN structure adjustments

Table 2 — Eight byte LUN structure adjustments

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

The eight byte LUN structure requirements as viewed from the application client are shown in table 3.

Table 3 — Eight Byte LUN structure

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

The FIRST LEVEL ADDRESSING field indicates the first level address of a device. See table 4 for a definition of the FIRST LEVEL ADDRESSING field.

The SECOND LEVEL ADDRESSING field indicates the second level address of a device. See table 4 for a definition of the SECOND LEVEL ADDRESSING field.

The THIRD LEVEL ADDRESSING field indicates the third level address of a device. See table 4 for a definition of the THIRD LEVEL ADDRESSING field.

The FOURTH LEVEL ADDRESSING field indicates the fourth level address of a device. See table 4 for a definition of the FOURTH LEVEL ADDRESSING field.

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

Table 4 — Format of addressing fields

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

The ADDRESS METHOD field defines the contents of the ADDRESS METHOD SPECIFIC field. See table 5 for the address methods defined for the ADDRESS METHOD field. The ADDRESS METHOD field only defines address methods for entities that are directly addressable by an application client.

Table 5 — ADDRESS METHOD field values

Code	Description	Clause
10b	Logical unit addressing method	4.11.3
00b	Peripheral device addressing method	4.11.4
01b	Device type specific	
11b	Reserved	

4.11.3 Logical unit addressing method

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

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

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

See table 6 for the definition of the ADDRESS METHOD SPECIFIC field used when the logical unit addressing method is selected.

Table 6 — Logical unit addressing

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

The TARGET field, BUS NUMBER field, and LUN field address the logical unit to which the received command shall be relayed. The command shall be relayed to the logical unit (LUN field value) within target (TARGET field value) located on bus (BUS NUMBER field value). The target information in the TARGET field may be a target identifier (see 4.9.2) or it may be a mapped representation of a target identifier, when the range of possible target identifiers is too large to fit in the TARGET field.

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

4.11.4 Peripheral device addressing method

All SCSI commands are allowed when the peripheral device address method is selected, however peripheral devices are only required to support mandatory SCSI commands. Devices are not required to relay commands, from the application client, to a lower layer. Any command that is not supported or relayed shall be terminated with

a CHECK CONDITION status. The sense key shall be set to ILLEGAL REQUEST and the additional sense code shall be set to INVALID COMMAND OPERATION CODE.

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

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

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

Table 7 — Peripheral device addressing

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

The BUS IDENTIFIER field identifies the bus or path that the SCSI device shall use to relay the received command. The BUS IDENTIFIER field may use the same value encoding as the BUS NUMBER field (see 4.11.3). However, bus identifier zero shall indicate that the command is to be relayed to a logical unit within the SCSI device at the current level.

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

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

A BUS IDENTIFIER field greater than zero represents physical SCSI interconnect that connects a group of SCSI devices to the current level SCSI device. Each physical interconnect shall be assigned a unique number from 1 to 63. These bus identifiers shall be used in the BUS IDENTIFIER field when assigning addresses to peripheral devices attached to the physical interconnects. When the BUS IDENTIFIER field is greater than zero, the command shall be relayed to the logical unit zero within target (TARGET/LUN field value) located physical interconnect (BUS IDENTIFIER field value). The target information in the TARGET/LUN field may be a target identifier (see 4.9.2) or it may be a mapped representation of a target identifier, when the range of possible target identifiers is too large to fit in the TARGET/LUN field.

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

The SCSI device located within the current level shall be addressed by a BUS IDENTIFIER field and a TARGET/LUN field of all zeros, also known as LUN 0 (see 4.11.1).

4.12 The SCSI model for distributed communications

The SCSI 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 26.
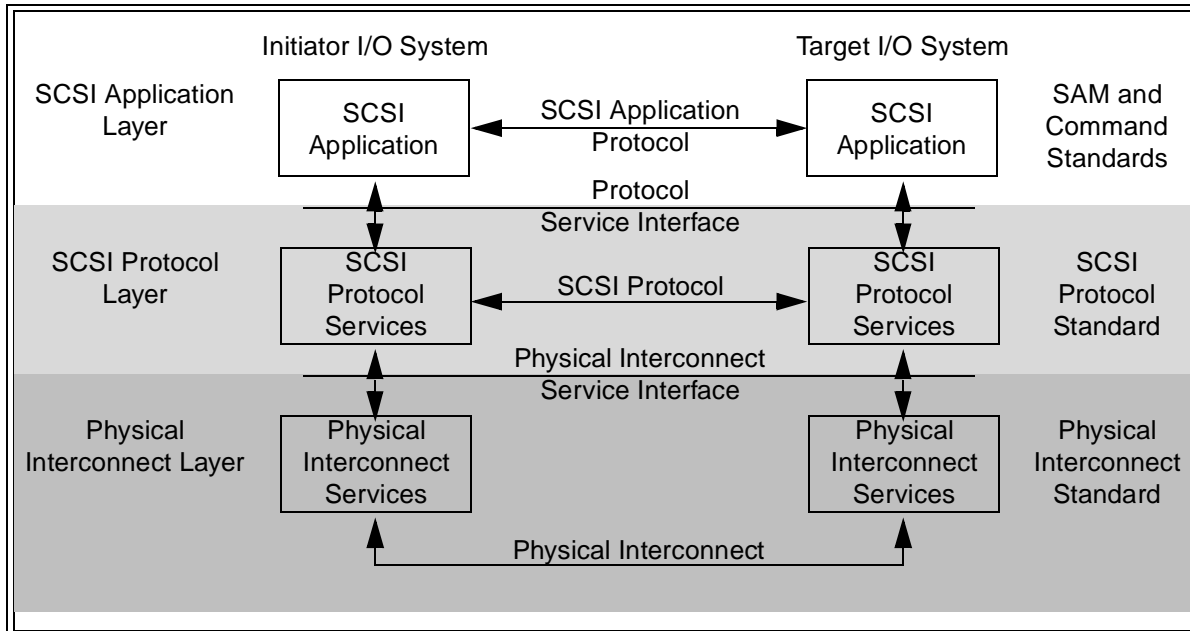
Figure 26 — Protocol service reference model

The layers comprising this model and the specifications defining the functionality of each layer are denoted by horizontal sequences. A layer consists of peer entities 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) <u>SCSI application layer</u>: Contains the clients and servers that originate and execute SCSI I/O operations by means of an SCSI application protocol;
  b) <u>SCSI protocol layer</u>: Consists of the services and protocols through which clients and servers communicate; and
  c) <u>Physical interconnect layer</u>: 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 protocol specifications. While these protocol services may serve as a guide for designing reusable software or firmware that can be adapted to different SCSI protocols, there is no requirement for an implementation to provide the service interfaces specified in this standard.

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 3.6.2. In this model, input arguments are defined relative to the layer receiving an inter-action. 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.

Figure 27 shows the relationships between the four protocol service types.
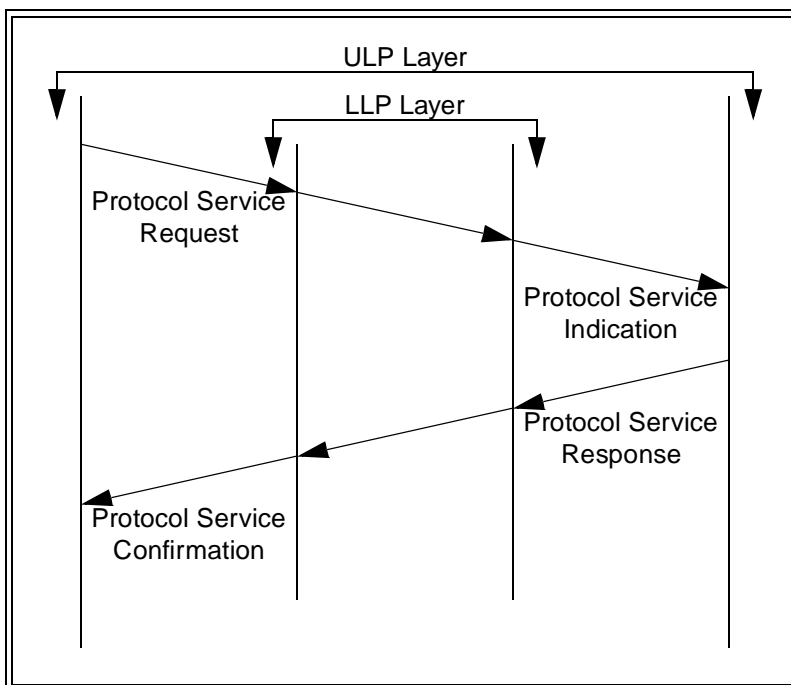
Figure 27 — Protocol service model

Figure 24 shows how protocol services may be used to execute a client-server request-response transaction at the SCSI application layer.
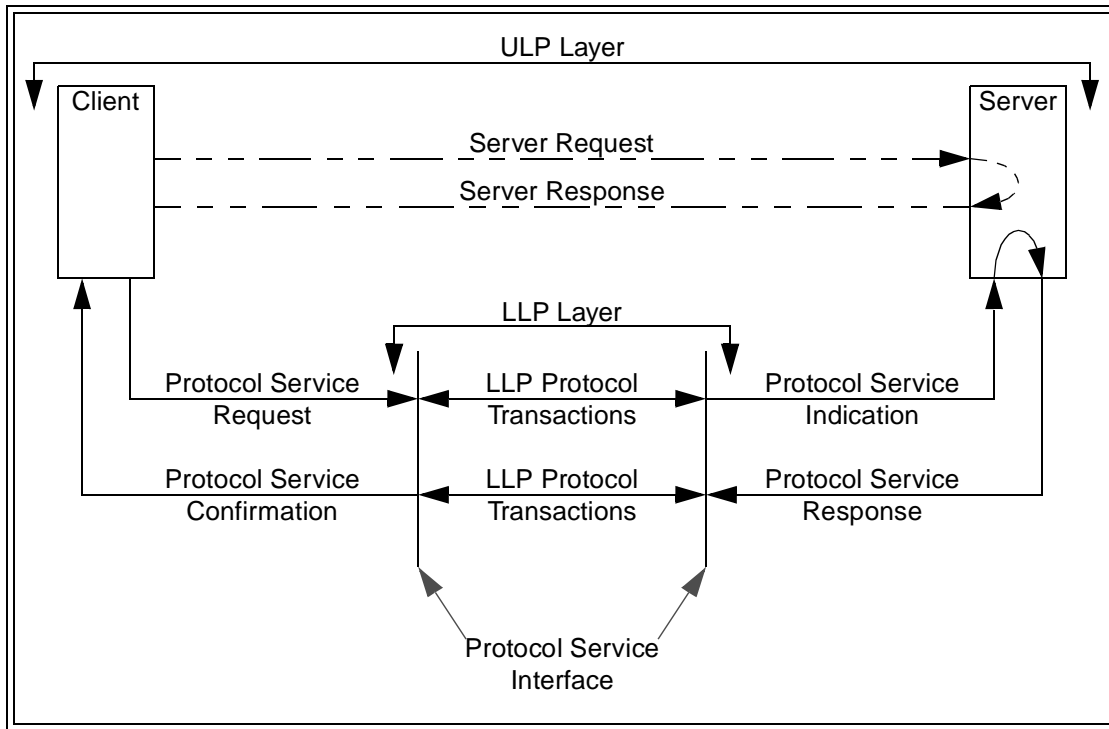
Figure 28 — Request-Response ULP transaction and related LLP services

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

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 4.9.3.
CDB:	Command descriptor block (see 5.1).
Task Attribute:	A value specifying one of the task attributes defined in 7.6. 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 5.6.4.2. It is not an error for the application client to provide this argument when autosense is not supported by the SCSI protocol or logical unit.

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 5.6.4.2).
Status:	A one-byte field containing command completion status (see 5.2). 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 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 5.2 other than INTERMEDIATE or INTERMEDIATE-CONDITION MET.

LINKED COMMAND COMPLETE: Logical unit responses indicating that a linked command has completed successfully. As specified in 5.2, the status parameter shall have a value of INTERMEDIATE or INTERMEDIATE-CONDITION MET.

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

The actual protocol events corresponding to a response of TASK COMPLETE, LINKED COMMAND COMPLETE or SERVICE DELIVERY OR TARGET FAILURE shall be specified in each 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 5.1.2. 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, 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 5.6.2 may result.

5.1 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 clause 5. The parameters required for each command are specified in the applicable SCSI 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-2 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 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.

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

The general format for all command descriptor blocks except the command descriptor block for operation code 7Fh is shown in table 8   The remaining parameters depend on the command to be executed. All SCSI protocol speci-

fications shall accept command descriptor blocks less than or equal to 16 bytes in length. Command descriptor blocks using the format shown in table 8 shall not exceed sixteen bytes in length.

Table 8 — 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							

5.1.1 OPERATION CODE **byte**

The first byte of an SCSI command descriptor block shall contain an operation code. The OPERATION CODE (see table 9) 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 table 10.

Table 9 — OPERATION CODE **byte**

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

The value in the GROUP CODE field specifies one of the groups shown in table 10.

Table 10 — Group Code values

Group Code	Meaning
0	6 byte commands
1	10 byte commands
2	10 byte commands
3	reserved **[1]**
4	16 byte commands
5	12 byte commands
6	vendor specific
7	vendor specific
Note: **[1]** The format commands using the group code 3 and operation code 7Fh is described in SPC-2. With the exception of operation code 7Fh, all group code 3 operation codes are reserved.	

5.1.2 CONTROL **byte**

The CONTROL byte is the last byte of every command descriptor block. The CONTROL byte is defined in table 11.

Table 11 — CONTROL **byte**

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

All SCSI protocol specifications and protocol implementations shall provide the functionality needed for a logical unit to implement the NACA bit and LINK bit as described herein.

The NACA (Normal ACA) bit is used to control the rules for handling an ACA condition caused by the command. Clause 5.6.1.1 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 NACA value of zero and may support the NACA value of one. The ability to support a NACA value of one is indicated in standard INQUIRY data (see the SPC-2 standard).

If the NACA bit is set to a value that 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. 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 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 5.2). The logical unit shall complete the command with a status of CHECK CONDITION and a sense key of ILLEGAL REQUEST if the LINK bit is set to one and the logical unit does not support linked commands.

Bit 1 provides an obsolete way to request interrupts between linked commands. If bit 1 is equal to one, device servers not implementing the obsolete capability shall terminate the command with CHECK CONDITION status and the sense key shall be set to ILLEGAL REQUEST.

5.2 Status

The status codes are specified in table 12. Status shall be sent from the logical unit to the application client whenever a command ends with a service response of TASK COMPLETE or LINKED COMMAND COMPLETE. The receipt of any status, except INTERMEDIATE or INTERMEDIATE-CONDITION MET, shall indicate that the associated task has ended.

Table 12 — Status codes

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

Definitions for each status 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 or contingent allegiance condition has occurred (see 5.6.1). Optionally, autosense data may be delivered (see 5.6.4.2).

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

BUSY. This status indicates that the logical unit is busy. This status shall be returned whenever a logical unit is unable to accept a command from an otherwise acceptable initiator (i.e., no reservation conflicts). The recommended initiator recovery action is to issue the command again at a later time.

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

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

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

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

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

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

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

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 5.6.1).

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

5.2.1 Status precedence

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

5.3 Protocol Services in Support of Execute Command

This clause describes the protocol services that support the remote procedure call. All SCSI 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 protocol standard is optional. All SCSI 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.

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 speci-
   fied in the **Send SCSI Command** call and autosense delivery is supported by
   the SCSI 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 5.6.4.2).

Protocol Service Confirmation:

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

5.3.1 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 protocol standards shall define
the protocols required to implement these services.

The application client's buffer appears to the device server as a single, logically contiguous block of memory large
enough to hold all the data required by the command (see figure 29). The model requires unidirectional data
transfer. That is, the execution of an SCSI command shall not require the transfer of data for that command both to
and from the application client.
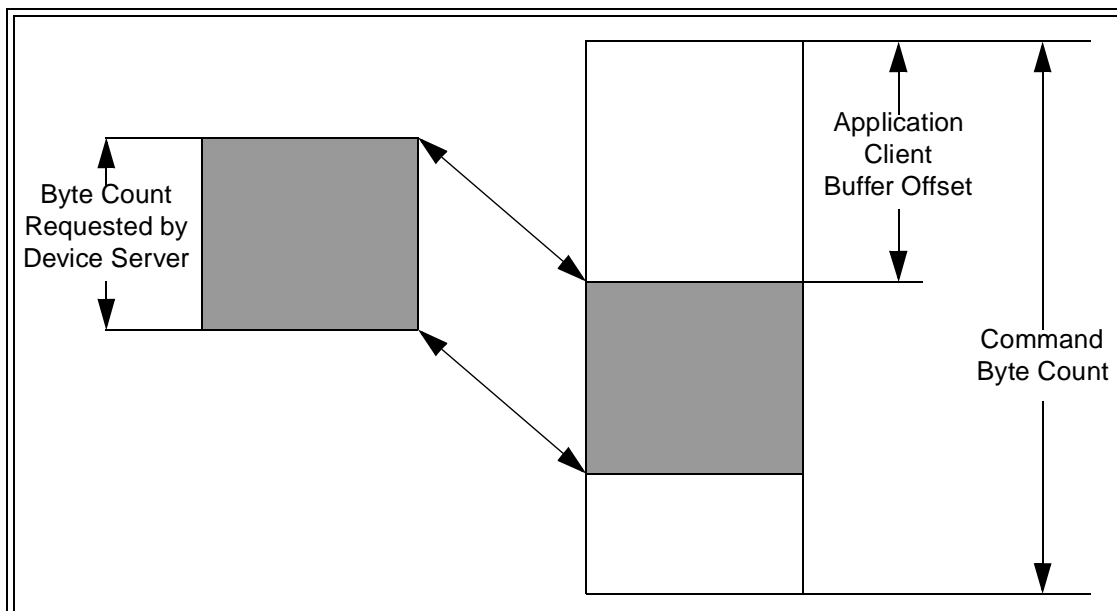
Figure 29 — Model for buffered data transfers

It is assumed that the buffering resources available to the logical unit 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 29 shows the model for such incremental data transfers.

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 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 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 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.

5.3.2 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 4.9.2.
Device Server Buffer:	Buffer from which data is to be transferred.
Application Client Buffer Offset:	Offset in bytes from the beginning of the application client's buffer to the first byte of transferred data.
Request Byte Count:	Number of bytes to be moved by this request.

Confirmation:

> **Data-In Delivered (Task Identifier ||)**

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

5.3.3 Data-Out Delivery service

Request:

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

Argument Descriptions: See 5.3.2.

Confirmation:

  Data-Out Received (Task Identifier ||)

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

5.4 Task and command lifetimes

This clause specifies the events delimiting the beginning and end of a task or pending SCSI 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.

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 logical unit executes a logical unit reset operation as described in 5.6.7;
    d)   The task manager executes an ABORT TASK referencing the specified task; or
    e)   The task manager executes an ABORT TASK SET or a CLEAR TASK SET task management function directed to the task set containing the specified task.

An SCSI 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.

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)   Notification of a unit attention condition with one of the following additional sense codes:
        a)   COMMANDS CLEARED BY ANOTHER INITIATOR (if in reference to the task set containing the task);
        b)   POWER ON;
        c)   RESET; or
        d)   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 a CLEAR TASK SET task management function directed to the task set containing the specified task; or
    f)   A service response of FUNCTION COMPLETE in response to a 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 is received.

As discussed in 4.6.1, when an SCSI 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.

5.5 Command processing examples

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

5.5.1 Unlinked command example

An unlinked command is used to show the events associated with the processing of a single device service request (see figure 30). This example does not include error or exception conditions.
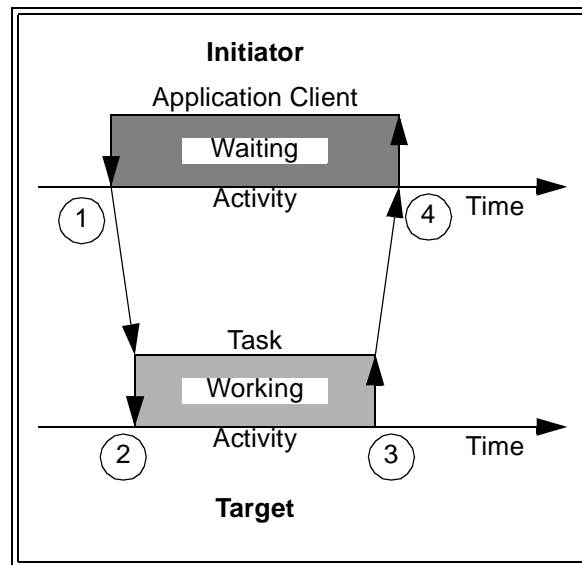
Figure 30 — Command processing events

The numbers in figure 30 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 TASK COMPLETE.

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

5.5.2 Linked command example

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

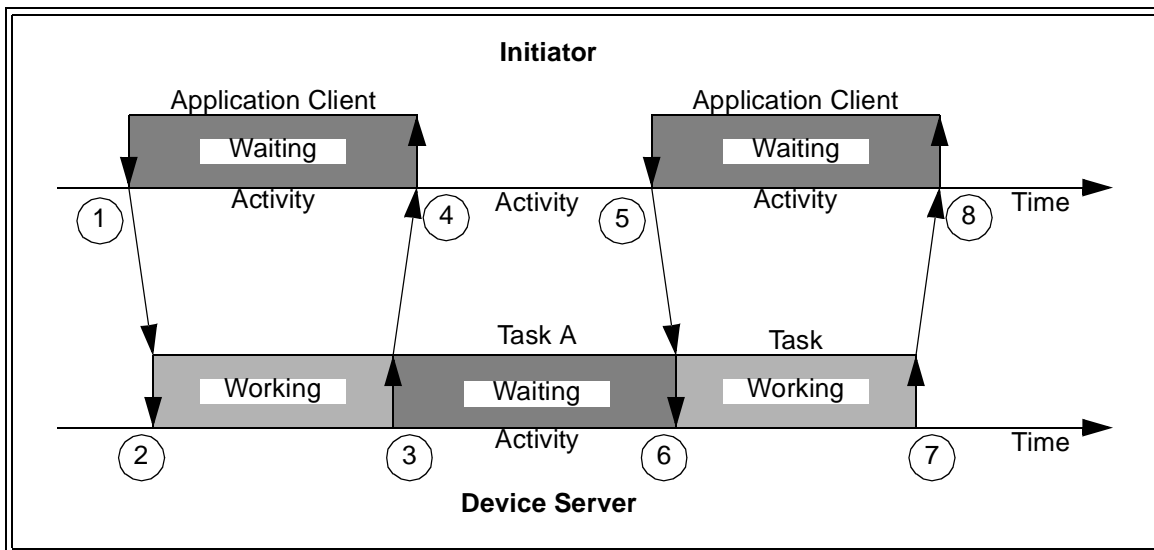The example in figure 31 shows the events in a sequence of two linked commands.

Figure 31 — Linked command processing events

The numbers in figure 31 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 5.1.2).

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

3. Upon completion of the first command, the device server invokes the **Send Command Complete** 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 **Send Command Complete** protocol service response of TASK COMPLETE, with status GOOD, is sent to the application client.

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

5.6 Command processing considerations and exception conditions

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

5.6.1 Auto Contingent Allegiance or Contingent Allegiance

The auto contingent allegiance (NACA=1, see 5.1.2) or contingent allegiance (NACA=0) condition shall exist within the task set when the logical unit completes a command by returning a CHECK CONDITION status (see 5.2).

5.6.1.1 Logical Unit response to Auto Contingent Allegiance or Contingent Allegiance

The auto contingent allegiance (NACA=1, see 5.1.2) or contingent allegiance (NACA=0) condition shall not cross task set boundaries and shall be preserved until it is cleared as described in 5.6.1.2. If requested by the application client and supported by the protocol and logical unit, sense data shall be returned as described in 5.6.4.2.

> Notes:
> 6   The SCSI-2 contingent allegiance condition has had an alternate added and the extended contingent allegiance condition has been replaced in SCSI by auto contingent allegiance in conjunction with the NACA bit.
> 7   If the SCSI protocol does not enforce state synchronization as described in 4.6.1, there may be a time delay between the occurrence of the auto contingent allegiance or contingent allegiance condition and the time 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.

As described in 5.6.1.2, the setting of the NACA bit in the CONTROL byte of the faulting command CDB determines the rules that apply to an ACA or CA condition caused by that command.

If the NACA bit was set to zero the SCSI-2 contingent allegiance rules shall apply.

If the NACA bit was set to one in the CONTROL byte of the faulting command, then a new task created by the faulted initiator while the ACA condition is in effect shall not be entered into the faulted task set unless all of the following conditions are true:

   a)   The task has the ACA attribute; and
   b)   No other task from the faulted initiator having the ACA attribute is in the task set.

If the task is from the faulted initiator and any of 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 (i.e., a contingent allegiance condition is in effect), 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 5.6.1.2, any existing contingent allegiance condition shall be cleared and a new auto contingent allegiance (NACA=1) or contingent allegiance (NACA=0) condition shall be established.

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

If TST=000b, tasks created by other initiators while the ACA or CA condition is in effect shall not be entered into the faulted task set (except for a PERSISTENT RESERVE command with a Preempt and Clear action as described in

5.6.1.2). Tasks rejected from the task set due to the presence of an ACA or CA condition shall be completed with a status of ACA ACTIVE (if NACA=1 in the new command's CDB CONTROL byte, see 5.1.2) or BUSY (if NACA=0).

If TST=001b, tasks created by one initiator shall not be rejected based on an ACA or CA condition in effect for another initiator. Only ACA or CA condition for the sending initiator (as well as other task set management considerations described in clause 7) shall affect acceptance into the task set or rejection for a task from that initiator.

5.6.1.2 Clearing an Auto Contingent Allegiance condition

If the NACA bit is set to zero in the CONTROL byte of the faulting command, then the SCSI-2 rules for clearing contingent allegiance shall apply. In addition, the logical unit shall clear the associated contingent allegiance condition upon sending sense data by means of the autosense mechanism described in 5.6.4.2.

While the SCSI-2 rules for clearing the CA 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 6.3).

If the logical unit accepts a value of one for the NACA bit and this bit was set to one in the CONTROL byte of the faulting command, then the SCSI-2 rules for clearing a contingent allegiance condition shall not apply. In this case, the ACA condition shall only be cleared:

   a) As the result of a power on or a logical unit reset (see 5.6.7);
   b) Through a CLEAR ACA task management function issued by the faulting initiator as described in 6.3;
   c) Through a Preempt and Clear action of a PERSISTENT RESERVE OUT command that clears the tasks of the faulting initiator (see the SPC-2 standard); or
   d) A command with the ACA attribute terminates with a CHECK CONDITION status.

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.

5.6.2 Overlapped commands

An overlapped command occurs when an application client reuses a Task Address (see 4.9.3) in a new command before a previous task to which that address was assigned completes its task lifetime as described in 5.4. Each SCSI 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.

> Notes:
> 8   An overlapped command may be indicative of a serious error and, if not detected, could result in corrupted data. This is considered a catastrophic failure on the part of the initiator. Therefore, vendor-specific error recovery procedures may be required to guarantee the data integrity on the medium. The target logical unit may return additional sense data to aid in this error recovery procedure (e.g., sequential-access devices may return the residue of blocks remaining to be written or read at the time the second command was received).
> 9   Some logical units may not detect an overlapped command until after the command descriptor block has been received.

5.6.3 Incorrect Logical Unit selection

The target's response to an incorrect logical unit identifier is described in the following paragraphs.

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 in item b below;

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

   In response to an INQUIRY command the target shall return the INQUIRY data with the peripheral qualifier set to the value required in the SPC-2 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;

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

   In response to an INQUIRY command the target shall return the INQUIRY data with the peripheral qualifier set to the value required in the SPC-2 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; or

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

   In response to an INQUIRY command the target shall return the INQUIRY data with the peripheral qualifier set to the value specified in the SPC-2 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.

5.6.4 Sense data

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

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

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

a) The REQUEST SENSE command specified in the SPC-2 standard;
b) An asynchronous event report; or
c) Autosense delivery.

The following clauses describe the last two transfer methods.

5.6.4.1 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 describe 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 clause, references to Asynchronous Event Reporting assume that the device to be notified has enabled asynchronous event reports from the target.) Support for asynchronous event reporting is a logical unit option.

> NOTE 10 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-2 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; or
d) an asynchronous event has occurred.

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

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

An example of 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 5.6.4).

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 an error condition encountered after command completion shall be returned only to the initiator that sent the affected task or tasks.

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

5.6.4.2 Autosense

Autosense is the automatic return of sense data to the application client coincident with the completion of an SCSI 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 protocol standard is optional.

As specified in clause 5, 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. After autosense data is sent, the sense data and the CA (NACA=0), if any, shall then be cleared. Autosense shall not affect ACA (NACA=1), see 5.6.1.

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 clause 5).

It is not an error for the application client to request the automatic return of sense data when autosense is not supported by the SCSI protocol or logical unit implementation. If the application client requested the return of sense data through the autosense facility and the 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.

5.6.5 Unit Attention condition

Each logical unit shall generate a unit attention condition whenever the logical unit has been reset as described in 5.6.6 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 logical unit inventory has been changed;
   g) The mode parameters in effect for the initiator have been restored from non-volatile memory;
   h) A change in the condition of a synchronized spindle; or
   i) Any other event requiring the attention of the initiator.

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

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

If an INQUIRY or a REPORT LUNS 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 or contingent allegiance

condition), the logical unit shall perform the command and shall not process the unit attention condition. If the unit attention condition was established in response to a change in the logical unit inventory, the unit attention condition shall be cleared for all logical units for the initiator that sent the REPORT LUNS command. In all other cases, the INQUIRY or REPORT LUNS command 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 auto contingent allegiance or 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 or contingent allegiance condition for the unit attention condition, the logical unit shall perform the second action listed above. If NACA for the REQUEST SENSE command is zero and the command is untagged the contingent allegiance condition shall be cleared.

If an initiator issues a command other than INQUIRY, REPORT LUNS, or REQUEST SENSE while a unit attention condition exists for that initiator (prior to generating the auto contingent allegiance or contingent allegiance condition for the unit attention condition), the logical unit shall not perform the command and shall report ACA ACTIVE (NACA=1, see 5.1.2) or BUSY (NACA=0) status.

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 5.6.4.1).

5.6.6 Target hard reset

A target 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 target reset events is protocol and interconnect specific. Each SCSI protocol standard shall specify the response to a target reset event including the conditions under which a target hard reset shall be executed.

To execute a hard reset a target shall initiate a logical unit reset for all attached logical units as described in 5.6.7.

5.6.7 Logical Unit reset

A logical unit reset is a response to a LOGICAL UNIT RESET task management request (see 6.5), or a some other logical unit reset event, such as a target hard reset (see 5.6.6). The definition of such events may be device-specific or dependent on the protocol and interconnect. Each appropriate SCSI standard shall specify the conditions under which a logical unit reset shall be executed.

To execute a logical unit reset the logical unit shall:

    a)   Abort all tasks in its task set(s);
    b)   Clear an auto contingent allegiance (NACA=1, see 5.1.2) or contingent allegiance (NACA=0) condition, if one is present;
    c)   Release all reservations established using the reserve/release management method (persistent reservations shall not be affected);
    d)   Return the device's operating mode to the appropriate initial conditions, similar to those conditions that would be found following device power-on. The MODE SELECT parameters (see the SPC-2 standard) shall be restored to their last saved values if saved values have been established. MODE SELECT parameters for which no saved values have been established shall be returned to their default values;

e) Set a unit attention condition (see 5.6.5); and

f) Initiate a logical unit reset for all dependent logical units (see 4.10.4).

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

6 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 task manager response indicating that the requested function is complete. The task manager shall unconditionally return this response upon completion of a task management request supported by the logical unit or 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 task management functions are summarized as follows (see the clauses below for detailed definitions of each task management function):

ABORT TASK (Task Address ||) - Abort the task identified by the **Task Address** parameter (see 4.9.3). 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.

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 a NACA bit value of one in the CDB CONTROL byte (see 5.1.2).

CLEAR TASK SET (Logical Unit Identifier ||) - Abort all tasks in the specified task set. This function shall be supported by all logical units, except in the following cases, when support for this function is optional:

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

LOGICAL UNIT RESET (Logical Unit Identifier ||) - Perform a logical unit reset as described in 5.6.7 by termi-nating all tasks in the task set(s) and propagating the reset to all dependent logical units (see 3.1.22). Support for this function is mandatory for hierarchical logical units (see 4.10.4) and may be supported by non-hierarchical logical units**.**

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

Argument descriptions:

           Target Identifier: Target device identifier defined in 4.7.2.

    Logical Unit Identifier: Logical Unit identifier defined in 4.8.

            Task Address: Address address defined in 4.9.3.

> NOTE 11 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.

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

6.1 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.

6.2 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 which 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. A contingent allegiance (NACA=0) shall be cleared by the ABORT TASK SET function.

6.3 CLEAR ACA

Function Call

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

Description:

This function shall only be implemented by a logical unit that accepts a NACA bit value of one in the CDB CONTROL byte (see 5.1.2).

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 5.6.1.2. If successful, this function shall 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.

6.4 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 4.9) and may be supported by logical units that do not support tagged tasks.

If the TST field equals 000b in the Control mode page (see SPC-2), the target shall perform an action equivalent to receiving a series of ABORT TASK requests from each initiator. If the TST field equals 001b the target shall perform an action equivalent to receiving a series of ABORT TASK requests from only the requesting initiator.

All tasks in the appropriate task set shall be aborted. The medium may have been altered by partially executed commands. All pending status and sense data for the appropriate task set shall be cleared.

No status shall be sent for any task. A unit attention condition shall be generated for all other initiators with aborted tasks (if any). 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, reservations, and auto contingent allegiance (NACA=1, see 5.1.2) shall not be changed by the CLEAR TASK SET function. A contingent allegiance (NACA=0) shall be cleared by the CLEAR TASK SET function.

6.5 LOGICAL UNIT RESET

Function Call:

> **Service Response = LOGICAL UNIT RESET (Logical Unit Identifier ||)**

Description:

This function shall be supported by all logical units that support hierarchical logical units (see 4.10.4) and may be supported by non-hierarchical logical units.

Before returning a FUNCTION COMPLETE response the logical unit shall perform the logical unit reset functions specified in 5.6.7. A unit attention condition for all initiators shall be created on each logical unit as specified in 5.6.5.

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 target hard reset functions specified in 5.6.6. A unit attention condition for all initiators shall be created on each logical unit as specified in 5.6.5.

6.7 Task management protocol services

The confirmed service described in this clause 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.
Initiator Identifier:	See 4.7.1.
Function Identifier:	Parameter encoding the task management function to be performed.

All SCSI protocol standards shall define the protocol-specific requirements for implementing the Send Task Management Request 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 protocol standard is optional. All SCSI I/O systems shall implement these protocols as defined in the applicable protocol specification.

The argument definitions correspond to those of clause 6.

Request sent by application client:

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

Indication received by task manager:

> **Task Management Request Received (Initiator Identifier, 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:

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

Confirmation received by application client:

> **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.

6.8 Task management function example

Figure 32 shows the sequence of events associated with a task management function.
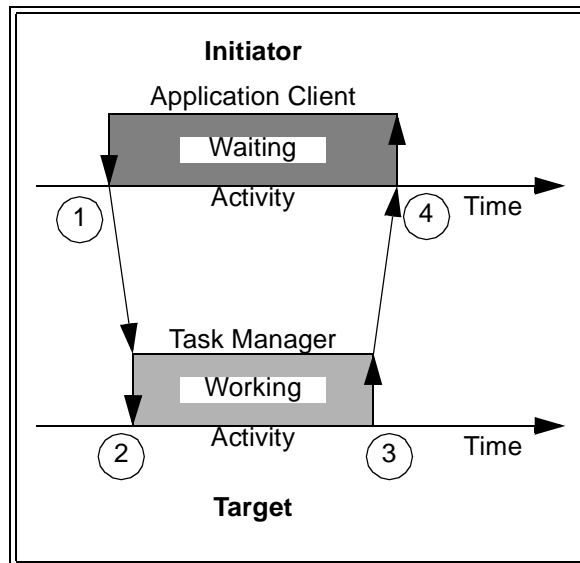
Figure 32 — Task management processing events

The numbers in figure 32 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** 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 **Received Function-Executed** confirmation is received by the application client.

7 Task Set Management

This clause describes some of the controls application clients have over task set management behaviors (see 7.2). This clause also specifies task set management requirements in terms of task states (see 7.5), task attributes (see 7.6), the events that cause task state transitions (see 7.3 and 7.4), and a map of task state transitions (see 7.7). The clause concludes with several task set management examples (see 7.8).

Task behavior, as specified herein, refers to the functioning of a task as observed by an application client within the initiator -- including the results of command execution and interactions with other tasks. Examples of behavior not observable by the application client are the physical activity on the interconnect or the format of transmitted data packets associated with a command. To define these and other aspects of behavior, SCSI protocol and interconnect standards may impose other requirements, outside the scope of this standard, which are related to observable behavior within the protocol or interconnect layers.

The rules for task set management only apply to a task after it has been entered into a task set. A task shall be entered into a task set unless a condition exists which causes that task to be completed with a status of BUSY, RESERVATION CONFLICT, TASK SET FULL, ACA ACTIVE or CHECK CONDITION (if caused by the detection of an overlapped command). A task may also be completed because of a CHECK CONDITION status caused by certain protocol-specific errors.

7.1 Terminology

The following definitions are used extensively in this clause.

7.1.1 suspended information: Information within the logical unit that is not available to a pending task.

7.1.2 current task: A task that has a data transfer protocol service request in progress (see 5.3.1) or is in the process of sending command status. Each SCSI protocol standard shall define the protocol-specific conditions under which a task is considered a current task.

7.1.3 pending task: Any task that is not a current task.

7.2 Controlling task set management

The Control mode page (see SPC-2) contains fields that specify particular task set management behaviors. The standard inquiry data CmdQue bit (see SPC-2) indicates support for tagged tasks (command queuing). One specific combination of task set management behaviors is identified as the basic task management model. Support for the basic task management model is indicated by values returned in the CMDQUE and BQUE bits in the standard inquiry data (see SPC-2). The basic task management model requires the following task set management behaviors:

   a) The only task attribute supported shall be SIMPLE;
   b) The device server may reorder the actual execution sequence of tasks in any manner. Any data integrity exposures related to task sequence order shall be explicitly handled by the application client using the appropriate commands;
   c) All the blocked tasks in the task set shall be aborted after an ACA or CA condition is cleared;
   d) It shall not be possible to disable tagged queuing; and
   e) Support for the CLEAR TASK SET task management function is optional.

7.3 Task management events

The following describe the events that drive changes in task state.

All older tasks ended:	All tasks have ended that were accepted into the task set earlier in time than the referenced task.
All older Head of Queue and older Ordered tasks ended:	All Head of Queue and Ordered tasks have ended that were accepted into the task set earlier in time than the referenced task.
ACA:	An auto contingent allegiance condition has occurred (NACA=1, see 5.1.2).
CA:	An contingent allegiance condition has occurred (NACA=0).
task abort:	One of the events described in 7.4 has occurred.
task completion:	The device server has sent a service response of TASK COMPLETE for the task (see clause 5 and 5.4).
task ended:	A task has completed or aborted.
ACA cleared:	An ACA condition has been cleared.
CA cleared:	An CA condition has been cleared.

7.4 Task Abort events

A Task Abort event is one of the following:

a) Completion of an ABORT TASK task management function directed to the specified task;
b) Completion of an ABORT TASK SET task management function under the conditions specified in 6.2;
c) Completion of a CLEAR TASK SET task management function referencing the task set containing the specified task;
d) Completion of a PERSISTENT RESERVE with a Preempt and Clear action directed to the specified task;
e) An ACA or CA condition was cleared and the QERR field was set to 01b or 11b in the control mode page (see the SPC-2 standard);
f) An ACA condition was cleared and the task had the ACA attribute;
g) A logical unit reset (see 5.6.7);
h) The return of an **Execute Command** service response of SERVICE DELIVERY OR TARGET FAILURE as described in clause 5; or
i) A power on condition.

7.5 Task states

The next several clauses identify and describe the states in the task state model. Following them, is a description of two task time-lines involving three of the four states.

7.5.1 Enabled

A task in the Enabled state may become a current task and may complete at any time, subject to the task completion constraints specified in the Control mode page (see the SPC-2 standard). A task that has been accepted into the task set shall not complete or become a current task unless it is in the enabled state.

Except for the use of target resources required to preserve task state, a task shall produce no effects detectable by the application client before the task's first transition to the Enabled state. Although, before entering this state for the first time, the task may perform other activities visible to lower layers -- such as pre-fetching data to be written to the media -- this activity shall not result in a detectable change in device state as perceived by an application client. In addition, the behavior of a completed task, as defined by the commands it has executed, shall not be affected by the task's states before it became enabled.

7.5.2 Blocked

A task in the Blocked state is prevented from completing due to an auto contingent allegiance or contingent allegiance condition. A task in this state shall not become a current task. While a task is in the Blocked state, any information the logical unit has or accepts for the task shall be suspended. If the TST field in the Control mode page (see SPC-2) equals 000b the blocked state is independent of the initiator. If the TST field equals 001b the blocked state applies only to the faulted initiator.

7.5.3 Dormant

A task in the Dormant state is prevented from completing due to the presence of certain other tasks in the task set. A task in this state shall not become a current task. While a task is in the Dormant state, any information the logical unit has or accepts for the task shall be suspended.

7.5.4 Ended

A task in the Ended state is removed from the task set.

7.5.5 Task states and task lifetimes

Figure 33 shows the events corresponding to two task execution sequences. Except for the Dormant state between times A and B in case 1, logical unit conditions and the commands executed by the task are identical.

Assuming in each case the task completes with a status of GOOD at time C, the system state observed by the application client for case 1 shall be indistinguishable from the state observed for case 2.
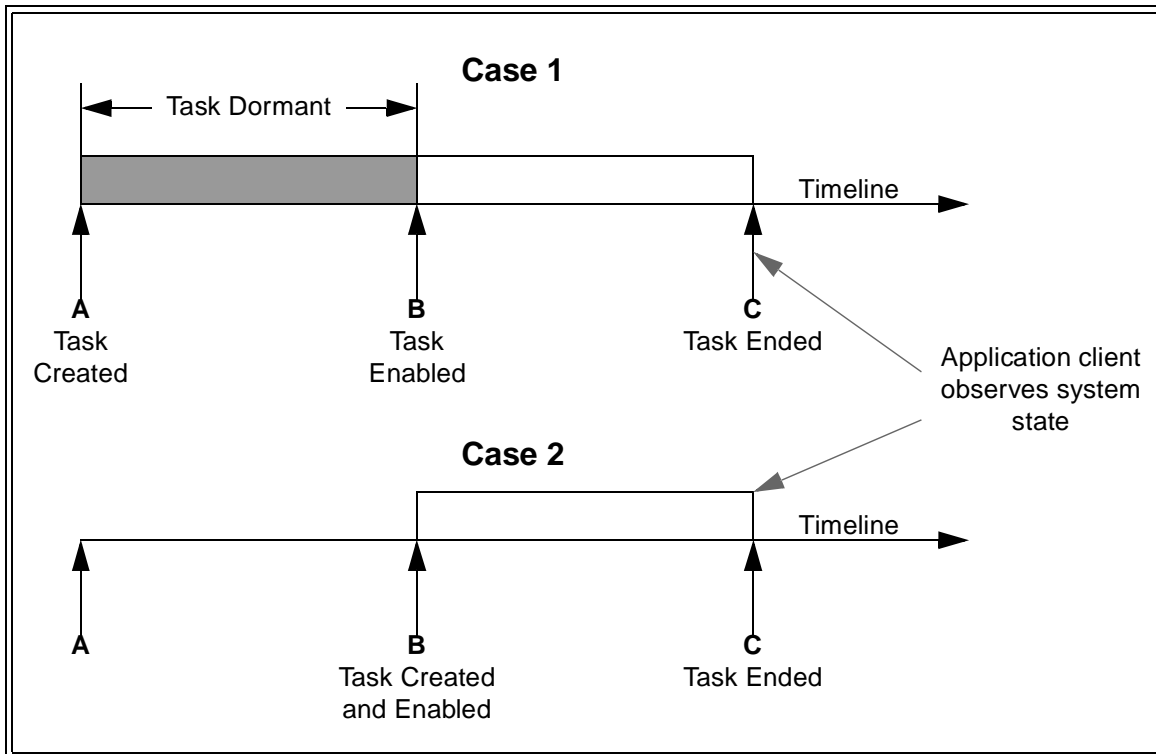

Figure 33 — Example of Dormant state task behavior

7.6 Task Attributes

A task shall have one of the attributes defined below.

7.6.1 SIMPLE Task

A task having the Simple attribute shall be accepted into the task set in the Dormant state. The task shall not enter the Enabled state until all older Head of Queue and older Ordered tasks in the task set have ended (see 7.3).

7.6.2 ORDERED Task

A task having the Ordered attribute shall be accepted into the task set in the Dormant state. The task shall not enter the Enabled state until all older tasks in the task set have ended (see 7.3).

7.6.3 HEAD OF QUEUE Task

A task having the Head of Queue attribute shall be accepted into the task set in the Enabled state.

7.6.4 ACA Task

A task having the ACA attribute shall be accepted into the task set in the Enabled state. There shall be no more than one ACA task per task set (see 5.6.1.1).

7.7 Task state transitions

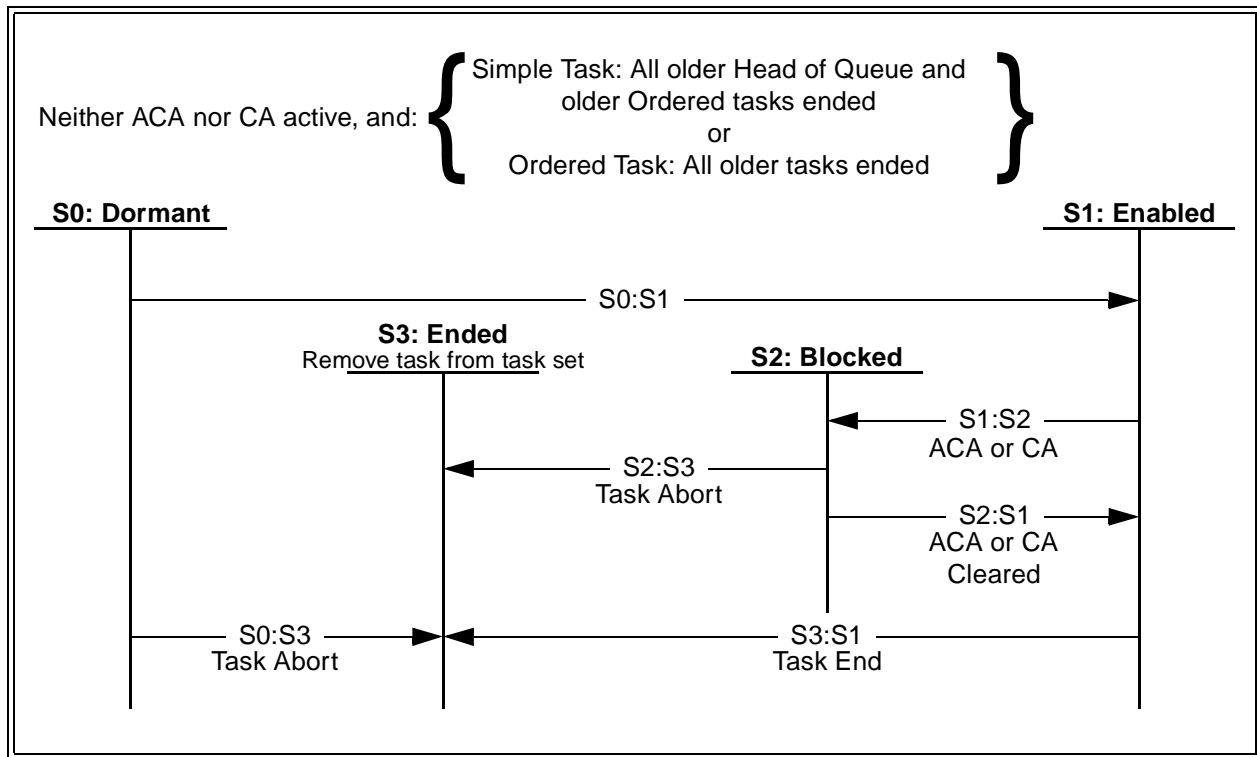The task state diagram of figure 34 shows the behavior of a single task in response to an external event.

Figure 34 — Task states

The following clauses describe task state transitions, actions and associated triggering events as they appear to an application client. Although the logical unit response to events affecting multiple tasks, such as a CLEAR TASK SET, may be different from the response to an event affecting a single task, from the viewpoint of the application client the collective behavior appears as a series of state changes occurring to individual tasks.

In the discussion below, "dormant task" refers to a task in the Dormant state, "enabled task" to a task in the Enabled state, and so forth.

7.7.1 Transition S0:S1 (Ordered Task): Provided an ACA or a CA condition does not exist or if TST=001b in the Control mode page (see SPC-2) provided the task is not for the faulted initiator and the QE RR field is not 01b in the Control mode page, a dormant task having the ORDERED attribute shall enter the Enabled state when all older tasks have ended. If TST=000b in the Control mode page, this transition shall not occur while an ACA or a CA condition is in effect for any initiator. If TST=001b in the Control mode page, this transition shall not occur while an ACA or a CA condition is in effect for the initiator that created the ordered task.

7.7.2 Transition SO:S1 (Simple task): Provided an ACA or a CA condition does not exist or if TST=001b in the Control mode page (see SPC-2) provided the task is not for the faulted initiator, a dormant task having the SIMPLE attribute shall enter the Enabled state when all older Head of Queue and older Ordered tasks have ended. If TST=000b in the Control mode page, this transition shall not occur while an ACA or a CA condition is in effect for any initiator. If TST=001b in the Control mode page, this transition shall not occur while an ACA or a CA condition is in effect for the initiator that created the simple task.

7.7.3 Transitions S0:S3, S2:S3: A task abort event shall cause the task to unconditionally enter the Ended state.

7.7.4 Transition S1:S2: If TST=000b in the Control mode page, an ACA or a CA condition shall cause an enabled task to enter the Blocked state. If TST=001b in the Control mode page, an ACA or a CA condition shall cause an enabled task for the faulted initiator to enter the Blocked state.

7.7.5 Transition S1:S3: A task that has completed or aborted shall enter the Ended state. This is the only state transition that applies to an ACA task.

7.7.6 Transition S2:S1: When an ACA or a CA condition is cleared and the QERR field is 00b in the Control mode page, a task in the Blocked state shall re-enter the Enabled state. When an ACA or a CA condition is cleared and the QERR field is 11b in the Control mode page, a task in the Blocked state for other than the faulting initiator shall re-enter the Enabled state.

7.8 Task set management examples

The following clauses give several task set management scenarios. The examples are valid for single or multi-initiator cases, when TST=000b. In this case, the interaction among tasks in a task set is independent of the initiator originating a task. The examples are also valid for a single initiator, when TST=001b. In this case, task set management proceeds independently for each initiator and the events and transitions in one initiator's task set do not affect the task set management for another initiator's task set. Throughout these examples, the scope of the task set box drawn in each snapshot depends on the setting of the TST field in the Control mode page (see SPC-2).

The figure accompanying each example shows successive snapshots of a task set after various events, such as task creation or completion. In all cases, the constraints on task completion order established using the QUEUE ALGORITHM MODIFIER field and DQUE bit in the Control mode page (see SPC-2) are not in effect.

A task set is shown as an ordered list or queue of tasks with the head of the queue towards the top of the page. A new Head of Queue task always enters the task set at the head, displacing older Head of Queue tasks. Simple, Ordered and ACA tasks always enter the task set at the end of the queue.

Tasks, denoted by rectangles, are numbered in ascending order from oldest to most recent. Fill, shape and line weight are used to distinguish task states and attributes as follows:

Task attributes:

   a)   Simple tasks -- rounded corners;
   b)   Ordered -- square corners and thin lines;
   c)   Head of Queue -- square corners and thick lines; or
   d)   ACA tasks -- square corners and thin dashed lines.

Task states:

   a)   Enabled -- no fill;
   b)   Dormant -- grey (50 percent fill); or
   c)   Blocked -- black.

7.8.1 Blocking boundaries

The conditions preventing a dormant task from becoming enabled (except for ACA and CA conditions) are shown by means of "blocking boundaries". Such boundaries appear as horizontal lines with an arrow on both ends. The text below the line identifies the tasks causing the barrier condition. A task is impeded by the barrier if it is between the boundary and the end of the queue. When no ACA or CA is in effect, a task enters the Enabled state after all intervening barriers have been removed.

7.8.2 Head of Queue tasks

Figure 35 shows task set conditions when several Head of Queue tasks are executed.
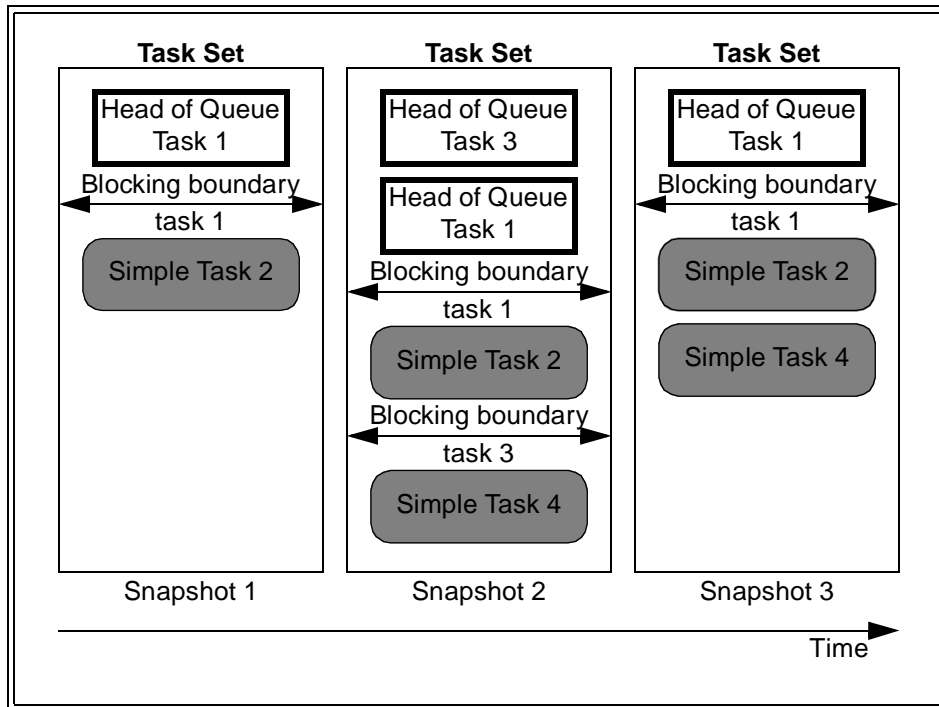

Figure 35 — Head of Queue tasks and blocking boundaries (example 1)

In snapshot 1 the task set initially contains one Head of Queue and one Simple task. As shown by the blocking boundary, simple task 2 is Dormant because of the older Head of Queue task. Snapshot 2 shows the task set after Head of Queue task 3 and Simple task 4 are created. The new Head of Queue task is placed at the front of the queue in the Enabled state, displacing task 1. Snapshot 3 shows the task set after task 3 completes. Since the conditions indicated by the task 1 blocking boundary are still in effect, tasks 2 and 4 are held in the Dormant state.

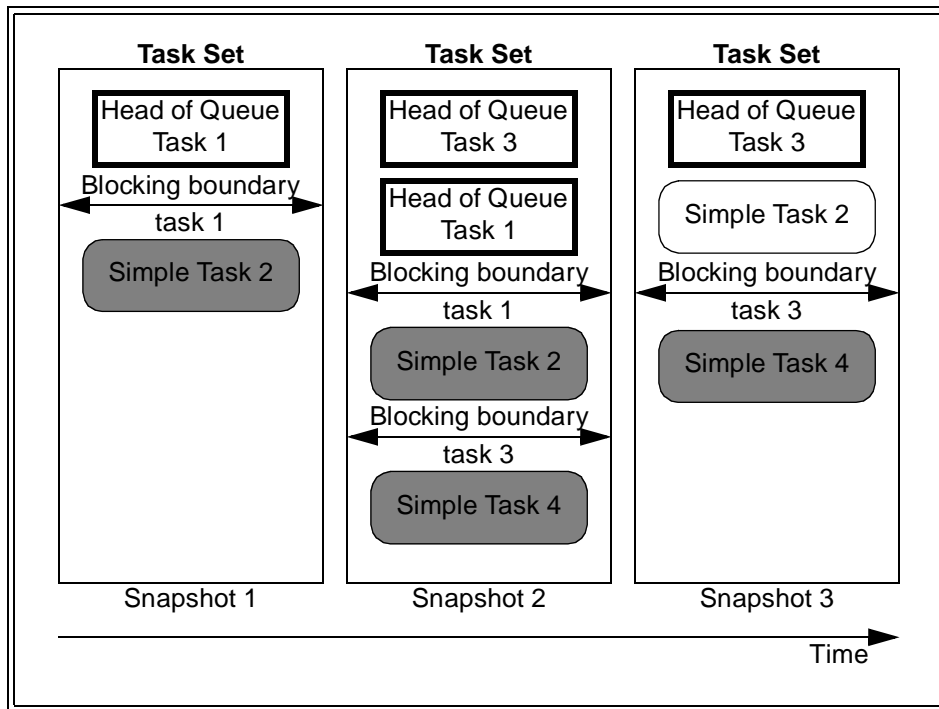Figure 36 is the same as the previous example, except that task 1 completes instead of task 3.

Figure 36 — Head of Queue tasks and blocking boundaries (example 2)

The completion of task 1 allows task 2 to enter the Enabled state. Simple task 4 is held in the Dormant state until task 3 completes.

7.8.3 Ordered tasks

An example of Ordered and Simple task interaction is shown in figure 37.
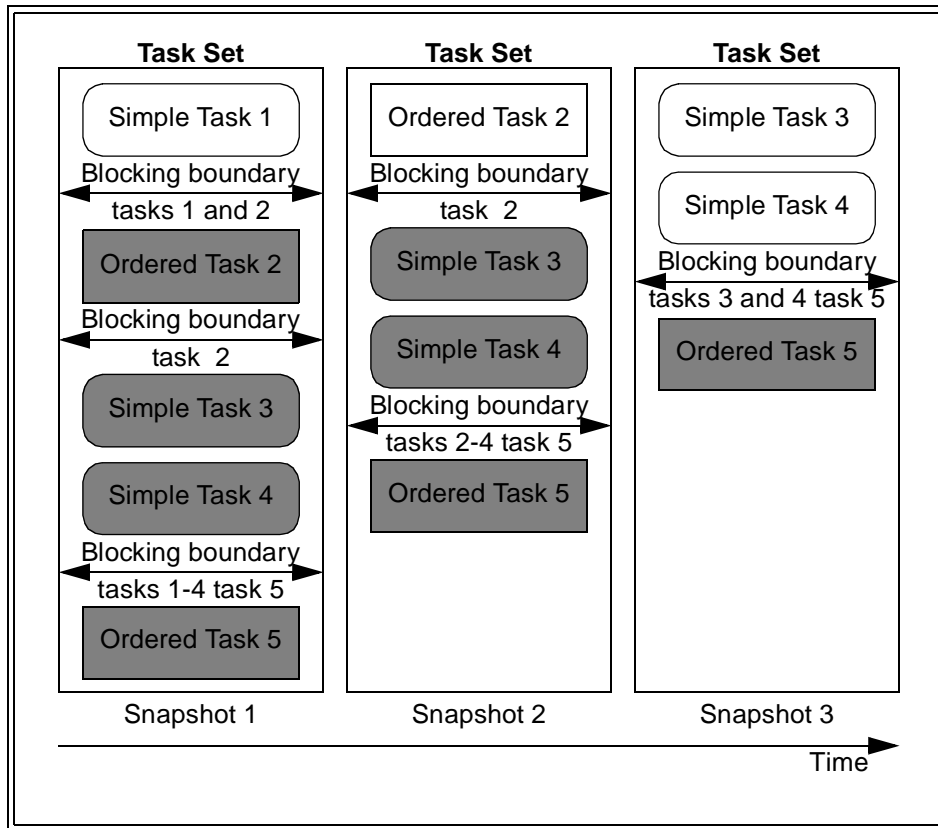

Figure 37 — Ordered tasks and blocking boundaries

The state of dormant tasks 2 through 5 is determined by the following rules:

Tasks 2 and 5 -- An Ordered task cannot enter the Enabled state until all older tasks have ended.
Tasks 3 and 4 -- A Simple task cannot enter the Enabled state until all older Head of Queue and older Ordered tasks have ended.

These constraints are shown by the blocking boundaries in snapshot 1.

In snapshot 2, the completion of task 1 allows ordered task 2 to become Enabled. Since the initial constraints on tasks 3, 4 and 5 are still in effect, these tasks must remain Dormant. As shown in snapshot 3, the completion of task 2 triggers two state changes: -- namely, the transitions of task 3 and task 4 to the Enabled state. Task 5 must be held in the Dormant state until these tasks end.

7.8.4 ACA task

Figure 38 shows the effects of an ACA condition on the task set. This example assumes the QE$_{RR}$ field is set to 00b in the Control mode page (see the SPC-2 standard). Consequently, clearing an ACA condition will not cause tasks to be aborted.
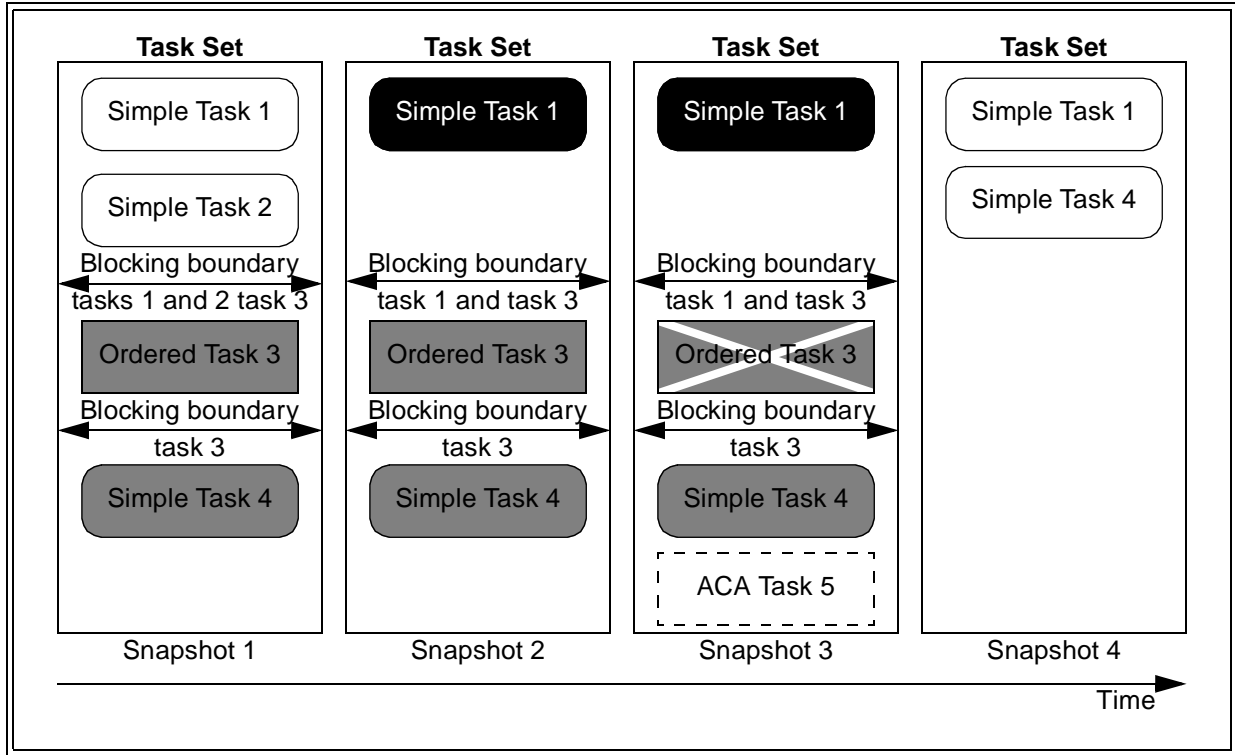

Figure 38 — ACA task example

The completion of task 2 with CHECK CONDITION status causes task 1 to enter the Blocked state shown in snapshot 2. In snapshot 3, Ordered task 3 is aborted using the ABORT TASK task management function and ACA task 5 is created to perform additional handling for the exception. Once the ACA condition is cleared, (snapshot 4) Simple task 1 can reenter the Enabled state. Since there are no Head of Queue or Ordered tasks older than task 4, it too can be placed in the Enabled state.