ENDL
TEXAS

Date: 22 June 2000
To: T10 Technical Committee
From: Ralph O. Weber
Subject: SAM-2 Task Identifier Readability Enhancements and Addition of Nexus to SAM-2

While reviewing FCP-2, I discovered a problem in the SAM-2 definition of Task Identifier. The problem adversely affects protocol standards such as SPI-4 and FCP-2 when they attempt to define a mapping between SAM-2 terms and protocol specific entities. As things stand right now, a protocol standard cannot show a mapping to the Task Identifier object with wording that will read sensibly.

There is an additional, less severe, problem with the SAM-2 object called Logical Unit Identifier, the actual definition of the object is not what most people think of when the name 'Logical Unit Identifier' is mentioned. While it is possible to describe the Logical Unit Identifier object sensibly in protocol standards, the lack of an intuitive definition for Logical Unit Identifier forces careful review and consideration in order to get the definitions right.

r0 of this proposal presented the problems and suggested solutions.

r1 proposed a collection of specific changes, including the addition of a clause describing the SCSI nexus concept and relating nexus to other SAM-2 objects. Once introduced, it is proposed that the nexus concept (object) be used at several convenient locations in SAM-2, most particularly in the definitions of the task set management functions. N.B. Preparation of this proposal included a review SCSI-2 to ensure minimal changes between the proposed nexus usage and the SCSI-2 usage.

r2 contains corrections and enhancements proposed by the May working group. r2 also includes the changes needed to meet the new routine prototype notation for input and output parameters.

Review of the Current Object Definitions

SAM-2 (and SAM) define several objects for identifying tasks:

Targets use
  Task Identifier
    Untagged Task Identifier = Initiator Identifier + Logical Unit Identifier
    Tagged Task Identifier = Untagged Task Identifier + Tag

Initiators use
  Task Address
    Untagged Task Address = Logical Unit Identifier
    Tagged Task Address = Untagged Task Identifier + Tag

See SAM-2 clauses 4.9.2 and 4.9.3 (PDF page 49 in sam2r13.pdf).

Note that the Target's Task Identifier requires an Initiator Identifier while the Initiator's Task Address does not require a Target Identifier. This is because the Logical Unit Identifier is defined to include the Target Identifier, as follows:

Logical Unit Identifier = Target Identifier + Logical Unit Number

See SAM-2 clauses 4.8 (PDF page 48 in sam2r13.pdf).

Inspection of the SAM-2 task management function definitions shows that the Logical Unit Identifier definition is more than a little fortuitous. By compounding the Target Identifier and Logical Unit Number in a single object, most of the task set management function definitions require only one argument. (Note: other compound objects are defined for other cases.)

The Problems

Replacing the Logical Unit Identifier object with its definition in the Target's Task Identifier definition we see that:

Task Identifier
Untagged Task Identifier = Initiator Identifier + Target Identifier + Logical Unit Number
Tagged Task Identifier = Untagged Task Identifier + Tag

That is, the target (and a protocol's description of a target's operation) is required to define a Task Identifier in terms of both the initiator and target identifiers. FCP-2 got this wrong, and might very well have an arduous task getting it right. As far as I can tell, there is no reason to require Task Identifier to have the definition currently in SAM-2 (and SAM). From a target's perspective, Initiator Identifier plus Logical Unit Number is a sufficient identifier for a task.

Furthermore, everybody or nearly everybody equates Logical Unit Identifier with Logical Unit Number, whereas Logical Unit Identifier is really a bookkeeping object defined to allow convenient constructions for the definitions of the task set management functions. It is even possible to view confusion between the Logical Unit Identifier and Logical Unit Number definitions during the development of SAM as the source of the task identifier problems.

Summary of Proposed Changes

The proposed changes can be broken down to six groups:

1) Make Untagged Task Identifier = Target Identifier + Logical Unit Number (removing the Initiator Identifier);
2) Add Nexus as SAM-2 object, with necessary definitions and a clause devoted to describing the relationships between nexus and the existing SAM-2 identifier objects;
3) Use the nexus object to simplify and/or clarify existing SAM-2 addressing requirements, particularly for the task set management functions;
4) Remove the Logical Unit Identifier object and change uses of the Logical Unit Identifier object to either uses of the nexus object or uses of Target Identifier + Logical Unit Number (which ever is clearer);
5) Other potentially clarifying changes noticed while preparing this proposal;
6) Changes to instantiate the new input/output notation for routine prototypes.

Unless otherwise stated, all references are to SAM-2 revision 13 and sam2r13.pdf. The clause number or page number order in which the following change affect SAM-2 has not been considered in establishing the order in which the changes are described. Changes are described in an order that the author believes makes understanding the effects of the changes easier.

Specific Proposed Changes for SAM-2

Change 1.1, Group(2): Add a clause describing the Nexus object between the current clauses 4.9 and 4.10. The text of the added clause to be as follows:

4.x The Nexus object

The Nexus object is a relationship between and initiator and a target that is viewed from the perspective of the SCSI Domain (as opposed to being viewed from either the target's perspective or the initiator's perspective). Knowledge of the perspective from which that Nexus is being viewed is required in order to determine the actual physical meaning of a Nexus.

The Nexus object has additional flexibility in that, depending on the context in which it is used, a Nexus may refer to any one or all of the following relationships:

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

When the Nexus object appears without a qualifier, then any one of the relationships may apply and further understanding of the context is required to determine which one or several of the relationships apply. When the Nexus object is qualified (as in I_T_L nexus), then the specific named relationship applies.

The SCSI Architecture Model uses the Nexus object because it has the flexibility to convey complex relationships for the presentation of general concepts. Yet when properly qualified, the Nexus object can be very specific. Table xx relates the Nexus object to several identifier objects presented elsewhere in this standard.

Table xx — Mapping I_T_L_Q to SAM-2 Identifiers

Nexus	Identifiers That Form Nexus			
	From Initiator Perspective		From Target Perspective	
	Single Port Model	Optional Multiple Port Model	Single Port Model	Optional Multiple Port Model
I_T	Target Identifier	[Initiator] Port Identifier Target Identifier	Initiator Identifier	Initiator Identifier [Target] Port Identifier
I_T_L	Target Identifier Logical Unit Number	[Initiator] Port Identifier Target Identifier Logical Unit Number	Initiator Identifier Logical Unit Number	Initiator Identifier [Target] Port Identifier Logical Unit Number
I_T_L_Q	Target Identifier Logical Unit Number Tag	[Initiator] Port Identifier Target Identifier Logical Unit Number Tag	Initiator Identifier Logical Unit Number Tag	Initiator Identifier [Target] Port Identifier Logical Unit Number Tag

Target Identifier is defined in 4.7.2. Initiator Identifier is defined in 4.7.1. Port Identifier is defined in 4.10.2 for targets and 4.9.3 for initiators. Logical Unit Number is defined in 4.8. Tag is defined in 4.9.1.

Change 1.2, Group(2): Add the following glossary definitions

3.1.n1 nexus: a relationship between an initiator and a target that is viewed from the perspective of the SCSI Domain (see 4.x).

3.1.n2 I_T nexus: a nexus between an initiator and a target (see 4.x).

3.1.n3 I_T_L nexus: a nexus between an initiator, a target, and a logical unit (see 4.x).

3.1.n4 I_T_L_Q nexus: a nexus between an initiator, a target, a logical unit, and a tagged task (see 4.x).

3.1.n5 I_T_L_x nexus: either an I_T_L nexus or an I_T_L_Q nexus (see 4.x).

Change 1.3, Group(4): In 4.8 (Logical Units on pdf page 48), delete the following:

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.

Change 1.4, Group(4): In 4.9 (Tasks on pdf page 48), include the I_T_L_x object in the definition of a Task by change text as described below.

In the first paragraph change from:

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).

to (changes identified by strikeouts and underlines):

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) and an I_T_L_Q Nexus (see 4.x). 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), an I_T_L Nexus and implicitly a SIMPLE task attribute (see 7.6).

Change 1.5, Group(4): In 4.9 (Tasks on pdf page 48), change text as described below. Note: this is one example of the problem of misusing the Logical Unit Identifier object described in the introductory paragraphs of this proposal.

Change from:

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.

to (changes identified by strikeouts and underlines):

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 identifier and logical unit number ~~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.

Change 1.6, Group(1 & 4): In 4.9.2 (Target identification of tasks pdf page 49) replace all instances of Logical Unit Identifier with Logical Unit Number, with the result being as shown below (each instance of a replacement has the new text underlined).

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 Number (see 4.8) and a Tag (see 4.9.1). An Untagged Task Identifier is composed of an Initiator Identifier and a Logical Unit Number.

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 Number (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 Number.

Change 1.7, Group(4): In 4.9.3 (Initiator identification of tasks on pdf page 49) replace all instances of Logical Unit Identifier with Target Identifier and Logical Unit Number, with the result being as shown below (each instance of a replacement has the new text underlined).

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 Target Identifier (see 4.7.2), Logical Unit Number (see 4.8) and a Tag (see 4.9.1). An Untagged Task Address is composed of a Target Identifier and a Logical Unit Number.

Change 1.8, Group(4): In 5.6.3 (Incorrect Logical Unit Selection on pdf page 77), replace all instances of Logical Unit Identifier with Logical Unit Number, with the result being as shown below (each instance of a replacement has the new text underlined). Note: this is another example of the problem of misusing the Logical Unit Identifier object described in the introductory paragraphs of this proposal.

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

The logical unit number may be incorrect because:

Change 1.9, Group(3, 5 & 6): In clause 6 (Task Management Functions on pdf page 82), change Object Identifier to Nexus in the general task management function procedure call format, with the result being as shown below (the new text is underlined).

Service response = Function name (input(Nexus [,Input-1] [,Input-2-]), output([Output-1] [,Output-2] …))

Also, there are no task management functions that have input or output parameters; the nexus is the only parameter for all task management functions. This situation is likely to continue for at least as long as the SCSI Architecture supports the parallel bus. Therefore, simplification of the procedure call format by removal of the input

and output parameters is recommended. Why confuse SAM-2 readers with model elements that are not required by the model? Following removal of the input/output parameter text, the procedure call format would look like:

Service response = Function name (input(Nexus))

Change 1.10, Group(3, 5 & 6): In clause 6 (Task Management Functions on pdf page 82), change the first parameter of every task management function procedure call example to the appropriate qualified nexus (shown with underlined text below), and change any wording in the descriptions that names the parameter (shown with strikeouts and underlines). Also, in keeping with the second clarification suggested in change 1.9, the notation indicating the absence of input and output parameters has been removed from the procedure call formats (shown with strikeouts). The combined results of all these changes are as follows.

ABORT TASK (input(I_T_L_Q Nexus)~~)~~ - Abort the identified task. ~~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 (input(I_T_L Nexus)~~)~~ - Abort all tasks in the task set for the I_T_L nexus~~requesting initiator~~. This function shall be supported by all logical units.

CLEAR ACA (input(I_T_L Nexus)~~)~~ - 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 (input(I_T_L Nexus)~~)~~ - 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 (input(I_T_L Nexus)~~)~~ - Perform a logical unit reset as described in 5.6.7 by terminating 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 (input(I_T Nexus)~~)~~ - Reset the target device and terminate all tasks in all task sets. All target devices shall support this function.

Change 1.11, Group(3): In clause 6 (Task Management Functions on pdf page 83), change the argument descriptions to from the identifier notation to the nexus notation. (Note how this change uses the nexus object to clarify the general task set function definition by enhancing the list of argument descriptions with an easily referenced concept.)

Change:

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.

to:

Nexus: A general initiator-target nexus (see 4.x).

I_T Nexus: An initiator and target nexus (see 4.x).

I_T_L Nexus: An initiator, target, and logical unit nexus (see 4.x).

I_T_L_Q Nexus: An initiator, target, logical unit, and tag nexus (see 4.x).

Change 1.12, Group(3, 5 & 6): In 6.1 through 6.6 (descriptions of each task set management function on pdf pages 83 through 86), change each task set function procedure format to match the equivalent format shown in change 1.10. The results of these changes are not shown in this proposal because the necessary review information can be found in change 1.10.

The description text in all clauses 6.1 through 6.6 has been reviewed for additional changes resulting from the use of the nexus object(s) and no requirements for text changes have been found. Reviewers of this proposal may wish to verify this for themselves.

Change 1.13, Group(3, 5 & 6): Rewrite 6.7 (pdf pages 85 and 86) to use the nexus object and to clarify other statements (such as the first sentence of the clause, which describes the clause's contents too narrowly). The rewritten clause is as follows (strikeouts and underlines are used to show changes).

6.7 Task management protocol services

The ~~confirmed service~~ protocol services described in this clause is used by an ~~application client to issue~~ initiator and target to process 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.~~

Nexus: A general initiator-target nexus (see 4.x).

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 an initiator and application client to a target's task manager:

~~**Send Task Management Request (Object Identifier, Function Identifier ||)**~~

Send Task Management Request (input(Nexus, Function Identifier))

Indication received by the task manager:

~~**Task Management Request Received (Initiator Identifier, Object Identifier, Function Identifier ||)**~~

Task Management Request Received (input(Nexus, Function Identifier))

Note that the nexus perspective has changed from the initiator's perspective in the **Send Task Management Request** request to the target's perspective in the **Task Management Request Received** indication (see 4.x). However, the same qualified nexus is maintained through all steps of task set management transaction. That is, if the **Send Task Management Request** uses an I_T_L nexus, then the **Task Management Request Received** and all subsequent protocol functions also use an I_T_L nexus.

Response from task manager to initiator and application client:

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

Task Management Function Executed (input(Nexus, 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 ||)**~~

Received Function-Executed (input(Nexus, Service Response))

Note that the nexus perspective has again changed from the target's perspective in the **Task Management Function Executed** response to the initiator's perspective in the **Received Function-Executed** confirmation.

Since the ~~object identifier~~ nexus ~~does not~~ may 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~~ I_T_L nexus.

Optional Proposed Changes for SAM-2

You will notice that no proposal has been made yet to change the protocol functions defined for processing SCSI commands to use nexus. The SCSI command protocol functions in SAM-2 revision 13 are consistently defined using the Task Address and Task Identifier objects, have no reliance on the Logical Unit Identifier object, make no use of the Object Identifier object (a confusing hand wave that is extremely close to Nexus in meaning), and as such are not in serious need of changes to use the Nexus object.

For these reasons, the following set of changes are put to the committee as optional. If these changes are made in addition to those above, then the use of the various Initiator, Target, etc. Identifier objects will be confined to Clause 4, which may be viewed as an improvement.

For the SCSI-2 purists, it is important to note that the I_T_L_x Nexus object is new to this proposal. The nearest SCSI-2 equivalent is the I_T_x_y nexus, which was defined as an I_T_L, I_T_R, or I_T_L_Q nexus. Since the I_T_R nexus is obsolete, it seems reasonable to substitute the I_T_L_x nexus for the I_T_x_y nexus.

N.B. if these changes are not adopted, the list entry d) can be removed from the new clause proposed in change 1.1. If these changes are not adopted, the routine prototype changes for inputs and outputs (group 6) still must be made.

Change 2.1, Group(3 & 6): In clause 5 (SCSI Command Model on pdf page 64), replace two occurrences of Task Address with I_T_L_x Nexus and adjust descriptive text appropriately. The result of the changes is as follows (with new text underlined).

Service response = Execute Command (<u>input(I_T_L_x Nexus</u>, CDB, [Task Attribute], [Data-Out Buffer], [Command Byte Count], [Autosense Request])<u>, output(</u>[Data-In Buffer], [Sense Data], Status)

Input Arguments:

<u>I_T_L_x Nexus</u>: either an I_T_L nexus or an I_T_L_Q nexus (see 4.x).

Change 2.2, Group(3): In last paragraph of clause 5 (SCSI Command Model on pdf page 65), replace task identifier with I_T_L_x Nexus (note: 'task identifier' probably was incorrect because 'task address' was the function argument, but these issues are rendered moot by the use of the nexus object). The result of the changes is as follows (with new text underlined).

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 **<u>I_T_L_x Nexus</u>** ~~task identifier~~.

Change 2.3, Group(3 & 6): Rewrite 5.3 (pdf pages 69 and 70) to use the I_T_L_x nexus object. The rewritten clause is as follows (strikeouts and underlines are used to show changes).

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] ||)**~~

Send SCSI Command (<u>input(I_T_L_x Nexus</u>, CDB, [Task Attribute], [Data-Out Buffer], [Command Byte Count], [Autosense Request]))

Protocol Service Indication:

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

SCSI Command Received (<u>input(I_T_L_x Nexus</u>, [Task Attribute], CDB, [Autosense Request]))

<u>Note that the **I_T_L_x Nexus** perspective has changed from the initiator's perspective in the **Send SCSI Command** request to the target's perspective in the **SCSI Command Received** indication (see 4.x).</u>

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

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

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

Send Command Complete (input(I_T_L_x Nexus, [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 ||)**~~

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

Note that the **I_T_L_x Nexus** perspective has again changed from the target's perspective in the **Send Command Complete** response to the initiator's perspective in the **Command Complete Received** confirmation.

Change 2.4, Group(3 & 6): Rewrite 5.3.2 and 5.3.3 (pdf pages 71 and 72) to use the I_T_L_x nexus object. The rewritten clauses are as follows (strikeouts and underlines are used to show changes).

5.3.2 Data-In Delivery Service

Request:

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

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

Argument descriptions:

~~**Task Identifier:** See 4.9.2.~~

I_T_L_x Nexus: Either an I_T_L nexus or an I_T_L_Q nexus (see 4.x).

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 ||)**~~

Data-In Delivered (input(I_T_L_x Nexus))

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 ||)**~~

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

Argument Descriptions: See 5.3.2.

Confirmation:

~~**Data-Out Received (Task Identifier ||)**~~

Data-Out Received (input(I_T_L_x Nexus))

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

Change 2.5, Group(3): In 5.6.2 (Overlapped Commands on pdf page 76), **do not change** the following first sentence of the clause. A change to nexus usage probably would be wrong in that the nexus relationship can be used over and over again during the processing of a single command. (N.B. not changing this text ensures that clauses 4.9.2 and 4.9.3 cannot be removed from SAM-2.)

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.

Suggested Changes for FCP-2

Since problems coordinating FCP-2 and SAM-2 initiated developing this proposal, it seems only appropriate that some changes be suggested for FCP-2. Adoption of this proposal should not constitute a requirement that the FCP-2 editor incorporate these changes. Changes to FCP-2 should be handled under the letter ballot and comments resolution process (with letter ballot comments reviewed in 00-150). Notation with each change proposal indicates which letter ballot comments the change addresses.

There is a note from the May working group: "Unused part of the nexus are ignored" near change 3.2 but it's not clear how that note should have been applied in that or any of the nearby changes.

No attempt has been made to verify that this proposal lists all the sites needing the routine prototype changes for inputs and outputs.

All pdf page references are to fcp2r04.pdf.

Change 3.1, Group(3 & 4): Delete the glossary definition for 'logical unit identifier' (3.1.25 pdf page 19) and the following definitions (addresses ENDL-7).

3.1.n1 nexus: a relationship between and initiator and a target that is viewed from the perspective of the SCSI Domain (see SAM-2).

3.1.n2 I_T nexus: a nexus between an initiator and a target (see SAM-2).

3.1.n3 I_T_L nexus: a nexus between an initiator, a target, and a logical unit (see SAM-2).

3.1.n4 I_T_L_Q nexus: a nexus between an initiator, a target, a logical unit, and a tagged task (see SAM-2).

Change 3.2, Group(3 & 4): In 9.1.1.3 (Task Attribute on pdf page 53), replace logical unit identifier in the description of the **UNTAGGED** attribute with a nexus reference (addresses ENDL-7). Note: these two changes represent the only occurrences of 'logical unit identifier' in FCP-2 revision 4.

Change from:

> Only one untagged task can exist for each logical unit identifier / initiator pair. Requesting a second untagged command for the same logical unit identifier / initiator pair shall be treated as an overlapped command.

to:

> Only one untagged task can exist for each I_T_L nexus. Requesting a second untagged command for the same I_T_L nexus shall be treated as an overlapped command.

Change 3.3, Group(1, 3 & 4): Here's what I think A.1 (Definition of procedure terms on pdf page 87) needs to contain in place of table A.1 and the sentence that introduces it (addresses ENDL-38, ENDL-39, ENDL-40, ENDL-41, and ENDL-42).

See table A.1 for the mapping of the addressing and frame identifiers used in this standard to the equivalent addressing identifier objects found in the SCSI Architecture Model-2 standard.

Table A.1 — FCP-2 identifiers mapped to SAM-2 identifier objects

FCP-2 addressing and frame identifiers	**SAM-2 addressing identifier objects**
address identifier of initiator port (see 3.1.2)	initiator identifier
address identifier of target port (see 3.1.2)	target identifier
fully qualified exchange identifier (see 3.1.20)	tag **[1]**
fully qualified exchange identifier (see 3.1.20) + logical unit number (see 3.1.26)	task address (initiator usage) or task identifier (target usage)
Notes: [1] Until the target has assigned an RX_ID for the received FCP_CMND IU, the OX_ID is the tag.	

See table A.2 for the mapping of addressing and frame identifiers used in this standard to the nexus objects found in the SCSI Architecture Model-2 standard.

Table A.2 — FCP-2 identifiers mapped to SAM-2 nexus objects

FCP-2 Addressing and Frame Identifiers That Form Nexus		**SAM-2 Nexus**
From Initiator Perspective	From Target Perspective	
address identifier of target port (see 3.1.2)	address identifier of initiator port (see 3.1.2)	I_T
address target of initiator port (see 3.1.2) logical unit number (see 3.1.26)	address identifier of initiator port (see 3.1.2) logical unit number (see 3.1.26)	I_T_L
fully qualified exchange identifier (see 3.1.20) + logical unit number (see 3.1.26)	fully qualified exchange identifier (see 3.1.20) + logical unit number (see 3.1.26)	I_T_L_Q

Change 3.4, Group(3 & 6): Replace A.2 (Application client SCSI command services on pdf page 89) with the following.

A.2 Application client SCSI command services

The SCSI command services shall be requested by the application client using a procedure call defined as:

Service response = Execute Command (**input(**I_T_L_Q nexus, command descriptor block, [task attribute], [data-out buffer], [command byte count]), output([data-in buffer], status)).

In FCP-2, the I_T_L_Q nexus is used regardless of whether the command is tagged or untagged. That is, a tag is always present in FCP-2, but the command still may be untagged.

Change 3.5, Group(3): In A.3, A.4, A.4.1, and A.4.2 (pdf pages 89 and 90), replace every instance of 'fully qualified exchange identifier + logical unit number' with 'I_T_L_Q nexus'. This affects tables A.3, A.4 and A.5 as well as the text in the body of clause A.4. The effects of these changes are not shown but they are nearly identical to the effects shown in change 3.4.

Change 3.6, Group(3 & 5): Having dug deeply in this topic now, it is clear that A.5 (Task management services on pdf pages 90 and 91) needs a complete rewrite. So, replace clause A.5 with the following (addresses ENDL-53, ENDL-54, ENDL-55, and ENDL-56).

A.5 Task management services

A.5.1 Application client view of task management services

The task management services shall be requested from the application client using a procedure call defined as:

Service response = Function name (input(Nexus))

Nexus is defined in 3.1.n1. The Service response shall be as defined in SAM-2.

The task management functions are summarized as follows (see the clauses below for detailed definitions of each task management function):

ABORT TASK (input(I_T_L_Q nexus)) - Abort the identified task. All FCP-2 devices shall support this function.

ABORT TASK SET (input(I_T_L nexus)) - Abort all tasks in the task set for the requesting initiator. All FCP-2 devices shall support this function.

CLEAR ACA (input(I_T_L nexus)) - 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 SAM-2).

CLEAR TASK SET (input(I_T_L nexus)) - Abort all tasks in the specified task set. All FCP-2 devices shall support this function.

LOGICAL UNIT RESET (input(I_T_L nexus)) - Perform a logical unit reset as described in SAM-2 by terminating all tasks in the task set(s) and propagating the reset to all dependent logical units (see SAM-2). Support for this function is mandatory for hierarchical logical units (see SAM-2) and may be supported by non-hierarchical logical units.

TARGET RESET (input(I_T nexus)) - Reset the target device and terminate all tasks in all task sets. All FCP-2 devices shall support this function. In FCP-2, a Target Reset task management function must be addressed to an I_T_L nexus even though it affects an I_T nexus.

A.5.2 FCP-2 implementations for task management services

A.5.2.1 ABORT TASK

The Abort Task task management function requests the sending of an ABTS ELS using the fully qualified exchange identifier (see 3.1.20).

A.5.2.2 ABORT TASK SET

The Abort Task Set task management function requests the sending of an FCP_CMND IU with the ABORT TASK SET bit set to one in TASK MANAGEMENT FLAGS field (see 9.1.1.4).

A.5.2.3 CLEAR ACA

The Clear ACA task management function requests the sending of an FCP_CMND IU with the CLEAR ACA bit set to one in TASK MANAGEMENT FLAGS field (see 9.1.1.4).

A.5.2.4 CLEAR TASK SET

The Clear Task Set task management function requests the sending of an FCP_CMND IU with the CLEAR TASK SET bit set to one in TASK MANAGEMENT FLAGS field (see 9.1.1.4).

A.5.2.5 LOGICAL UNIT RESET

The Logical Unit Reset task management function requests the sending of an FCP_CMND IU with the LOGICAL UNIT RESET bit set to one in TASK MANAGEMENT FLAGS field (see 9.1.1.4).

A.5.2.6 TARGET RESET

The Target Reset task management function requests the sending of an FCP_CMND IU with the TARGET RESET bit set to one in TASK MANAGEMENT FLAGS field (see 9.1.1.4).

A.5.3 Task Management Protocol Services

Processing of a Task Set Management procedure call shall be composed of a four step confirmed service as shown in table A.x.

Table A.x — Processing of Task Set Management procedure

Step	Protocol Service Name	FCP-2 Service Interface procedure call
request	Send Task Manage-ment Request	Send Task Management Request (nexus, function identifier)
indication	Task Management Request Received	Task Management Request Received (nexus, function identifier)
response	Task Management Function Executed	Task Management Function Executed (nexus, service response)
confirmation	Received Function-Executed	Received Function-Executed (nexus, service response)

For all task set management functions except Abort Task, the request is communicated from the initiator to the target via an FCP_CMND IU (where it becomes the indication) and the response is communicated from the target to the initiator via an FCP_RESP IU (where it becomes the confirmation). The service response parameter is the contents of the RSP_CODE field in the FCP_RESP IU. If the FCP_RESP IU does not include the RSP_CODE field then the service response is function complete.

For the Abort Task task management function, the request is communicated from the initiator to the target via an ABTS ELS (where it becomes the indication) and the response is communicated from the target to the initiator via an either a BA_ACC ELS or a BA_RJT ELS (where it becomes the confirmation). If the response is communi-cated in a BA_ACC ELS then the service response is function complete. If the response is communicated in a BA_RJT then the service response is contained in the Reason Code and Reason Explanation fields.