CONGRUENT SOFTWARE, INC.
3998 Whittle Avenue
Oakland, CA 94602
(510) 531-5472
(510) 531-2942 FAX

FROM: Peter Johansson

TO: NCITS T10

DATE: March 21, 1998

RE: More SBP-2 Revision 3 Comment Resolution

This memorandum addresses the letter ballot comments on SBP-2 Revision 3 not previously resolved by 98-109r2 and SBP-2 Revision 3a. Those two previously published documents concern themselves with editorial issues, only. The technical issues were reserved for separate discussion.

One comment submitted by Steve Finch, that all or most of the isochronous material in SBP-2 be deleted, does not lend itself to the response format used in both 98-109r0 and this document. A separate response may be found in 98-120r0.

The comment responses are organized in the same order as the draft standard, not in order by submitter. When responses span multiple comments or are complex, the proposed changes to SBP-2 are shown separately at the end of this document.

The first revision of this document was discussed at the SBP-2 *ad hoc* working group meeting in San Diego on March 17. Some changes were agreed and are incorporated in this revision. With these changes, the working group recommends to the T10 plenary that this document be accepted as resolution of the referenced technical comments.

General

The following comments apply to the draft standard as a whole, not to a particular section.

Reference	Ballot comment	Proposed resolution
GM 61	Since all the other SCSI standards use ISO/IEC rules for decimal values and any desired ISO/IEC standard for SBP-2 will use the ISO/IEC rules, I suggest SBP-2 be revised to include and conform with those rules at this time (e.g. '24.576 MHz' changes to 24,576 MHz'). Be sure to add the conventions statement to avoid US 24 GHz implementations.	I've discussed this with some other document editors and do not find strong consistency within T10 documents prior to their internationalization. My personal preference is to retain United States usage in an ANSI document and make the change later for ISO/IEC—but I refer this question to the working group or plenary for discussion and resolution.

References, definitions and model

The comments below apply to sections 2 through 4, inclusive, of the draft standard.

Reference	Ballot comment	Proposed resolution
SF 4	Section 2 lists normative references. Is there a method of listing informative references? I think we should add this to cover cases such as the reference in Annex G to the "1394 Trade Associate Specification for AV/C Digital Interface Command Set, Version 2.0, April 22, 1997". Note that this document reference is now out of date.	The style guides published by ANSI do not provide a method to list informative references. I'll consult with the ANSI editor to see if we could title section 2 "Normative and other references" and accommodate this request. If ANSI cannot accommodate such a request I will create an informative annex (bibliography) for these references.
SF 6	Section 3.1.2 The term "initiator" is used extensively through out this document and is not defined. Add a definition for initiator. I suggest the following definition: "3.1.2.x initiator: A device containing application clients that originate commands to be processed in a target."	Added the following definition: "**initiator:** A node that originates device service or task management requests and signals these requests to a target for processing."
SF 7	Section 3.1.2 The term "request" is used extensively through out this document. This term has specific a meaning in IEEE 1394-1995 which is different from the usage in this document. Add a definition for "request" and for a second term "request subaction". I suggest the following definitions: "3.1.2.x request: An ORB constructed by an initiator for processing by a target. The term request is not a request packet as defined by IEEE 1394-1995." "3.1.2.x request subaction: A request packet as defined by IEEE 1394-1995."	After a review of SBP-2 I believe that clarity and readability would not be increased by making the exacting distinctions proposed. The SAM and SAM-2 documents are examples of how excessive formalism can decrease the value of the text. I think the distinction between various types of requests (and there are more than just the one suggested by the first definition at left) and a Serial Bus request subaction are clear from context. In some cases I have made the "request subaction" explicit, since that is a formal term with a narrow meaning. I've added the following to the definitions: "**request subaction:** A packet transmitted by a node (the requester) that communicates a transaction code and optional data to another node (the responder) or nodes."

Reference	Ballot comment	Proposed resolution
SF 8	Section 3.1.2 The term "response" is used extensively through out this document. This term has specific a meaning in IEEE 1394-1995 which is different from the usage in this document. Add a definition for "response" and for a second term "response subaction". I suggest the following definitions: "3.1.2.x response: An block of data constructed by a target in response to a request which is stored in the initiators system memory by the target. Login Response and Status are examples of responses." The term response is not a response packet as defined by IEEE 1394-1995" "3.1.2.x response subaction: A response packet as defined by IEEE 1394-1995."	See the response to comment SF 7; an analogous argument holds here. I've added the following to the definitions: "**response subaction:** A packet transmitted by a node (the responder) that communicates a response code and optional data to another node (the requester). A response subaction may consist of either an acknowledge packet or a response packet."
SF 18	Section 3.1.2.28: the use of the term "stored" is not clear. Wherever "store" is used, replace with "is written to the initiator by the target *via* a write quadlet request or a write block request" or "is written to the target by the initiator *via* a write quadlet request or a write block request". It might be easier to include a definition of "store".	Added the following definition: "**store:** When any form of this verb is used in the context of data transferred by the target to the system memory of either an initiator or other device, it indicates both the use of Serial Bus write request subaction(s), quadlet or block, to place the data in system memory and the corresponding response subaction(s) that complete the write(s)."
SF 5	Section 3.1.2 The term "target" is used extensively through out this document and is not defined. Add a definition for target. In some places in the document target is used when node is meant. I have tried to note these in my comments. I suggest the following definition: "3.1.2.x target: A device containing logical units that receive and execute commands from one or more initiators. A target is a unit as defined in CSR architecture and Serial Bus terminology."	Added the following definition: "**target:** A node that receives device service or management requests from an initiator. In the case of device service requests, the commands are directed to one of the target's logical units to be executed. Management requests are serviced by the target. A CSR Architecture unit is synonymous with a target."
GM 25	The definition 'A task has a lifetime, which commences when the task is signaled to the target, proceeds through a period of execution by the target and finishes when completion status is stored at the initiator. While a task is active, it makes use of both target resources and initiator resources.' is very clear but my understanding of a task was that it began when a nexus or I/O operation was entered into the task set.	Change part of the definition from "…commences when the task is signaled to the target…" to "…commences when the task is entered into the target's task set…". A nexus is a parallel SCSI concept not applicable to SBP-2.

Reference	Ballot comment	Proposed resolution
GM 26	The abbreviations should include msb and lsb and the use of MSB and LSB as in all other SCSI standards should be considered.	In Figure 1, *msb* and *lsb* are used to label fields for description in the paragraph below; they are not used as formal abbreviations anywhere in the SBP-2 draft. I don't think they need be defined. Also, I scanned for instances of "significant" and found most apply to bits but few apply to the singular most or least significant bit.
SF 34	Section 4.7.2, the last paragraph states: "Stream controller actions may be queued by the target. This permits time-critical operations to be specified in advance and avoids latency problems that could arise if the stream controller could accept no more than one request at a time. Within the queue of requests to the stream controller, each is executed in order as the preceding stream control ORB completes." While I agree that this may be a valid implementation, I don't think it has been established that this is the only implementation nor that it is the implementation desired when using isochronous streams. Delete this paragraph or at least the last sentence of the paragraph. The remainder of this section stands by itself.	Independent of the other comments, recall that all of section 4 is informative. With respect to ordered execution of stream control ORB's, this is the model supported by SBP-2. Out-of-order execution of these requests is not meaningful within the standard as it is defined today. Of all the sentences in the paragraph, it is the last that is, perhaps, most important. The preamble is helpful because it advises implementers that they may need to prefetch ORB's to sustain the performance of their device.

Data structures

The following comments apply to section 5 of the draft standard.

Reference	Ballot comment	Proposed resolution
GM 43	In clause 5 change 'The *node_ID* field shall specify the Serial Bus node for which the address pointer is valid, as defined by IEEE Std 1394-1995.' to 'The *node_ID* field shall contain the Serial Bus node for which the address pointer is valid, as defined by IEEE Std 1394-1995.'	Change the text to "The *node_ID* field shall identify…"

Reference	Ballot comment	Proposed resolution
SF 43	Section 5.1.2.2, the *cm* field and the associated text that follows are intertwined in a way that I find confusing. I believe that this field should be divided into two fields: *cycle_mark_start* and *stream_start*. If *cycle_mark_start* is one, then *cycle_mark_offset* specifies the location of the first CYCLE MARK as an offset, in quadlets, relative..., else the location of the CYCLE MARK, even if present, is undefined. If *stream_start* is one, the *cycle_mark_offset* specifies the location of the first quadlet of the isochronous data as an offset, in quadlets, relative..., else the offset is zero and the data begins with the first data byte specified by the command. I see no reason to tie these two conditions together and I can visualize a case where the stream offset is non-zero and there is no CYCLE MARK to be pointed to.	There are three combinations of cycle mark offset and stream offset that are meaningful: a) Cycle marks are not present and the stream offset can range from zero to less than *stream_length*; b) In the first stream command block ORB, the stream offset is nonzero and coincident with a cycle mark; and c) In all other stream command block ORB's, the stream offset is zero and the cycle mark offset locates a the first cycle mark within the data. In this way the two fields are intertwined. The last part of the comment is correct; I have updated the table (see 5.1.2.2 at the end of this document to permit nonzero stream offsets when cycle marks are absent.
SF 44	Section 5.1.2.2, general comment: if a cycle offset mark is specified, I think it MUST be limited to pointing to a location within the first block of a block oriented device. If a device has 512 byte blocks, then the maximum value of the offset is 127 (quadlets). Currently, there is no restriction on the value in this field, so if the command transfers 256 blocks (32k quadlets), the offset could be a large number. This makes the target's life unduly complicated. If the above isn't acceptable, at least limit the value of the *cycle_mark_offset* field to be less than or equal to the *stream_length*.	I agree it is pointless to permit the stream offset to specify a location outside of the first block—the target will read and discard data unnecessarily. The following has been added to the description of stream offset: "For a block device, the stream offset, expressed in bytes, shall be less than the block size of the device." In cases without block sizes, I've also restricted *cycle_mark_offset*: "The *cycle_mark_offset* field, when *cm* has a value of two or three, specifies the location of the first CYCLE MARK packet as an offset, in quadlets, relative to the starting medium location indicated by the *command_block*. When *cm* has a value of one, *cycle_mark_offset* specifies the stream offset instead. In either case, the value of *cycle_mark_offset*, converted to bytes, shall be less than *stream_length*."

Reference	Ballot comment	Proposed resolution
SF 47	Section 5.1.3, in the paragraph describing the STOP control function: if a stream which is set up for recording (listening) is stopped its data is "made available to the isochronous commands previously enqueued at the stream command block agent." (In the above, isochronous commands should be changed to Stream Command ORB's.) There is no statement, here or elsewhere, that this data must written to the media nor that the current stream command ORB should be considered complete and a status returned to indicate a partial write, nor that any commands currently pending on the stream command linked list should be discarded or completed with an error. In other words, how does one gracefully stop a stream? Similarly, if a device is talking, the text says the buffers are flushed, but what of the stream command ORB's still pending? Are these commands, *i.e.* the task set, also flushed? In both cases, should the stream command block agent be transitioned to the dead state?	The relevant descriptions of the STOP control function have been expanded to read "If the target had been listening, any isochronous data already received from Serial Bus shall be made available to the stream commands previously queued at the stream command block agent. Unless prevented by other errors, the device server shall transfer all flushed isochronous data to the medium in accordance with the queued stream commands. Completion status for the stream command that completes the transfer of isochronous data shall be stored at the initiator's status_FIFO and shall reflect the length of the data transfer. Unexecuted stream commands that remain in the task set shall be aborted. If the target had been talking, the stream command task set shall be aborted and any untransmitted isochronous data obtained from the stream commands shall be discarded. The state of the stream command fetch agent is unaffected by the STOP control function."
SF 50	Section 5.1.3: why do we have both a Channel Mask and a Configure Channels? If we had only a Configure Channel function and added a one bit field for this function, we could accomplish the same thing. In addition, this would allow changing the channel mask functionality when the stream is not stopped.	The reason for segregating the two functions was to permit "live" channel enable / disable but to require channel set-up or tear-down to occur only while the stream is stopped. In discussions about expected target capabilities, the latter seemed less likely to be supported on-the-fly.
SF 53	Section 5.1.3: SET ERROR MODE and the *rpt* field are insufficient. A general concept was established within SBP-2 that errors cause: 1) the associated fetch engine to go to the dead state, and 2) the task set to be flushed. Errors in the isochronous case are not exempted from this. The SET ERROR MODE command and the *rpt* field appear to change this for ALL errors. For the list of errors in section 12.3, I believe we want a granularity. For example, we might want to ignore missing CYCLE START packets and Data CRC errors, but cause fatal errors if the other errors listed occur.	The "general" SBP-2 concept applies to normal task sets and stream command task sets, not to stream control functions. I have had numerous discussions with designers of consumer applications who have indicated that these error modes are precisely the ones needed. If more granularity were needed in future, it could always be added.

Reference	Ballot comment	Proposed resolution
SF 56	Section 5.1.4, Table 2: delete the entry for TERMINATE TASK, make the value Reserved for future standardization. Section 10.2 states "Targets shall support, at a minimum, a basic task management model.", section 10.4.6 states "Targets that implement the basic task management model shall not support TERMINATE TASK…". This means that TERMINATE TASK shall not be supported by any SBP-2 device. Why are we defining a command that can not be used? Remove this command from the table and delete section 10.4.6 and any references in SBP-2 to TERMINATE TASK through out this standard including any reference in any changes I may have recommended.	TERMINATE TASK may be supported by an SBP-2 device if that device implements other than the basic task management model (see the *q* bit in configuration ROM). I've added the following editorial text to 10.2 to clarify this relationship: "Targets may implement other task management models so long as they support all of the features of the basic model (see the *q* bit in 7.4.10)."
PJ p. 33	The paragraph below Figure 21 should specify that the *notify* bit shall be one.	Because the management agent is a single-threaded server, any value for *notify* other than one is of questionable usefulness. This new requirement should be accepted.
SF 57	Section 5.1.4, first paragraph below Figure 21: add the following sentence: "The notify bit shall be set to one by the initiator and the target need not check the notify bit and shall always return a status for any completed management ORB." I can find no method that an initiator can determine if a management command completed if a target should decide to not return a status when a command completes without error. The above behavior must be mandatory.	
SF 58	(N) Section 5.1.4, Table 2: I believe we need to allow command sets the freedom to define an additional management command if such a command is required by the functionality of the device type. Change "5-6 Reserved for future standardization" to two entries "5 Command set specific" and "6 Reserved for future standardization".	Based upon experience with the SET PASSWORD function and upon conversations with OS implementers, I think that command set-dependent functions within the management ORB's are contra-indicated.
PJ p. 35	The paragraph below Figure 22 hasn't been corrected to reflect the inclusion of Annex C on password security.	The *password* information, when present, is no longer command-set dependent but part of SBP-2. Strike "~~command set-dependent~~" and replace "~~beyond the scope of this standard~~" with "specified by Annex C".
SF 59	Section 5.1.4.1, first paragraph after Figure 22: states that the usage of password data is beyond the scope of this standard, yet Annex C (Normative) defines it. This is inconsistent.	

Reference	Ballot comment	Proposed resolution
SF 60	Section 5.1.4.1, second paragraph after Figure 22: add "The *login_response_length* shall be set to a value of 12 by the initiator and may be ignored by the target." The length of the login response is fixed and it make no sense to allow the initiator to set a different value nor cause the target device to check for a minimum length.	Added the sentence "The initiator shall set *login_response_length* to a value of at least 12; the target may ignore this field."
SF 63	Section 5.1.4.1, Figure 23 show the length field to be a value of 12, and in the paragraph below Figure 23 change the paragraph to read "The length field shall specify the length, in bytes, of the login response data and shall be 12." The login response data is useless unless complete and there is no reason to force checking for odd conditions.	Accepted. Also deleted the last sentence of the same paragraph.
SF 65	Section 5.1.4.2, first paragraph after Figure 25, move the last sentence to be the second sentence and then add the following sentences after the last: "The target shall limit the transmission of the login response to the length specified in the query logins ORB. If the query login response is smaller than the buffer allocated by the initiator, the target shall not pad the response to meet the allocated length."	The first part of the comment applies equally to all buffers referenced by the target and has been added to the start of section 5 as "The size of a data structure or buffer addressed by a pointer that conforms to Figure 11 is either explicitly specified by an associated length field or implicitly known from context. Whichever the case, the target shall not initiate any Serial Bus request subactions (read, write or lock) that reference system memory outside of the range determined by the address pointer and length." The second requirement for target implementations (about padding) adds no benefit for initiators. Any target is free to implement the algorithm suggested; there is nothing in SBP-2 to preclude it.
SF 67	Section 5.1.4.3, there is no way to indicate during the create stream process whether stream control is to be accomplished *via* a stream control agent or *via* some other means as indicated in section 4.5, 4.7, and Annex F. A 2 bit field in the create stream ORB should be assigned to indicate the method of stream control. A value of zero can mean *via* a stream control agent, a value of one can mean *via* AV/C as defined in IEC 61883, the value of 2 and 3 should be reserved.	Based upon prior discussions in the working group, the use of a stream control agent *vs*. other means is expected to be command set- or device-dependent and therefore not require specification in the create stream ORB.

Reference	Ballot comment	Proposed resolution
SF 68	Section 5.1.4.3, the table after Figure 26 has for value of zero an unstructured format. In discussions within the committee, it was suggested that such an unstructured format may mean to meter out the data in the file in a fashion that would imply the contents of the data on the media did not include any isochronous packet headers and was encapsulated into isochronous packets of fixed size by the stream engine. If this is so, how is the packet size communicated to the stream engine? To do this as part of the create stream is one option, another is to have a stream control command which does it. Neither are defined in this standard.	As is the case with the optional stream control agent, discussions presupposed that the command set or device type would determine the interpretation of the "unstructured" data; *i.e.*, it is beyond the scope of SBP-2. Perhaps "unspecified" is a better name for this type of data; I've made the appropriate editorial changes.
SF 69	Section 5.1.4.3, in the table after Figure 26 change the term "structured" in the name entry for values of 2 and 3 to isochronous data interchange format", as it is called in section 11. Also add "(See Clause 11.)" I believe other structured formats may be defined in future versions or addenda to this standard, so the name should be explicit and not general.	The name is changed to "isochronous data interchange".
SF 71	Section 5.1.4.3, in the paragraph describing *create_stream_response* and *create_stream_response_length*, add the following sentence: "The value of *create_stream_response_length* shall be set to 24 by the initiator and may be ignored by the target." The length of the create stream response is fixed and it make no sense to allow the initiator to set a different value nor cause the target device to check for a minimum length.	Added the sentence "The initiator shall set *login_response_length* to a value of at least 24; the target may ignore this field."
SF 72	Section 5.1.4.3, Figure 27, change *command_block_agent* to *stream_command_block_agent*. The *command_block_agent* was returned in the login response, this field is not the same and should have a unique name to prevent confusion. Also, change all occurrences of *command_block_agent* to *stream_command_block_agent* where appropriate in this section (and throughout the document?).	Accepted.

Reference	Ballot comment	Proposed resolution
SF 73	Section 5.1.4.3, in the second paragraph below Figure 27, add a sentence "If the create stream request indicated that flow control was by a means other than a stream control agent, then *stream_control_agent* field is reserved."	Added, as the last sentence of the paragraph, "If the target does not implement a stream control agent, the contents of *stream_control_agent* are unspecified and shall be ignored by the initiator."
SF 76	Section 5.1.4.6, change "TERMINATE TASK or ABORT TASK" to "ABORT TASK".	See the response to SF 56; TERMINATE TASK is retained as an optional feature of SBP-2 devices.
SF 77	Section 5.1.4.6, in the paragraph defining the function field, delete "TERMINATE TASK".	
SF 78	Section 5.1.4.6, add a new paragraph after the NOTE: "If the *login_ID* is that obtained from a create stream response, then the task management function applies to both the stream control and stream command task sets."	The first sentence of the last paragraph of the clause is modified to read "The *login_ID* shall be set to the value returned in login response data or to the value of *stream_ID* returned in create stream response data. In either case, *login_ID* identifies the task set(s) to which the task management request is directed."
PJ p. 47	The table of possible *sbp_status* values should be updated to include 0x0B to correspond to *ack_tardy* defined by IEEE P1394a. The error occurs if the addressed node continues to return *ack_tardy* after some time limit.	Incorporate a definition for tardy retry limit exceeded, namely "An *ack_tardy* was received for the request and the vendor-dependent retry limit (which may be based upon either time or number of occurrences) for tardy responses has been exceeded."
SF 85	Section 5.3, first paragraph, add the following to the end of this paragraph "Unsolicited status associated with a specific task set, but not a specific task, shall be reported to each initiator logged in to that task set."	Added the following to the end of the first paragraph of 5.3.2: "If a target stores unsolicited status for any initiator logged-in to a logical unit it shall attempt to store status for all initiators logged-in to the same logical unit."
SF 86	Section 5.3, first table after Figure 33, add the following text to the description for value 3 "Such a status block can only be placed on a *status_FIFO* identified during a create stream command."	We don't mention what sort of status FIFO's the other types may or may not pertain; it seems inappropriate to mention this here.

Configuration ROM

The following comments apply to section 7 and Annex D of the draft standard.

Reference	Ballot comment	Proposed resolution
GM 63	In 7.4.1, 'The value indicates that the NCITS Secretariat is responsible for the software interface definition.' Say what?	Change the text to "The value indicates that the NCITS Secretariat and its Technical Committee T10 are responsible for the maintenance of this standard."

Reference	Ballot comment	Proposed resolution
SF 92	Section 7.4.8, Figure 52 and related text, and section 7.5.4: The *mgt_ORB_timeout* field is not a logical unit characteristic, it is a unit (target) characteristic. In fact, if there are multiple logical units with different characteristics, there could be multiple instances of this entry, and there is no mandate that this field be consistent throughout. I suggest a "Unit_Characteristics" entry be defined that contains this field.	Changes to configuration ROM entries could adversely affect early adopters and should be discussed within the working group. I've added changed 7.4.8 and 7.4.10 add the end of this document for the sake of discussion. I recommend these changes; the only drawback is that there is no way to specify different ORB sizes for different logical units.
LJL 1	Figure D-2 (SCSI configuration ROM) in section D.2 (SCSI command set target) of Annex D (informative) (Sample configuration ROM) uses a key value which does not have a definition in any other standard… Immediately following the Model_ID entry is a textual descriptor leaf entry, with a key field of 81 (subscript 16), whose *indirect_offset* value of 5 points to a leaf that contains the ASCII string "QQQQ"".	This is some work in SBP-2 that was done in advance of anticipated changes in the CSR Architecture. At the last meeting of IEEE P1212r, the working group voted to define 17_{16} as a Model_ID *key_value* meaningful for immediate *key_type*. The meaning of the immediate value is determined by the device vendor but it is expected to be a model number. P1212r recommends that a textual descriptor string associated with the Model_ID entry descriptively name the model (in this example, the "QQQQ" model).
LJL 2	There are issues in the Configuration ROM for target devices regarding the desire to provide textual descriptors and, perhaps, special identifier values for the class driver and type class driver of the Microsoft device driver stack. Various proposals which have made "loose" use of textual descriptor leaves, bus_dependent_info and "model_name" entries. Compatibility among many target device types as well as many initiator types need these issues need to be cleared up.	P1212r will contain recommendations for the consistent use of textual descriptor strings so they can be presented to the user and assist in the user's configuration choices. Text is always considered. The uniformity necessary to use configuration ROM with one vendor's operating system is, I think, a matter for some other specification or profile and not SBP-2.

Access

The following comments apply to section 8 of the draft standard.

Reference	Ballot comment	Proposed resolution
GM 64	Referring to 8.1 'Targets shall implement a logical unit reservation protocol that by itself supports neither persistent reservations nor passwords;' does this mean it shall be vendor dependent except that it must not support persistent reservations? Does this mean an exception to SPC by making persistent reservations prohibited?	Now that Annex C specifies an optional mechanism for passwords and persistent access control, I think this text should be corrected to read "Targets shall implement a logical unit reservation protocol ~~support neither persistent reservations nor passwords; it is a simple mechanism that can~~ which may be used to guarantee single initiator access to the logical unit and to preserve that initiator's access rights across a Serial Bus reset. Targets may optionally implement the extensions to the logical unit reservation protocol specified by Annex C, which support both passwords and persistent reservations. Neither of these mechanisms preclude additional, command set-dependent methods that control access to a target."
SF 98	Section 8.2.1, second paragraph after the Note: change "The target shall perform the following..." to "The target shall perform, in any order, the following...".	Accepted; see the changes to 8.2.1 at the end of this document.
SF 99	Section 8.2.1, in the list of items to be validated, labeled "a)" through "e)", item "a)" is not a validation but a step in performing the validation contained in "b)". Merge these two bullets to make on validation step.	Accepted; items a) and b) have been merged into a single nonprocedural list item.
SF 100	Section 8.2.1, in the list of items to be validated, item "e)" is not a validation step, but a function performed by the target.	Accepted; this step is no longer part of the nonprocedural list.
JW 10	Section 8.2.1, items b, c, d define situations where the login request is rejected. No specific return status is indicated for login rejections. Which status return code is appropriate for each of the conditions outlined?	See 8.2.1 at the end of this document for proposed clarifications.
SF 101	Section 8.2.1, item "e)", change the first sentence to "The target shall determine if a free *login_descriptor* is available for the logical unit." The target might choose to support multiple initiator access to its logical units, but reserve at least one *login_descriptor* for each logical unit thus preventing the initiators using up all of the *login_descriptors* to access a single logical unit.	I think a target that elected to reserve a *login_descriptor* for anticipated logins by other initiators would not consider that descriptor free. The current language permits the implementation contemplated and requires no change.

Reference	Ballot comment	Proposed resolution
SF 102	Section 8.2.1, after the list of validation items, add the following note: "NOTE - In the case of an unsuccessful login request, target access of the EUI-64 may or may not have occurred."	Many things may or may not occur when a login request fails. An access to the initiator's EUI-64 does not merit separate mention. A large class of initiators will not even be aware when their EUI-64 is read.
SF 105	Section 8.2.2, item "b)" in the list of things a target validates for a create stream is not a validation item, but a resource allocation. Merge the lead in sentence to the list with item "a)" and make item "b)" a separate paragraph.	Both comments accepted in a modified form; see 8.2.2 at the end of this document.
SF 110	Section 8.3, in the list of items "a)" to "d)", combine items a) and b) as item a) is not a validation, but what must be done before the validation in step b) Is performed. Items c) and d) are also not validation steps.	
SF 106	Section 8.2.2 … The term *login_descriptor* should be reserved for login operations, and the term *stream_descriptor* for create streams.	Accepted; see 8.2.2 at the end of this document.
SF 108	Section 8.3, second paragraph introduces the "one second" reconnect time. This value needs a minimum and a maximum number. No one can guarantee exactly one second for many reasons such as the reference clocks are typically ± 100 ppm, and the timeout is likely to be implemented in software. The target detection should be 1.0 second minimum, 1.25 (?) seconds maximum, and the initiator should reconnect within 1.0 seconds.	Accepted as "For at least one second subsequent to a bus reset the target shall retain sufficient information to permit an initiator to reconnect its login ~~ID~~ (and, implicitly, any associated streams ~~ID's~~). After two~~one~~ seconds the target shall perform an implicit logout for all login ID's and stream ID's that have not been successfully reconnected to their original initiator(s)."
SF 111	Section 8.3, insert the following sentence after items a) through d): "The fetch agent for the logical unit to which the reconnect was accomplish shall be in the reset state upon completion of a successful reconnect."	Accepted as "Upon successful completion of a reconnect request, the fetch agent associated with *login_ID* shall be in the reset state; the state of the fetch agent(s), if any, for the streams dependent upon *login_ID* is not affected by the reconnect request."
SF 113	Section 8.4, add an additional paragraph to the end of this section: "Upon a successful logout specifying a *login_ID* obtained during a login, any streams associated with that logical unit connection are also logged out."	Accepted as "A logout whose *login_ID* was obtained as the result of a login request implicitly causes the logout of all streams associated with the *login_ID*."

Command execution

The following comments apply to section 9 of the draft standard.

Reference	Ballot comment	Proposed resolution
SF 114	Section 9.1, first paragraph, second sentence, change "and refuses subsequent Serial Bus transactions" to "and indicates a resource conflict *via* either an *ack_conflict* or *resp_conflict_error* to another writes to the address of the management ORB"	The first paragraph has been modified to read "Management requests (which include login and logout requests) are signaled to the target agent by means of a Serial Bus block write request that specifies the address of the management ORB. The management agent becomes busy while executing a request and refuses subsequent Serial Bus requests with *ack_conflict_error* or *resp_conflict_error* until the current transaction is completed. The management agent does not require any initialization procedures."
SF 119	Section 9.1.2, add the following to the NOTE: "Write requests to the agent registers by nodes other than the logged in initiator have no effect upon the fetch agent state machine."	This section provides an informative description of how dynamic appends are effected; it would dilute its focus to mention this important fact here. Instead I have added the following at the end of 6.4: "A target shall ignore or reject Serial Bus request subactions addressed to any of a fetch agent's CSR's unless the *source_ID* matches the node ID of the initiator logged-in to that initiator."
SF 121	Section 9.1.4, Figure 58: add the following changes…	See 9.1.4 and a redrawn state diagram at the end of this document.
	…F0:F1, change the transition condition from "TR_DATA.indication(WRITE, ORB_POINTER)" to "Logged In and TR_DATA.indication(WRITE, ORB_POINTER)"	The text that precedes Figure 60 makes it clear that *all* writes to the fetch agent CSR's require the initiator to be logged-in. There's no need to add this extra text to the diagram itself.
SF 122	Section 9.1.4, Text describing transition Any:F0b: Add the following sentence at the end: "If a read request for an ORB is outstanding, the subsequent response (if any) should be accepted but the content of the response should be discarded."	Added "Transaction label(s) for outstanding request subactions shall not be reused until either the corresponding response subaction completes or a split time-out expires; in the former case, the response data shall be discarded."
SF 123	Section 9.1.4, Text describing transition F0:F1, change in last sentence "with a response of COMPLETE" to "with an ack_complete or an ack_pending followed by a response containing resp_complete."	In the service model used by IEEE Std 1394-1995, a response is understood to mean either an acknowledgment (other than pending) or a pending acknowledgment followed by a response packet.
SF 124	Section 9.1.4, Text describing transition "F1:F1" to "F1:Fnew".	The changes recommended by SF 124 through SF 127, inclusive, are reflected in 9.1.4 at the end of this document.
SF 125	Section 9.1.4, Text describing transition "F1:F2" to "Fnew:F2".	

Reference	Ballot comment	Proposed resolution
SF 126	Section 9.1.4, Text describing transition "F3:F2" to "F3:F1". Delete the phrase ", set the next_ORB variable to the value of the ORB_POINTER register", change "F2" to "F1" and delete the phrase ", from which state an immediate F2:F1 transition follows."	
SF 127	Section 9.1.4, Text describing transition F4:F2, delete the last sentence.	
SF 128	Section 9.1.4, Text describing transition Any:F5, the list of errors gives an indication that only errors associated with the fetching process cause this transition. Add two new bullets "- any device level error" and "- any transport error encountered during the transfer of data or status related to a command being processed that was fetched *via* this fetch agent."	The errors listed are intended to encompass errors that the fetch agent might directly encounter in its operations. It is also correct that a command error causes the fetch agent to transition to the dead state; see the modified text at the end of this document.
SF 129	Section 9.1.4, Text describing transition F5:Dead, change "AGENT_STATE" to "DOORBELL". The AGENT_STATE register is read only.	Accepted with modifications; see 9.1.4 at the end of this document.
SF 131	Section 9.2, second paragraph, add a new sentence to the end of the paragraph "The target is also responsible for generating the proper system memory address appropriate for the transfer being performed."	There is no way the target can performed the required data transfer functions without generating appropriate system memory addresses. The proposed text doesn't add anything.
SF 132	Section 9.3, related to the last paragraph before the note. We have never included any text to define how to reclaim an ORB that returns a status with the *src* field equal to one. After this last block is updated with a new ORB pointer value and the doorbell is rung, when can the old ORB be discarded? One way is to monitor the ORB pointer and see if it has changed. This would require polling. Another is to wait until the ORB whose address was placed in the old ORB completes. This could be a long time if a linked list is appended and the device does out of order transactions. I don't have a fixed solution, but we need one.	It could be a *very* long time, regardless of the execution model! I've added "When *src* has a value of one, the system memory shall not be reused or deallocated until the target stores completion status for some subsequent ORB in the linked list.". I've also strengthened the note as follows: "NOTE – For targets that support the ordered model of task execution, the return of completion status for an ORB implicitly indicates that all preceding ORB's in the linked list have completed successfully, are no longer part of the task set and that the initiator may reuse or deallocate their system memory."

Task management

The following comments apply to section 10 of the draft standard.

Reference	Ballot comment	Proposed resolution
SF 133	Section 10.1, third paragraph, change the fourth sentence from "There is a one-to-one relationship between a stream identifier and a stream task set; there may be multiple stream task sets associated with a logical unit." to "There is a one-to-one relationship between a stream identifier and a stream task set. There may be zero or more stream task sets and one normal command task associated with a logical unit. If there is a stream task set associated with a logical unit, there will be a normal command task set as well." In the last sentence of the paragraph, change "there are no interactions between tasks that belong to different stream task sets." To "there are no interactions between tasks that belong to different task sets."	Accepted as "Each time target resources are allocated for isochronous operations (by means of a create stream request), a task set, identified by a stream ID, is created that is associated ~~both~~ with a logical unit ~~and a stream ID. There is a one-to-one relationship between a stream ID and a stream task set;~~ There may be zero or more~~multiple~~ stream task sets associated with a logical unit. Each stream task set is separate and distinct from the normal task set and from other stream task sets: there are no interactions between tasks that belong to different ~~stream~~ task sets." Also added the following "NOTE – A successful login, which establishes a normal task set, is a prerequisite to the creation of a stream task set—even if the initiator never signals any commands to the normal task set."
SF 134	Section 10.2, fourth bullet, change "All tasks within a task set", to "All active tasks".	Rejected; they are nothing but active tasks within the task set.
GM 69	The paragraph above the note in 10.2 is redundant to the model clause 4.6. The redundancy can be resolved by changing the wording of 4.6 to eliminate the mandatory requirements in the informative clause.	The (informative) text in 4.6 was edited in Revision 3a to reflect this suggestion. I would also suggest the removal of the duplicated NOTE— in 10.2. There is still some replication between the two clauses, but I do not think it causes any problems.
SF 135	Section 10.3, item "a)" change "of the fetch agent" to "of all fetch agents".	When is more than one fetch agent associated with a task set?
SF 136	Section 10.3, between item "b)" and "c)" add an additional item to read "If more than one initiator is logged in to the logical unit, a unit attention condition is declared for each of the other initiators. For each such initiator, if the unsolicited_status_enable variable is set an unsolicited status is sent, else the attention condition is retained until it can be signaled to the associated initiator."	This has not been the intent of the working group. Fault(s) in one initiator'' task set do not affect other initiator(s). Although SBP-2 is not a SCSI protocol, the precise behaviors intended are well described by a QErr value of 11b in the control mode page (see SPC).

Reference	Ballot comment	Proposed resolution
SF 142	Section 10.4.1, add the following paragraph to the end of this section: "The fetch agent associated with the login ID shall not be transitioned to the dead state due to the processing of an abort task or a Dummy ORB."	I do not understand why this is needed, since there's nary a suggestion in the entire clause that the fetch agent should become dead. None the less, I've changed the first sentence of the first paragraph to read "Abort task is a task management function that permits an initiator to abort a specified task without otherwise affecting the task set or its fetch agent."
JW 14	Section 10.4.1, paragraph beginning "A Second method to abort tasks...", The wording implies that this is a second method of aborting tasks when it is just a clarification of the relationship between Abort Task and stream task sets. Also it mandates Abort Task be supported by targets that support streams. This means hard disks supporting streams must support Abort Task. Why is this necessary? I would rather not have the requirement to support Abort Task in association with Streams.	See 10.4.1 at the end of this document for editorial clarification. I think mandatory support for targets that implement streams is necessary because stream operations may be time-consuming. Without a method to abort a task (other than changing *rq_fmt* to three) the user might have no way to cancel outstanding isochronous operations. Also, user actions at a control interface may require changes to a queue of commands and not merely aborting all commands.
GM 70	The combination of suggestions and mandatory requirements in 10.4.1 … may lead to confusion. I think the "shalls" in this sequence should be "shoulds."	I believe the intent is that ABORT TASK *via* an ORB is optional but that if the target elects to abort the task it shall perform the steps as described. See 10.4.1 at the end of this document for editorial clarification.
SF 144	Section 10.4.5, first sentence, change "all initiators" to "all logged in initiators".	Done in SBP-2 Revision 3a but not noted in 98-109r0.
GM 71	I suggest that the Terminate Task definition should end immediately after 'FUNCTION REJECTED.'	In order for some classes of SCSI device to be supported by SBP-2 it is necessary to permit implementation of terminate task. This comment is rejected because it would leave terminate task unspecified.
SF 145	Section 10.5, the table does not include a write to the AGENT RESET register.	I've added fetch agent reset (write to AGENT_RESET); it aborts the initiator's task set if it's not already aborted.
SF 146	Section 10.5, the table: for Faulted command, CLEAR TASK SET and LOGICAL UNIT RESET, should transition all tasks in the task set for all logged in initiators and transition all associated fetch engines to dead.	I've added the following to the paragraph immediately below the table: "When an event affects more than one task set, all of the associated fetch agents transition to the state indicated by the table."
SF 147	Section 10.5, the table: for ABORT TASK SET, should transition only the initiator's fetch engine to dead and abort the only those commands in the task set which belong to the initiator.	Not necessary to specify, since all the tasks in a task set *de facto* belong to one initiator.

Reference	Ballot comment	Proposed resolution
SF 148	Section 10.5, the table: for TARGET RESET, transition all fetch agents to dead.	See the resolution for SF 146.
SF 149	Section 10.5, in third paragraph below the table, add to the end of the first sentence "and any pending status are discarded."	Modified the first sentence of that paragraph to read "Immediately upon detection of a bus reset, all normal command block fetch agents transition to the reset state and their associated task sets are cleared without the return of completion status."

Isochronous data interchange format

The following comments apply to section 11 of the draft standard.

Reference	Ballot comment	Proposed resolution
SF 153	Section 11.1, the first sentence of the first paragraph and the last paragraph in this section both contain information on operational characteristic, not formats. This information should be moved to section 12.	Although the editorial style in the draft attempts to introduce static building blocks before describing their operational use, flexibility is useful. See in particular descriptions of ORB data structures that reference their intended functionality. In my opinion the reader is better able to understand why a certain data structure or format exists if the description makes some reference to its use. Perhaps an additional clause in section 12 on cycle marks would be helpful? I defer this to discussion in the working group.
SF 158	Section 11.4, the first paragraph seems to operational information and should be moved to section 12.	
SF 159	Section 11.5 repeats a definition contained in a normative reference which is still in the development stage. This provides an opportunity to have inconsistency between these two standards. Also see section 12.2.3 which says "The common isochronous packet (CIP) format, as standardized by IEC 61883/FDIS, ...".	The working group expressed a desire to have most relevant information in one document. To satisfy this and avoid duplicate normative references, this information has been relocated to an informative annex.
SF 161	Section 11.5, paragraph below figure 64 says the sid field "shall specify the Serial Bus physical ID of the source (talker) for the isochronous data." and then continues on to say that, when talking, we substitute in a value for this field from a table and not that we should use our bus ID. This is inconsistent.	Corrected to read "The *sid*, or source ID, field identifies the node whose plug control register(s) control the source (talker) for the isochronous data."
SF 162	Section 11.5, second paragraph below figure 64, add an additional sentence: "The application data field shall not contain partial data blocks."	Modified the definition of *dbs* to include "Individual data blocks shall be entirely contained within a single Serial Bus isochronous packet (which may encapsulate more than one data block) ~~may be encapsulated within a single Serial Bus isochronous packet~~."

Isochronous operations

The following comments apply to section 12 of the draft standard.

Reference	Ballot comment	Proposed resolution
SF 170	Section 12.2.1, second paragraph states that the channel mask can only be changed if the stream is stopped, but I can visualize some cases where this it may be desirable to change it on the fly. Did we impose this restriction? If not, who/what/where did this restriction come from?	Changing the channel mask on-the-fly makes no sense when the target is a talker; there is no synchronization point that correlates with the data coming from the medium. This functionality could be useful to a listener: "If the target is a talker, the channel mask may be updated only when the stream controller is stopped or paused. At all other times a target that is a talker shall reject any attempts to modify the channel mask and shall leave it unchanged."
SF 171	Section 12.2.2, second paragraph, delete the last 2 sentences. This operation in the case of STOP is in conflict with section 5.1.3. In the case of pause, the information stated is very much implementation dependent.	Modified to read "If the stream controller is in a state where isochronous data transmissions are suspended~~disabled~~ (~~typically~~ as the result of a stream control ORB with a ~~STOP or~~ PAUSE control function), isochronous data transfer from the medium may continue ~~within the limitations of target buffer resources~~. ~~Once these buffers are full, the execution of stream command block ORB's shall be suspended.~~"
SF 175	Section 12.3, in the second set of bullets, for all three bullets, delete that portion of the items that refers to "the current stream control ORB". The paragraph above talks about the ORB used to set the mode, so the term "the current stream control ORB" must be referring to the ORB setting the mode. This is not what was intended.	The bullet items have been modified to refer to stopping or continuing the "execution of the stream control ORB's" without reference to the "current" ORB.
SF 176	Section 12.3, in the second set of bullets, the first bullet, does "and stop execution" mean the same thing as a STOP command?	The intent is the same, *i.e.*, as if a STOP command were instantaneously received that preempted the active stream control ORB.
SF 180	Section 12.3, in the table after Figure 67, add a not to the descriptions for error values 2 and 3 "the isochronous packet is not recorded on the media."	I think this is contrary to discussions in the working group. It is certainly simpler for the target to record the erroneous data than to back it out of internal FIFO's.
SF 181	Section 12.3, add to the table after Figure 67 an error meaning "found a data record which exceeded the maximum record size allowed for the speed allowed for transmission."	The table (which was relocated to clause 5.3.3 in Revision 3a) has been modified to include *sbp_status* values of 6 and 7, defined as "Format error in recorded isochronous data with the result that data was not transmitted on Serial Bus" and "Data payload in recorded isochronous data too large for transmission on Serial Bus at the requested speed", respectively.

SCSI Architecture Model conformance

The following comments apply to Annex F of the draft standard.

Reference	Ballot comment	Proposed resolution
SF 186	Section F.8, a serial bus reset may cause the normal command block agent set to be initialized, but does not affect the isochronous streams. The definition of hard reset, I believe, is to make the unit go to the same state as a power on reset. This isn't true for an SBP-2 device.	Change the definition of hard reset to "A ~~Serial Bus reset~~ write to the RESET_START register (see 6.1) or a task management function of TARGET RESET causes the target to execute a hard reset, as defined by SAM-2."

5.1.2.2 Stream command block ORB

The *cm* field (together with the *cycle_mark_offset* field) specifies the location of the first quadlet of isochronous data (stream offset) as encoded by the table below.

Value	*cycle_mark_offset*	Stream offset
0	Undefined	Zero
1	~~Reserved (not to be used)~~ Undefined	~~Reserved (not to be used)~~ *cycle_mark_offset*
2	Location of first CYCLE MARK	Zero
3	Location of first CYCLE MARK	*cycle_mark_offset*

The stream offset derived from the combination of *cm* and *cycle_mark_offset* specifies the location of the first quadlet of the isochronous data as an offset, in quadlets, relative to the starting medium location indicated by the *command_block*. For a block device, the stream offset, expressed in bytes, shall be less than the block size of the device.

NOTE – The command transported by the stream command block ORB specifies a starting location on the medium and an associated transfer length. Particularly in the case of block devices, the relevant isochronous data may be a subset of the data length and may commence at a nonzero offset relative to the natural block boundaries of the medium—hence the necessity for the additional values, *stream_length* and stream offset, to completely characterize the request.

The *cycle_mark_offset* field, when *cm* has a value of two or three, specifies the location of the first CYCLE MARK packet as an offset, in quadlets, relative to the starting medium location indicated by the *command_block*. When *cm* has a value of one, CYCLE MARK packets are not present in the data and *cycle_mark_offset* specifies the stream offset instead. In either case, the value of *cycle_mark_offset*, converted to bytes, shall be less than *stream_length*.

7.4.8 ~~Logical~~ Unit_Characteristics entry

The ~~Logical~~ Unit_Characteristics entry is an immediate entry ~~that, when present~~ in the unit directory~~,~~ which specifies characteristics of the target implementation. Figure 54 shows the format of this entry.

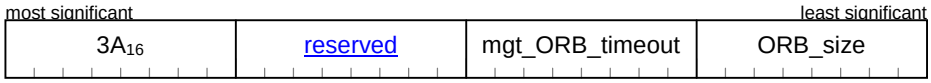

Figure 54 – ~~Logical~~ Unit_Characteristics entry format

$3A_{16}$ is the concatenation of *key_type* and *key_value* for the ~~Logical~~ Unit_Characteristics entry.

~~The *q* bit shall specify the task management (queuing) model implemented by the target. If *q* is zero, the target implements the basic task management model defined by this standard in 10.2. When *q* is one, the task management model is dependent upon the command set specified by the Command_Set_Spec_ID and Command_Set entries.~~

~~The *ordered* bit (abbreviated as *o* in the figure above) specifies the manner in which the target executes tasks signaled to the normal command block agent. If the target executes and reports completion status without any ordering constraints, the *ordered* bit shall be zero. Otherwise, if the target both executes all tasks in order and reports their completion status in the same order, the *ordered* bit shall be one.~~

~~The *isochronous* bit (abbreviated as *i* in the figure above) specifies whether or not the target supports isochronous operations. When *isochronous* is one, create stream requests and stream command block~~

~~requests shall be supported; stream control requests may be supported. If the *isochronous* bit is one, the *irmc*, *cmc* and *isc* bits in the bus information block shall also be one, as described in 7.2.~~

The *mgt_ORB_timeout* field shall specify, in units of 500 milliseconds, the maximum time an initiator shall allow for a target to store a status block in response to a management ORB. The time-out commences when the initiator receives either *ack_complete* or *resp_complete* from the target in response to the block write of the management ORB address to the MANAGEMENT_AGENT register.

The *ORB_size* field shall specify, in quadlets, the fetch size used by the target to obtain ORB's from initiator memory. The initiator shall allocate, on a quadlet aligned boundary, at least this much memory for each ORB signaled to the target.

7.4.10 Logical_Unit_Number entry

The Logical_Unit_Number entry is an immediate entry that, when present in the unit directory, specifies the characteristics, peripheral device type and logical unit number of a logical unit implemented by the target. Figure 56 shows the format of this entry.

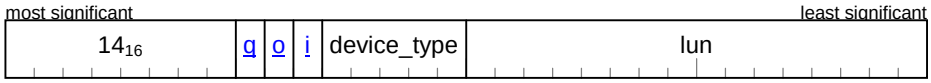

Figure 56 – Logical_Unit_Number entry format

14_{16} is the concatenation of *key_type* and *key_value* for the Logical_Unit_Number entry.

The *q* bit shall specify the task management (queuing) model implemented by the logical unit. If *q* is zero, the logical unit implements the basic task management model defined by this standard in 10.2. When *q* is one, the task management model is dependent upon the command set specified by the Command_Set_Spec_ID and Command_Set entries.

The *ordered* bit (abbreviated as *o* in the figure above) specifies the manner in which the logical unit executes tasks signaled to the normal command block agent. If the logical unit executes and reports completion status without any ordering constraints, the *ordered* bit shall be zero. Otherwise, if the logical unit both executes all tasks in order and reports their completion status in the same order, the *ordered* bit shall be one.

The *isochronous* bit (abbreviated as *i* in the figure above) specifies whether or not the logical unit supports isochronous operations. When *isochronous* is one, create stream requests and stream command block requests shall be supported; stream control requests may be supported. If the *isochronous* bit is one, the *irmc*, *cmc* and *isc* bits in the bus information block shall also be one, as described in 7.2.

The *device_type* field indicates the peripheral device type implemented by the logical unit. This field shall contain a value specified by the table below.

Value	Peripheral device type
$0 – 1E_{16}$	The meaning of *device_type* is command set-dependent
$1F_{16}$	Unknown device type; command set-dependent means are necessary to determine the peripheral device type

The *lun* field shall identify the logical unit to which the information in the Logical_Unit_Number entry applies.

8.2.1 Login

Before an initiator may signal any other requests to a target it shall first perform a login. The login request, whose format is specified in 5.1.4.1, shall be signaled to the target's MANAGEMENT_AGENT register by means of an eight-byte block write transaction that specifies the Serial Bus address of the login request. The address of the management agent shall be obtained from configuration ROM.

The speed at which the block write request to the MANAGEMENT_AGENT register is received shall determine the speed used by the target for all subsequent requests to read the initiator's configuration ROM, fetch ORB's from initiator memory or store status at the initiator's *status_FIFO*. Command block ORB's separately specify the speed for requests addressed to the data buffer or page table.

The login ORB shall specify the *lun* of the logical unit for which the initiator desires access.

The target shall perform the following steps (in any order) to validate a login request:

- The target shall read the initiator's unique ID, EUI-64, from the bus information block by means of two quadlet read transactions. The *source_ID* from the write transaction used to signal the login ORB to the target's MANAGEMENT_AGENT register shall be used as the *destination_ID* in the quadlet read transactions;
- The target shall determine whether or not the initiator already owns a login by comparing the EUI-64 just obtained against the *login_owner_EUI_64* for all *login_descriptors*. If the initiator is currently logged-in to the same logical unit, the login request shall be rejected with an *sbp_status* of access denied;
- If the *exclusive* bit is set in the login ORB and there are any active *login_descriptors* for the logical unit, the target shall reject the login request with an *sbp_status* of access denied;
- If an active *login_descriptor* with the *exclusive* attribute exists for the *lun* specified in the login ORB, the target shall reject the login request with an *sbp_status* of access denied; else
- The target shall determine if a free *login_descriptor* is available and, if none are available, reject the login request with an *sbp_status* of resources unavailable.

~~If a *login_descriptor* is free~~Once the above conditions have been met and a *login_descriptor* allocated, the initiator's *source_ID* is stored in *login_owner_ID*, the initiator's EUI-64 is stored in *login_owner_EUI_64*, the *lun* and *status_FIFO* fields from the login ORB are stored in the *login_descriptor*, the *exclusive* variable in the *login_descriptor* is set to the value of the *exclusive* bit from the login ORB and the addresses of the fetch agent(s) are stored in the *login_descriptor*. Lastly the target assigns a unique *login_ID* to this login and stores it in the *login_descriptor*.

If the target is able to satisfy the login request, it shall return a login response as specified in 5.1.4.1. A critical component of a login response returned to the initiator is the base address of the target agent that the initiator shall use to signal any subsequent requests to the target for the indicated *login_ID*.

8.2.2 Create stream

An isochronous stream may be created for an initiator only after completion of the login process just described. The initiator shall supply a *login_ID* previously obtained as the result of a successful login as well as other information in the create stream request that characterizes the isochronous operations to be performed.

The information consists of four~~three~~ items:

- whether the target is to function as a talker or a listener;
- the isochronous data format;
- the maximum number of channels that may be simultaneously enabled; and
- the aggregate maximum isochronous payload for all channels to be transferred between Serial Bus and the device medium in a single isochronous period.

The aggregate maximum isochronous payload is the worst-case amount of data the target may have to transfer to or from Serial Bus and from or to the medium in an isochronous period. Implementation-dependent constraints may limit the performance of the target, which requires this information in order to determine if the login may be accepted. Upon playback (when the target is a talker), the aggregate maximum isochronous payload shall reflect the total of all channels recorded on the medium—not just the aggregation of payload(s) for the channels to be transmitted on Serial Bus. This is essential since the target reads all of the data from the medium even though the channel mask may select a small subset for playback.

These parameters—listener *vs.* talker, isochronous data format, maximum channels and aggregate maximum isochronous payload—may be used by the target to determine if sufficient resources are available to create the stream and, if so, the manner in which they are to be configured.

The target shall perform the following to validate a create stream request:

– The target shall validate the *login_ID* supplied in the create stream ORB by comparing the destination_ID in the read request(s) used to fetch the ORB with the source_ID retained when *login_ID* was assigned to the initiator. If the node ID's do not match, the *login_ID* is invalid.

  If the *login_ID* is valid, the target shall determine if a free *stream_descriptor*~~*login_descriptor*~~ is available and, if none are available, reject the create stream request with an *sbp_status* of resources unavailable.

~~If a *login_descriptor* is free~~Once the above conditions have been met and a *stream_descriptor* allocated, the *stream_descriptor* is associated with the appropriate *login_descriptor*~~initiator's *source_ID* is stored in *login_owner_ID*, the initiator's EUI-64 is stored in *login_owner_EUI_64*, the *lun* from the *login_descriptor* is copied to the *login_descriptor* for the create stream request~~ and the addresses of the fetch agent(s) are ~~also~~ stored in the *stream_descriptor*~~*login_descriptor*~~. Lastly the target assigns a unique *stream_ID* to this stream~~login~~ and stores it in the *stream_descriptor*~~*login_descriptor*~~.

NOTE – The *stream_descriptor* may also hold other information from the create stream request, such as listener *vs.* talker, isochronous data format, number of channels or aggregate maximum payload as dictated by the target implementation.

In addition to the addresses of the stream command block and stream control fetch agents, the target shall also specify in the *create_stream_response*~~*login_response*~~ data the minimum transfer length that the initiator should specify in the *stream_length* field of any stream command block request signaled to the target.

9.1.4 Fetch agent state machine

The operations of a target fetch agent are specified by the figure below. The state of a fetch agent is visible in the context displayed by the AGENT_STATE and ORB_POINTER registers described in 6.4. The state machine diagram and accompanying text explicitly specify the conditions for transition from one state to another and the actions taken within states.

The target shall qualify all writes to fetch agent CSR's by the *source_ID* of the currently logged-in initiator. A write to a fetch agent CSR by any other Serial Bus node shall be rejected by the target by one of the following methods:

– an acknowledgment of *ack_type_error*;

– an acknowledgment of *ack_complete* (although the write is ignored); or

– an acknowledgment of *ack_pending*. When the target subsequently responds, the response code shall be *resp_type_error*.

The recommended target action is to indicate a type error, either by an acknowledgment of *ack_type_error* or an acknowledgment of *ack_pending* followed by *resp_type_error*.

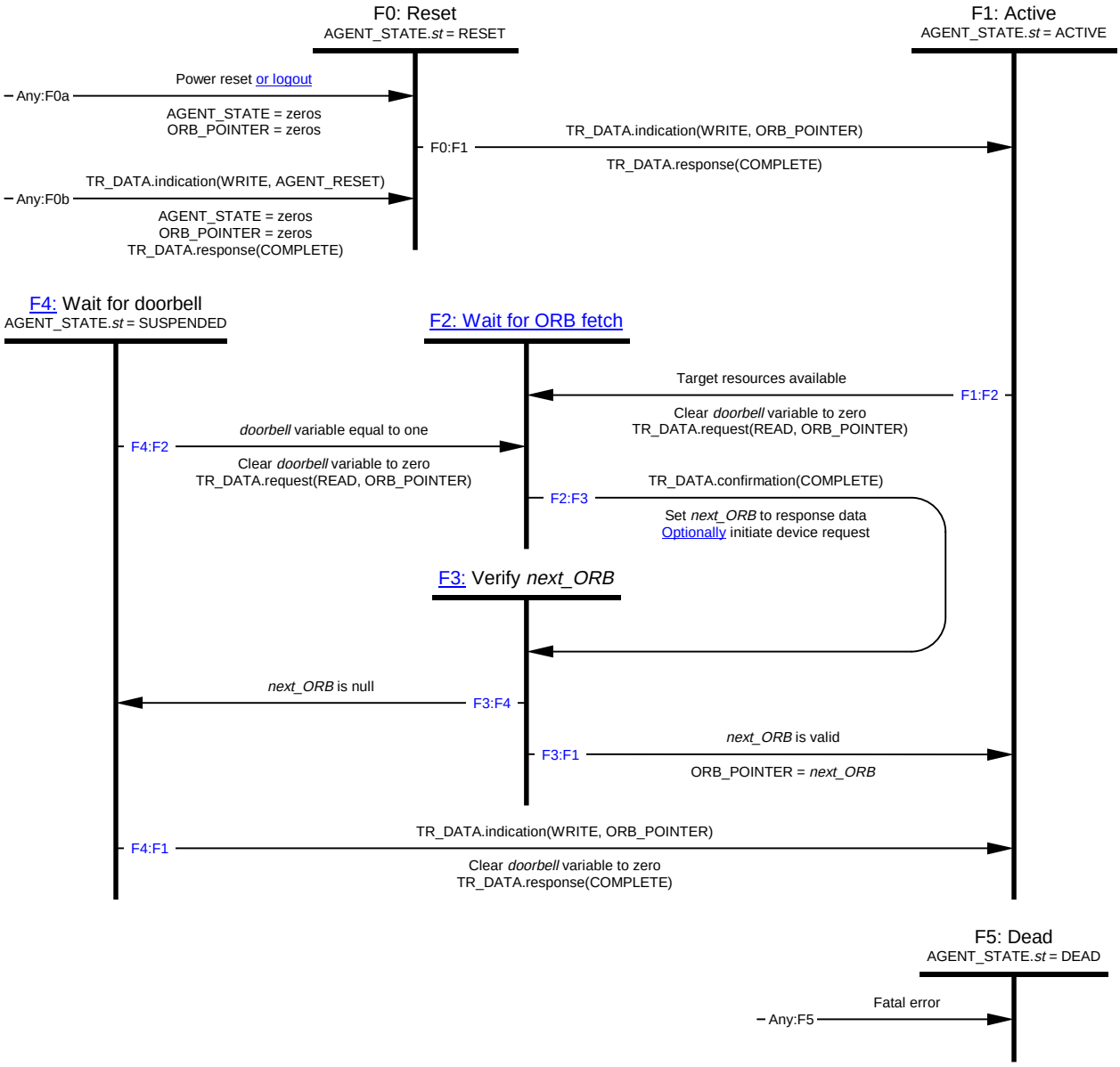

Figure 60 – Fetch agent state machine

Transition Any:F0a. A power reset shall cause the fetch agent to transition to the RESET state from any other state. The AGENT_STATE and ORB_POINTER registers (that control and make visible the operations of the fetch agent) shall be reset to zeros.

Transition Any:F0b. A quadlet write request by the initiator to the AGENT_RESET register shall cause the fetch agent to transition to state F0 from any other state. The fetch agent shall zero the AGENT_STATE and ORB_POINTER registers before the transition to state F0. Transaction label(s) for outstanding request subactions shall not be reused until either the corresponding response subaction completes or a split time-out expires; in the former case, the response data shall be discarded.

State F0: Reset. Upon entry to this state, the *st* field in the AGENT_STATE register shall be set to RESET. The fetch agent is inactive and available to be initialized by an initiator.

Transition F0:F1. An 8-byte block write of a valid *ORB_offset* to the ORB_POINTER register shall update the register and cause the fetch agent to transition to state F1. The target shall confirm the block write request with a response subaction of COMPLETE.

NOTE – When the fetch agent is reset, it is not necessary to write to the DOORBELL register when a transition is made to the ACTIVE state.

State F1: Active. Upon entry to state F1, the *st* field in the AGENT_STATE register shall be set to ACTIVE. In this state, the fetch agent may use the address information in the ORB_POINTER register to fetch ORB's from the initiator as resources permit.

Transition F1:F2. The availability of target resources is an implementation-dependent decision. Typically, the resources might be space in device memory to hold an image of the ORB while the command is scheduled for execution and subsequently completed. In any case, the fetch agent clears the *doorbell* variable to zero and then issues a block read request to obtain the ORB from system memory.

State F2: Wait for ORB fetch. The fetch agent is suspended and awaiting a read response for a block read directed to the address contained in the ORB_POINTER register.

Transition F2:F3. Subsequent to a block read request, issued as described above, the fetch agent may accept a block read response that contains either the *next_ORB* data or an entire ORB intended for execution by the device server~~the desired ORB~~. If a read response is received whose *source_ID*, *destination_ID* and *tl* fields match the *destination_ID*, *source_ID* and *tl* fields, respectively, of the read request, the fetch agent shall copy the *next_ORB* field from the response data to the *next_ORB* variable before making the transition to state F3. When the response data contains an entire ORB not yet in the device server's working set, the fetch agent shall make the ORB available to the device server for execution ~~and shall copy the *next_ORB* field from the response data to the *next_ORB* variable before making the transition to state F2. The target shall not initiate execution of the command contained within the ORB until these actions are complete.~~

State F3: Verify *next_ORB*. The *next_ORB* variable contains information about a subsequent ORB that may be linked in order after the one just fetched. As described in 5.1, the *next_ORB* pointer encodes the address of the next ORB. The actions of this state determine whether or not the *next_ORB* pointer is null.

Transition F3:F1. If the *next_ORB* variable does not indicate a null pointer the fetch agent shall update the ORB_POINTER register with the value of *next_ORB*.

Transition F3:F4. The fetch agent shall transition to a suspended state, F4, if *next_ORB* contains a null pointer. A null pointer is defined in 5.1 and exists if the most significant bit of the variable is one.

State F4: Wait for doorbell. The fetch agent is suspended; the ORB_POINTER register contains the address of the ORB that, at the time state F4 was entered contained a null pointer.

Transition F4:F1. If an indication of a write to the ORB_POINTER register is received, the fetch agent shall clear the *doorbell* variable to zero~~, set the *next_ORB* variable to the value of the ORB_POINTER register~~ and ~~then~~ confirm the write transaction with a response subaction of COMPLETE. After the confirmation, the fetch agent shall transition to state F1~~, from which state an immediate F3:F1 transition follows~~.

Transition F4:F2. Whenever the *doorbell* variable is equal to one, the fetch agent shall clear the *doorbell* variable to zero, ~~and then~~ issue a read request to obtain a fresh copy of the *next_ORB* field from the ORB whose address is contained in the ORB_POINTER register and then~~shall~~ transition to state F2. The *doorbell* variable is set to one as the result of a quadlet write request of any value to the DOORBELL register, whether the write request is received in this or any other state.

The fetch agent may issue either an 8-byte block read request (to fetch just the *next_ORB* field) or it may reread the entire ORB. The initiator shall insure that system memory occupied by the ORB remains accessible, as described in 9.3.

Transition Any:F5. Upon the detection of any fatal error, the fetch agent shall transition to state F5. Examples of fatal errors include, but are not limited to:

- the failure of the addressed node to acknowledge a read request;
- the failure of the addressed node to respond to a read request (split time-out);
- a busy condition at the addressed node that exceeds the target's busy retry limit;
- a data CRC error in a response subaction.

Some of these errors may be recoverable if retried by the target.

The fetch agent may also be instructed to transition to the dead state as a result of an error in command execution detected by the device server.

State F5: Dead. The dead state is a unique state that preserves fetch agent information in the AGENT_STATE and ORB_POINTER registers. ~~All~~ Writes to any fetch agent register except AGENT_RESET~~these registers~~ shall have no effect while in state F5.

10.4.1 Abort task

Abort task is a task management function that permits an initiator to abort a specified task without otherwise affecting the task set or its fetch agent. A modification to the *rq_fmt* field of the ORB to be aborted is the basic method; in addition, targets may also recognize task management ORB's to abort tasks. All targets shall support abort task.

Because the task to be aborted may not have been fetched by the target when the initiator wishes to abort the task, the following procedure shall be used to abort the task:

a) The *rq_fmt* field shall be set to a value of three in the ORB for the task to be aborted. This field and the *next_ORB* field are the only two portions of an ORB that may be modified by the initiator once the ORB is linked into an active request list;

b) The initiator may construct a management ORB in system memory for the abort task function. The initiator shall set the appropriate values in the *rq_fmt*, *login_ID* and *ORB_offset* fields of the ORB, as described in 5.1.4.6. The *function* field shall be set to ABORT TASK; *ORB_offset* shall contain the Serial Bus address of the ORB for the task to be aborted;

c) The initiator may signal the abort task management ORB to the management agent.

Mandatory support for abort task requires the target to recognize an *rq_fmt* value of three in an ORB and take the actions described below.

- If the ORB to be aborted has already been fetched by the target, the task may be completed by the target without recognition of the abort task request; otherwise
- When the ORB is first fetched, the target shall recognize the *rq_fmt* field value of three and shall not execute the command. That target shall store completion status for the aborted ORB; the request status shall be REQUEST COMPLETE and the *sbp_status* field shall indicate dummy ORB completed.

A second method to abort task(s) may be available by means of task management ORB's with a *function* of ABORT TASK. ~~Target support for this method of abort task is determined by the type of task set to which the task to be aborted belongs. The *login_ID* field in the ORB identifies both the fetch agent and the corresponding task set.~~ If the *login_ID* specified in the ORB was returned in a *login_response*, target support for this method of abort task is optional. Otherwise the *login_ID* was returned in a *create_stream_response* (in which case it is a *stream_ID*) and the target shall ~~implement~~ support this method to abort task(s), as specified below. Targets that implement this method~~support abort task by means of task management ORB's~~ shall store a completion status of REQUEST COMPLETE for the abort task request in the status buffer specified by the ORB.

If the task to be aborted, identified by *ORB_offset*, is not recognized by the target as part of its working set, one of two conditions may exist: either the ORB has not been fetched or completion status has already been stored. In either case the target is not required to take any immediate action. In the first case, when the ORB is ultimately fetched, the *rq_fmt* field has a value of three and the target shall not execute the command. The target shall store completion status for the aborted ORB; the request status shall be REQUEST COMPLETE and the *sbp_status* field shall indicate dummy ORB completed. In the second case, no action whatsoever need be taken by the target.

If the task to be aborted is recognized by the target as part of its working set, the target should attempt to abort the task according to the steps below. Note that timing conditions may exist that prevent targets from aborting the specified task. In particular, if the target has already issued a write request to store completion status for the task to be aborted, the target shall take no other action in response to the abort task request Otherwise, if the target undertakes to abort the task it shall~~should~~ perform the following actions in response to a task management ORB with the ABORT TASK *function*:

a) The target should not issue additional data transfer requests for the task;

b) The target shall wait for response subactions to pending data transfer requests and, once all such response subactions are received, shall not issue additional data transfer requests for the task;

c) So long as none of the target medium, data buffer or status FIFO have been modified as the result of partial execution of the task, the target shall store completion status of REQUEST COMPLETE with an *sbp_status* field that indicates dummy ORB completed;

d) Otherwise, if task execution has commenced and any one of the target medium, data buffer or status FIFO has been modified, then the target shall store completion status of REQUEST COMPLETE with an *sbp_status* field that indicates request aborted.

Regardless of which abort task methods are supported by the target, the initiator shall not reuse the system memory occupied by the ORB, data buffer or page table of the task to be aborted until completion status is returned for that ORB.