Working Draft

X3T10
Project 1155D

Revision 2
January 9, 1997

Information technology —
Serial Bus Protocol 2 (SBP-2)

X3T10 Technical Editor:

Peter Johansson
Congruent Software, Inc.
3998 Whittle Avenue
Oakland, CA 94602
USA

(510) 531-5472
(510) 531-2942 FAX

pjohansson@aol.com

Reference numbers
ISO/IEC xxxxx:199x
ANSI X3.xxx-199x

Printed January 9, 1997

Points of contact

X3T10 Chair:

John B. Lohmeyer
Symbios Logic, Inc.
4420 Arrows West Drive
Colorado Springs, CO 80907
USA

(719) 533-7560
(719) 533-7036
john.lohmeyer@symbios.com

X3T10 Vice-chair:

Lawrence J. Lamers
Adaptec, Inc.
691 South Milpitas Boulevard
Milpitas, CA 95035

(408) 957-7817
(408) 957-7193 FAX
ljlamers@aol.com

X3T10 Secretariat:

X3 Secretariat
1250 I Street NW, Suite 200
Washington, DC 2000
USA

(202) 737-8888
(202) 638-4922 FAX

X3T10 Bulletin board: (719) 533-7950

X3T10 FTP: ftp.symbios.com/pub/standards/io/x3t10

X3T10 Home page: http://www.symbios.com/x3t10

X3T10 Reflector: scsi@symbios.com
majordomo@symbios.com (to subscribe)

IEEE 1394 Reflector: p1394@sun.com
bob.snively@sun.com (to subscribe)

Document distribution:

Global Engineering
15 Inverness Way East
Englewood, CO 80112-5704
USA

(800) 854-7179
(303) 792-2181
(303) 792-2192 FAX

ANSI®
X3.xxx-199x

American National Standard
for Information Systems –

Serial Bus Protocol 2 (SBP-2)

Secretariat

Information Technology Industry Council

Not yet approved

American National Standards Institute, Inc.

Abstract

This standard specifies a protocol for the transport of commands, data and status between devices connected by Serial Bus, a memory-mapped split-transaction bus defined by IEEE Std 1394-1995. In order to take advantage of unique capabilities of Serial Bus for the transport of isochronous data, this standard provides methods to manage isochronous connections and to control the flow of isochronous data between devices.

American National Standard

Approval of an American National Standard requires verification by ANSI that the requirements for due process, consensus, and other criteria for approval have been met by the standards developer.

Consensus is established when, in the judgment of the ANSI Board of Standards Review, substantial agreement has been reached by directly and materially affected interests. Substantial agreement means much more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that effort be made towards their resolution.

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

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

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

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

Published by

American National Standards Institute
1430 Broadway, New York, NY 10018

Printed in the United States of America

Contents

Page

Tables

Figures

Annexes

Foreword (This foreword is not part of American National Standard X3.xxx-199x)

This standard defines such a transport protocol within the domain of Serial Bus, IEEE Std 1394-1995, that is designed to permit efficient, peer-to-peer operation of input output devices (disks, tapes, printers, *etc.*) by initiator(s) such as operating systems or embedded applications. Vendors that wish to implement devices that connect to Serial Bus may follow the requirements of this and other standards to manufacture an SBP-2 compliant device.

This standard was developed by X3T10 during 1996 and 1997. Although some early SBP mock-up devices were demonstrated in 1993, significant proof-of-concept, in the form of prototype implementations, has proceeded contemporaneously with the development of this standard.

There are two annexes in this standard. Annex A is normative and is considered part of this standard. Annex B is informative and is not considered part of this standard.

Requests for interpretation, suggestions for improvement and addenda, or defect reports are welcome. They should be sent to the X3 Secretariat, Information Technology Industry Council, 1250 I Street NW, Suite 200, Washington, DC 20005-3922.

This standard was processed and approved for submittal to ANSI by Accredited Standards Committee on Information Processing Systems, X3. Committee approval of this standard does not necessarily imply that all committee members voted for approval. At the time it approved this standard, the X3 Committee had the following members:

James D. Converse, Chair
Donald C. Loughry, Vice-chair
Joanne M. Flanagan, Secretary

Organization Represented	*Name of Representative*
American Nuclear Society	Geraldine C. Main
AMP, Inc	Edward Kelly
Apple Computer	Karen Higginbottom
Association of the Institute for Certification of Professionals	Kenneth Zemrowski
AT&T/NCR	Thomas W. Kern
Boeing Company	Catherine Howells
Bull HN Information Systems, Inc.	William George
Compaq Computer Corporation	James Barnes
Digital Equipment Corporation	Delbert Shoemaker
Eastman Kodak	James D. Converse
GUIDE International	Frank Kirshenbaum
Hewlett-Packard	Donald C. Loughry
Hitachi America, Ltd.	John Neumann
Hughes Aircraft Company	Harold L. Zebrack
IBM Corporation	Joel Urman
National Communication Systems	Dennis Bodson
National Institute of Standards and Technology	Robert E. Roundtree
Northern Telecom, Inc.	Mel Woinsky
Neville & Associates	Carlton Neville
Recognition Technology Users Association	Herbert P. Schantz
Share, Inc.	Gary Ainsworth
Sony Corporation	Michael Deese
Storage Technology Corporation	Joseph S. Zajaczkowski
Sun Microsystems	Scott Jameson

3M Company	Eddie T. Morioka
Unisys Corporation	John L. Hill
US Department of Defense	William C. Rinehuls
US Department of Energy	Alton Cox
US General Services Administration	Douglas Arai
Wintergreen Information Services	Joun Wheeler
Xerox Corporation	Dwight McBain

Technical Committee X3T10 on Lower Level Interfaces, which developed and reviewed this standard, had the following members:

John B. Lohmeyer, Chair
Lawrence J. Lamers, Vice-chair
Ralph O. Weber, Secretary

I. D. Allan
P. D. Aloisi
G. Barton
R. Bellino
C. Brill
J. Chen
R. Cummings
Z. Daggett
J. Dambach
J. V. Dedek
E. Fong
E. A. Gardner
L. Grantham
D. Guss
K. J. Hallam
N. Harris
E. Haske
S. F. Heil
S. Holmstead
P. Johansson
G. Johnsen
S. Jones
T. J. Kulesza
E. Lappin
R. Liu
B. McFerrin
J. McGrath
P. McLean
P. Mercer
G. Milligan
C. Monia
D. Moore
I. Morrell
J. Moy
S. Nadershahi
E. Oetting
D. Pak
G. Penokie
A. E. Pione
D. Piper
R. Reisch
S. D. Schueler
R. N. Snively
G. R. Stephens
C. E. Strang, Jr.
T. Totani
D. Wagner
D. Wallace
J. L. Williams
M. Wingard
M. Yokoyama

Revision history

Revision 1 (July 17, 1996)

First release of working draft.

Revision 1a (August 13, 1996)

Changes were incorporated from *ad hoc* discussions with diverse participants. These were presented at the Redmond, WA, SBP-2 Working Group meeting for discussion.

Data structure locations have been constrained to enable cost-reductions in target silicon. ORB's and associated parameter and response buffers shall be in the same node as the initiator that logged-in to the target. The same restriction shall also apply to the status FIFO. In a similar fashion, the data buffer and the page table that describe it shall reside both in the same node—although this node does not have to be the same as the initiator's.

The ORB fields that describe the data buffer and page table were enhanced to permit the description of data transfer alignment requirements in the case where the data buffer is directly addressable as a contiguous range of Serial Bus addresses.

Interrupt notification was modified to permit the return of status to be optional. An error condition overrides this parameter; a status block shall always be stored in the event of an error.

The ORB data structures were modified to compact the stream CDB and stream control ORB's to 32 bytes from 64 bytes.

New management ORB's have been defined for security management and access control. The accompanying work in section 8 still remains to be completed.

The status block has been expanded to permit the return of autosense data when appropriate to the device class. Targets are permitted to return portions of the status block when appropriate.

Unsolicited status was added as a feature, along with an interlock through a new register, STATUS_ACKNOWLEDGE, to let the initiator pace the receipt of unsolicited status reports.

The login and management agents have been collapsed into one agent, the management agent. Requests are signaled to the management agent *via* a new register, the MANAGEMENT_AGENT register, whose address is obtained from configuration ROM.

The target fetch agent (for normal CDB, stream CDB and stream control requests) has been enhanced to permit its reactivation from a SUSPENDED state by a single write to the ORB_POINTER register. This is an improvement over the previous approach where a write to the DOORBELL register would cause the fetch agent to refetch the *next_ORB* address before fetching the new request. A

Serial Bus transaction is eliminated and the restart latency is significantly improved.

A new configuration ROM entry, Unit_Unique_ID was defined to support SBP-2 devices that have multiple Serial Bus connections.

The basic task management model, discussed by the X3T10 SCSI-3 Working Group in Colorado Springs, CO, July 17, 1996, is now part of the draft.

Revision 1b (September 9, 1996)

Changes were incorporated as a result of working group discussions in Redmond, WA and were subsequently presented in Natick, MA.

The definition of a logical unit has been expanded. A target shall always implement logical unit zero.

Alignment restrictions on SBP-2 data structures (*i.e.*, anything referenced by an address pointer in an ORB other than the data buffer itself) have been relaxed from 16- to 4-byte alignment.

The normal and stream CDB ORB's have been simplified to permit a variable length CDB to follow the first five quadlets of the ORB, whose definition remains constant. This eliminated the need for both a 32- and 64-byte ORB.

The names *page_table* and *page_table_elements* have been changed throughout the document to *data_descriptor* and *data_size*, respectively. The meaning and usage of these fields has not changed. These global changes are not marked with change bars.

Block read transactions used to access page tables shall not cross page alignment boundaries expressed as 2^{page_size+8} bytes.

SBP-2 status has been redefined into two parts, one dependent upon the command set of the device and another used to present transport protocol status common to all devices.

As a result of discussions in Redmond, the fetch agent CSR's have been simplified and mistakes in the fetch agent state machine corrected. The figure that illustrates the fetch agent state machine and the accompanying text have been relocated to be closer to the descriptions of the usage of target fetch agents by initiators.

Section 8 has been expanded to document the usage of the login and security ORB's defined in 5.1.4.

Revision 1c (September 18, 1996)

Changes made *per* working group discussions in Natick, MA.

The acronym CDB has been changed throughout to command block or *command_block*, as appropriate. This global change is not marked by change bars.

The conformance glossary has been expanded to define the terms “reserved” and “ignored” and to clarify the implications of “shall.”

A note has been added to emphasize that device designers are encouraged to use 32-byte ORB’s.

The *fetchable* bit in the status block has been renamed *end_of_list* and its meaning has been redefined. The status block has been modified to permit SBP-2 errors to be reported concurrently with command set errors. The *sense_key*, *asc* and *ascq* fields have been deleted and redefined as command set-dependent.

The MANAGEMENT_AGENT register has been redefined so that a write transaction, rather than a lock, is used to signal a request to the target.

The *doorbell*, *fetched* and *status_acknowledge* bits have been removed from AGENT_STATE register.

Configuration ROM definitions in the unit directory have been modified and a logical unit directory added to permit greater flexibility in the specification of targets that implement multiple logical units. The sample configuration ROM in the informative annex reflects the changes.

A new clause has been added to section 9 to describe the expected use of a target fetch agent by the BIOS or similar single-threaded application at an initiator.

The section on task management has been updated to improve clarity and to indicate that support for task management ORB’s with a *function* of ABORT TASK is optional. Targets are still required to recognize an abort task request when the initiator sets the value of *rq_fmt* to three.

A normative annex has been added to specify the minimum Serial Bus requirements for both initiators and targets.

Revision 1d (October 5, 1996)

Editorial comments discussed in Irvine, CA, have been incorporated in this revision.

The definition of “reserved” has been changed so that a target shall not check the values of reserved fields.

The *notify* bit is advisory. That is, a target may return status even if *notify* is zero.

The initiator shall insure that *max_payload* does not specify a maximum data transfer larger than the speed code permits.

The circumstances under which a target may retry a block write transaction to an initiator’s *status_FIFO* have been clarified.

The requirement for targets to implement the STATE_CLEAR.*dreq* bit has been stated in Annex A.

Revision 1e (November 9, 1996)

Minor editorial changes throughout, *per* discussions in Redmond, WA. The name of the CURRENT_ORB register has been changed to ORB_POINTER; this global change is unmarked by change bars.

After substantial discussion, the *ad hoc* working group concluded that security issues are best handled at the command set level. SBP-2 need provide only an access control mechanism that is sufficient to validate the actual identity of the initiator, EUI-64, and to provide a means whereby an initiator that had access rights before a Serial Bus reset has priority to reestablish the same access rights ahead of other, competing initiators. As a consequence, section 8 has been substantially revised and corresponding changes made to the data structure and configuration ROM descriptions in 5.1.4 and 7.3.6 respectively.

The fetch agent state machine diagram has been updated to simplify the actions a target shall take upon a write to the DOORBELL register.

The error conditions under which a target shall not attempt a retry of a block write transaction to store completion status have been clarified.

In Annex A, the target is required to support 8-byte block read and block write requests only for the MANAGEMENT_AGENT or ORB_POINTER register.

Former Annex B, “SCSI-3 Architecture Model compliance," has been removed to a separate document under development by X3T10 that includes a description of the use of SBP-2 facilities to implement SCSI devices.

Revision 1f (November 14, 1996)

The definition of “reserved” has been updated to bring it into conformance with contemporaneous standards such as X3T10 Project 1048D, SCSI-3 Multimedia Commands.

A *function* value for management ORB’s has been set aside for command set-dependent use.

New *password* and *password_length* fields have been defined in the login ORB. The usage of these fields is command set-dependent but is intended to permit additional validation of the login ORB by a target.

An *exclusive* bit has been defined in the login ORB. When *exclusive* is set to one it causes multiple initiator targets to behave as if they supported only one login at a time.

In order to enable lower cost target hardware implementations, the format of the page table has been expanded and redundant information has been added. The net result is that the parsing of page tables may be normalized by target hardware. A requirement for octlet alignment of the page table elements was also added.

Targets shall not support broadcast write requests except as already required by IEEE Std 1394-1995 or future standards.

Logout requests are to be rejected if the *source_ID* does not match that of the currently logged-in initiator.

Revision 1g (December 4, 1996)

The response status returned by a target when an ORB with *rq_fmt* equal to three is processed (also known as a dummy ORB) is REQUEST ABORTED.

The previous revision had errors in the description of constraints that apply to page table elements, dependent upon their position within the page table. These errors have been corrected.

The descriptions of login and logout in section 8 have been clarified.

Portions of 10.4.1 have been rewritten in a simpler fashion that also permits greater target implementation flexibility in response to a task management ORB with the ABORT TASK *function*.

Revision 2 (January 9, 1997)

Subsequent to a vote by the X3T10 plenary to stabilize portions of SBP-2, this revision has been prepared; it is essentially identical to Revision 1g but without the change bars.

The sections stabilized by the plenary *exclude* the portions of SBP-2 concerned with isochronous data streams. The stabilized sections are enumerated below:

Section	Description
1	Scope and purpose
2	Normative references
3	Definitions and notation
4	Model (with the exception of 4.5)
5	Data structures (with the exception of 5.1.2.2 and 5.1.3)
6	Control and status registers (with the exception of 6.5)
7	Configuration ROM
8	Access (with the exception of 8.2.2)
9	Command execution
10	Task management
Annex A	Minimum Serial Bus node capabilities
Annex B	Sample configuration ROM

For readers unfamiliar with X3T10 process, stabilization is a significant milestone in the development of a standard. Once a document or portions thereof are stabilized they are not to be modified unless either a) there is a demonstrable flaw in the draft standard or b) the changes are agreed to by a two-thirds vote of the X3T10 plenary in which at least half of the membership votes.

American National Standard for Information Systems –

Serial Bus Protocol 2 (SBP-2)

1 Scope and purpose

1.1 Scope

This standard defines a command and data transport protocol for High Performance Serial Bus, as specified by IEEE Std 1394-1995. The transport protocol, Serial Bus Protocol 2 or SBP-2, conforms to the requirements of the aforementioned standard as well as to ISO/IEC 13123:1994, Control and Status Register (CSR) Architecture for Microcomputer Buses, and permits the exchange of commands, data and status between initiators and targets connected to Serial Bus.

1.2 Purpose

Original development work for Serial Bus Protocol (SBP) was initiated out of a desire to adapt SCSI capabilities and facilities to a serial environment. Serial interconnects offer a migration path for SCSI into the future because they may be better suited to cost reduction and speed increases than the parallel interconnects first utilized by SCSI.

As development of the standard progressed, the working group recognized the solutions provided by SBP-2 were of general applicability to large classes of Serial Bus peripheral devices. With this in mind, the development work was redirected to provide mechanisms for the delivery of commands, data and status independent of the command set or device class of the peripheral. SBP-2 provides a generic framework that may be referenced by other documents or standards that address the unique requirements of a particular class of devices. Ranked below are enhanced goals set for the design of SBP-2:

- The protocol should permit the encapsulation of commands, data and status from a diversity of command sets, legacy as well as future, in order to preserve the investment in an existing application and operating system software base;
- The protocol should enable the initiator to form an arbitrarily large set of tasks without consideration of implementation limits in the target;
- The protocol should allow the initiator to dynamically add tasks to this set while the target is active in execution of earlier tasks. The addition of new tasks should not interfere with the target's processing of tasks currently active;
- Although the protocol should enable varying levels of features and performance in target implementations, strong focus should be kept on a minimal set deemed adequate for entry-level environments;
- Within the constraints posed by the preceding goal, the hardware and software design of the initiator should not be unduly affected by variations in target capabilities;
- The protocol should take advantage of features of Serial Bus that offer improvement of functionality or permit the inclusion of new functionality. In particular, the isochronous features of Serial Bus are one of its key differentiators and should be fully supported by the protocol; and
- In order to promote the scalability of aggregate system performance, the protocol should distribute the DMA context from the initiator adapter to the target devices.

Although SBP-2 has been designed for Serial Bus as currently specified by IEEE Std 1394-1995, the Technical Committee anticipates that it will be appropriate for use with future extensions to Serial Bus as they are standardized.

2 Normative references

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

Copies of the following documents can be obtained from ANSI:

Approved ANSI standards;

Approved and draft regional and international standards (ISO, IEC, CEN/CENELEC and ITUT); and

Approved and draft foreign standards (including BIS, JIS and DIN).

For further information, contact the ANSI Customer Service Department by telephone at (212) 642-4900, by FAX at (212) 302-1286 or *via* the world wide web at http://www.ansi.org.

Additional contact information for document availability is provided below as needed.

2.1 Approved references

The following approved ANSI, international and regional standards (ISO, IEC, CEN/CENELEC and ITUT) may be obtained from the international and regional organizations that control them.

IEEE Std 1394-1995, Standard for a High Performance Serial Bus

ISO/IEC 9899:1990, Programming Languages—C

ISO/IEC 13213:1994, Control and Status Register (CSR) Architecture for Microcomputer Buses

2.2 References under development

At the time of publication, no referenced standards were still under development.

3 Definitions and notation

3.1 Definitions

3.1.1 Conformance

Several keywords are used to differentiate levels of requirements and optionality, as follows:

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

3.1.1.2 ignored: A keyword that describes bits, bytes, quadlets, octlets or fields whose values are not checked by the recipient.

3.1.1.3 may: A keyword that indicates flexibility of choice with no implied preference.

3.1.1.4 reserved: A keyword used to describe objects—bits, bytes, quadlets, octlets and fields—or the code values assigned to these objects in cases where either the object or the code value is set aside for future standardization. Usage and interpretation may be specified by future extensions to this or other standards. A reserved object shall be zeroed or, upon development of a future standard, set to a value specified by such a standard. The recipient of a reserved object shall not check its value. The recipient of a defined object shall check its value and reject reserved code values.

3.1.1.5 shall: A keyword that indicates a mandatory requirement. Designers are required to implement all such mandatory requirements to assure interoperability with other products conforming to this standard.

3.1.1.6 should: A keyword that denotes flexibility of choice with a strongly preferred alternative. Equivalent to the phrase “is recommended.”

3.1.2 Glossary

The following terms are used in this standard:

3.1.2.1 byte: Eight bits of data.

3.1.2.2 doublet: Two bytes, or 16 bits, of data.

3.1.2.3 initial node space: The 256 terabytes of Serial Bus address space that is available to each node. Addresses within initial node space are 48 bits and are based at zero. The initial node space includes initial memory space, private space, initial register space and initial units space. See either ISO/IEC 13213:1994 or IEEE Std 1394-1995 for more information on address spaces.

3.1.2.4 initial register space: A two kilobyte portion of initial node space with a base address of FFFF F000 0000_{16}. Core registers defined by ISO/IEC 13213:1994 are located within initial register space as are Serial Bus-dependent registers defined by IEEE Std 1394-1995.

3.1.2.5 initial units space: A portion of initial node space with a base address of FFFF F000 0400_{16}. This places initial units space adjacent to and above initial register space. The CSR’s and other facilities defined by unit architectures are expected to lie within this space.

3.1.2.6 isochronous channel: A relationship between a node that is the talker and one or more nodes that are listeners, identified by a channel number. One isochronous packet, identified by the channel

number, may be sent by the talker during each isochronous cycle. Channel numbers are allocated cooperatively through isochronous resource management facilities.

3.1.2.7 isochronous cycle: An operating mode of Serial Bus that occurs, on average, every 125 microseconds. During an isochronous cycle, the bus is available to isochronous talkers, only. Cooperative allocation of isochronous bandwidth guarantees a bounded worst-case latency for isochronous data.

3.1.2.8 kilobyte: A quantity of data equal to 2^{10} bytes.

3.1.2.9 listener: A node that receives an isochronous packet for an isochronous channel during an isochronous cycle. There may be zero, one or more listeners for any given isochronous channel.

3.1.2.10 login: The process by which an initiator obtains access to a set of target fetch agents. The target fetch agents and their control and status registers provide a mechanism for an initiator to convey ORB's to the target.

3.1.2.11 login ID: A value assigned by the target during the login process. For all logins except isochronous, the login ID establishes a relationship between an initiator and a task set; in this case it is an initiator ID. In the case of an isochronous login, the login ID establishes a relationship between an initiator and an isochronous stream. The login ID is used to identify subsequent requests from an initiator; in some cases the login ID is not present in the operation request block and its value is implicit.

3.1.2.12 node ID: The 16-bit node identifier defined by IEEE Std 1394-1995 that is composed of a bus ID portion and a physical ID portion. The physical ID is uniquely assigned as a consequence of Serial Bus initialization.

3.1.2.13 octlet: Eight bytes, or 64 bits, of data.

3.1.2.14 operation request block: A variable length data structure fetched from system memory by a target in order to execute the command encapsulated within it.

3.1.2.15 quadlet: Four bytes, or 32 bits, of data.

3.1.2.16 register: A term used to describe quadlet aligned addresses that may be read or written by Serial Bus transactions. In the context of this standard, the use of the term register does not imply a specific hardware implementation. For example, in the case of split transactions that permit sufficient time between the request and response subactions, the behavior of the register may be emulated by a processor within the module.

3.1.2.17 split transaction: A transaction that consists of separate request and response subactions. Subactions are considered separate if ownership of the bus is relinquished between the two. A transaction that is not split is called a unified transaction.

3.1.2.18 status block: A fixed length data structure written to system memory by a target when an operation request block has been completed.

3.1.2.19 stream: An object that represents a target's functions and resources necessary to transfer isochronous data from one or more Serial Bus channels to the device medium (the target is a listener) or to transfer isochronous data from the device medium to one or more Serial Bus channels (the target is a talker).

3.1.2.20 system memory: The portions of any node's memory resource that are directly addressable by a Serial Bus address and which accepts, at a minimum, quadlet read and write access. Computers are the most common example of nodes that make system memory addressable from Serial Bus, but any node, including those usually thought of as peripheral devices, may have system memory.

3.1.2.21 talker: A node that transmits an isochronous packet for an isochronous channel during an isochronous cycle. There shall be no more than one talker for any given isochronous channel.

3.1.2.22 task: A task is an organizing concept that represents the work to be done by a target to carry out a command. In order to perform a task, a target maintains context information for the task, which includes (but is not limited to) the command, parameters such as data transfer addresses and lengths, completion status and ordering relationships to other tasks. 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 signaled to the initiator. While a task is active, it makes use of both target resources and initiator resources.

3.1.2.23 task set: A group of tasks available for execution by a logical unit of a target. There may be dependencies between individual tasks within the task set specified by this standard.

3.1.2.24 terabyte: A quantity of data equal to 2^{40} bytes.

3.1.2.25 transaction: An exchange between a requester and a responder that consists of a request and a response subaction. The request subaction transmits a Serial Bus transaction such as quadlet read, block write or lock, from the requesting node to the node intended to respond. Some Serial Bus commands include data as well as transaction codes. The response subaction returns completion status and sometimes data from the responding node to the requesting node.

3.1.2.26 unified transaction: A transaction in which the request and response subactions are completed as an indivisible sequence. Between the initiation of the request and the completion of the response, subactions by nodes other than the requester or the responder are blocked. A transaction that is not unified is called a split transaction.

3.1.2.27 unit: A component of a Serial Bus node that provides processing, memory, I/O or some other functionality. Once the node is initialized, the unit provides a CSR interface that is typically accessed by device driver software at an initiator. A node may have multiple units, which normally operate independently of each other.

3.1.2.28 unit architecture: The specification of the interface to and the behaviors of a unit implemented within a Serial Bus node. This standard is a unit architecture for SBP-2 targets.

3.1.2.29 unit attention: A state that a logical unit maintains while it has unsolicited status information to report to one or more logged-in initiators. A unit attention condition shall be created as described elsewhere in this standard or in the applicable command set- and device-dependent documents. A unit attention condition shall persist for a logged-in initiator until a) unsolicited status that reports the unit attention condition is successfully stored at the initiator or b) the initiator's login becomes invalid or is released. Logical units may queue unit attention conditions. After the first unit attention condition is cleared, another unit attention condition may exist.

3.1.3 Abbreviations

The following are abbreviations that are used in this standard:

CDB Command descriptor block

CIP Common isochronous packet format

CSR Control and status register

CRC Cyclical redundancy checksum

DVCR Digital video cassette recorder

EUI-64 Extended Unique Identifier, 64-bits

LUN Logical unit number

MPEG Motion picture experts group

ORB Operation request block

SBP-2 Serial Bus Protocol 2 (this standard itself)

3.2 Notation

The following conventions should be understood by the reader in order to comprehend this standard.

3.2.1 Numeric values

Decimal, hexadecimal and, occasionally, binary numbers are used within this standard. By editorial convention, decimal numbers are most frequently used to represent quantities or counts. Addresses are uniformly represented by hexadecimal numbers. Hexadecimal numbers are also used when the value represented has an underlying structure that is more apparent in a hexadecimal format than in a decimal format. Binary numbers are used infrequently and generally limited to the representation of bit patterns within a field.

Decimal numbers are represented by Arabic numerals without subscripts or by their English names. Hexadecimal numbers are represented by digits from the character set 0 – 9 and A – F followed by the subscript 16. Binary numbers are represented by digits from the character set 0 and 1 followed by the subscript 2. For the sake of legibility, binary and hexadecimal numbers are separated into groups of four digits separated by spaces.

As an example, 42, $2A_{16}$ and $0010\ 1010_2$ all represent the same numeric value.

3.2.2 Bit, byte and quadlet ordering

SBP-2 is defined to use the facilities of Serial Bus, IEEE Std 1394-1995, and therefore uses the ordering conventions of Serial Bus in the representation of data structures. In order to promote interoperability with memory buses that may have different ordering conventions, this standard defines the order and significance of bits within bytes, bytes within quadlets and quadlets within octlets in terms of their relative position and not their physically addressed position.

Within a byte, the most significant bit, *msb*, is that which is transmitted first and the least significant bit, *lsb*, is that which is transmitted last on Serial Bus, as illustrated below. The significance of the interior bits uniformly decreases in progression from *msb* to *lsb*.

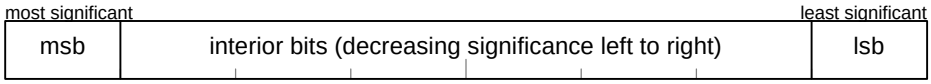

Figure 1 – Bit ordering within a byte

Within a quadlet, the most significant byte is that which is transmitted first and the least significant byte is that which is transmitted last on Serial Bus, as shown below.

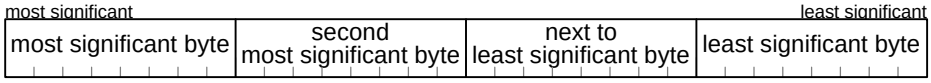

Figure 2 – Byte ordering within a quadlet

Within an octlet, which is frequently used to contain 64-bit Serial Bus addresses, the most significant quadlet is that which is transmitted first and the least significant quadlet is that which is transmitted last on Serial Bus, as the figure below indicates.

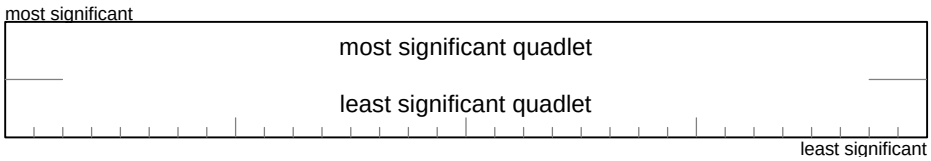

Figure 3 – Quadlet ordering within an octlet

Increasing Serial Bus addresses for quadlets correspond to increasing addresses on other buses bridged to Serial Bus, but the correlation of addresses is problematical when block transfers take place that are not quadlet aligned or not an integral number of quadlets. In such cases, no assumptions can be made about the ordering (significance within a quadlet) of bytes at the unaligned beginning or fractional quadlet end of such a block transfer, unless an application has knowledge (outside of the scope of this standard) of the ordering conventions of the other bus.

3.2.3 Register specifications

This standard precisely defines the format and function of control and status registers, CSR's. Some of these registers are read-only, some are both readable and writable and some generate special side effects subsequent to a write.

In order to precisely define CSR's, their bit fields, their initial values and the effects of read, write or other transactions, the format illustrated by Figure 4 below is used.

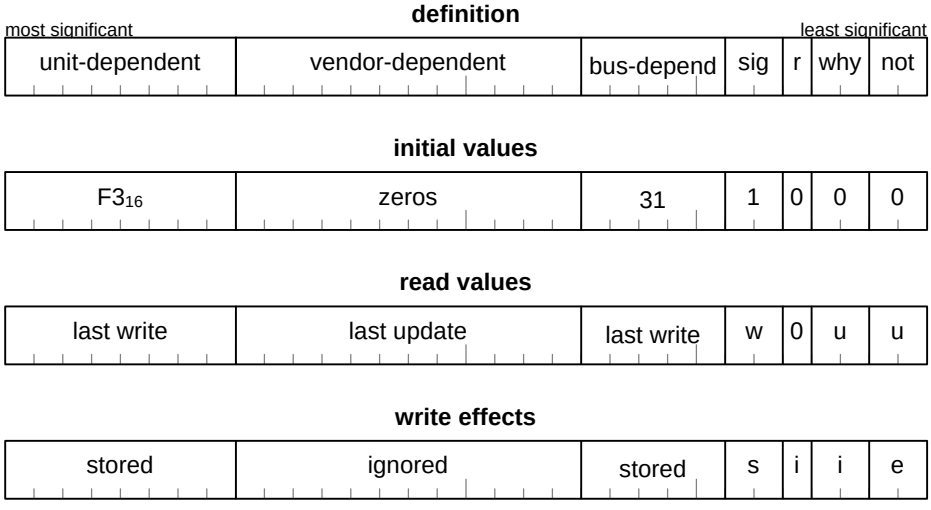

Figure 4 – CSR specification example

The register definition contains the names of register fields. The names are intended to be descriptive, but the fields are defined in the text; their function should not be inferred solely from their names. However, the following register definition field names have defined meanings.

Name	Abbreviation	Definition
bus-dependent	bus-depend	The meaning of the field shall by defined by the bus standard, in this case IEEE Std 1394-1995
reserved	r	The field is reserved for future standardization (see definitions)
unit-dependent	unit-depend	The meaning of the field shall be defined by the company or organization responsible for the unit architecture
vendor-dependent	vendor-depend or v	The meaning of the field shall be defined by the node's vendor

CSR's shall assume initial values upon the restoration of power (a power reset) or upon a write to the node's RESET_START register (a command reset). If the power reset values differ from the command reset values, they shall be separately and explicitly defined. Initial values for register fields may be described as numeric constants or with one of the terms defined for the register definition. Values for register fields subsequent to a reset may be described in the same terms or as defined below.

Name	Abbreviation	Definition
unchanged	x	The field retains whatever value it had just prior to the power reset, bus reset or command reset.

In addition to numeric values for constant fields, the read values returned in response to a quadlet read transaction may be specified by the terms below.

Name	Abbreviation	Definition
last write	w	The value of the field shall be either the initial value or, if a write or lock transaction addressed to the register has successfully completed, the value most recently stored in the field.[1]
last update	u	The value of the field shall be that most recently updated by the node hardware or software. An updated field value may be the result of a write effect to the same register address, a different register address or some other change of condition within the node.

The effects of data written to the register shall be specified by the terms below.

[1] For clarity, read values for a field in a register that accepts lock transactions may be described as *last successful lock* rather than *last write*. However, the abbreviation in both cases remains *w*. Similar liberties may be taken with the use of *conditionally stored* in place of *stored* when the action occurs as the result of a lock transaction, but the corresponding one-letter abbreviation, *s*, is also unchanged.

Name	Abbreviation	Definition
effect	e	The value of the data written to the field may have an effect on the node's state, but the effect may not be immediately visible by a read of the same register. The effect may be visible in another register or may not be visible at all.
ignored	i	The value of the data written to the field shall be ignored; it shall have no effect on the node's state.
stored	s	The value of the data written to the field shall be immediately visible by a read of the same register; it may also have other effects on the node's state.

Reserved fields within a register shall be explicitly described with respect to initial values, read values and write effects. Initial values and read values shall be zero while write effects shall be ignored. CSR's that are not implemented, either because they are optional or they fall within a reserved address space, shall abide by these same conventions if a successful completion response is returned for a read, write or lock transaction.

3.2.4 State machines

All state machines in this standard are defined in the style illustrated by Figure 5.

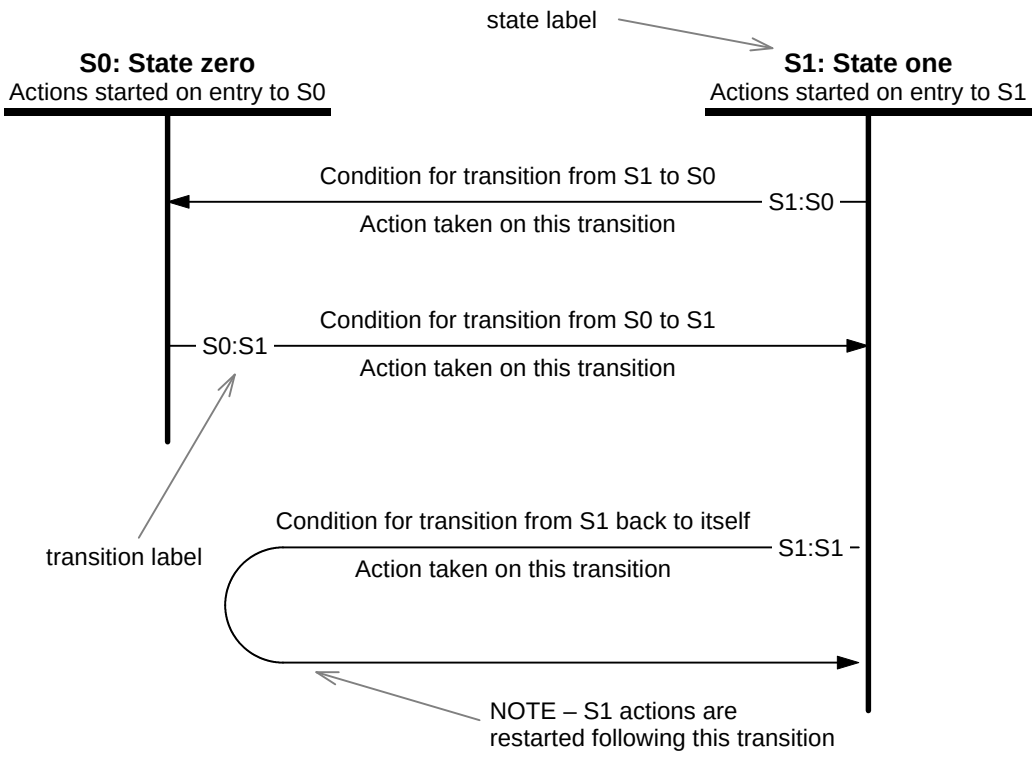

Figure 5 – State machine example

The state machines in this standard make three assumptions:

– Time elapses only within a discrete state;

- State transitions are conceptually instantaneous; the only actions taken during the transition are the setting of flags or variables and the sending of signals; and
- Each time a state is entered (or reentered from itself), the actions of that state are performed.

Multiple transitions may connect two states. In this case, the transitions are uniquely labeled by appending a character to the transition label, *e.g.*, S0:S1a and S0:S1b.

4 Model (informative)

Serial Bus Protocol 2 (SBP-2) is a transport protocol defined for IEEE Std 1394-1995, Standard for a High Performance Serial Bus. It defines facilities for requests originated by Serial Bus devices (initiators) to be communicated to other Serial Bus devices (targets) as well as the facilities required for the transfer of data or status information between the devices.

The remainder of this clause is informative and describes components of the SBP-2 model. It is intended to enhance the usefulness of the other, normative parts of this standard. In addition to the information in this clause, users of this standard should also be familiar with the CSR architecture and Serial Bus standards.

4.1 Unit architecture

In CSR architecture and Serial Bus terminology, devices implemented to this standard (targets) are units. A Serial Bus node that implements a target shall have a unit directory in configuration ROM that identifies the presence and capabilities of the target.

The unit directory in configuration ROM permits initiators to detect the presence of targets during Serial Bus configuration, whether part of system initialization or subsequent to a Serial Bus reset. The node's 64-bit identifier, EUI-64, permits detected targets to be uniquely recognized despite changes in physical addresses that may occur as the result of Serial Bus resets.

4.2 Logical units

A logical unit is part of the unit architecture and is an instance of a device model, *e.g.*, mass storage, CD-ROM or printer. A logical unit consists of a device server that is responsible to execute commands for the device, one or more stream controllers, one or more task sets that hold commands available for execution by the device server or stream controller(s) and an identifier that is unique within the domain of the target.

A target shall implement at least one logical unit, addressable as logical unit number (or LUN) zero. Additional logical units may be implemented, which may be addressable by their logical unit numbers. The logical units may implement different device models; for example, a single unit architecture might contain both a CD-ROM logical unit and an associated medium-changer logical unit. The presence of logical units within a target may be described by configuration ROM or may be discoverable by command set-dependent requests directed to the target.

4.3 Requests and responses

Target actions, such as a disk read that transfers data from device medium to system memory, are specified by means of requests created by the initiator and signaled to the target. The request is contained within a data structure called an operation request block or ORB. The eventual completion status of a request is indicated by means of a status block stored by the target at an address provided by the initiator.

This standard defines several different formats for request blocks, whose principal uses are:

- to obtain access to target resources (login requests);
- to transport command blocks (normal and stream command block requests);
- to manage task sets or to release target resources (management requests); or
- to control the flow of isochronous data (stream control requests).

Login and management requests are directed to agents that can service only a single request at a time. The ORB's for the other requests, normal command block, stream command block and stream control, provide a field that shall contain a null pointer or the address of another ORB. This permits these requests to be in a linked list, as illustrated below.

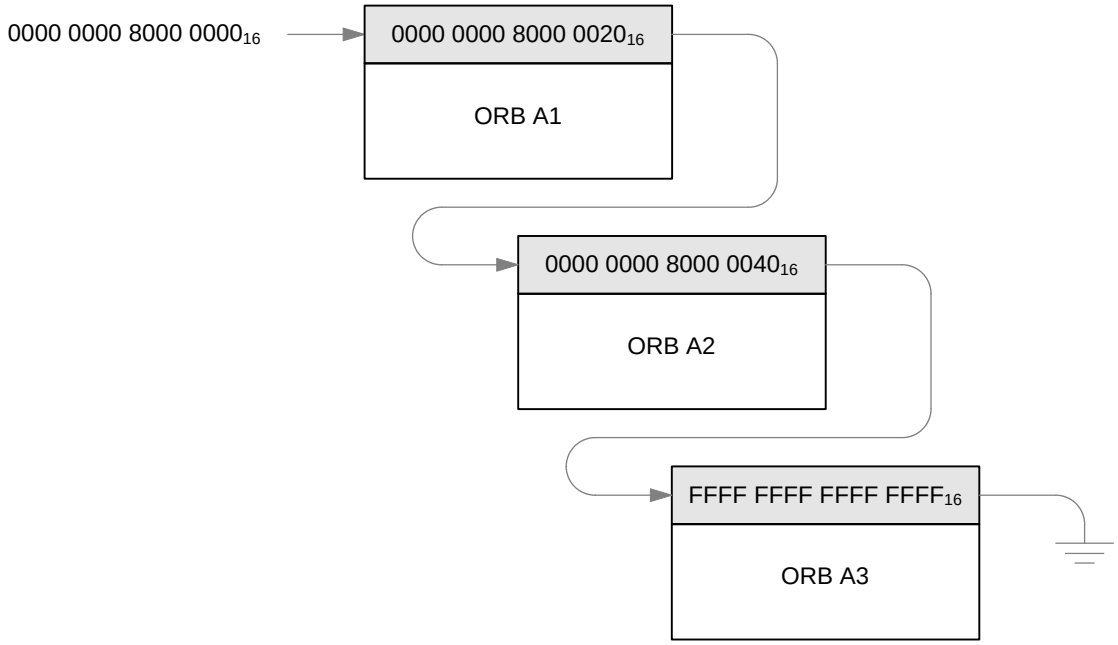

Figure 6 – Linked list of ORB's

Requests in a linked list are serviced by a target fetch agent, which reads the request(s) from initiator memory when the initiator signals the availability of request(s). The target is permitted to read ahead in the linked list; consequently the device server may reorder the execution of requests to improve performance.

When the request is complete, either in success or failure, the target stores a status block at an address specified by the initiator.

4.4 Target agents

Target agents are facilities provided by SBP-2 devices that enable initiators to signal the availability of requests. There are two fundamental types of target agent, one that can execute a single request at a time and the other that can manage queues (linked lists) of requests, as illustrated by Figure 6 above. In the first case, the initiator signals the request to the agent by means of a Serial Bus block write request with the address of the request. In the other case, the initiator appends new requests to an active list and the target agent in turn fetches the requests from system memory as target resources permit their execution.

Those target agents that manage linked lists of requests utilize context maintained at both the initiator and target to fetch requests from memory and make the request locally available to the target for execution. Although other components not visible to an initiator may form part of this context, the context minimally consists of these three elements:

- a linked list of ORB's;
- a current ORB address; and
- a doorbell.

This standard defines procedures for both the initiator and the target that permit new requests to be added to a linked list of ORB's while the target is actively fetching or executing previously enqueued requests. The procedures avoid the possibility of race conditions between the producer (initiator) and consumer (target) of the ORB's.

There are three types of target agent:

- management;
- command block; and
- stream control.

Management agents accept a variety of requests: login, task management and logout. Before any other requests can be made, it is necessary for an initiator to complete a login *via* the management agent. Once this is done, the management agent also accepts isochronous login requests and task management requests directed to either a normal (or asynchronous) task set or to a task set associated with an isochronous stream. Ultimately, management agents accept logout requests; these indicate the initiator‘s intent to release target resources previously acquired by a login. Management agents service a single request at a time and do not support linked lists.

Command block agents service the majority of target requests, either normal or stream command block requests, according to the type of login that granted access to the command block agent. Command block agents manage linked lists of requests.

Stream control agents are associated with isochronous operations, only, and are one of two agents necessary to coordinate isochronous operations. The other, the stream command block agent, accepts commands that govern the movement of isochronous data to or from the device medium. Its associated stream control agent accepts requests that meter the flow of isochronous data to or from Serial Bus. The time-critical nature of these operations requires that stream control agents support linked lists of requests, just as command block agents. Both agents are necessary to completely support isochronous operations for a target.

4.5 Streams

Streams are objects that are based upon the isochronous capabilities of Serial Bus. A stream consists of all of the target functions and resources that are necessary to transfer isochronous data from one or more Serial Bus channels to the device's medium (the target is a listener) or to transfer isochronous data from the device's medium to one or more Serial Bus channels (the target is a talker).

Streams differ fundamentally from the data transfers described by normal command blocks in two important respects. First, streams do not require any address context for the transfer of data to or from system memory; stream data is identified by a channel number and the time-ordered location of the data within the stream. Second, streams permit flow control that is synchronized to time or other time-dependent events.

Because of these differences from normal operations, two functional components are required within the target to fully control a stream: a stream task set and a stream controller. Both are described in more detail below.

4.5.1 Stream task set

Just as a normal (or asynchronous) task set consists of a set of commands that request data transfer to or from a device's medium, a stream task set also consists of a set of commands that specify medium locations for isochronous data. There are two differences:

- no system memory addresses; and

- implicit order relationships.

No system memory addresses are needed because data transferred to or from the device medium is associated with one or more Serial Bus isochronous channels. When isochronous data is read from the device medium it is made available, in order, to the stream controller described below. When isochronous data is written to the device medium it is obtained, in order, from the stream controller. In neither case is a system memory address necessary.

Isochronous data is essentially time-ordered. As a consequence, the isochronous data transferred to or from the device medium must be presented in correct order. Therefore no reordering of isochronous commands is permitted within the task set associated with an isochronous stream and the failure of any one task requires that all subsequent tasks be aborted.

This behavior required of a stream task set is that all tasks shall be executed in order and their completion status reported in the same order.

4.5.2 Stream controller

By means that are implementation dependent, an ordered data pipe is assumed to exist between a target's stream task set and the associated stream controller. The function of the stream controller is to mediate the flow of isochronous data between this data pipe and Serial Bus.

The format of data transported through the data pipe is similar to that of Serial Bus isochronous packets. The data is identified by time stamps (cycle times) and channel numbers and the payload is described in terms of its length, in bytes.

The stream controller shall:

- filter isochronous data according to channel numbers;
- transform time stamps and channel numbers in the isochronous data; and
- synchronize the flow of the isochronous data with external, time-dependent events.

Any of these operations may take place whether the stream controller is a listener or a talker. Stream control ORB's that specify these operations are independent of the stream command block ORB's in the stream task set. The stream task set and the stream controller communicate with each other through the data pipe.

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.

4.5.3 Error reporting

In addition to the data transfer errors that may be encountered by any of the stream command block ORB's, errors may occur within the isochronous stream itself as it is transferred to or from Serial Bus. These errors might include, but are not limited to:

- a missing isochronous packet or cycle start indication;
- an isochronous packet with a header CRC error;
- when the target is a talker, an underflow in the availability of data from the stream command block ORB's that causes no data to be transmitted for one or more channels during an isochronous cycle; or

- when the target is a listener, an overflow in which isochronous data from Serial Bus must be discarded because of an internal buffer overflow or a lack of stream command block ORB(s) to transfer the data to the medium.

In any of these cases, the initiator may wish to ignore all errors, report all errors but continue the isochronous stream or report the first error and halt isochronous operations. All of these are possible; if an error reporting option is selected, each stream control ORB may specify a buffer to hold an error log generated by the target. The error log is available to the initiator once the associated stream control ORB has completed.

5 Data structures

There are three classes of data structures defined by Serial Bus Protocol:

- operation request blocks (ORB's);
- page tables;
- status blocks.

These data structures may be allocated and initialized by an initiator in system memory at Serial Bus nodes. ORB's and status blocks shall be allocated at the initiator's node; page tables shall be allocated at the same node as the data buffer to which they refer.

All data structures defined by this standard shall be aligned on quadlet boundaries. These alignment requirements permit all 64-bit address pointers within ORB's to conform to the format specified below.

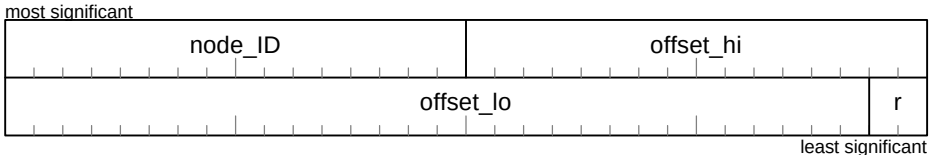

Figure 7 – Address pointer

The *node_ID* field shall specify the Serial Bus node for which the address pointer is valid, as defined by IEEE Std 1394-1995. In many cases, additional constraints on the location of data structures render the information in *node_ID* redundant. In these cases, *node_ID* is considered a reserved field or is explicitly redefined for other uses.

The *offset_hi* and the *offset_lo* fields shall together specify the most significant 46 bits of the Serial Bus offset and shall be combined with two low-order bits of zero to derive the 48-bit Serial Bus offset.

ORB's are a special case of data structures: they shall all be allocated at the initiator's node and may be organized into a linked list. Since the node ID is known for all ORB's in a given list, the address pointer format is redefined to reuse the *node_ID* field. An address pointer that references an ORB shall follow the format below.

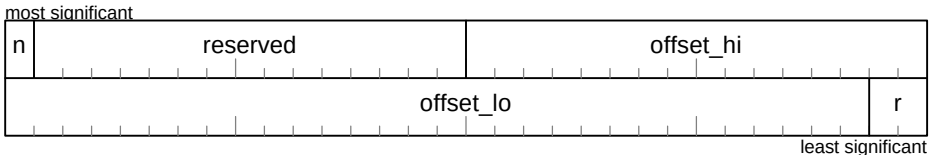

Figure 8 – ORB pointer

The *null* bit (abbreviated as *n* in the figure above) shall indicate a null pointer when it is one. In this case the target shall ignore the ORB offset fields.

5.1 Operation request blocks (ORB's)

All initiator requests for target actions are expressed within ORB's fetched by the target *via* Serial Bus read transaction(s). This standard defines different ORB formats for different uses; these ORB formats may be viewed in hierarchical relationship to each other, as illustrated below.

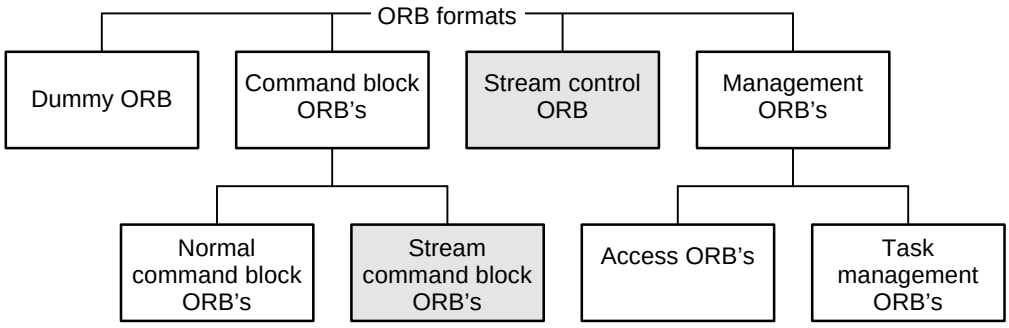

Figure 9 – ORB family tree

In the preceding figure, the ORB's that pertain solely to isochronous operations are shown shaded in gray The formats of all of the ORB's are described in the clauses that follow. This clause specifies fields that are common to all ORB's, illustrated in the figure below.

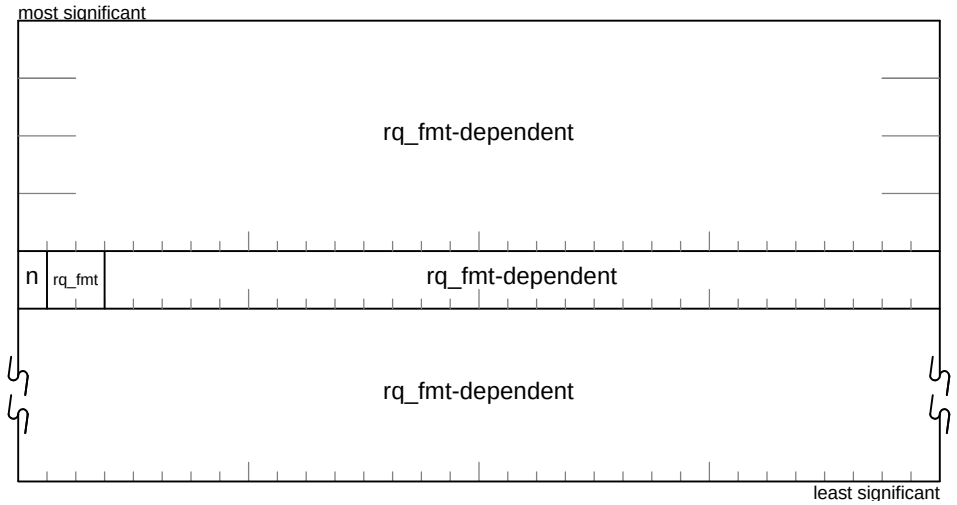

Figure 10 – ORB format

The *notify* bit (abbreviated as *n* in the figure above) advises the target whether or not completion notification is required upon request completion. When *notify* is zero, the target may elect to suppress completion notification (except in the case of an error). If *notify* is one, the target shall always store a status block in initiator memory. When the target stores a status block, it shall store it at the *status_FIFO* address specified in the ORB or (if not specified in the ORB to which the status pertains) at the address supplied in the login parameters. If the request completes with an error condition, the value of *notify* is ignored and a status block shall be unconditionally stored at the *status_FIFO* address.

The *rq_fmt* field specifies ORB format, as defined by the table below.

Value	ORB format
0	Format specified by this standard
1	Reserved for future standardization
2	Vendor-dependent
3	Dummy (NOP) request format

The format of an ORB is uniquely determined by a combination of *rq_fmt*, the command set implemented by the target and the target agent to which the ORB is signaled. This standard specifies those parts of the ORB that are invariant across command set and device type differences between targets.

5.1.1 Dummy ORB

Dummy ORB's are most frequently used as place holders within linked lists of requests. A typical example is the use of a dummy ORB in the initialization of a target fetch agent (see 9.1.1). Although the only meaningful information within a dummy ORB is contained within the first 20 bytes, the target may fetch more than this amount of data (see 7.4.3). The initiator shall insure that system memory beyond the address of a dummy ORB is accessible to the target for at least the maximum ORB fetch size implemented by the target. The format of a dummy ORB is illustrated below.

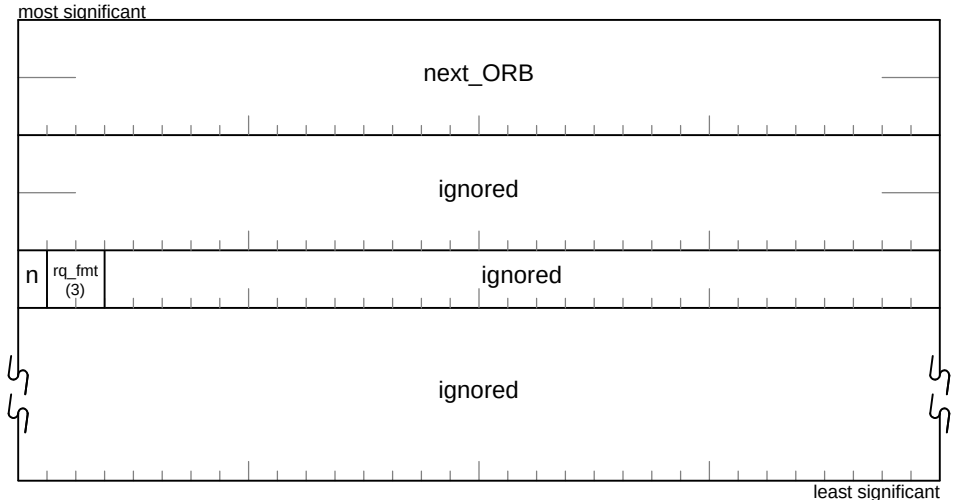

Figure 11 – Dummy ORB

The *next_ORB* field shall specify a null pointer or the address of an ORB and shall conform to the address pointer format illustrated by Figure 8. In typical usage, a dummy ORB is initialized with a null *next_ORB* pointer.

The *notify* bit is as previously defined for all ORB formats.

The *rq_fmt* field is as previously defined for all ORB formats and shall be three.

Barring catastrophic target failure, dummy requests shall complete with a status of REQUEST ABORTED. This is the normal completion status for an ORB whose *rq_fmt* field is equal to three; it is not an error.

5.1.2 Command block ORB's

Command block ORB's are used to encapsulate data transfer or device control commands for transport to the target. A target's command set and device type determine the length of the commands; this consequently determines the length of the command block ORB, which shall be fixed for a particular command set and device type. A target reports this size in configuration ROM (see 7.3.6).

NOTE – Although device designers may select arbitrary ORB lengths, system considerations may favor some ORB sizes over others. For example, as a result of commonly implemented cache line sizes, a 32-byte ORB is particularly favored by contemporary systems.

There are two kinds of command block ORB, one for normal (sometimes referred to as asynchronous) operations and one for isochronous operations.

Normal command block ORB's permit the specification of a data buffer in system memory, from which or to which data is transferred by the target.

Stream command block ORB's do not specify a data buffer in system memory. The essential nature of isochronous operations is that they involve a stream of data without system memory address context. Data bytes within a stream have relative ordering with respect to each other, but there is no explicit system memory address that is the source or the sink for the stream. Instead, an isochronous stream is coupled to a stream controller that can start, stop or pause the isochronous stream on Serial Bus. For this reason, stream command block ORB's have no data buffer address, only a stream length that governs the data transfer.

5.1.2.1 Normal command block ORB's

The format of the normal command block ORB is illustrated by the figure below.

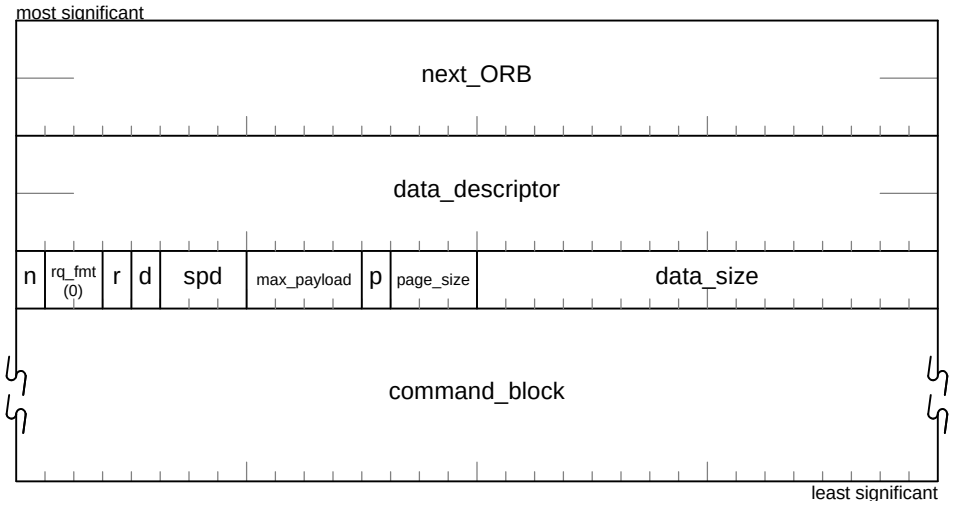

Figure 12 – Normal command block ORB

The *next_ORB* field shall specify a null pointer or the address of a dummy ORB or a normal command block ORB and shall conform to the address pointer format illustrated by Figure 8.

The *data_descriptor* field shall specify, directly or indirectly, the address of a buffer in system memory. If *data_size* is zero, the contents of *data_descriptor* are undefined and shall be ignored by the target. The format of the *data_descriptor* field shall be as specified by Figure 7. If the *page_table_present* bit is zero,

data_descriptor shall contain the address of the data buffer associated with the ORB. If the *page_table_present* bit is one, *data_descriptor* shall contain the address of the page table that describes the (possibly discontiguous) memory segments that make up the data buffer. When *data_descriptor* specifies the address of a page table, the format of the page table shall conform to that described in 5.2.

The *notify* bit and *rq_fm*t field are as previously defined for all ORB formats. The *rq_fmt* field shall be zero.

The *direction* (abbreviated as *d* in the figure above) bit specifies direction of data transfer for the buffer described by the *data_descriptor* and *data_size* fields. If the *direction* bit is zero, the target shall use Serial Bus read transactions to fetch data destined for the device medium. Otherwise, when the *direction* bit is one, the target shall use Serial Bus write transactions to store data obtained from the device medium.

The *spd* field specifies the speed that the target shall use for data transfer transactions addressed to the data buffer or page table associated with the ORB, as encoded by Table 1 below.

Table 1 – Data transfer speeds

Value	Speed
0	S100
1	S200
2	S400
3 – 7	Reserved for future standardization

The *max_payload* field specifies the maximum data transfer length, in bytes, that may be requested by the target in a single Serial Bus read or write transaction addressed to the data buffer associated with the ORB. The maximum data payload is specified as $2^{max\_payload + 2}$ bytes. The initiator shall insure that *max_payload* specifies a maximum data transfer length less than or equal to that permissible at the data transfer rate specified by *spd*.

The *page_table_present* bit (abbreviated as *p* in the figure above) shall be zero if *data_descriptor* directly addresses the data buffer associated with the ORB. When *data_descriptor* indirectly addresses the data buffer, this bit shall be one.

If the *page_table_present* bit is zero, *page_size* shall specify the underlying page size of the data buffer memory directly addressed by the *data_descriptor* field. A *page_size* value of zero indicates that the underlying page size is not specified. When *page_table_present* is one, *page_size* shall specify the page size of elements described by the page table. In both cases, if *page_size* is nonzero the page size is calculated as 2^{page_size+8} bytes.

If *page_table_present* is zero, the *data_size* field shall specify the size, in bytes, of the system memory addressed by the *data_descriptor* field. Otherwise *data_size* shall contain the number of elements in the page table addressed by *data_descriptor*.

The *command_block* field provides room for a command descriptor block whose content and meaning are not specified by this standard.

5.1.2.2 Stream command block ORB

A stream command block ORB is a structure that has the format illustrated below.

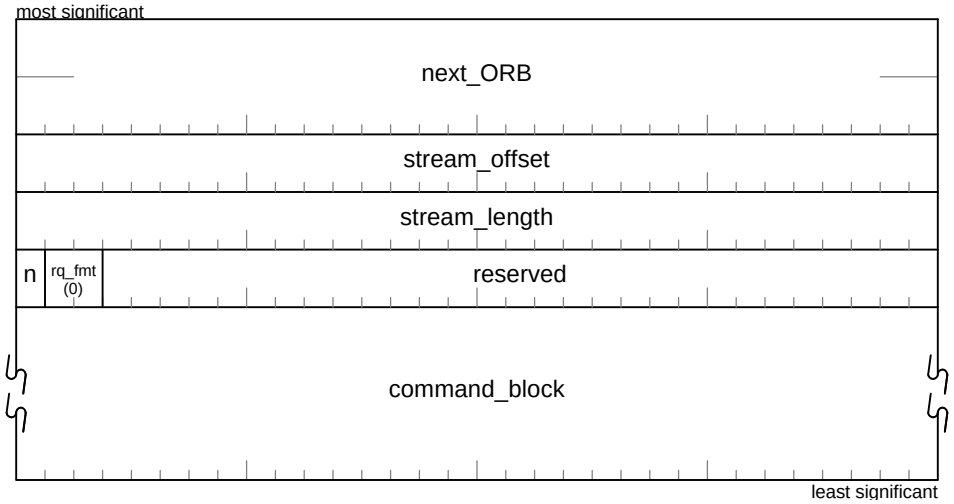

Figure 13 – Stream command block ORB

The *next_ORB* field shall specify a null pointer or the address of a dummy ORB or a stream command block ORB and shall conform to the address pointer format illustrated by Figure 8.

The *stream_offset* field specifies the location of the first byte of the isochronous data as a byte offset relative to the starting medium location indicated by the command_block. The value of *stream_offset* shall be a multiple of four.

NOTE – The command block 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, such as mass storage, 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 fields, *stream_length* and *stream_offset*, to completely characterize the request.

The *stream_length* field specifies the quantity of isochronous data, in bytes, that is to be transferred to or from the logical unit's stream controller.

The *notify* bit and *rq_fm*t field are as previously defined for all ORB formats. The *rq_fmt* field shall be zero.

The *command_block* field provides room for a command descriptor block.

5.1.3 Stream control ORB

Stream control ORB's are used to direct the action of a logical unit stream controller. The stream controller is configured at login as either a talker or a listener. When listening, the stream controller accepts isochronous data from Serial Bus in accordance with stream control ORB's, transforms the isochronous stream and then records the data on the medium as specified by isochronous requests. When talking, this process is reversed and an isochronous data stream obtained from the medium is filtered and transformed by the stream controller before isochronous packets are transmitted on Serial Bus.

The format of the stream control ORB is illustrated below.

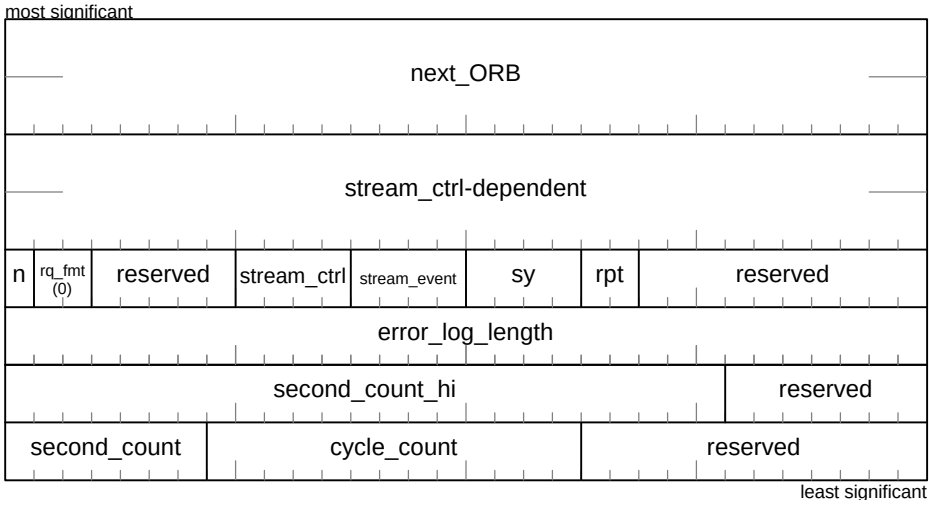

Figure 14 – Stream control ORB

The *next_ORB* field shall specify a null pointer or the address of a dummy ORB or a stream control ORB and shall conform to the address pointer format illustrated by Figure 8.

The usage of the *stream_ctrl*-dependent field varies according to the value of *stream_ctrl* and is described in more detail for each stream control function.

The *notify* bit and *rq_fm*t field are as previously defined for all ORB formats. The *rq_fmt* field shall be zero.

The *stream_ctrl* field shall specify a stream control function for the stream, as encoded below.

Value	Stream control function
0	START
1	STOP
2	PAUSE
3	UPDATE CHANNEL MASK
4	CONFIGURE PLUG
5	SET ERROR MODE
6	QUERY STREAM STATUS
7 – F_{16}	Reserved for future standardization

The START control function instructs the logical unit's stream controller to commence (or resume) talking or listening on Serial Bus. The time at which the action is to occur shall be specified by the *stream_event* field in conjunction with other stream control ORB fields.

The STOP control function instructs the logical unit's stream controller to terminate the isochronous stream and to flush the stream buffers. If the target had been listening, any isochronous data already received from Serial Bus shall be made available to the isochronous commands previously enqueued at the stream command block agent. If the target had been talking, any isochronous data obtained from isochronous commands shall be discarded. The time at which the action is to occur shall be specified by the *stream_event* field in conjunction with other stream control ORB fields.

The PAUSE control function instructs the logical unit's stream controller to suspend the transfer of isochronous data with the expectation that isochronous data transfer will resume. If the target is a talker, the stream controller shall pause on the requested stream event and shall not send any isochronous packets for the stream while paused. Subject to target implementation limitations, data from isochronous commands previously enqueued at the stream command block agent may continue to accumulate at the target while the data stream is paused. If the target is a listener, the target shall pause on the requested stream event and shall discard any isochronous packets for the stream while paused. The target may flush any isochronous data already received from Serial Bus in order to make it available to any isochronous commands previously enqueued at the stream command block agent.

The UPDATE CHANNEL MASK control function instructs the logical unit's stream controller to change the set of enabled channels. The enabled channels shall be specified by the *channel_mask* field. The time at which the action is to occur shall be specified by the *stream_event* field in conjunction with other stream control ORB fields. When *stream_ctrl* specifies a value of UPDATE CHANNEL MASK, the *stream_ctrl*-dependent field shall contain a 64-bit channel mask, as shown below.

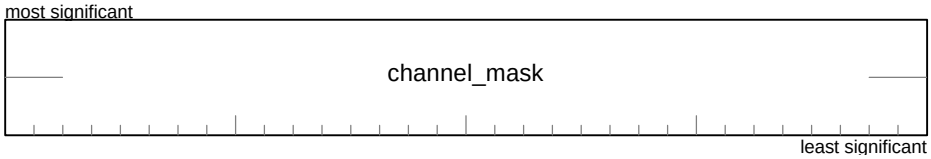

Figure 15 – Channel mask

A one in the bit position that corresponds to one of the numbered Serial Bus isochronous channels, zero to 63, indicates that the channel is to be enabled. When a channel is enabled for listening, isochronous packets observed for that channel are transferred to the medium under control of isochronous commands for the stream specified by *login_ID*. Conversely, when a channel is enabled for talking, an isochronous stream is obtained from the medium as directed by isochronous commands and isochronous packets are transmitted on Serial Bus for the enabled channel. The channel number specified is the channel number prior to any transformation that is a result of values in the logical unit's stream controller channel map.

The CONFIGURE PLUG control function instructs the logical unit's stream controller to update the specified plug control register and also to update the 64-entry channel map maintained internally for the isochronous stream. The plug control register is used in conjunction with connection management protocol to characterize aspects of isochronous operations for a single channel. The channel map specifies a transformation from a source channel to a destination channel. When listening, channel numbers observed in Serial Bus isochronous packets are replaced with numbers specified by the channel map before the isochronous data is recorded on the medium. When talking, channel numbers encountered in recorded isochronous data are replaced with numbers specified by the channel map before the isochronous packets are transmitted on Serial Bus.

NOTE – It is possible for a mapping of two or more source channels into a single destination channel to be meaningful. For example, isochronous data recorded at different times from different channels may be concatenated on the medium and subsequently replayed as a single channel.

When *stream_ctrl* specifies a value of CONFIGURE PLUG, the *stream_ctrl*-dependent field shall contain a plug and channel map configuration entry, as illustrated below.

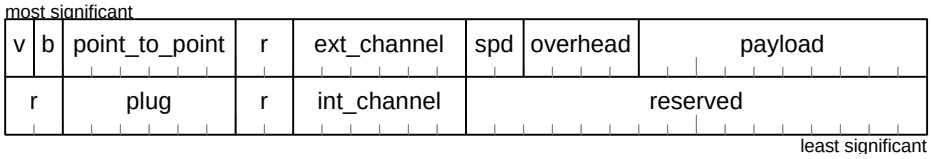

Figure 16 – Plug configuration data

The *valid* bit (abbreviated as *v* in the figure above) shall be zero if the remainder of the 8-byte entry does not contain valid plug configuration information, otherwise it shall be one.

The *broadcast* bit (abbreviated as *b* in the figure above) corresponds to the bit of the same name in the plug control register. The *broadcast* bit in the plug control register shall be updated with the value from the plug configuration entry.

The *point_to_point* field corresponds to the field of the same name in the plug control register. The value of the *point_to_point* field shall be interpreted as a two's complement number. The target shall increment or decrement the *point_to_point* field in the plug control register by the value specified. Any overflow in the *point_to_point* field in the plug control register shall be ignored by the target and no error indication shall be returned.

The *ext_channel* field corresponds to the *channel* field in the plug control register. The target shall update the *channel* field with the value of *ext_channel*.

The *spd*, *overhead* and *payload* fields correspond to the fields of the same name in the plug control register. If *plug* specifies an output plug control register, the target shall update the plug control register with the values supplied in the plug configuration entry.

NOTE – In order to specify meaningful values for *spd*, *overhead* and *payload*, the initiator may require information whose acquisition is beyond the scope of this standard. For example, application-dependent knowledge of the largest isochronous payload for any of the channel(s) in a stream of isochronous data recorded on the medium is necessary before the target, as a talker, can successfully play back the data.

The *plug* field shall specify the plug control register that shall be updated, as summarized below.

Value	Plug control register
$0 – E_{16}$	OUTPUT_PLUG[*plug*] register
$0F_{16}$	Invalid
$10 – 1E_{16}$	INPUT_PLUG[*plug* - 32] register
$1F_{16}$	Invalid

The *int_channel* field shall specify the channel number by which the channel is identified internally to the target.

NOTE – The *ext_channel* and *int_channel* fields together specify a mapping that the target shall maintain for each of the stream's 64 possible channels. This mapping specifies channel transformations that take place when the target is a listener or a talker. When the target is a listener, enabled channels are recognized according to *ext_channel* (the channel number observed in isochronous packets received by the target). Before the isochronous data is recorded on the medium, the channel number is remapped to the value specified by *int_channel*; this may be an identity map. When the target is a talker, enabled channels are recognized according to *int_channel* (the channel number recorded on the medium in the isochronous data). Before the isochronous data is transmitted on Serial Bus, the channel number is remapped to the value specified by *ext_channel*.

The usage described above for *ext_channel* and *int_channel* assumes the presence of an implementation-dependent, 64-entry channel map maintained by the target for each isochronous stream. The channel map shall describe a transformation from a source channel to a destination channel. At the time a stream identifier (*login_ID*) is obtained by means of an isochronous login, the channel map for the stream shall reset to the identity map.

The results of a CONFIGURE PLUG control function in a stream control ORB may be confirmed by a read of the plug control register specified by plug. Note that some of the fields in a plug control register-*broadcast*, *point_to_point*, *channel*, *spd* and *overhead*-may be directly stored by a Serial Bus lock transaction addressed to the register as well as updated by a stream control ORB.

The SET ERROR MODE control function instructs the logical unit's stream controller to configure its error handling mode as specified by the *rpt* field. Dependent upon the value of the *rpt* field, an error log in system memory may also be specified.

The QUERY STREAM STATUS control function instructs the logical unit's stream controller to return status information that indicates the whether or not the stream controller is ready to accept a START control function.

NOTE – Assume that a target is to be instructed to listen to isochronous data and transfer the stream to device medium. If the starting medium location is at a nonzero byte offset relative to a block boundary, some implementations may require time to read previously recorded data from the medium before being ready to commence recording the new isochronous data. Subsequent to enqueuing an isochronous command at the stream command block agent, the QUERY STREAM STATUS control function may be used to determine if the target is ready to accept a START control function.

The *stream_event* field is valid only if the *stream_ctrl* field specifies a value of START, STOP, PAUSE or UPDATE CHANNEL MASK. When one of these control functions is specified, the *stream_event* field specifies the time at which the action is to take place, as encoded below.

Value	Stream event code
0	IMMEDIATE
1	CYCLE MATCH
2	SY MATCH
3	FIRST DATA
$4 – F_{16}$	Reserved for future standardization

A value of IMMEDIATE instructs the logical unit's stream controller to perform the specified action as soon as possible, within the capabilities of the target implementation.

A value of CYCLE MATCH instructs the logical unit's stream controller to perform the specified action at the cycle time specified by *second_count_hi*, *second_count* and *cycle_count*.

A value of SY MATCH instructs the logical unit's stream controller to perform the specified action on the isochronous cycle for which the *sy* field of an isochronous packet for any enabled channel matches the *sy* field in the stream control ORB. A *stream_event* value of SY MATCH is valid only if the logical unit's stream controller is configured as a listener.

A value of FIRST DATA instructs the logical unit's stream controller to perform the specified action when isochronous data is observed for any enabled isochronous channel. A *stream_event* value of FIRST DATA is valid only if the logical unit's stream controller is configured as a listener.

NOTE – A *stream_event* field value of FIRST DATA may have effects similar to IMMEDIATE, in that it is possible for isochronous data to be recorded immediately. The difference between the two stream events is

apparent if no isochronous packets for any of the enabled channels are present when the stream control ORB is executed. If IMMEDIATE is specified, CYCLE MARK packets are recorded as each cycle start is observed. If FIRST DATA is specified, no packets are recorded until the first isochronous packet for an enabled channel is observed. When this event occurs, a CYCLE MARK packet with the most recent cycle start data is recorded followed by a DATA packet for the enabled channel.

The *sy* field is valid only if the logical unit's stream controller is configured as a listener and the *stream_event* field specifies SY MATCH. See the preceding description of *stream_event*.

The *rpt* field specifies an operational mode for the logical unit's stream controller, as described in the table below. The *rpt* field is valid only if the *stream_ctrl* field specifies SET ERROR MODE.

Value	Error handling mode
0	Report errors and halt stream
1	Report errors and continue stream
2	Ignore all errors
3	Reserved for future standardization

Different sorts of errors may be detected when the logical unit's stream controller is configured as a talker or a listener. If an error occurs, the stream controller shall take one of three actions, as specified by the value of *rpt*:

- Report the error by writing an entry to the error log and then halting isochronous data transfers by performing the equivalent of a STOP control function with a *stream_event* value of IMMEDIATE;
- Report the error by writing an entry to the error log but continue isochronous data transfers;
- Ignore the error (the error log, if provided, is ignored) and continue isochronous data transfers.

A more detailed description of isochronous errors and how they are handled is provided in 12.4.

The *error_log_length* field is valid only if *stream_ctrl* is equal to SET ERROR MODE. In this case, the *stream_ctrl*-dependent field shall specify the address of a buffer in system memory and *error_log_length* shall specify the size, in bytes, of the buffer. The most significant 16 bits of the buffer address shall be zeroed by the initiator and shall be ignored by the target; the target shall address the error log with the node ID of the initiator.

If *error_log_length* is zero, the *rpt* field shall be treated as if it has a value of two and no isochronous data transfer errors shall be reported.

The *second_count_hi*, *second_count* and *cycle_count* fields are valid only if the *stream_ctrl* field specifies START, STOP, PAUSE or UPDATE CHANNEL MASK and the *stream_event* field specifies CYCLE MATCH. Together, these fields specify a cycle time for comparison with the target's cycle clock. An equal comparison occurs if the *second_count_hi* field matches the field of the same name in the target's BUS_TIME register and if both the *second_count* and *cycle_count* fields match their corresponding fields in the target's CYCLE_TIME register.

5.1.4 Management ORB's

Management ORB's are 32-byte data structures that encapsulate several types of management request:

- access requests (which include login and logout requests); and
- task management requests.

Unlike the normal command block, stream command block and stream control ORB's (which are all implicitly associated with a particular task set or stream by virtue of the fetch agent to which they are addressed), most management ORB's explicitly declare the task set or stream for which they are intended.

Management ORB's have the general format illustrated below. Note that since they lack a *next_ORB* field, these ORB's cannot be linked together to form a list.

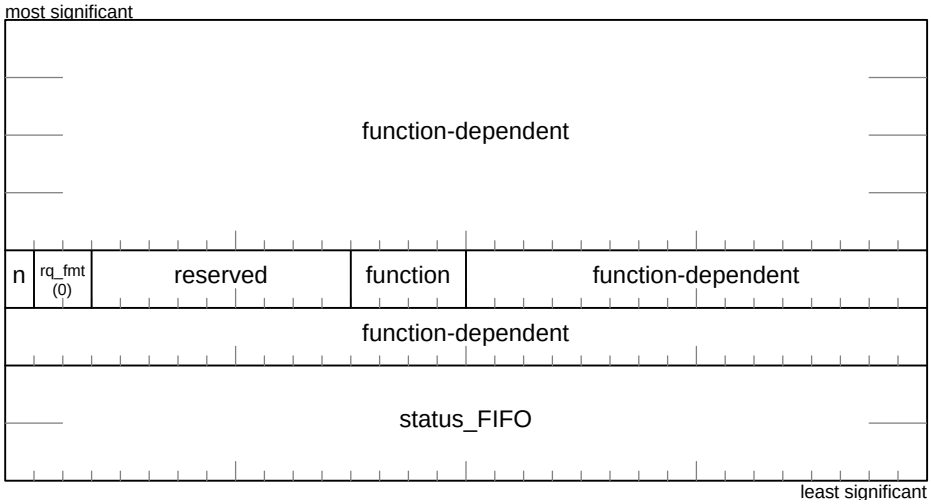

Figure 17 – Management ORB

The *notify* bit and *rq_fm*t field are as previously defined for all ORB formats. The *rq_fmt* field shall be zero.

The *function* field specifies the management function requested, as defined by the table below.

Table 2 – Management request functions

Value	Management function
0	LOGIN
1	QUERY LOGINS
2	ISOCHRONOUS LOGIN
3	RECONNECT
4	Command set-dependent
5 – 6	Reserved for future standardization
7	LOGOUT
8 – 9	Reserved for future standardization
A_{16}	TERMINATE TASK
B_{16}	ABORT TASK
C_{16}	ABORT TASK SET
D_{16}	CLEAR TASK SET
E_{16}	LOGICAL UNIT RESET
F_{16}	TARGET RESET

The *status_FIFO* field shall specify an address allocated for the return of status information. Except when an ORB format explicitly specifies a *status_FIFO* address, this same address shall be used by the target for the return of status for all subsequent requests associated with this login. The *status_FIFO* field shall conform to the format for address pointers specified by Figure 7 and shall address the same node as the initiator; consequently the *node_ID* field of this address pointer shall be reserved.

5.1.4.1 Login ORB

Before any other requests (except QUERY LOGINS) can be made of a target, the initiator shall first complete a login procedure that uses the ORB format shown below.

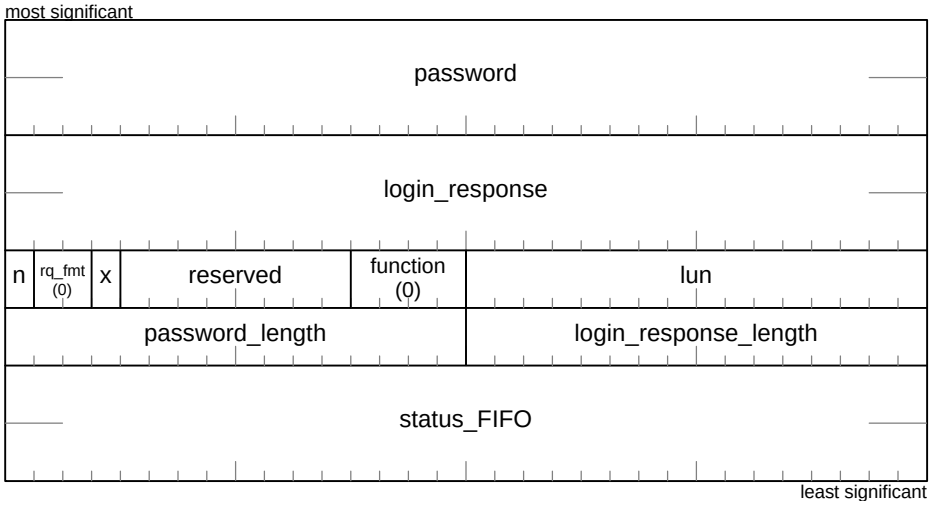

Figure 18 – Login ORB

The *password* and *password_length* fields may specify optional, command set-dependent information used to validate the login request. If *password_length* is zero, the *password* field may contain immediate data. When *password_length* is nonzero, the *password* field shall specify the address of a buffer. The buffer shall be accessible to a Serial Bus block read request with a data transfer length less than or equal to *password_length*. The format and usage of password data, whether immediate or indirectly addressed, are beyond the scope of this standard.

The *login_response* and *login_response_length* fields specify the address and size of a buffer allocated for the return of the login response. The buffer shall be accessible to a Serial Bus block write request with a data transfer length less than or equal to *login_response_length*. If the status block stored at the *status_FIFO* address indicates an unsuccessful login, no login response data shall be stored.

The *notify* bit, *rq_fmt* and *status_FIFO* fields are as previously defined for management ORB formats.

The *exclusive* bit (abbreviated as *x* in the figure above) shall specify target behavior with respect to concurrent login to a logical unit. When *exclusive* is zero, the target, subject to its own implementation capabilities, may permit more than one initiator to login to a logical unit. If *exclusive* is one the target shall permit only one login to a logical unit at a time; see 8.2.1 for a description of target behavior.

The *login_response* and *status_FIFO* fields shall conform to the format for address pointers specified by Figure 7. All of these buffers shall be in the same node as the initiator; consequently the *node_ID* field of these addresses shall be reserved.

The *lun* field specifies the logical unit number (LUN) to which the request is addressed.

Upon successful completion of a login, the login response is returned in the format illustrated below.

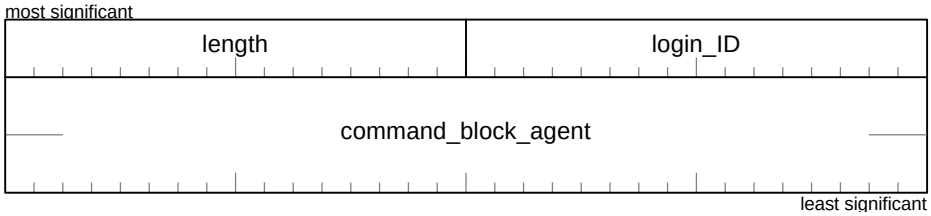

Figure 19 – Login response

The *length* field shall specify the length, in bytes, of the login response data. If *login_response_length* in the login request is too small for the transfer of all the login response data, the *length* field shall not be adjusted to reflect the truncation.

The initiator shall use the *login_ID* value returned by the target to identify all subsequent requests directed to the target's management agent that pertain to this login.

The *command_block_agent* field specifies the base address of the agent's CSR's, which are defined in 6.4.

5.1.4.2 Login query ORB

An initiator may determine the EUI-64 and node ID of all currently logged-in initiators by means of a login query request, whose format is illustrated below.

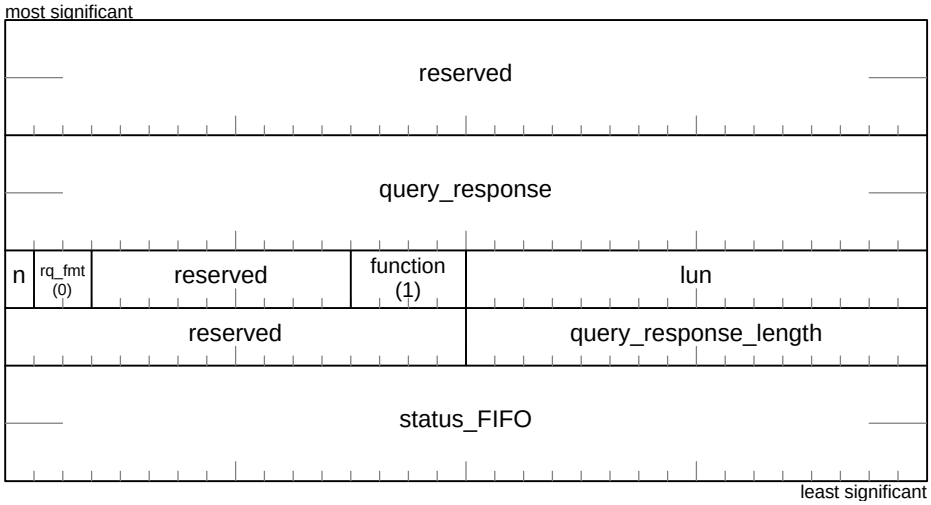

Figure 20 – Login query ORB

The *query_response* and *query_response_length* fields specify the address and size of a buffer for the return of the query results. The buffer shall be accessible to a Serial Bus block write request with a data transfer length less than or equal to *query_response_length*.

The *notify* bit, *rq_fmt* and *status_FIFO* fields are as previously defined for management ORB formats.

The *query_response* and *status_FIFO* fields shall conform to the format for address pointers specified by Figure 7. All of these buffers shall be in the same node as the initiator; consequently the *node_ID* field of these addresses shall be reserved.

The *lun* field specifies the logical unit number (LUN) to which the request is addressed.

The query response data returned shall have the following format.

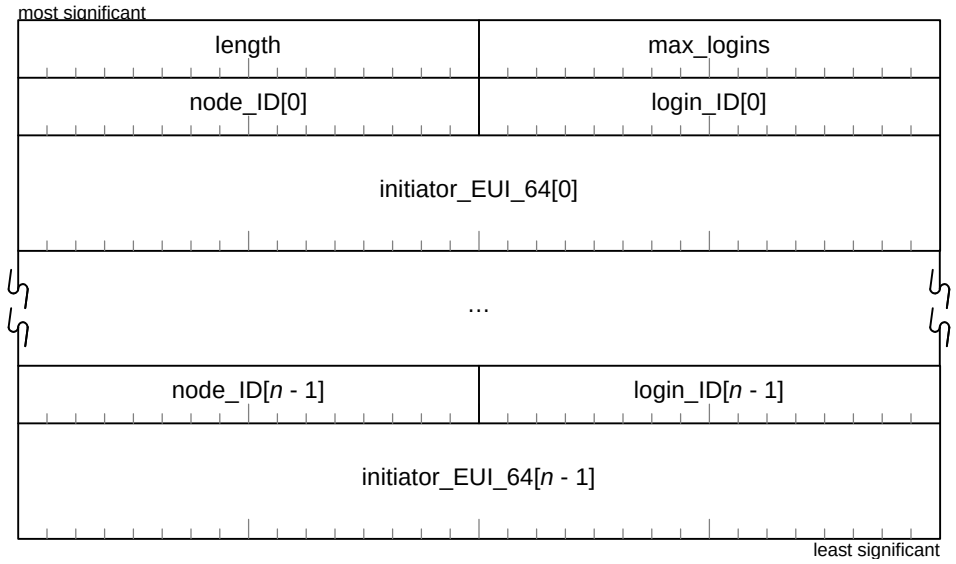

Figure 21 – Login query response format

The *length* field shall specify the length, in bytes, of the query response data. If *query_response_length* in the login query request is too small for the transfer of all the query response data, the *length* field shall not be adjusted to reflect the truncation. The value of the *length* field shall be equal to 4 + 12 * *n*, where *n* is the number of logged-in initiators.

The *max_logins* field shall specify the maximum concurrent logins that may be accepted by the logical unit.

The remainder of the query response is a variable-length array of 12-byte entries, each of which contains a *node_ID*, *login_ID* and *initiator_EUI_64* field, one for each logged-in initiator.

The *node_ID* field of an entry shall specify the node ID of a logged-in initiator. If a Serial Bus reset has occurred since the login was established and the initiator has not reconnected the login, the *node_ID* field shall have a value of $FFFF_{16}$.

The *login_ID* field of an entry shall specify the login ID provided to the initiator as a result of its successful login.

The *initiator_EUI_64* field of an entry shall specify the EUI-64 obtained by the target from the initiator's configuration ROM at the time the login was validated.

5.1.4.3 Isochronous login ORB

Before any stream requests can be made of a target, the initiator shall first complete an isochronous login procedure that uses the ORB format shown below.

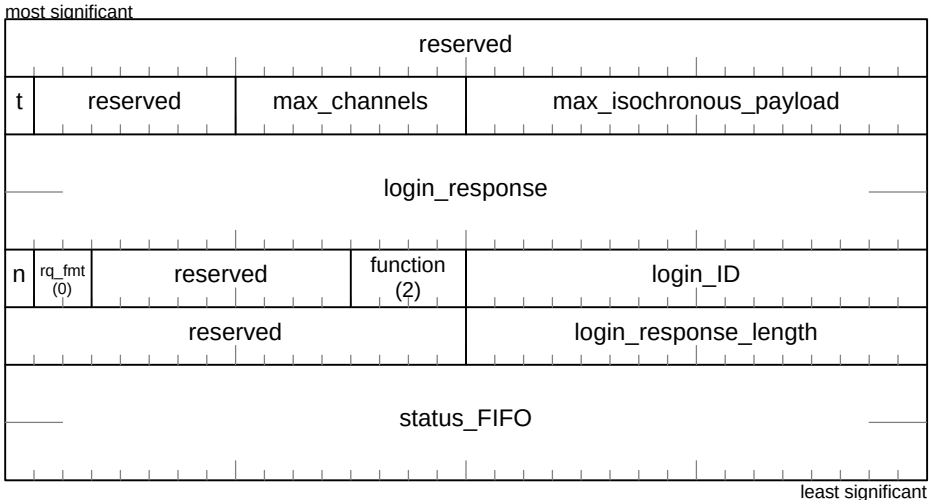

Figure 22 – Isochronous login ORB

The *talker* bit (abbreviated as *t* in the figure above) shall specify the type of isochronous login requested. If the target resources are to be configured for listening, *talker* shall be zero.

The *max_channels* field specifies the maximum number of isochronous channels that may be simultaneously transmitted or received. This field is ignored if the *type* field has a value other than two or three.

The *max_isochronous_payload* field is the aggregate maximum isochronous payload that the target is requested to support for the stream. That is, the sum of all the *data_length* fields of Serial Bus isochronous packets transmitted or received for all of the stream's isochronous channels shall not exceed *max_isochronous_payload* in a single isochronous cycle.

The *login_response* and *login_response_length* fields specify the address and size of a buffer allocated for the return of the login response. The buffer shall be accessible to a Serial Bus block write transaction with a data transfer length less than or equal to *login_response_length*. If the status block stored at the *status_FIFO* address indicates an unsuccessful login, no login response data shall be stored.

The *notify* bit, *rq_fmt* and *status_FIFO* fields are as previously defined for management ORB formats.

The *login_ID* field shall specify a login ID value obtained as the result of a successful login.

The *login_response* and *status_FIFO* fields shall conform to the format for address pointers specified by Figure 7. All of these buffers shall be in the same node as the initiator; consequently the *node_ID* field of these addresses shall be reserved.

Upon successful completion of an isochronous login, the login response is returned in the format illustrated below.

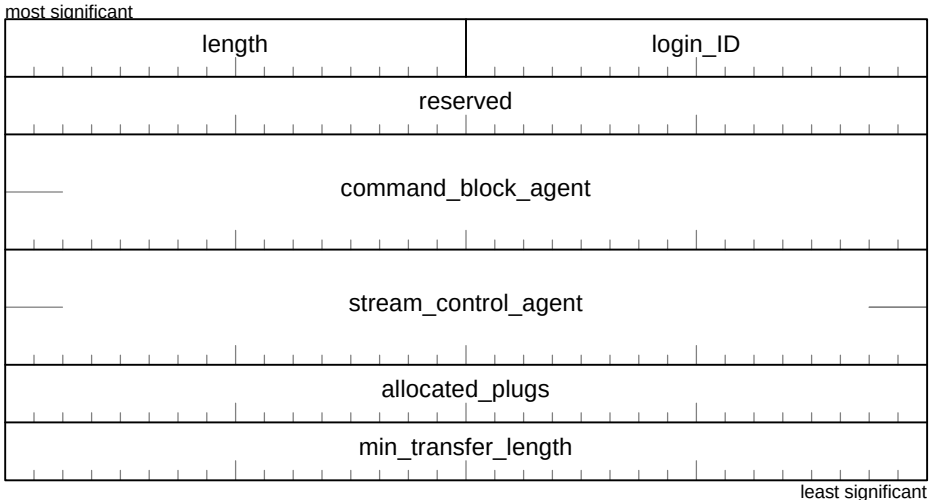

Figure 23 – Isochronous login response

The *login_ID* identifies an isochronous stream for which target resources have been allocated. The initiator shall use this value to identify all subsequent requests directed to the target's management agent that pertain to this login.

The *command_block_agent* and *stream_control_agent* fields specify the base address of the agent's CSR's, which are defined in 6.4.

The *allocated_plugs* field shall specify which plug control registers are allocated to the stream. The *allocated_plugs* field is valid only if the *type* field in the login parameters has a value of two or three. The *allocated_plugs* is a bit mask, where each bit represents one of the 31 OUTPUT_PLUG or INPUT_PLUG registers. The least significant bit corresponds to OUTPUT_PLUG[0] or INPUT_PLUG[0], according to the value of the *type* field in the login parameters. Bits of increasing significance correspond to the input or output plug of ordinal *n*, where *n* is incremented according to the significance of the bit within the *allocated_plugs* field. Since there are no more than 31 output or input plugs, the most significant bit of *allocated_plugs* shall be zero.

The *min_transfer_length* field is valid only if the *type* field in the login parameters has a value of two or three. The value of *min_transfer_length* specifies the minimum *stream_length* value required by the target in stream command block ORB's in order to sustain the isochronous data rate requested by the login. If the initiator presents any stream command block ORB's whose *stream_length* value is less than this minimum, the target may experience underflow or overflow in isochronous data while talking or listening at the requested rate.

5.1.4.4 Reconnect ORB

After a Serial Bus reset it is possible for an initiator's 16-bit node ID to change. Since a target validates all writes to agent CSR's by the node ID of the initiator, an initiator shall reestablish validated access before it may signal new requests to the target. This is accomplished by means of a reconnect request, with the format shown below.

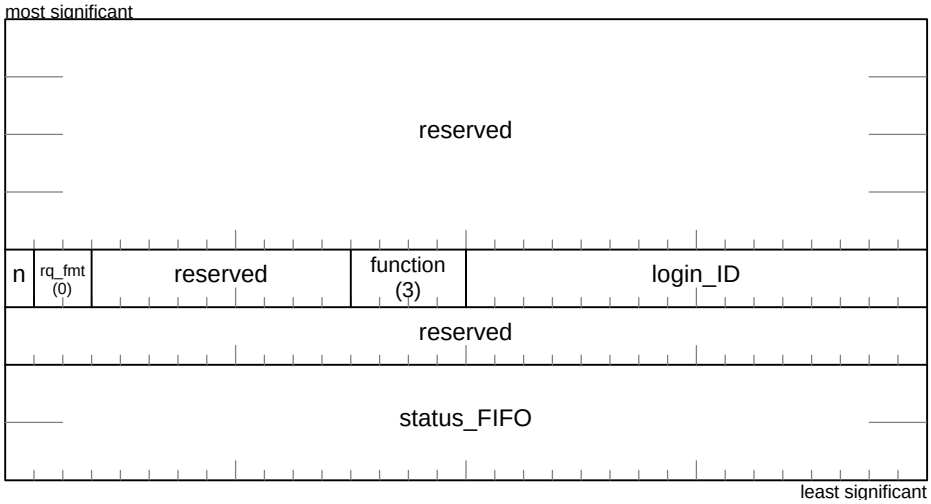

Figure 24 – Reconnect ORB

The *notify* bit, *rq_fmt* and *status_FIFO* fields are as previously defined for management ORB formats.

The *login_ID* field shall specify a login ID value obtained as the result of a successful login. The target shall verify that the EUI-64 of the initiator requesting the login reestablishment matches the EUI-64 previously saved by the target for the *login_ID*.

Upon successful reestablishment of the login, the initiator may signal requests to the target agent at the same CSR addresses returned in the original login response data. The initiator shall also use the *login_ID* value to identify all requests directed to the target's management agent that pertain to the reestablished login.

Any isochronous logins established with the same *login_ID* value specified in the reconnect ORB are also reestablished. The login ID's of the isochronous logins remain the same.

5.1.4.5 Logout ORB

When an initiator wishes to relinquish its access privileges for a logical unit or an isochronous stream, it shall perform a logout with the ORB format shown below.

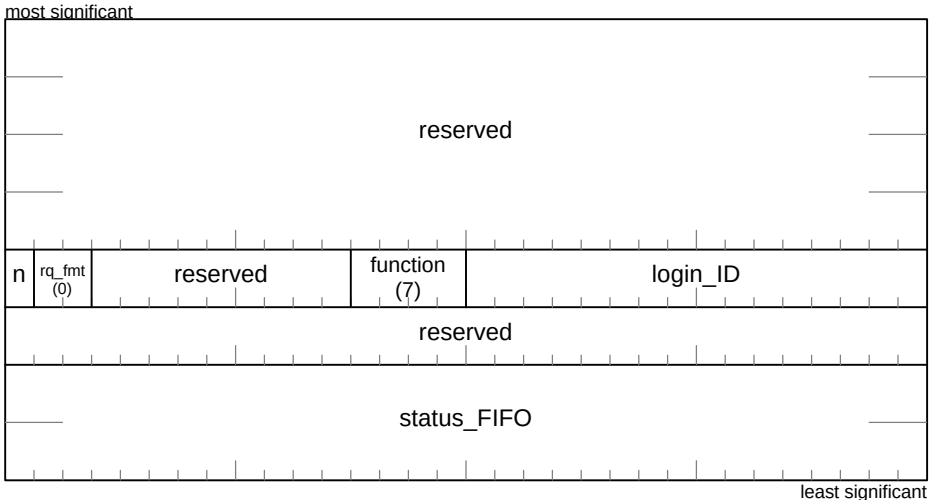

Figure 25 – Logout ORB

The *notify* bit, *rq_fmt* and *status_FIFO* fields are as previously defined for management ORB formats.

The *login_ID* field shall specify a login ID value obtained as the result of a successful login or isochronous login.

5.1.4.6 Task management ORB

The task management ORB is used to control task sets. This ORB shall have the format defined below.

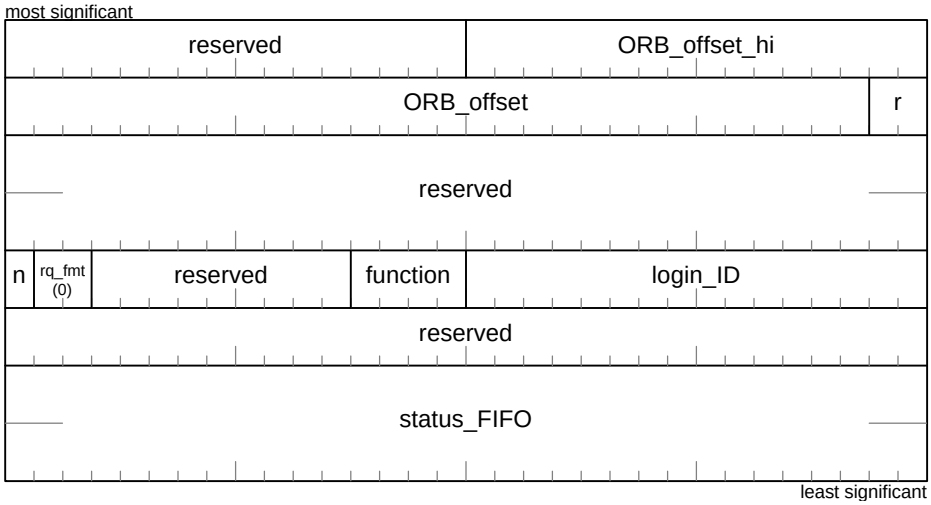

Figure 26 – Task management ORB

The *ORB_offset_hi* and *ORB_offset_lo* fields together form the *ORB_offset* field, which identifies the task to which the management function applies. *ORB_offset* is derived by taking the least significant 48 bits of the Serial Bus address of the ORB and discarding the least significant two bits. The *ORB_offset* field is ignored unless the *function* field is TERMINATE TASK or ABORT TASK. All tasks are uniquely identified by their Serial Bus address of the ORB that initiated the task.

The *notify* bit, *rq_fmt* and *status_FIFO* fields are as previously defined for management ORB formats.

The *function* field shall contain a value of TERMINATE TASK, ABORT TASK, ABORT TASK SET, CLEAR TASK SET, LOGICAL UNIT RESET or TARGET RESET, as defined by Table 2.

The *login_ID* shall be set to the value returned in the login response data for the task set to which the task management request is directed.

NOTE – In the case of TARGET RESET, which does not pertain to any one task set, *login_ID* shall be set to a value obtained as the result of any successful login completed by the initiator.

5.2 Page tables

The data buffer associated with an ORB is specified by the *data_descriptor*, *page_table_present*, *page_size* and *data_size* fields. The data buffer is a logically contiguous area in system memory. As previously described, when *page_table_present* is zero, the data buffer is also contiguous within Serial Bus address space. In this case, *data_descriptor* contains the 64-bit address of the data buffer and *data_size* specifies its length, in bytes.

In the other case, when *page_table_present* is equal to one, the data buffer is composed of segments that are discontiguous within Serial Bus address space and it is necessary to use a page table to describe the segments that form the data buffer. The page table is a variable-length array of elements whose format is shown below. Each element describes one segment that is contiguous within Serial Bus address space. Page table elements shall be octlet aligned.

Figure 27 – Page table element (when *page_size* equals four)

NOTE – In the figure above, the field widths of *segment_base_lo* and *segment_offset*, 20 and 12 bits, respectively, are chosen only for the purposes of illustration. The size of *segment_base_lo* and *segment_offset* vary according to *page_size*. The field width, in bits, of *segment_offset* shall be *page_size+8*. In the example shown above, the page size is assumed to be 4096 bytes.

The *segment_length* field shall specify the length, in bytes, of the portion of the data buffer described by the page table element. The value of *segment_length* shall be less than or equal to 2^{page_size+8}.

The *segment_base_hi* and *segment_base_lo* fields together shall specify the base address of the segment within the node's 48-bit system memory address range.

The *segment_offset* field shall specify the starting address for data transfer within the segment.

The 64-bit system memory address used to address the data is formed by the concatenation of the 16-bit *node_ID* field from the *data_descriptor* field in the ORB, *segment_base_hi*, *segment_base_lo* and *segment_offset*.

In all page table elements, the sum of *segment_length* and *segment_offset* shall be less than or equal to 2^{page_size+8}.

In addition to the preceding requirements, the values of *segment_length* and *segment_offset* are constrained by their position within the page table. These additional restrictions are summarized below.

Element	Total page table elements		
	1	**2**	***n* (where *n* >= 3)**
0	No additional restrictions	*segment_length* = 2^{page_size+8} - *segment_offset*	
1 – *n*-2	—	*segment_offset* = 0	segment_offset = 0 *segment_length* = 2^{page_size+8}
n-1	—	—	*segment_offset* = 0

The presence of a page table is indicated by the value of *page_table_present* in the ORB. When *page_table_present* is nonzero, the *data_descriptor* field in the ORB shall contain the address of the page table and the *data_size* field shall contain the number of elements in the page table.

When a page table is used it shall be located in the same node as the data buffer it describes. The *spd* and *max_payload* fields of the ORB shall describe data transfer capabilities for both the data buffer and the page table. The page table shall be contiguous within Serial Bus address space and shall be accessible to Serial Bus block read transactions with a *data_length* less than or equal to 2^{page_size+8} bytes so long as a block read transaction does not cross Serial Bus address boundaries that occur every 2^{page_size+8} bytes.

5.3 Status block

Upon completion of a request, if the *notify* bit in the ORB is one or if there is error status to report, the target shall signal the initiator by storing all or part of the status block shown below. If the *status_FIFO* address is explicitly provided as part of the ORB to which the status pertains, the target shall store the status block at the address specified. Otherwise, the target shall store the status block at the *status_FIFO* address provided by the initiator as part of the login parameters. The target may also store unsolicited status at this address, as defined in 9.4.

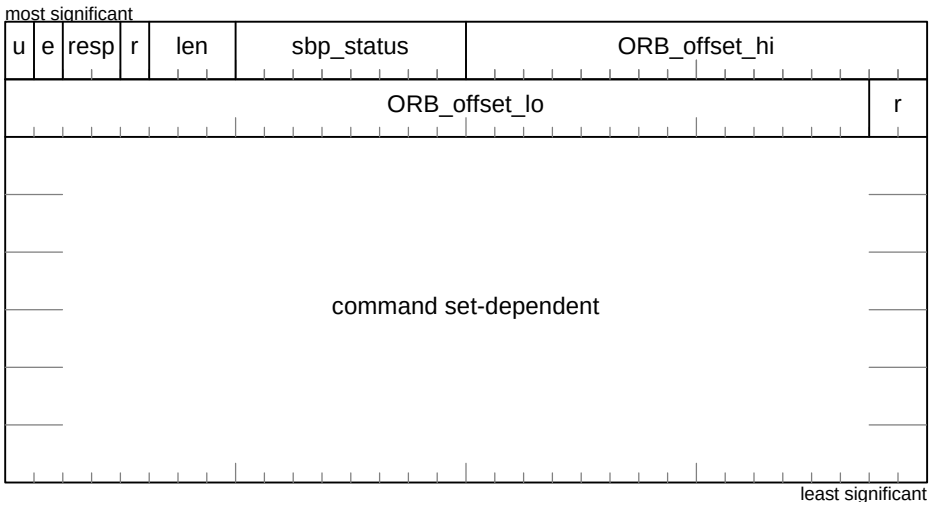

Figure 28 – Status block format

The target shall store a minimum of eight bytes of status information and may store up to the entire 32 bytes defined above so long as the amount of data stored is an integral number of quadlets. The target shall use a single Serial Bus block write transaction to store the status block at the *status_FIFO* address.

The *unsolicited* bit (abbreviated as *u* in the figure above) shall specify the usage of the *ORB_offset* field. If the *unsolicited* bit is zero, the status block pertains to a request identified as described below. When

unsolicited is one, the status block is not related to any outstanding request and the contents of *ORB_offset* shall be ignored.

The *end_of_list* bit (abbreviated as *e* in the figure above) shall specify the value of the *next_ORB* field of the ORB to which the status block pertains at the time the ORB was most recently fetched. When *end_of_list* is zero, the *next_ORB* field did not contain a null pointer. Otherwise *next_ORB* was null when last fetched.

The *resp* field shall specify the SBP-2 response status for the request identified by *ORB_offset*. Response values are encoded by *resp* as shown by the table below.

Value	Name	Description
0	REQUEST COMPLETE	The request completed without transport protocol error
1	TRANSPORT FAILURE	The target detected a nonrecoverable transport failure that prevented the completion of the request
2	ILLEGAL REQUEST	There is an unsupported field or bit value in the ORB; the *sbp_status* field may provide additional information
3	VENDOR DEPENDENT	The meaning of *sbp_status* shall be specified by the vendor

The *len* field shall specify the quantity of valid status block information stored at the *status_FIFO* address. The size of the status block is encoded as *len* + 1 quadlets.

The *sbp_status* field provides additional information that qualifies the response status in *resp*. The meanings assigned to *sbp_status* are specified by the table below.

Value	Description
0	No additional sense to report
1	Invalid request type
2	Speed not supported
3	Page size not supported
4	Access denied
5	Logical unit not supported
6	Maximum payload too small
7	Too many channels
8	Resources unavailable
9	Function rejected
10	Login ID not recognized
FF_{16}	Unspecified error

The *ORB_offset_hi* and *ORB_offset_lo* fields together form an *ORB_offset* field that uniquely identifies the ORB to which the status block pertains. If *unsolicited* is zero, the target shall form *ORB_offset* from the least significant 48 bits of the Serial Bus address used to fetch the ORB; the least significant two bits shall be discarded. When the status block contains unsolicited status, the *ORB_offset* field shall be ignored.

The remainder of the status block, up to a maximum of 32 bytes, is command set-dependent.

6 Control and status registers

The control and status registers (CSR's) implemented by a target shall conform to the requirements defined by this standard and its normative references. The CSR's may be arranged in four principal categories:

- core registers required by ISO/IEC 13213:1994;
- bus-dependent registers required by IEEE Std 1394-1995;
- unit architecture registers required by this standard; and
- bus-dependent registers required by supplements to IEEE Std 1394-1995 (principally the plug control registers, or PCR's).

Unless otherwise specified, all registers shall support quadlet read and quadlet write transactions. The registers defined in 6.3 and 6.4 shall ignore broadcast write requests.

These registers are described in turn in the clauses that follow.

6.1 Core registers

The CSR architecture standardizes the locations and functions of core registers. The addresses of these registers are specified in terms of byte offsets within initial register space, where the base address of initial register space is FFFF F000 0000_{16} relative to initial node space. IEEE Std 1394-1995 should be consulted for detailed descriptions of these core registers; the table below summarizes which core registers are mandatory for targets.

Offset	Register name	Description
0	STATE_CLEAR	State and control information
4	STATE_SET	Sets STATE_CLEAR bits
8	NODE_IDS	Contains the 16-bit *node_ID* value used to address the node
$0C_{16}$	RESET_START	Resets the node's state
18_{16} – $1C_{16}$	SPLIT_TIMEOUT	Time limit for split transactions

6.2 Serial Bus-dependent registers

The CSR architecture reserves a portion of initial register space for bus-dependent uses. Serial Bus defines registers within this address space, whose addresses are specified in terms of byte offsets within initial register space, where the base address of initial register space is FFFF F000 0000_{16} relative to initial node space. IEEE Std 1394-1995 should be consulted for detailed descriptions of these core registers; the table below summarizes which Serial Bus-dependent registers are mandatory for targets.

Offset	Register name	Description
210_{16}	BUSY_TIMEOUT	Controls transaction layer retry protocols

Isochronous capabilities are optional for targets. If a target supports isochronous operations, it shall be cycle master capable and isochronous resource manager capable as well as isochronous capable. These capabilities require that additional Serial Bus-dependent registers shall be implemented, as summarized by the table below.

Offset	Register name	Description
200_{16}	CYCLE_TIME	24.576 MHz clock required for isochronous operation
204_{16}	BUS_TIME	System time in seconds
$21C_{16}$	BUS_MANAGER_ID	Contains the *node_ID* of the bus manager, if one is present
220_{16}	BANDWIDTH_AVAILABLE	Known location for Serial Bus isochronous bandwidth allocation
224_{16} – 228_{16}	CHANNELS_AVAILABLE	Known location for Serial Bus isochronous channel allocation

6.3 MANAGEMENT_AGENT register

The MANAGEMENT_AGENT register permits the initiator to signal the address of a management ORB to the target. This register shall support 8-byte block write requests whose *destination_offset* is equal to the address of the MANAGEMENT_AGENT register and shall reject quadlet write requests and all other block write requests. The format of this register is illustrated below.

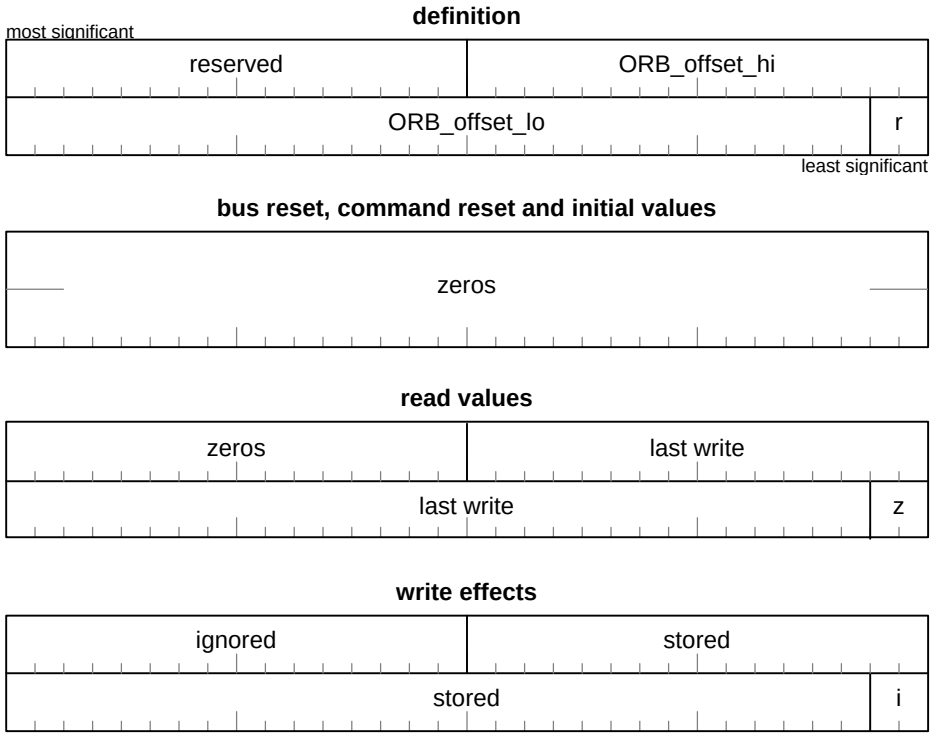

Figure 29 – MANAGEMENT_AGENT format

The *ORB_offset_hi* and *ORB_offset_lo* fields together form an *ORB_offset* field from which a Serial Bus address is derived when the management ORB is fetched. The Serial Bus address shall be formed from the concatenation of the 16-bit node ID of the initiator (available to the target as the *source_ID* field of the block write request that updated the register), the *ORB_offset* field and two least significant bits of zero.

An initiator may signal a request by means of an 8-byte block write transaction that specifies the address of the request. If the management agent is busy with another request, the block write shall be rejected

with a response of *resp_conflict_error*. If the write transaction is successful, the management agent shall fetch the request specified by *ORB_offset* and execute it. Unsuccessful write transactions shall not affect the execution of any request(s) in progress.

IEEE Std 1394-1995 reserves a portion of initial units space for bus-dependent use; the MANAGEMENT_AGENT register shall be located at address FFFF F001 0000_{16} or above within the node's 48-bit address range. The address of the management agent is specified by the *csr_offset* field in the Management_Agent entry in configuration ROM (see 7.3.3).

6.4 Command block and stream control agent registers

Unlike the management agent, which services a single request at a time, the command block and stream control agents manage linked lists of requests from which they fetch requests. For this reason they are referred to as fetch agents. Each target fetch agent has a set of control and status registers that lie within the target's initial units space; the fetch agent CSR's shall be located at address FFFF F001 0000_{16} or above within the node's 48-bit address range.

Although the location of each fetch agent's CSR's is not fixed, the relative relationship of the registers is fixed within a contiguous block of eight quadlets, as defined by the table below.

Relative offset	Name	Description
00_{16}	AGENT_STATE	Reports fetch agent state
04_{16}	AGENT_RESET	Resets fetch agent
08_{16}	ORB_POINTER	Address of ORB
10_{16}	DOORBELL	Signals fetch agent to refetch an address pointer
14_{16}	STATUS_ACKNOWLEDGE	Acknowledges the initiator's receipt of unsolicited status
18_{16} – $1C_{16}$		Reserved for future standardization

The base address of each set of fetch agent's CSR's is obtained from the login response returned by the target as part of a successful login.

6.4.1 AGENT_STATE register

The AGENT_STATE register is a read-only register that provides information about the current condition of the fetch agent. The definition is given by Figure 30 below.

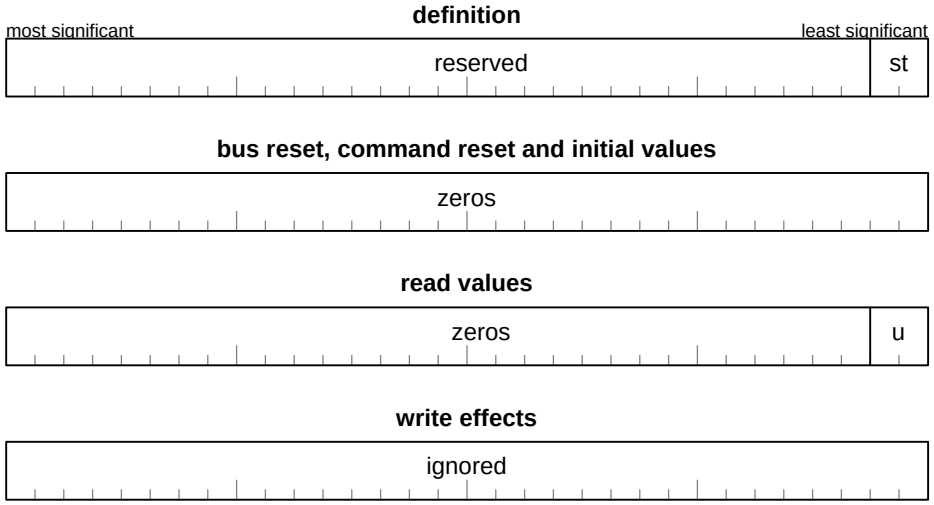

Figure 30 – AGENT_STATE format

The *st* field shall specify the current operational state of the fetch agent, as encoded by the values in the table below.

Value	Fetch agent state
0	RESET
1	ACTIVE
2	SUSPENDED
3	DEAD

6.4.2 AGENT_RESET register

The AGENT_RESET register permits an initiator to reset the operational state of a target fetch agent. The definition of this write-only register is given by Figure 31 below.

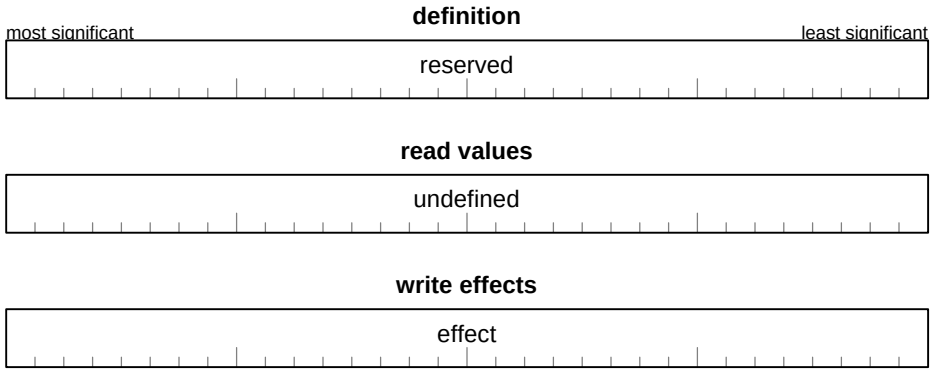

Figure 31 – AGENT_RESET format

A quadlet write of any value to this register shall cause all fetch agent CSR's to be reset to their initial values, after which the fetch agent shall transition to the reset state.

6.4.3 ORB_POINTER register

The ORB_POINTER register contains the address of an ORB in system memory. This register shall support 8-byte block write requests whose *destination_offset* is equal to the address of the MANAGEMENT_AGENT register and shall reject quadlet write requests and all other block write requests. The definition is given by Figure 32 below.

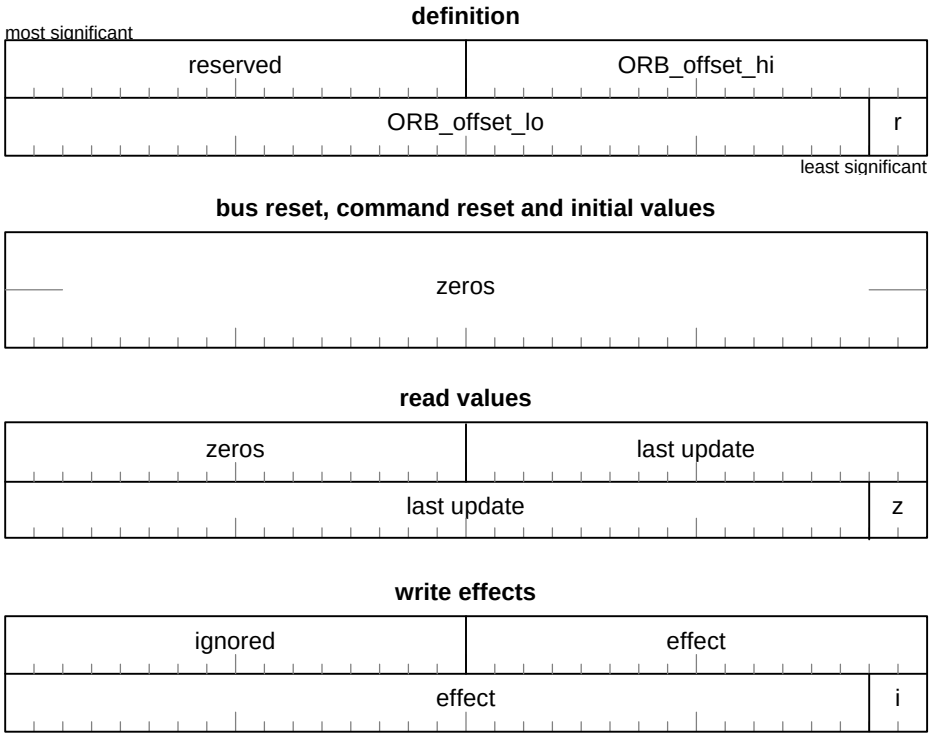

Figure 32 – ORB_POINTER format

The *ORB_offset_hi* and *ORB_offset_lo* fields together form an *ORB_offset* field from which a Serial Bus address is derived when the ORB is fetched. The Serial Bus address shall be formed from the concatenation of the 16-bit node ID of the initiator (available to the target as a result of a login), the *ORB_offset* field and two least significant bits of zero.

The effects of a write transaction to the ORB_POINTER register are dependent upon the value of *st* in the AGENT_STATE register. If the target agent is in the DEAD state, writes to the ORB_POINTER register shall be ignored. If the target agent is in the RESET or SUSPENDED state, a write to this register shall cause the *ORB_offset* to be stored and the agent to transition to the ACTIVE state. If the target agent is in the ACTIVE state, a write to the ORB_POINTER register may cause unpredictable target behavior.

6.4.4 DOORBELL register

The DOORBELL register provides a location at which the initiator may signal the target that a linked list of requests has been updated. The definition of this write-only register is given by Figure 33 below.

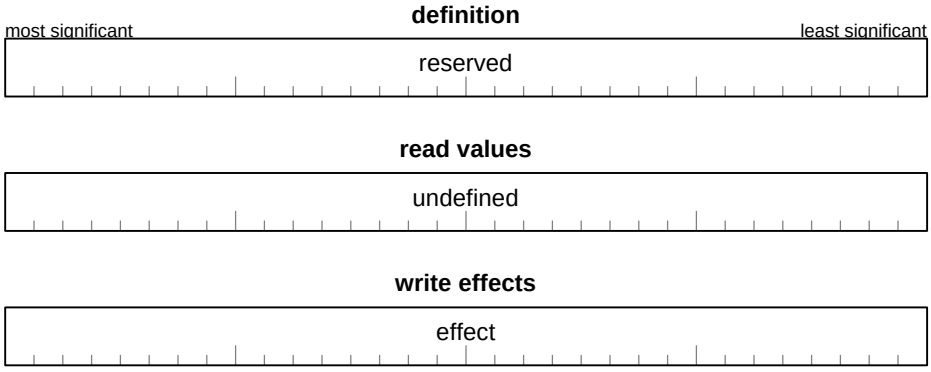

Figure 33 – DOORBELL format

A quadlet write of any value to this register shall cause the fetch agent's *doorbell* variable to be set to one.

6.4.5 STATUS_ACKNOWLEDGE register

The STATUS_ACKNOWLEDGE register provides a location at which the initiator may signal the target that unsolicited status has been received. The definition of this write-only register is given by Figure 34 below.

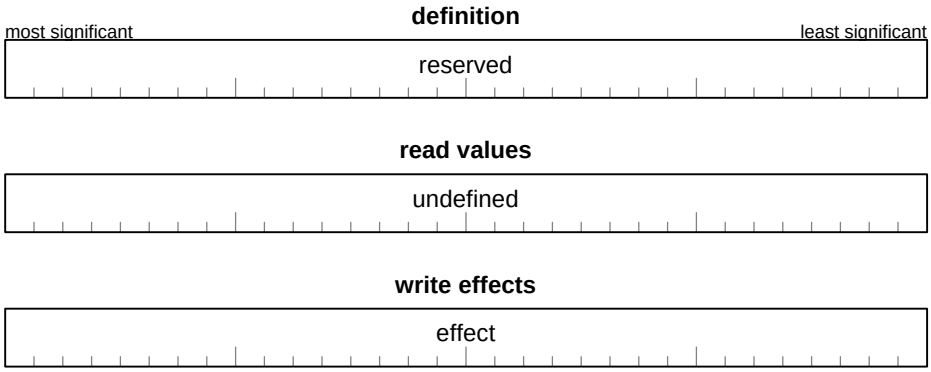

Figure 34 – STATUS_ACKNOWLEDGE format

A quadlet write of any value to this register shall cause the fetch agent's *status acknowledgment* variable to be set to one. A successful login or isochronous login shall set the *status acknowledgment* variable to one.

6.5 Plug control registers

In addition to the Serial Bus-dependent registers described above, Serial Bus also reserves a portion of initial units space for bus-dependent uses. Addresses within this space are specified in terms of byte offsets within initial register space, where the base address of initial register space is FFFF F000 0000_{16} relative to initial node space. Optional registers for connection management protocol are defined within this address space.

Isochronous capabilities are optional for targets. If a target supports isochronous operations, it shall support connection management protocol. Connection management protocol requires that additional registers shall be implemented, as summarized by the table below.

Offset	Register name	Description
900_{16}	OUTPUT_MASTER_PLUG	Common output plug controls for the node
904_{16} – $97C_{16}$	OUTPUT_PLUG	Output plug control registers for individual isochronous channels, OUTPUT_PLUG[0] through OUTPUT_PLUG[30]
980_{16}	INPUT_MASTER_PLUG	Common input plug controls for the node
984_{16} – $98C_{16}$	INPUT_PLUG	Input plug control registers for individual isochronous channels, INPUT_PLUG[0] through INPUT_PLUG[30]

The plug control registers shall support quadlet read and lock transaction and shall reject write transactions. The plug control registers may support block read access.

6.5.1 OUTPUT_MASTER_PLUG register

The OUTPUT_MASTER_PLUG register provides information about and permits control of common aspects of a node's OUTPUT_PLUG registers. If a node can function as an isochronous talker , it shall implement the OUTPUT_MASTER_PLUG register. The register definition is given by Figure 35 below.

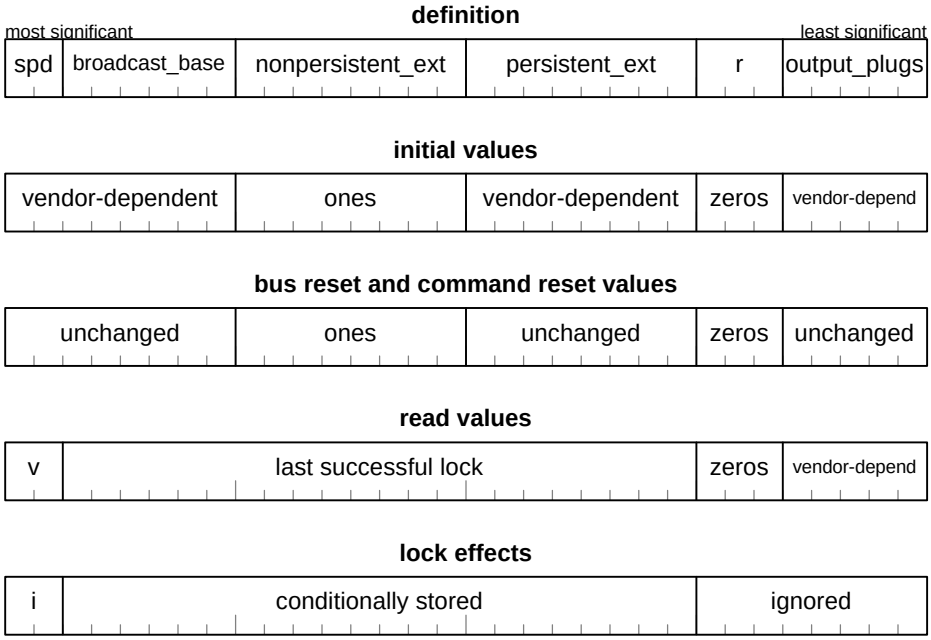

Figure 35 – OUTPUT_MASTER_PLUG format

The *spd* field shall specify the maximum speed for isochronous data transmission that any of the OUTPUT_PLUG registers may use, as encoded by Table 1.

The *broadcast_base* field shall specify the base isochronous channel number used to determine the channel number transmitted for broadcast out connections. When broadcast out connections are established for plug(s) for which a point-to-point connection does not simultaneously exist, the channel field of the OUTPUT_PLUG register(s) shall be set to 63 if *broadcast_base* equals 63 and otherwise shall be set to (*broadcast_base* + *n*) modulo 63, where *n* is the ordinal of OUTPUT_PLUG[*n*]. See 12.1 for more information on broadcast out and point-to-point connections.

The *nonpersistent_ext* and *persistent_ext* fields are reserved for future standardization.

The *output_plugs* field shall specify the total count of OUTPUT_PLUG registers implemented by a node. Between zero and 31 OUTPUT_PLUG registers may be implemented. If one or more OUTPUT_PLUG registers are implemented, they shall lie within the contiguous address range FFFF F000 0904_{16} to FFFF F000 0900_{16} + 4 * *output_plugs*, inclusive.

6.5.2 OUTPUT_PLUG register

The OUTPUT_PLUG register permits the description and control of both broadcast and point-to-point connections that originate with the associated plug. OUTPUT_PLUG registers are optional and are required only if the node can function as an isochronous talker. OUTPUT_PLUG registers shall be implemented within a contiguous address space; OUTPUT_PLUG[*n*] refers to the register addressable at FFFF F000 0904_{16} + 4 * *n*. The register definition is given by Figure 36 below.

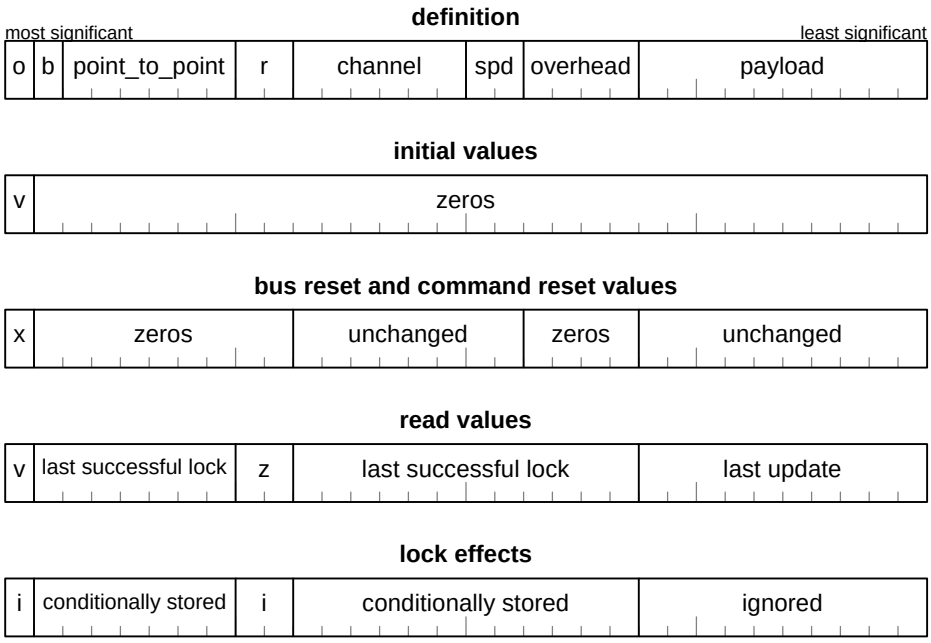

Figure 36 – OUTPUT_PLUG format

The *online* bit (abbreviated as *o* in the figure above) shall specify the on-line status of the plug resources controlled by the OUTPUT_PLUG register. An *online* bit value of zero shall indicate that the plug is off-line and not capable of transmitting isochronous data. An *online* value of one shall indicate that the plug may be configured and used for isochronous data transmission.

NOTE – Plug status may change dynamically from off-line to on-line as device resources become unavailable or available. The causes of a change in plug status reported by the *online* bit are vendor-dependent.

The *broadcast* bit (abbreviated as *b* in the figure above) shall specify whether or not a broadcast connection exists for the output plug, where a value of zero indicates that no such connection exists.

The *point_to_point* field shall specify the number of point-to-point connections that exist for the output plug.

The *channel* field shall specify the isochronous channel number used in isochronous data transmissions for the plug.

The *spd* field shall specify the speed to be used for isochronous data transmissions for the plug, as encoded by Table 1. If *spd* is set to a value greater than the value of the *spd* field in the OUTPUT_MASTER_PLUG register, isochronous data transmissions shall be disabled for the plug.

The *overhead* field shall encode a value used in the calculation of the isochronous bandwidth necessary to allocate for isochronous data transmissions associated with the plug. Isochronous bandwidth is expressed in terms of bandwidth allocation units, as defined by IEEE Std 1394-1995. One bandwidth allocation unit represents the time required to transmit one quadlet of data at a future (larger than the present maximum definition) S1600 data rate, roughly 20 nanoseconds. If *overhead* is nonzero, the total bandwidth allocation necessary is expressed as *overhead* * 32 + (*payload* + 3) * $2^{4 - spd}$. Otherwise, the total bandwidth allocation may be obtained from 512 + (*payload* + 3) * $2^{4 - spd}$. In the preceding formulae, *overhead*, *payload* and *spd* represent the values of these fields in the OUTPUT_PLUG register.

The *payload* field shall specify the maximum number of quadlets that may be transmitted in a single isochronous packet for this plug. A *payload* value of zero indicates a maximum of 1024 quadlets; all other values for *payload* represent a maximum of the value itself.

NOTE – The value of *payload* does not include the isochronous header, header CRC or data CRC required as part of an isochronous packet; it counts only those quadlets that are part of the isochronous data payload.

6.5.3 INPUT_MASTER_PLUG register

The INPUT_MASTER_PLUG register provides information about and permits control of common aspects of a node's INPUT_PLUG registers. If a node can function as an isochronous listener, it shall implement the INPUT_MASTER_PLUG register. The register definition is given by Figure 37 below.

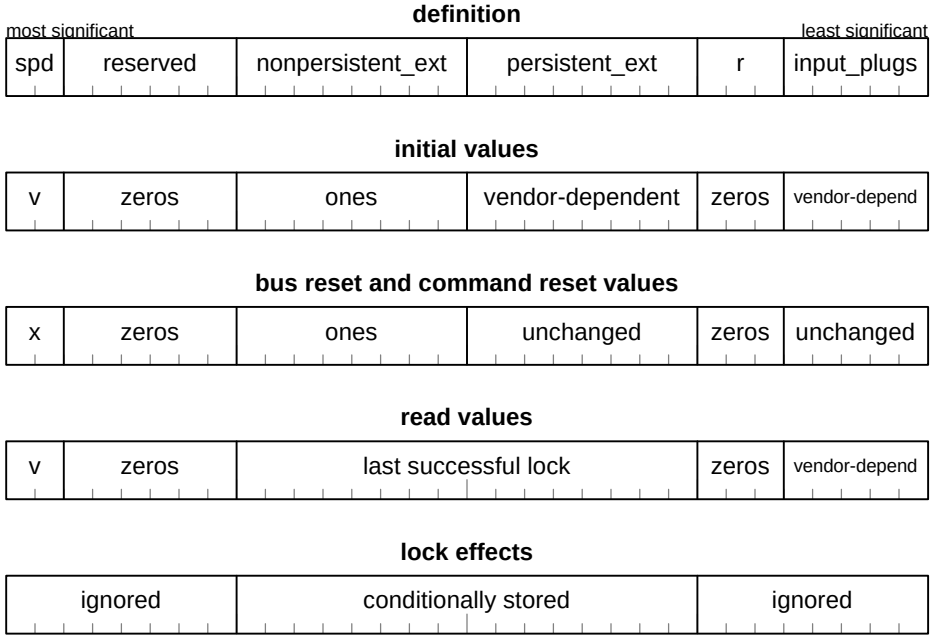

Figure 37 – INPUT_MASTER_PLUG format

The *spd* field shall specify the maximum speed at which any of the node's input plugs may receive isochronous data, as encoded by Table 1.

The *nonpersistent_ext* and *persistent_ext* fields are reserved for future standardization.

The *input_plugs* field shall specify the total count of INPUT_PLUG registers implemented by a node. Between zero and 31 INPUT_PLUG registers may be implemented. If one or more INPUT_PLUG registers are implemented, they shall lie within the contiguous address range FFFF F000 0984_{16} to FFFF F000 0980_{16} + 4 * *input_plugs*, inclusive.

6.5.4 INPUT_PLUG register

The INPUT_PLUG register permits the description and control of point-to-point connections that terminate at the associated plug. INPUT_PLUG registers are optional and are required only if the node can function as an isochronous listener. INPUT_PLUG registers shall be implemented within a contiguous address space; INPUT_PLUG[*n*] refers to the register addressable at FFFF F000 0984_{16} + 4 * *n*. The register definition is given by Figure 38 below.

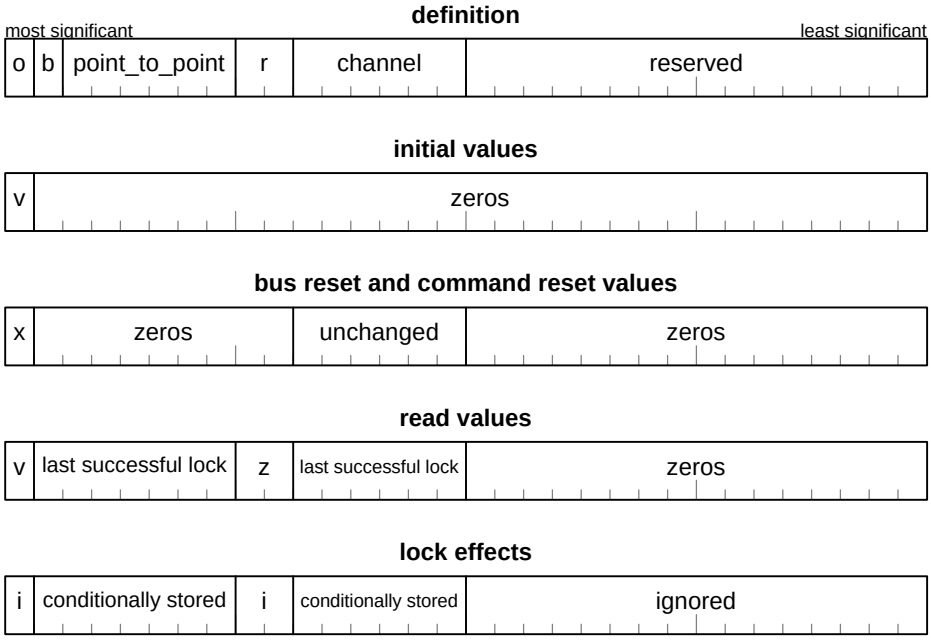

Figure 38 – INPUT_PLUG format

The *online* bit (abbreviated as *o* in the figure above) bit shall specify the on-line status of the plug resources controlled by the INPUT_PLUG register. An *online* bit value of zero shall indicate that the plug is off-line and not capable of receiving isochronous data. An *online* value of one shall indicate that the plug may be configured and used for isochronous data transmission.

NOTE – Plug status may change dynamically from off-line to on-line as device resources become unavailable or available. The causes of a change in plug status reported by the *online* bit are vendor-dependent.

The *broadcast* bit (abbreviated as *b* in the figure above) shall specify whether or not a broadcast connection exists for the input plug, where a value of zero indicates that no such connection exists.

The *point_to_point* field shall specify the number of point-to-point connections that exist for the input plug.

The *channel* field shall specify the isochronous channel number used in isochronous data reception for the plug.

7 Configuration ROM

All nodes that implement SBP-2 targets as a unit architecture shall implement general format configuration ROM in accordance with ISO/IEC 13213:1994, IEEE Std 1394-1995 and this standard. General format configuration ROM is a self-descriptive structure as illustrated below. The bus information block and root directory are at fixed locations; all other directories and leaves are addressed by entries in their parent directory.

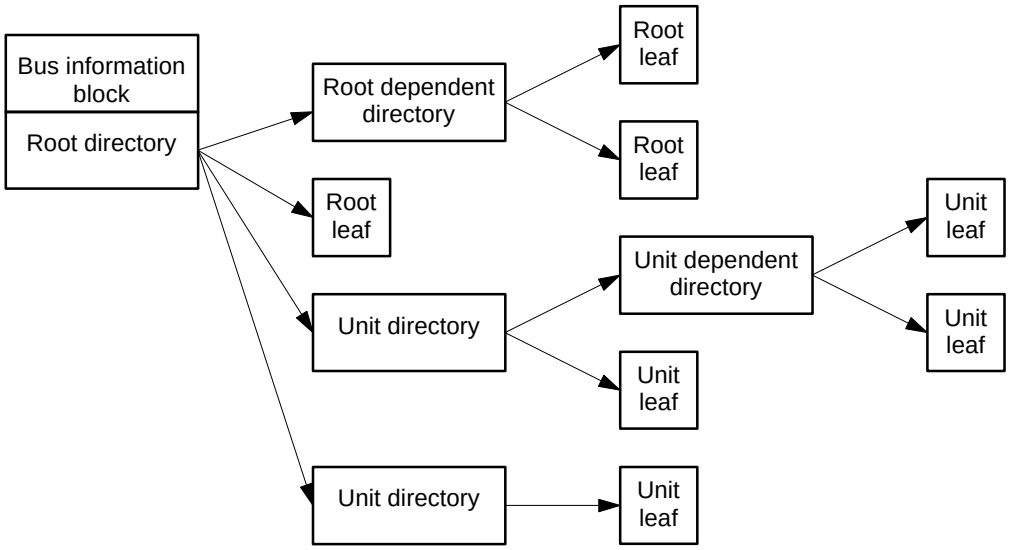

Figure 39 – Configuration ROM hierarchy

The figure above shows the potential of the general ROM format to accommodate a diversity of directory and leaf entries in a tree structure. In practice a target need implement only those configuration ROM entries described in the clauses that follow.

7.1 Bus information block

All targets shall implement a bus information block at a base address of FFFF F000 0404_{16}. For convenience of reference, the format of the bus information block defined by IEEE Std 1394-1995 is reproduced below. However, the current version of the referenced standard shall be consulted for the most recent information.

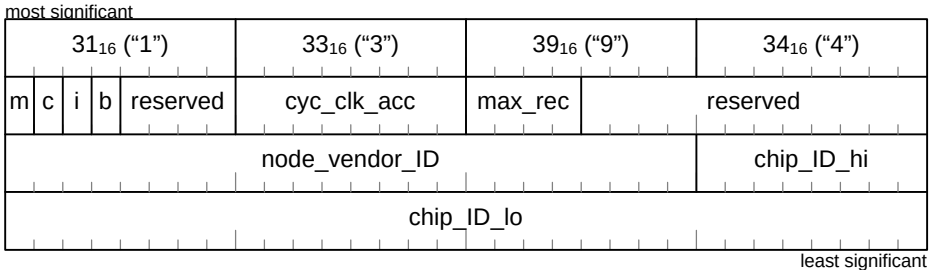

Figure 40 – Bus information block format

The first quadlet contains the string “1394” in ASCII characters.

The *irmc* bit (abbreviated as *m* in the figure above) shall be one if the node is isochronous resource manager capable; otherwise, the *irmc* value shall be zero.

The *cmc* bit (abbreviated as *c* in the figure above) shall be one if the node is cycle master capable; otherwise, this value shall be zero.

The *isc* bit (abbreviated as *i* in the figure above) shall be one if the node supports isochronous operations; otherwise, this value shall be zero.

The *bmc* bit (abbreviated as *b* in the figure above) shall be one if the node is bus manager capable; otherwise, this value shall be zero.

The *cyc_clk_acc* field specifies the node's cycle master clock accuracy in parts per million. If the *cmc* bit is one, the value in this field shall be between zero and 100. If the *cmc* bit is zero, this field shall be all ones.

The *max_rec* field defines the maximum payload size of a block write transaction addressed to the node. The range of the maximum payload size is from four bytes to 2048 bytes. A *max_rec* value of zero indicates that the maximum payload size is not specified. Otherwise, within the range of defined payload sizes, the maximum size is equal to $2^{max\_rec + 1}$. The *max_rec* field does not place any limits on the maximum payload size in asynchronous data packets, either requests or responses, that the node may transmit.

The *node_vendor_ID* field is a copy of the company ID present in the node unique ID leaf of configuration ROM.

The *chip_ID_hi* and *chip_ID_lo* fields are copies of the 40-bit chip ID present in the node unique ID leaf of configuration ROM.

Taken as a whole, the *node_vendor_ID*, *chip_ID_hi* and *chip_ID_lo* form a 64-bit node unique identifier. Because physical addresses on Serial Bus may change after a bus reset, this unique identifier is the only secure method of node identification.

7.2 Root directory

Configuration ROM for targets shall contain a root directory. The root directory immediately follows the bus information block and has a base address of FFFF F000 0414_{16}. The root directory shall contain Module_Vendor_ID, Node_Capabilities and Node_Unique_ID entries.

The root directory shall also contain a Unit_Directory entry that specifies the location of a unit directory whose format is specified by this standard.

7.2.1 Module_Vendor_ID entry

The Module_Vendor_ID entry is an immediate entry in the root directory that provides the company ID of the vendor that manufactured the module. Figure 41 shows the format of this entry.

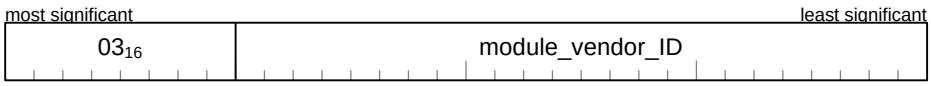

Figure 41 – Module_Vendor_ID entry format

03_{16} is the concatenation of *key_type* and *key_value* for the Module_Vendor_ID entry.

The IEEE/RAC uniquely assigns the *module_vendor_ID* to each module vendor, as specified by ISO/IEC 13213:1994. Unique identifiers for a company or organization may be obtained from:

Institute of Electrical and Electronic Engineers, Inc.
Registration Authority Committee
445 Hoes Lane
Piscataway, NJ 08855-1331

7.2.2 Node_Capabilities entry

The Node_Capabilities entry is an immediate entry in the root directory that describes node capabilities. Figure 42 shows the format of this entry.

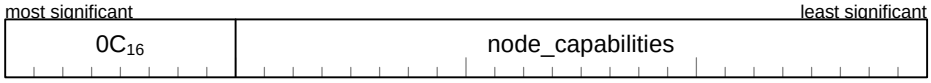

Figure 42 – Node_Capabilities entry format

$0C_{16}$ is the concatenation of *key_type* and *key_value* for the Node_Capabilities entry.

The *node_capabilities* field contains subfields specified by ISO/IEC 13213:1994. Targets shall implement the *spt*, *64*, *fix*, *lst* and *drq* bits.

Targets shall set the *spt*, *64*, *fix*, *lst* and *drq* bits to one. These indicate, respectively, that the node implements the SPLIT_TIMEOUT register, the 64-bit fixed addressing scheme, the STATE_CLEAR.*lost* bit and the STATE_CLEAR.*dreq* bit.

7.2.3 Node_Unique_ID entry

The Node_Unique_ID entry is a leaf entry in the root directory that describes the location of the node unique ID leaf within configuration ROM. Figure 43 shows the format of this entry.

Figure 43 – Node_Unique_ID entry format

$8D_{16}$ is the concatenation of *key_type* and *key_value* for the Node_Unique_ID entry.

The *indirect_offset* field specifies the number of quadlets from the address of the Node_Unique_ID entry to the address of the node unique ID leaf within configuration ROM.

7.2.4 Unit_Directory entry

The Unit_Directory entry is a directory entry in the root directory that describes the location of a unit directory within configuration ROM. There may more than one unit directory; each unit directory shall be located by a separate Unit_Directory entry. Figure 44 shows the format of this entry.

Figure 44 – Unit_Directory entry format

$D1_{16}$ is the concatenation of *key_type* and *key_value* for the Unit_Directory entry.

The *indirect_offset* field specifies the number of quadlets from the address of the Unit_Directory entry to the address of the unit directory within configuration ROM.

7.3 Unit directory

Configuration ROM for targets shall contain a unit directory in the format specified by this standard. The unit directory shall contain Unit_Spec_ID and Unit_SW_Version entries, as specified by ISO/IEC 13213:1994, and a Management_Agent entry, as specified by this standard.

Targets shall implement at least one logical unit, logical unit zero. Additional logical units may be implemented. A logical unit is described by entries in the unit directory or by entries in a logical unit directory dependent upon the unit directory or by entries taken in combination from both places. The properties of logical units are established by Command_Set_Spec_ID, Command_Set_Version and Logical_Unit_Characteristics entries; an instance of a specific logical unit is established by a Logical_Unit_Number entry.

The unit directory may also contain a Unit_Unique_ID entry.

7.3.1 Unit_Spec_ID entry

The Unit_Spec_ID entry is an immediate entry in the unit directory that specifies the organization responsible for the architectural definition of the target. Figure 45 shows the format of this entry.

Figure 45 – Unit_Spec_ID entry format

12_{16} is the concatenation of *key_type* and *key_value* for the Unit_Spec_ID entry.

00 $609E_{16}$ is the *unit_spec_ID* obtained from the IEEE/RAC. The value indicates that the X3 Secretariat is responsible for the software interface definition.

7.3.2 Unit_SW_Version entry

The Unit_SW_Version entry is an immediate entry in the unit directory that, in combination with the *unit_sw_version*, specifies the software interface of the target. Figure 46 shows the format of this entry.

Figure 46 – Unit_SW_Version entry format

13_{16} is the concatenation of *key_type* and *key_value* for the Unit_SW_Version entry.

01 0483_{16} is the *unit_sw_version* value that indicates that the target conforms to this standard.

7.3.3 Command_Set_Spec_ID entry

The Command_Set_Spec_ID entry is an immediate entry that, when present in the unit directory, specifies the organization responsible for the command set definition for the target. Figure 47 shows the format of this entry.

Figure 47 – Command_Set_Spec_ID entry format

38_{16} is the concatenation of *key_type* and *key_value* for the Command_Set_Spec_ID entry.

The *command_set_spec_ID* is an organizationally unique identifier obtained from the IEEE/RAC. The organization to which this 24-bit identifier has been granted is responsible for the definition of the command set implemented by the target.

7.3.4 Command_Set_Version entry

The Command_Set_Version entry is an immediate entry in the unit directory that, when present in the unit directory, in combination with the *command_set_spec_ID* specifies the command set implemented by the target. Figure 48 shows the format of this entry.

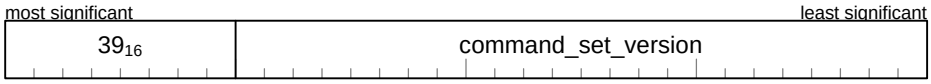

Figure 48 – Command_Set_Version entry format

39_{16} is the concatenation of *key_type* and *key_value* for the Command_Set_Version entry.

The meaning of *command_set_version* shall be specified by the owner of *command_set_spec_ID*.

7.3.5 Management_Agent entry

The Management_Agent entry is an immediate entry in the unit directory that specifies the base address of the target's management agent CSR. Figure 49 shows the format of this entry.

Figure 49 – Management_Agent entry format

54_{16} is the concatenation of *key_type* and *key_value* for the Management_Agent entry.

The *csr_offset* field shall contain the quadlet offset, from the base address of initial register space, FFFF F000 0000_{16}, to the base address of the MANAGEMENT_AGENT register for the target. All target CSR's shall be located at or above address FFFF F001 0000_{16}; therefore the value of *csr_offset* shall not be less than 4000_{16}.

NOTE – If a device implements additional control and status registers that are dependent upon the device class, it is recommended that these registers be placed at one of two locations within the device's address space. If the additional register(s) pertain to a logical unit, the recommended locations are at offset 20_{16} and above following the base address of the logical unit's command block agent registers. Additional register(s) that are associated with the device, and not a particular logical unit, may be located immediately after the MANAGEMENT_AGENT register. If this recommendation is used, there is no necessity for additional configuration ROM entries to describe the location of device-dependent registers.

7.3.6 Logical_Unit_Characteristics entry

The Logical_Unit_Characteristics entry is an immediate entry that, when present in the unit directory, specifies characteristics of the target implementation. Figure 50 shows the format of this entry.

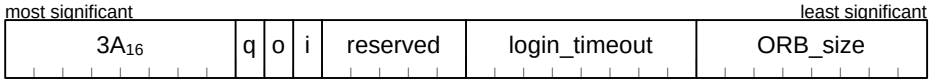

Figure 50 – Logical_Unit_Characteristics entry format

$3A_{16}$ 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_Version 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, isochronous login requests, stream command block requests and stream control requests shall all 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 7.1.

The *login_timeout* field shall specify, in units of 500 milliseconds, the maximum time an initiator allows for the completion, successful or in error, of a login request by a target.

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.

7.3.7 Logical_Unit_Directory entry

The Logical_Unit_Directory entry is an optional directory entry in the root directory that describes the location of the unit directory within configuration ROM. Figure 51 shows the format of this entry.

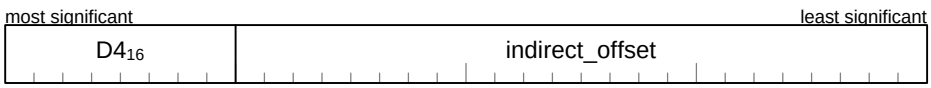

Figure 51 – Logical_Unit_Directory entry format

$D4_{16}$ is the concatenation of *key_type* and *key_value* for the Logical_Unit_Directory entry.

The *indirect_offset* field specifies the number of quadlets from the address of the Logical_Unit_Directory entry to the address of the unit directory within configuration ROM.

7.3.8 Logical_Unit_Number entry

The Logical_Unit_Number entry is an immediate entry that, when present in the unit directory, specifies the peripheral device type and logical unit number of a logical unit implemented by the target. Figure 52 shows the format of this entry.

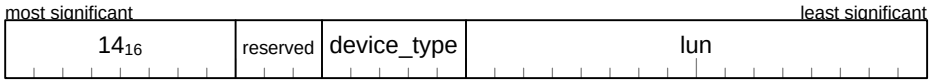

Figure 52 – Logical_Unit_Number entry format

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

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.

7.3.9 Unit_Unique_ID entry

The Unit_Unique_ID entry is an optional leaf entry in the unit directory that describes the location of the unit unique ID leaf within configuration ROM. If a vendor implements a device with multiple Serial Bus access paths, *i.e.*, multiple links to Serial Bus each of which receives a distinct *node_ID* as the result of Serial Bus initialization or bus enumeration, the Unit_Unique_ID entry shall be implemented. Figure 53 shows the format of this entry.

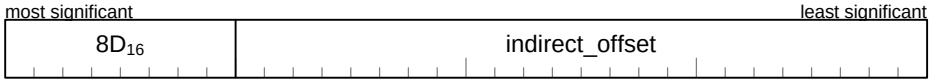

Figure 53 – Unit_Unique_ID entry format

$8D_{16}$ is the concatenation of *key_type* and *key_value* for the Unit_Unique_ID entry.

The *indirect_offset* field specifies the number of quadlets from the address of the Unit_Unique_ID entry to the address of the unit unique ID leaf within configuration ROM.

7.4 Logical unit directory

The logical unit directory provides one of two methods by which a logical unit implemented by the target may be described (the other is a Logical_Unit_Number entry in the unit directory, already described in 7.3.7).

The logical unit directory shall contain a Logical_Unit_Number entry.

The logical unit directory may additionally contain Command_Set_Spec_ID, Command_Set_Version or Logical_Unit_Characteristics entries.

7.4.1 Command_Set_Spec_ID entry

The Command_Set_Spec_ID entry is an immediate entry that, when present in a logical unit directory, specifies the organization responsible for the command set definition for the logical unit. If there is no Command_Set_Spec_ID entry in the logical unit directory, the Command_Set_Spec_ID entry in the unit directory shall apply; otherwise the entry in the logical unit directory shall take precedence. Figure 47 shows the format of this entry the fields are defined in 7.3.3.

7.4.2 Command_Set_Version entry

The Command_Set_Version entry is an immediate entry that, when present in a logical unit directory and in combination with the *command_set_spec_ID*, specifies the command set implemented by the logical unit. If there is no Command_Set_Version entry in the logical unit directory, the Command_Set_Version entry in the unit directory shall apply; otherwise the entry in the logical unit directory shall take precedence. Figure 48 shows the format of this entry; the fields are defined in 7.3.4.

7.4.3 Logical_Unit_Characteristics entry

The Logical_Unit_Characteristics entry is an immediate entry that, when present in a logical unit directory, specifies characteristics of the logical unit implementation. If there is no Logical_Unit_Characteristics entry in the logical unit directory, the Logical_Unit_Characteristics entry in the unit directory shall apply; otherwise the entry in the logical unit directory shall take precedence. Figure 50 shows the format of this entry; the fields are defined in 7.3.6.

7.4.4 Logical_Unit_Number entry

The Logical_Unit_Number entry is an immediate entry in a logical unit directory that specifies peripheral device type and logical unit number of the logical unit implementation. Figure 52 shows the format of this entry; the fields are defined in 7.3.8.

7.5 Node unique ID leaf

As specified by ISO/IEC 13213:1994, the node unique ID is a 64-bit number appended to a company ID value to create a globally unique 88-bit number. While conforming to this definition, Serial Bus additionally constrains the 64-bit node unique ID values so that they are unique within the global context of all Serial Bus nodes. Figure 54 shows the format of the node unique ID leaf.

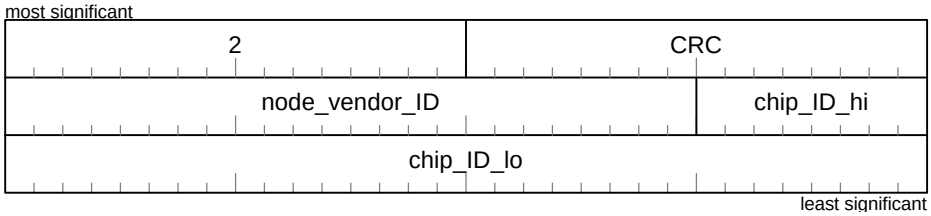

Figure 54 – Node unique ID leaf format

The first quadlet of the node unique leaf shall contain the number of following quadlets in the leaf and a CRC calculated for those quadlets, as specified by ISO/IEC 13213:1994.

The *node_vendor_ID* value shall be the same as the *module_vendor_ID* value from the root directory.

The *chip_ID_hi* field is concatenated with the *chip_ID_lo* field to create a 40-bit chip ID value. The vendor specified by the *node_vendor_ID* value shall administer the chip ID values. When appended to the *node_vendor_ID* value, these shall form a unique 64-bit value called EUI-64 (Extended Unique Identifier, 64 bits). These EUI-64 values are, by definition, unique from other EUI-64 identifiers derived from the IEEE/RAC-provided company ID value.

7.6 Unit unique ID leaf

Although the node unique ID described in the preceding section is sufficient to uniquely identify nodes attached to Serial Bus, it is insufficient to identify a target when a vendor implements a device with multiple Serial Bus node connections. In this case initiator software requires information by which a particular target may be uniquely identified, regardless of the Serial Bus access path used. The figure below shows the format of the unit unique ID leaf.

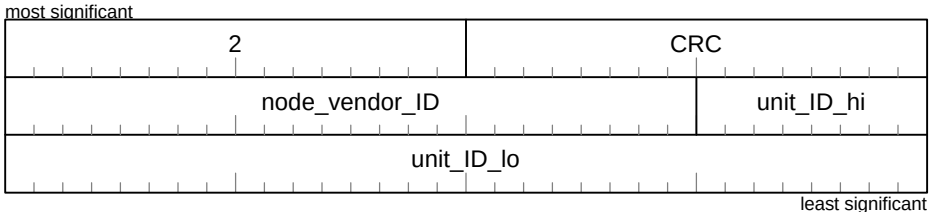

Figure 55 – Unit unique ID leaf format

The first quadlet of the unit unique leaf shall contain the number of following quadlets in the leaf and a CRC calculated for those quadlets, as specified by ISO/IEC 13213:1994.

The *node_vendor_ID* value shall be the same as the *module_vendor_ID* value from the root directory.

The *unit_ID_hi* field is concatenated with the *unit_ID_lo* field to create a 40-bit unit ID value. The vendor specified by the *node_vendor_ID* value shall administer the unit ID values. When appended to the *node_vendor_ID* value, these shall form a unique 64-bit value called EUI-64 (Extended Unique Identifier, 64 bits). These EUI-64 values are, by definition, unique from other EUI-64 identifiers derived from the IEEE/RAC-provided company ID value.

As a consequence of the implementation of multiple Serial Bus nodes, there is configuration ROM accessible for each node. Parts of these configuration ROM's shall differ from each other, *e.g.*, the node unique ID leaf, but the value in the unit unique ID leaf shall be the same regardless of which node is used to access the information.

8 Access

Before an initiator may signal commands or other requests to a target, access privileges shall first be granted by the target. The criteria for the grant of access may include resource availability or other target requirements. This section specifies the target facilities that support access control and the methods by which an initiator requests access to a target and eventually relinquishes access when it is no longer required.

8.1 Access protocols

Targets shall implement a logical unit reservation protocol that supports neither persistent reservations nor passwords; it is a simple mechanism that can be used to guarantee single initiator access to the logical unit and to preserve that initiator's access rights across a Serial Bus reset.

In order to support the logical unit reservation protocol, a target shall implement resources to manage one or more logins from initiators. These resources are described below and are used in the specification of target actions in response to login requests signaled by an initiator to the target's management agent:

- The target implements a set of one or more *login_descriptors* that are used to hold context for logins. The context of a login stored in a *login_descriptor* consists of the *lun*, the *login_owner_ID*, the *login_owner_EUI_64*, an *exclusive* variable, the base addresses of the fetch agent CSR's returned to the initiator in the *login_response* data and the *login_ID* used by the initiator to identify the login.
- The *login_owner_ID* is the 16-bit node ID of the current owner of a login. Upon either a Serial Bus reset or a power reset, the *login_owner_ID* for all *login_descriptors* is reset to all ones. The target shall use the *login_owner_ID* to qualify all write requests addressed to the *login_descriptor* fetch agent CSR's.
- The *login_owner_EUI_64* is the unique 64-bit identifier of the current owner of a login. Upon a power reset, the *login_owner_EUI_64* for all *login_descriptors* is reset to all ones. Upon a Serial Bus reset, the *login_owner_EUI_64* persists for two seconds and is then reset to all ones unless it has been reestablished.

A *login_descriptor* is considered free if both its *login_owner_ID* and *login_owner_EUI_64* are all ones. The resources of this *login_descriptor* may be allocated to any initiator that successfully completes a login or an isochronous login. If a *login_descriptor*'s *login_owner_ID* is all ones but its *login_owner_EUI_64* holds a valid EUI-64, the *login_descriptor* is reserved—the initiator identified by *login_owner_EUI_64* may reestablish the login. Active *login_descriptors* are those whose *login_owner_ID* and *login_owner_EUI_64* are both valid; the initiator that owns the login may signal requests to the fetch agent(s) associated with the *login_descriptor*.

8.2 Login requests

The clauses that follow describe the use of the login ORB's defined in 5.1.4.

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. The address of the management agent shall be obtained from configuration ROM.

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

The target shall perform the following to validate a login request:

a) 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;

b) 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.

c) If the *exclusive* bit is set in the login ORB, the target shall reject the login request if there are any active *login_descriptors* for the logical unit;

d) 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; and

e) The target shall determine if a free *login_descriptor* is available. If a *login_descriptor* is free, 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* from the login ORB is 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 also 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 Isochronous login

Isochronous login for an initiator may be granted 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 isochronous login request that characterizes the isochronous operations to be performed.

The isochronous information consists of three items:

- whether the target is to function as a talker or a listener;
- 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 medium in a single isochronous cycle.

The maximum number of channels are required in order for the target to allocate sufficient plug control registers. The initiator must also specify which role, talker or listener, the target shall assume in order that the target allocate OUTPUT_PLUG or INPUT_PLUG register(s), respectively.

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 cycle. Implementation-dependent constraints may limit the performance of the target, which requires this information in order to determine if the login may be accepted.

The target shall perform the following to validate an isochronous login request:

a) 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 isochronous login ORB to the target's MANAGEMENT_AGENT register shall be used as the *destination_ID* in the quadlet read transactions;

b) The target shall determine whether or not the *login_ID* is valid by comparing the just obtained EUI-64 against the *login_owner_EUI_64* for the *login_descriptor* identified by *login_ID*;

c) If the *login_ID* is valid, the target shall determine if a free *login_descriptor* is available. If a *login_descriptor* is free, 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* from the *login_descriptor* is copied to the *login_descriptor* for the isochronous login and the addresses of the fetch agent(s) are also stored in the *login_descriptor*. Lastly the target assigns a unique *login_ID* to this login and stores it in the *login_descriptor*,

In addition to the addresses of the stream command block and stream control fetch agents, the target shall also specify in the *login_response* data which plug control registers the initiator shall use for the isochronous stream as well as the minimum transfer length that the initiator should specify in the *stream_length* field of any stream command block request signaled to the target.

8.3 Reconnection

Upon a Serial Bus reset, the target shall abort all task sets for all command block agents. Requests signaled to stream control agents may continue for one second subsequent to a Serial Bus reset, the time permitted for the reallocation of isochronous bandwidth and channels and for the reestablishment of isochronous connections. If an initiator does not reestablish its login within this time period, then an implicit logout is performed on behalf of the initiator.

The target shall perform the following to validate a reconnect request:

a) 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 reconnect ORB to the target's MANAGEMENT_AGENT register shall be used as the *destination_ID* in the quadlet read transactions;

b) The target shall determine whether or not the *login_ID* is valid by comparing the just obtained EUI-64 against the *login_owner_EUI_64* for the *login_descriptor* identified by *login_ID*;

c) If the *login_ID* is valid, the target shall store the initiator's *source_ID* in *login_owner_ID*,

No *login_response* data is stored for a reconnect request; the completion status is indicated by the status block stored at the *status_FIFO* address.

8.4 Logout

When an initiator no longer requires access to a target's resources, it shall signal a logout request to the management agent. The login to be released shall be identified by *login_ID* in the logout ORB. A target shall reject a logout request if *login_ID* does not match that of any active *login_descriptor* or if the *source_ID* of the write request used to signal the logout ORB to the MANAGEMENT_AGENT register is not equal to the *source_ID* of the matching *login_descriptor*. If any tasks or stream control ORB's are active at the time of the logout request, they shall be aborted in the same fashion as if the task set had been aborted. Upon successful completion of a logout request, all resources allocated to the initiator are free once again and may be used by the target to satisfy subsequent login requests.

9 Command execution

This section describes the procedures used by an initiator to request command execution by a target. As described in the model, requests are specified by data structures in system memory that are subsequently fetched by the target. While a target executes a request, it is responsible for any data transfer associated with the request. Once a request completes, successfully or in error, a status block is stored in system memory by the target. The data structures are defined in section 5; the initiator procedures for the use of these request and status blocks are described in the clauses that follow

9.1 Requests and request lists

Management requests (which include login and logout requests) are signaled to the target agent by means of a Serial Bus block write transaction that specifies the address of the management ORB. The management agent becomes busy while executing a request and refuses subsequent Serial Bus transactions until the current request is completed. The management agent does not require any initialization procedures.

The other target agents, command block and stream control, are characterized as fetch agents since they manage linked lists of requests in system memory and are responsible to fetch the ORB's. For normal command block, stream command block and stream control ORB's, the initiator produces requests and the target consumes them. These processes are asynchronous and independent of each other. Target efficiency is improved if the target can be kept occupied with an ample working set of requests. To this end, the initiator is permitted to arrange ORB's in linked lists and to dynamically append new requests to the lists while the target remains active.

Each normal command block, stream command block or stream control ORB contains an address pointer, *next_ORB*, which shall either be null or point to another ORB. A linked list of ORB's, previously illustrated by Figure 6 , implicitly orders the ORB's—the fact that the ORB's are in order permits the target to execute them in order (or not) according to its device-dependent characteristics.

The target is responsible to fetch ORB's from system memory, as described in more detail in 9.1.3. This remainder of this clause describes what the initiator shall do to:

- initialize a target fetch agent; and
- dynamically append new requests to an active list and notify a target fetch agent of the new requests; or
- notify a target fetch agent of a single new request.

9.1.1 Fetch agent initialization (informative)

After successful completion of a login procedure and the return of the base address of the fetch agent CSR's, the initiator may initialize the fetch agent as follows:

a) The initiator allocates space for a dummy ORB and initializes it *per* the format described in 5.1.1. Although only the *next_ORB* field, *notify* bit and the *rq_fmt* field are significant within a dummy ORB, the initiator allocates at least the minimum ORB size specified by the target's configuration ROM. The initiator sets the *next_ORB* field to the null pointer value;

b) The initiator resets the target fetch agent by a quadlet write to the fetch agent's AGENT_RESET register;

c) The initiator writes the address of the dummy ORB to the fetch agent's ORB_POINTER register by means of an 8-byte block write request. In the example in Figure 56, this is the value 0000 0000 8004 $00C0_{16}$. This causes the fetch agent to transition to the ACTIVE state.

The figure below illustrates the result of these actions:

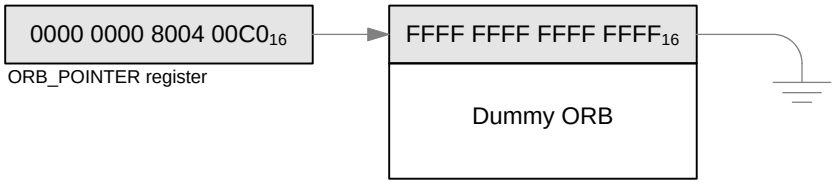

Figure 56 – Fetch agent initialization with a dummy ORB

When the fetch agent transitions to the active state as a result of the write to the ORB_POINTER register, it uses the value to fetch the dummy ORB (as target resources permit). The dummy ORB, by definition, completes immediately and the target fetch agent stores a status block for the request. However, the null pointer in the *next_ORB* field of the dummy ORB causes the fetch agent to transition to the suspended state. The ORB_POINTER register still points to the dummy ORB and the initiator may subsequently append additional requests, as described in 9.1.2.

9.1.2 Dynamic appends to request lists (informative)

Once a target fetch agent has been initialized and made active as described above, it is possible for the initiator to append new requests to the linked list while the fetch agent remains active. Assume that the initiator intends to add three new requests previously illustrated by Figure 6.

An initiator may append new requests to an active request list as follows:

a) The initiator constructs a linked list of ORB's in system memory, as illustrated in the example. The *next_ORB* field of the last ORB contains a null pointer. The *next_ORB* fields of all other ORB's contain a valid pointer to a subsequent ORB;

b) The initiator updates the *next_ORB* field of what had been the last ORB, in this example the dummy ORB in Figure 56, with the address of the first request in the new request list, in this example 0000 0000 8000 0000_{16}; and

c) Lastly, the initiator transmits a quadlet write request, with any data value, to the fetch agent's DOORBELL register.

The final step informs the target that address pointers in the request list have been updated by the initiator. If the target fetch agent had not encountered a null pointer, the activation of the doorbell is redundant. However, if the target fetch agent is already suspended at the time *next_ORB* is updated, the activation of the doorbell is essential to reactivate the fetch agent. In this latter case, it is necessary for the target fetch agent to refetch all or part of an ORB from system memory in order to ascertain if a previously null pointer contains a valid address of an ORB.

NOTE – If the initiator has knowledge that the fetch agent is in the suspended state, the algorithm described above may be modified to write the address of the new ORB to the ORB_POINTER register in place of the write to the DOORBELL register. This has the virtue of avoiding a refetch of the *next_ORB* field from the ORB at which the fetch agent is suspended, but would produce unpredictable results if the fetch agent were not in the suspended state.

9.1.3 Fetch agent use by the BIOS (informative)

The BIOS, or any similar initiator application that executes in a single-threaded environment, has little need of the target fetch agent's capabilities to manage multiple outstanding requests. The BIOS may take advantage of this and use a simpler procedure than that described in 9.1.2 to signal requests to the target. Subsequent to initialization of the target fetch agent by means of a write to the AGENT_RESET register, the BIOS may signal one request at a time to the target as follows:

a) The BIOS allocates space for the request in an ORB and initializes it according to the ORB format. The *next_ORB* field contains a null pointer;

b) The BIOS signals the request to the target agent by writing the address of the ORB to the ORB_POINTER register in an 8-byte block write transaction. This causes the target agent to transition to the ACTIVE state and to execute the request; and

c) Subsequent to the return of a status block to the *status_FIFO* address specified when the login was performed, the BIOS may signal additional requests by repeating this procedure.

The performance improvements yielded by the above procedure (which are accomplished by the elimination of a read transaction to fetch an ORB) are minor; the principal benefit to the BIOS is code simplification.

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*. If 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*.

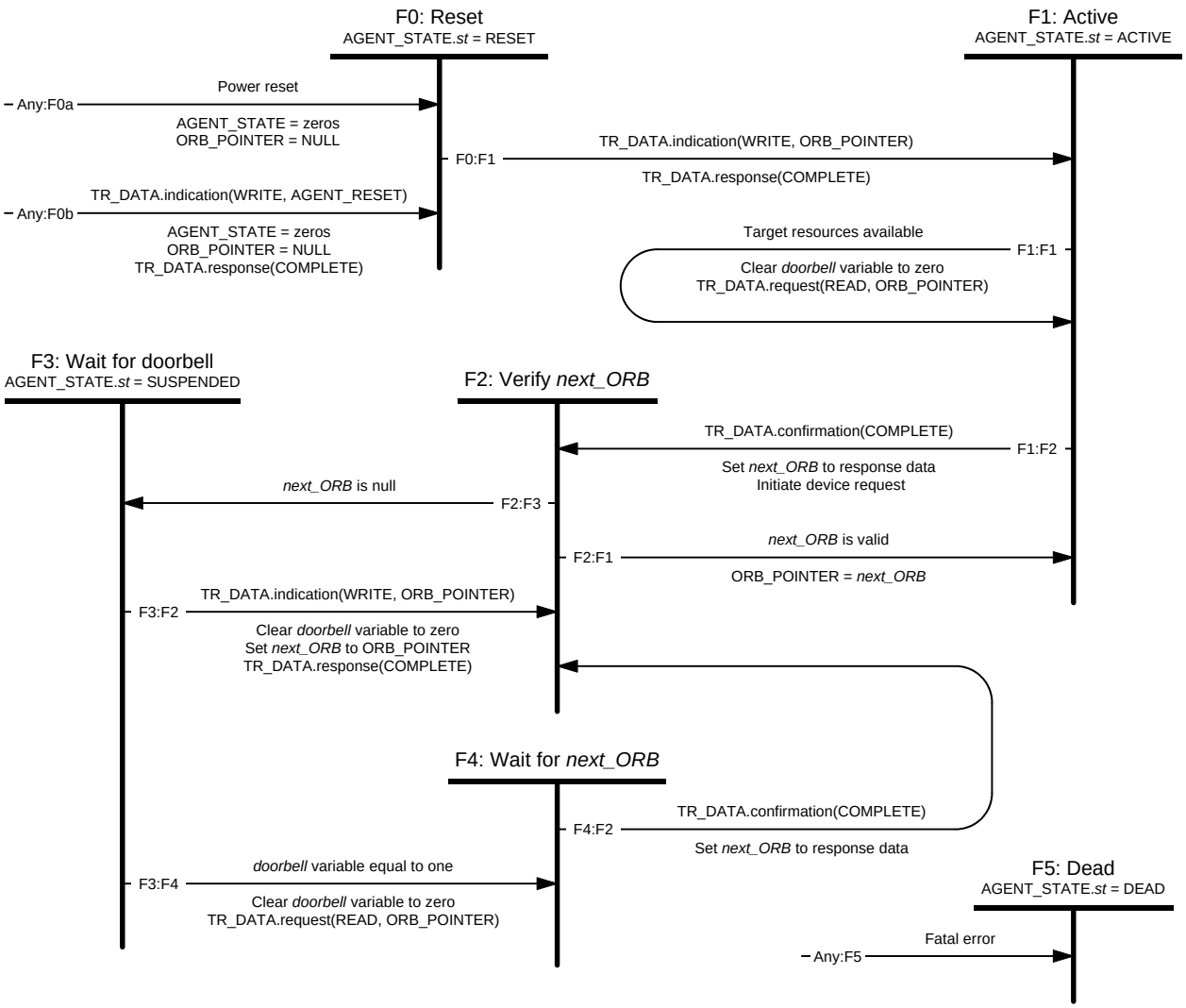

Figure 57 – Fetch agent state machine

Transition Any:F0a. A power reset shall cause the fetch agent to transition to the RESET state from any other state. The registers that control and make visible the operations of the fetch agent shall be reset to known values, zeros in the case of the AGENT_STATE register and a null pointer in the case of the ORB_POINTER register.

TransitionAny: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 register and set the ORB_POINTER register to a null pointer value before the transition to state F0.

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 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 is assumed to have valid address information in the ORB_POINTER register and may fetch ORB's from the initiator as resources permit.

Transition F1:F1. 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.

Transition F1:F2. Subsequent to a block read request, issued as described above, the fetch agent may accept a block read response that contains 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 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 F2: 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 F2:F3. The fetch agent shall transition to a suspended state, F3, 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.

Transition F2:F1. If the *next_ORB* variable does not indicate a null pointer, presumably it is a valid pointer. In this case, the fetch agent shall update the ORB_POINTER register with the value of *next_ORB*.

State F3: Wait for doorbell. The fetch agent is suspended, the ORB_POINTER register contains valid address information that should not be updated by the initiator and a null pointer has signaled the end of a linked list of ORB's in system memory.

Transition F3:F2. 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 of COMPLETE. After the confirmation, the fetch agent shall transition to state F2 in order to verify the *next_ORB* variable. If, as expected, *next_ORB* is not null an immediate F2:F1 transition follows.

Transition F3:F4. 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 shall transition to state F4. 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.

NOTE – 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.

State F4: Wait for *next_ORB*. 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 F4:F2. Subsequent to a block read request, issued as described above, the fetch agent may accept a block read response that contains the *next_ORB* data. 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 F2 in order to verify the *next_ORB* variable. If, as expected, *next_ORB* is not null an immediate F2:F1 transition follows.

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.

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

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 these registers shall have no effect while in state F5.

9.2 Data transfer

The transfer of data associated with a command is entirely the responsibility of the target. The target shall use Serial Bus read transactions to fetch data from system memory and Serial Bus write transactions to store data in system memory.

The total transfer length may be larger than the maximum data payload that can be accommodated in a single transaction. The target is responsible to manage the size and number of read or write transactions to transfer all the requested data. The target may choose any appropriate size for these data transfer transactions, subject to constraints specified by the ORB.

The target shall observe alignment requirements specified by the *page_table_present* bit and the *page_size* field. If *page_table_present* is one, the target shall observe alignment boundaries that occur every 2^{page_size+8} bytes; no single Serial Bus block read or block write transaction shall cross such a boundary. When *page_table_present* is zero, a *page_size* value of zero indicates that there are no alignment requirements. Nonzero *page_size* values specify alignment boundaries in the same fashion as when a page table is present.

The target shall issue data transfer requests with a speed equal to that specified by the *spd* field in the ORB. The target shall not issue block read or write requests with a data payload length greater than that specified by the *max_payload* field in the ORB.

Within the above speed and size constraints, the target is free to issue the data transfer requests in any order and to retry failed data transfer requests according to vendor-dependent algorithms.

9.3 Completion status

Upon completion of an ORB, the target shall examine the *notify* bit in the ORB to determine whether or not to store a status block. If *notify* is zero, the target may store a status block. Otherwise, if *notify* is one or if the ORB completed with an error condition, the target shall store a status block. The address for the status block is specified by *status_FIFO*, supplied by the initiator as part of the login parameters. The status block, previously described in 5.3, contains sufficient information to indicate successful command completion or, in the case of a faulted command, to permit the initiator to select the appropriate error handling strategy.

In all cases, the status FIFO allocated by the initiator shall be accessible to a single Serial Bus block write transaction with any *data_length* that is a multiple of four and less than or equal to 32 bytes. The target shall store the status block by means of a single block write and shall not attempt any retries if either:

a) no acknowledge packet is received immediately after the write request; or

b) subsequent to the receipt of an *ack_pending* immediately after the write request, no corresponding response packet is received within the split time-out limit.

Other errors, including the link layer busy conditions, *ack_data_error*, *resp_conflict_error* and *resp_data_error*, may be retried up to a vendor-dependent limit. If no retry is attempted or if the retry limit is exhausted without success, the target fetch agent shall transition to the DEAD state.

The return of completion status to the initiator may also signal that the system memory allocated to the ORB may be reused. If the *end_of_list* bit is clear the initiator may reuse or deallocate the system memory occupied by an ORB.

9.4 Unsolicited status

In addition to status associated with a particular ORB, described in the preceding section, a fetch agent may store unsolicited status at the address specified by *status_FIFO*. A status block that contains unsolicited status shall be identified by setting the *unsolicited* bit to one.

A fetch agent may store unsolicited status at any time that its *status acknowledgment* variable is one. Upon completion of the Serial Bus block write transaction used to store the status block, the fetch agent shall zero its *status acknowledgment* variable. The initiator may set the fetch agent's *status acknowledgment* variable to one by writing any data value to the STATUS_ACKNOWLEDGE register.

10 Task management

The preceding section describes the procedures used by the initiator to signal the target that tasks are to be executed and the procedures by which the target performs data transfer or device control for the tasks and ultimately signals their completion back to the initiator. Section 9 gives no consideration to the larger perspective of how these tasks interact with each other and how the initiator may manage the tasks.

This section defines how individual tasks are collected together as task sets and how both tasks and task sets may be managed by the initiator.

10.1 Task sets

A task set is a collection of tasks, each of which has an associated command in an ORB, that is available to the target for execution. The interactions among these tasks and the ordering relationships, if any, are governed by the task management model implemented by the target.

A task enters the task set when it is linked into an active request list. The extent of a task set includes all the uncompleted ORB's linked into a request list in system memory, not solely the ORB's already fetched by the target.

Historically, there has been one task set associated with each logical unit of a device. The concept of a task set is extended by SBP-2 to permit multiple stream task sets per logical unit. Each time target resources are allocated for isochronous operations (by means of an isochronous login), a task set is created that is associated both with a logical unit and a stream identifier. There is a one-to-one relationship between a stream identifier and a stream task set, but there may be 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.

10.2 Basic task management model

Targets shall support, at a minimum, a basic task management model. Under this model, the following rules apply:

- All tasks within a task set share the same execution characteristics: either they are all reorderable or else they are all ordered;
- The reorderable or ordered execution characteristics of a task set are implicit in the target implementation and are not subject to control by the initiator;
- For stream task sets, the target shall execute all tasks in order and report their completion status in the same order. For normal task sets, configuration ROM shall specify whether the target may reorder task execution or not;
- All tasks within a task set are uniquely identified by the Serial Bus address of the ORB that initiated the task. This address shall be unique for the life of the task;
- The abort task, abort task set and target reset task management functions, described later in this section, shall be implemented;

The only element of choice in the implementation of a task set is under this model is whether or not the target may reorder task execution.. An unordered model is usually appropriate for devices, such as mass storage, where no positional or other context information is inherited from one command to the next. An ordered model may be more appropriate for devices, such as sequential storage, where the outcome of one command affects the next. The same ordering considerations apply to stream task sets, within which the data is time-ordered by its very nature.

The unordered model is characterized by unrestricted reordering of the active tasks. The target may reorder the actual execution sequence of any tasks in a task set in any manner. Unrestricted reordering places the responsibility for the assurance of data integrity on the initiator. If the integrity of data on the device medium could be compromised by unrestricted reordering involving a set of active tasks, {T_0, T_1, T_2, … T_N} and a new task T´, the initiator shall wait until {T_0, T_1, T_2, … T_N} have completed before appending T´ to an active request list.

NOTE – In multitasking operating system environments, independent execution threads may generate tasks that have ordering constraints within each thread but not with respect to other threads. If this is the case, an initiator may manage the constraints of each thread yet still keep the target substantially busy. This avoids the undesirable latencies that occur if the target is allowed to become idle before new ORB's are signaled.

The ordered model requires both that tasks be executed in order and that completion status be returned in order. However, the split-transaction nature of Serial Bus inherently makes it possible for Serial Bus transactions to be reordered. Because of this, the target shall insure that completion status is reported in order. When a task in an ordered task set completes, the target shall successfully store the completion status in system memory before initiating a Serial Bus write transaction to store completion status for any other task in the task set.

10.3 Error conditions

Upon an error condition or fault detected during the execution of any task within a task set, the entire task set shall be cleared as follows:

a) The target shall halt the operation of the fetch agent associated with the task set by making a transition to the DEAD state;

b) For all recently completed tasks, the target shall wait until the completion status of each command has been successfully stored in system memory or until the implementation-dependent retry algorithms have been exhausted in the attempt to store completion status; and

c) Finally, the target shall return error completion status for the faulted task.

The return of error status for a faulted task is an indication to the initiator that the task set has been cleared and that any remaining active tasks in the request list have been aborted.

10.4 Task management requests

The clauses that follow describe the use of the task management ORB's defined in 5.1.4.6.

10.4.1 Abort task

Abort task is a task management function that permits an initiator to abort a specified task. 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. Note that 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 is dependent upon the target's ability 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 ultimately 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 ABORTED.

Targets may optionally support task management ORB's with a *function* of ABORT TASK. Targets that support abort task in this manner shall store a completion status of FUNCTION COMPLETE for the abort task request in the status buffer provided.

If the task to be aborted, identified by *ORB_offset*, is not recognized by the target as part of its local 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 ABORTED. 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 local 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, the target 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 responses to pending data transfer requests and, once all such responses are received, shall not issue additional data transfer requests for the task;

c) The target shall store completion status for the task to be aborted and shall wait for the transaction complete acknowledgment or response. If the target successfully aborted the task, the request status stored shall be 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. consensus

10.4.2 Abort task set

Abort task set is a task management function that permits an initiator to abort all of its tasks within a task set. All targets shall support abort task set.

To abort a task set, the initiator shall construct a management ORB in system memory for the abort task set function. The initiator shall set the appropriate values in the *rq_fmt* and *login_ID* fields of the ORB, as described in 5.1.4.6. The *function* field shall be set to ABORT TASK SET.

The initiator shall signal the abort task set ORB to the management agent.

Upon receipt of an abort task set request, the target shall perform the following actions:

a) The target shall halt the operation of the fetch agent associated with the task set by making a transition to the DEAD state;

b) The target shall not issue data transfer requests for any task in the task set whose *login_ID* is equal to that specified in the abort task set request;

c) The target shall wait for responses to pending data transfer requests for any task in the task set whose *login_ID* is equal to that specified in the abort task set request;

d) For all recently completed tasks whose *login_ID* is equal to that specified in the abort task set request, the target shall wait until the completion status of each command has been successfully stored in system memory or until the implementation-dependent retry algorithms have been exhausted in the attempt to store completion status; and

e) When all of the above events have completed, the target shall store completion status for the abort task set request in the status buffer provided. The completion status shall indicate FUNCTION COMPLETE.

The initiator shall not reuse the system memory occupied by any of the ORB's, data buffers or page tables of the tasks to be aborted until completion status is returned for the abort task set request.

10.4.3 Clear task set

Clear task set is a task management function that permits an initiator to abort all tasks within a task set. Targets may support clear task set.

To clear a task set, the initiator shall construct a management ORB in system memory for the clear task set function. The initiator shall set the appropriate values in the *rq_fmt* and *login_ID* fields of the ORB, as described in 5.1.4.6. The *function* field shall be set to CLEAR TASK SET.

The initiator shall signal the clear task set ORB to the management agent.

Upon receipt of a clear task set request, the target shall perform the following actions:

a) The target shall halt the operation of all fetch agents associated with the task set by making transitions to the DEAD state;

b) The target shall not issue data transfer requests for any task in the task set;

c) The target shall wait for responses to pending data transfer requests for any task in the task set;

d) For all recently completed tasks, the target shall wait until the completion status of each command has been successfully stored in system memory or until the implementation-dependent retry algorithms have been exhausted in the attempt to store completion status;

e) The target shall create a unit attention condition for all logged-in initiators other than the initiator, identified by *login_ID*, that signaled the clear task set request; and

f) When all of the above events have completed, the target shall store completion status for the clear task set request in the status buffer provided. The completion status shall indicate FUNCTION COMPLETE.

The initiator shall not reuse the system memory occupied by any of the ORB's, data buffers or page tables of the tasks to be aborted until completion status is returned for the clear task set request.

10.4.4 Target reset

Target reset is a task management function that causes a target to perform the actions described below and to create unit attention conditions for all initiators. All targets shall support target reset.

To reset a target, the initiator shall construct a management ORB in system memory for the target reset function. The initiator shall set the appropriate values in the *rq_fmt* and *login_ID* fields of the ORB, as described in 5.1.4.6. The *function* field shall be set to TARGET RESET.

The initiator shall signal the target reset ORB to the management agent.

Upon receipt of a target reset request, the target shall perform the following actions:

a) The target shall halt the operation of all fetch agents by making transitions to the DEAD state;

b) The target shall not issue data transfer requests for any task in any task set;

c) The target shall create a unit attention condition for all logged-in initiators other than the initiator, identified by *login_ID*, that signaled the target reset request; and

d) When all of the above events have completed, the target shall store completion status for the target reset request in the status buffer provided. The completion status shall indicate FUNCTION COMPLETE.

The initiator shall not reuse the system memory occupied by any of the affected ORB's, data buffers or page tables of the tasks until completion status is returned for the target reset request.

10.4.5 Terminate task

Terminate task is a task management function that permits an initiator to request early completion of a specified task. Targets that implement the basic task management model shall not support terminate task and shall reject all terminate task requests with a completion status of FUNCTION REJECTED.

To request task termination, the initiator shall construct a management ORB in system memory for the terminate 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 TERMINATE TASK; *ORB_offset* shall contain the Serial Bus address of the ORB for the task to be terminated. Once the terminate task ORB has been initialized, the initiator shall signal the ORB to the management agent.

Upon receipt of a terminate task request, the target shall store a completion status of FUNCTION COMPLETE or FUNCTION REJECTED for the terminate task request in the status buffer provided.

If the terminate task function is accepted by the target, the completion status of FUNCTION COMPLETE does not necessarily indicate that the specified task has completed. The ultimate completion of the specified task shall be signaled when the target stores completion status for the task.

If an error condition is detected for the specified task, the terminate task request shall be ignored and the target shall perform the actions previously described in 10.3.

If the specified task completes prior to the receipt of the terminate task request, the target shall wait until completion status is successfully stored for the specified task before completion status shall be stored for the terminate task request.

Otherwise, the target shall complete the specified task as follows:

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

b) The target shall wait for responses to pending data transfer requests;

c) The target shall store completion status of REQUEST COMPLETE and appropriate command set-dependent status that indicates command termination.

When a terminated task creates an error condition, the target shall clear the task set and take the actions described in 10.3.

The initiator shall not reuse the system memory occupied by the ORB, data buffer or page table of the task to be terminated until completion status is returned for that ORB.

11 Isochronous data formats

Isochronous data stored on the medium is kept in a form essentially similar to the format of isochronous packets on Serial Bus, but the *tcode* field present in Serial Bus packets is reused as the *type* field in recorded isochronous data. Three different packet formats may be present in recorded isochronous data, encoded by *type* as shown below.

Value	Name	Description
0	NULL	Null (or filler) packet
1	CYCLE MARK	Marks the time of a cycle start event
2	DATA	Isochronous data packet
3 – F_{16}	—	Reserved for future standardization

In addition to the reuse of the *tcode* field as *type*, the header and data CRC fields observed as part of Serial Bus isochronous packets are not recorded on the medium. Recorded isochronous data packets shall be stored on quadlet boundaries on the medium and shall contain an integral number of quadlets.

11.1 Null packets

When the *type* field has a value of NULL, the data that is stored on the medium shall be ignored during playback. The format of a NULL packet is shown below.

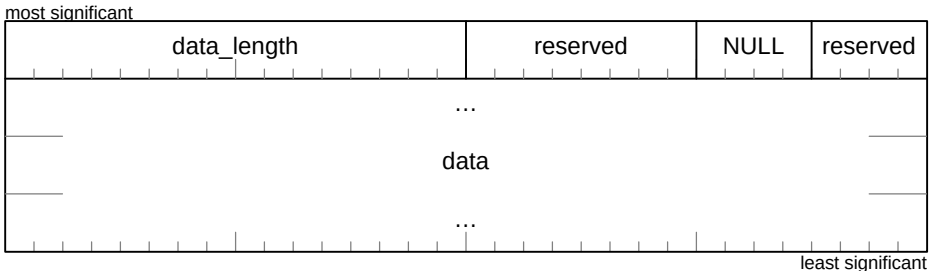

Figure 58 – NULL packet format

The *data_length* field shall be a multiple of four. Zero is a permissible value for *data_length*; in this case, the null packet shall consist of only the header and shall be a single quadlet in length.

When *data_length* is greater than or equal to four, the *data* field shall consist of *data_length* / 4 quadlets. The values of quadlets within the *data* field are unspecified for NULL packets.

NOTE – NULL packets serve no particular purpose for targets, but they may be useful to some applications, such as nonlinear editing. Excessive quantities or sizes of NULL packets may cause some target implementations to experience underflow or other errors in the playback of isochronous data.

11.2 Cycle marks

Whenever a cycle start packet is observed on Serial Bus for an enabled isochronous stream, a CYCLE MARK packet shall be recorded on the medium. The CYCLE MARK packet is a single quadlet that stores the time transported by the cycle start packet, as shown by the figure below.

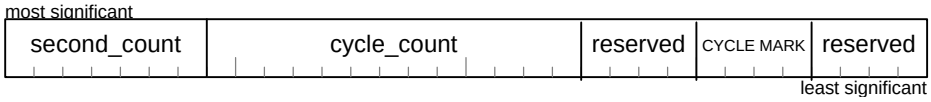

Figure 59 – CYCLE MARK format

The *second_count* and the *cycle_count* fields shall contain the values of the corresponding fields from the most recently observed cycle start packet. No more than one CYCLE MARK packet shall be recorded for a single cycle start packet.

NOTE – The time information in the CYCLE MARK packet is not necessary for a target to recreate an isochronous stream during playback, but it may be useful to applications that search for known time and cycle boundary locations in recorded isochronous data.

11.3 Isochronous data packets

The format of an isochronous data packet recorded on the medium is illustrated below.

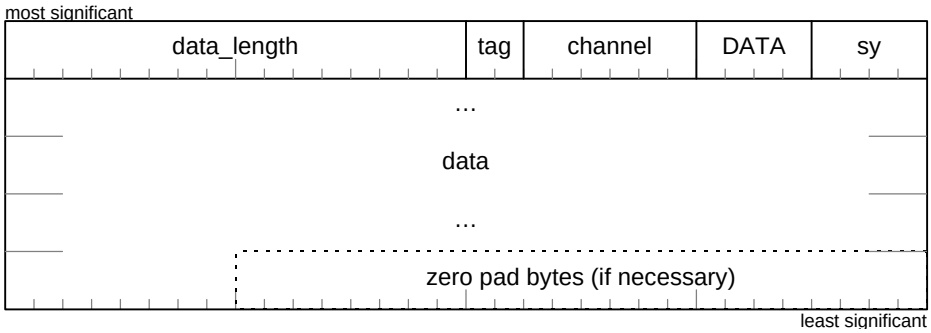

Figure 60 – Format for recorded isochronous data

The *data_length* field shall specify the length, in bytes, of the *data* field for the packet.

The *tag* field shall specify the format of the *data* field, encoded as indicated by the following table.

Value	Data field format
0	Unformatted data
1	Common isochronous packet (CIP) format
2 – 3	Reserved for future standardization

The *channel* field shall identify the isochronous channel number for the packet. The *channel* field recorded on the medium may have been transformed by the mapping from *ext_channel* to *int_channel* specified by a stream control ORB with a CONFIGURE PLUG control function (see 5.1.3). Upon playback, the *channel* field shall be transformed by the mapping from *int_channel* to *ext_channel* specified by a stream control ORB with a CONFIGURE PLUG control function.

The *sy*, or synchronization code, field is an application-dependent field, the details of whose use are beyond the scope of this standard.

NOTE – A synchronization point may be defined as a boundary between video or audio frames, or any other point in the isochronous stream the application may consider appropriate.

The *data* field shall contain *data_length* bytes of information and shall be padded with trailing zero bytes, as necessary, to occupy an integral number of quadlets on the medium.

Dependent upon the value of *tag*, the target may require additional knowledge of isochronous data formats. When *tag* is zero, the data payload of the isochronous packet is unformatted and requires no transformations upon either recording or playback. When *tag* is one, the format of the data payload shall conform to the common isochronous packet (CIP) format standardized by ISO/IEC 1883:199x. The components of the CIP format pertinent to targets are described in the clause that follows.

11.4 Common isochronous packets (CIP)

Isochronous data packets that conform to CIP format divide the data payload into two parts, the CIP header and the application-dependent data that follows. Figure 61 illustrates the organization of the common isochronous packet format.

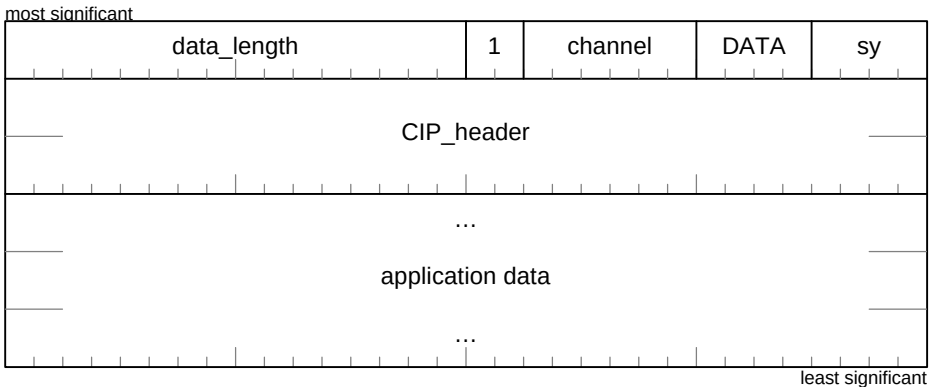

Figure 61 – Common isochronous packet (CIP) format

The CIP header is a variable number of quadlets (although only two are shown in the preceding figure). The most significant bit of each quadlet of the CIP header is called the *eoh* bit. For an *n* quadlet CIP header, *eoh* shall be zero for quadlets zero through *n* - 2, inclusive, and *eoh* shall be one for quadlet *n* - 1. The next most significant bit of each quadlet is called the *form* bit. Together, the *eoh* and *form* bits specify the format of the CIP header quadlet. At present, CIP header formats are defined for *form* values of zero; *form* values of one are reserved for future standardization.

The only CIP header format currently defined is a two-quadlet header shown below.

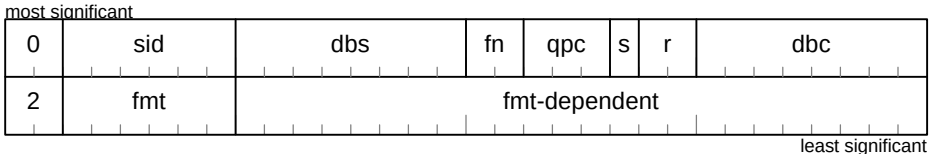

Figure 62 – Two-quadlet CIP header format

The *sid*, or source ID, field shall specify the Serial Bus physical ID of the source (talker) for the isochronous data. Upon playback, the target shall substitute its own physical ID for the recorded *sid* value in all transmitted isochronous packets.

The *dbs*, or data block size, field shall specify the size of the application-dependent data that follows the CIP header. A *dbs* value of zero encodes a size of 256 quadlets; for all other values of *dbs* the number of

quadlets is the value of *dbs* itself. More than one data block may be encapsulated within a single Serial Bus isochronous packet. Each encapsulated data block consists of a CIP header followed by application-dependent data.

The *fn*, or fraction number, field shall specify the number of data blocks that form a higher level, application-dependent object—the isochronous source packet. The number of data blocks that form an isochronous source packet is specified as 2^{fn}; when there is a one-to-one correspondence between isochronous source packets and data blocks, *fn* shall have a value of zero.

The *qpc*, or quadlet padding count, field shall specify the number of pad quadlets appended to an isochronous source packet before it is divided into data blocks. The quadlet padding count shall specify a value that results in equal sizes for the data blocks. If *fn* is zero, *qpc* shall also be zero.

The *sph*, or source packet header, bit (abbreviated as *s* in Figure 62) shall be one if the isochronous source packet begins with a header quadlet of the format shown below; otherwise, it shall be zero.

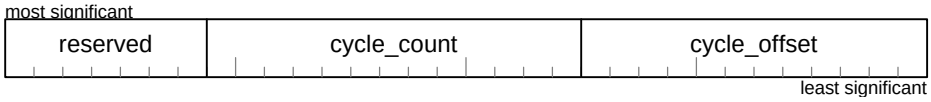

Figure 63 – Source packet header format

The source packet header shall contain a time stamp encoded in the same fashion as the least significant 25 bits of the CYCLE TIME register.

The *dbc*, or data block continuity counter, field shall specify the sequence number of the data block within the isochronous source packet. The range of permissible values for *dbc* is determined by the *fn* field and shall be between zero and 2^{fn} - 1, inclusive. Only the number of bits in the *dbc* field necessary to represent the sequence numbers are actually used; the content of the other bits of *dbc* is not specified by this standard. If *fn* is zero, the contents of *dbc* are not specified; otherwise, the *fn* least significant bits of *dbc* shall hold the sequence number.

The *fmt* field shall specify the formats of both the *fmt*-dependent field with the same quadlet of the CIP header and the application-dependent data contained within the common isochronous packets. An *fmt* value of $3F_{16}$ indicates that no application-dependent data follows the CIP header and that the *dbs*, *fn*, *qpc* fields, the *sph* bit and the *dbc* field in the CIP header shall all be ignored. Other values of *fmt* encode the application-dependent format of the isochronous data, *e.g.*, DVCR or MPEG. The details of most application-dependent formats are not relevant to targets and are beyond the scope of this standard. However, the most significant bit of *fmt* specifies the format of the *fmt*-dependent field within same quadlet of the CIP header; this field is meaningful to targets when it contains a time stamp, since the time stamp shall be transformed during playback. The table below summarizes the meanings of *fmt* for targets.

Value	Description
$0 – 1F_{16}$	Application data is present; the *fmt*-dependent field contains a time stamp defined by *syt* below
$20_{16} – 3E_{16}$	Application data is present; the contents of the *fmt*-dependent field are unspecified
$3F_{16}$	No application data is present

When *fmt* is in the range zero to $1F_{16}$, inclusive, the second quadlet of the CIP header has the format illustrated below.

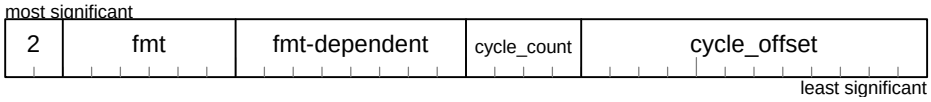

Figure 64 – Synchronization time (*syt*) format

The two fields, *cycle_count* and *cycle_offset*, are collectively referred to as the *syt*, or synchronization time, field. When *syt* has a value of $FFFF_{16}$, no synchronization time information is present and the *syt* field value shall be preserved upon playback. Otherwise, the *syt* field represents a time stamp encoded in the same fashion as the least significant 16 bits of the CYCLE_TIME register. Just as in the case of the CYCLE_TIME register, the value of *cycle_offset* is constrained to be in the range zero to 3071 inclusive; Values of *syt* for which *cycle_offset* is greater than 3071 are invalid. When *syt* contains valid *cycle_count* and *cycle_offset* fields, the target shall transform these values upon playback, as described in 12.3.4.

12 Isochronous operations

For each active channel on Serial Bus, isochronous data consists of zero or one isochronous packet transmitted by a talker each isochronous cycle and received by zero or more listeners in the same isochronous cycle. This section describes how an initiator may control isochronous data when a target participates, as either the talker or a listener, with other isochronous device(s) on Serial Bus..

Control of isochronous streams involves many different elements:

- the allocation of target resources (isochronous login requests);
- the establishment or breaking of connections between the target and other isochronous devices (connection management); and
- the transfer of isochronous data to or from the target's medium (stream command block requests);
- the starting, stopping and synchronization of isochronous data reception or transmission by the target from or to Serial Bus (stream control ORB's);
- the allocation of Serial Bus resources, such as channel numbers or bandwidth.

The first item, login, has already been defined as part of the access procedures in section 7.6. The last, channel and bandwidth allocation, are specified by IEEE Std 1394-1995. This section describes the remaining procedures of isochronous operations.

12.1 Connection management

All applications that control isochronous devices on Serial Bus shall conform to procedures defined in this clause for the management of isochronous connections between the devices. The plug control registers, previously defined in 6.5, provide the facilities upon which connection management operates. All isochronous devices shall implement plug control registers—this includes consumer electronic devices that do not implement SBP-2 as well as isochronous targets.

Connection management procedures are defined as a set of rules for the manipulation of plug control registers by direct access, through read and lock transactions, to these registers. This standard additionally defines indirect methods, such as the CONFIGURE PLUG control function in a stream control ORB, to modify the plug control registers. The same procedures shall apply whether a plug control register is modified directly or indirectly.

There are three fundamental connection types:

- point-to-point;
- broadcast out; and
- broadcast in.

A point-to-point connection exists when both endpoints of an isochronous data flow between a talker and a listener have been made visible in plug control registers at each device. That is, an OUTPUT_PLUG register at a talker and an INPUT_PLUG register at a listener both have the same value in their *channel* fields and both have nonzero values in their *point_to_point* fields.

A broadcast out connection exists when a talker transmits isochronous data but has no indication that there are listeners. In this case the talker's OUTPUT_PLUG register identifies an isochronous channel by the value in the *channel* field and indicates a broadcast out connection by a value of one for the *broadcast* bit.

A broadcast in connection exists when a listener is receiving (or is prepared to receive) isochronous data without indication that a talker exists. Analogously to a broadcast out connection, the listener's INPUT_PLUG register identifies an isochronous channel by the value in the *channel* field and indicates a broadcast in connection by a value of one for the *broadcast* bit.

Multiple point-to-point connections may exist simultaneously at a single output or input plug but only zero or one broadcast connection, out or in according to the nature of the plug, may exist at any time at a single plug. Broadcast and point-to-point connections may coexist simultaneously at a single plug.

The same isochronous data may be transported between two devices that share a point-to-point connection as may be transported if one has a broadcast out connection and the other a broadcast in connection. The essential difference is that point-to-point connections are considered protected: through conformance to these connection management procedures, only entities that establish point-to-point connections are permitted to break them. By contrast, broadcast connections may be terminated by any entity.

12.1.1 Plug states

Plugs are one of the concepts used in the description of connection management procedures. Each OUTPUT_PLUG or INPUT_PLUG register implemented by a target represents a plug. Plugs do not physically exist; they represent state information associated with the transmission or reception of an isochronous channel. The value of a plug control register is the visible manifestation of the state of that plug.

A plug may be in one of four states: IDLE, READY, SUSPENDED or ACTIVE. The state is evidenced by the values of the *online* and *broadcast* bits and the *point_to_point* field in the plug control register. State transitions are caused by modifications to the plug control register, either directly as the result of a Serial Bus lock transaction or indirectly as the result of a stream control ORB. Figure 65 below illustrates the state transitions. For the sake of clarity, the *broadcast* bit and the *point_to_point* field of the plug control register are described in the state diagram as if they comprised a single field named *connections*. If either *broadcast* or *point_to_point* is nonzero, *connections* is considered nonzero; this is also referred to as a connected condition for the plug. Otherwise, if both *broadcast* and *point_to_point* are zero, *connections* is deemed zero; the plug is called unconnected. Also, the symbol PLUG in the diagram indicates either an OUTPUT_PLUG or INPUT_PLUG register as appropriate.

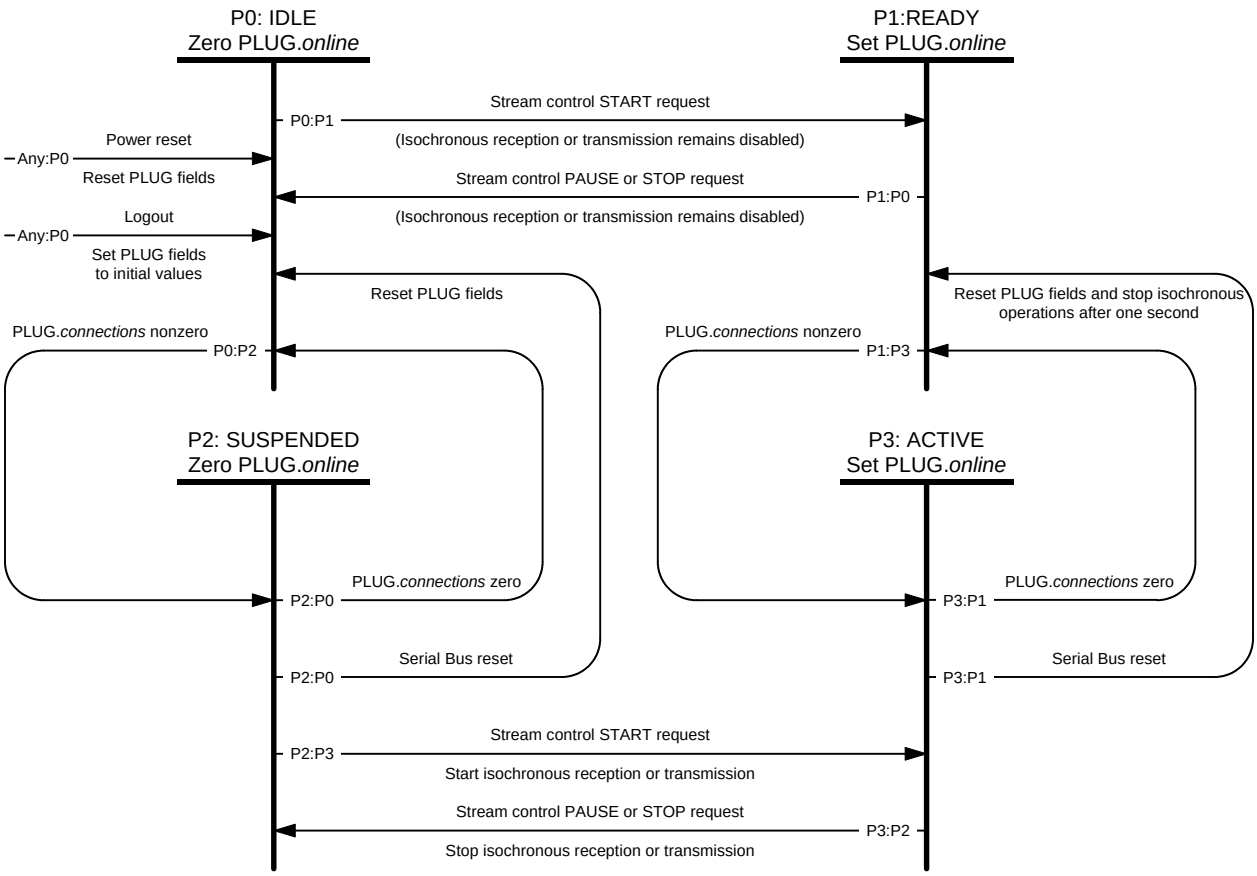

Figure 65 – Plug state transitions

Transition Any:P0. Any of a command reset, power reset or logout shall cause the plug control register to be reset to its initial values, as specified in 6.5, and shall place the plug in the idle state.

State P0: Idle. Upon entry to this state, the *online* bit in the plug control register shall be zeroed. The plug is available for use.

Transition P0:P1. The successful execution of a stream control ORB with a control function of START shall cause the *online* bit to be one. The plug shall transition to the READY state. The plug is still unable to transmit or receive isochronous data because no connections have been established.

Transition P0:P2. When either the *broadcast* bit or the *point_to_point* field becomes nonzero, the plug becomes connected and shall transition to the SUSPENDED state.

State P1: Ready. Upon entry to this state, the *online* bit in the plug control register shall be set to one. Since no connections exist, the plug is not yet configured to transmit or receive isochronous data.

Transition P1:P0. The successful execution of a stream control ORB with a control function of PAUSE or STOP shall cause the *online* bit to be zeroed. The plug shall transition to the IDLE state.

Transition P1:P3. When either the *broadcast* bit or the *point_to_point* field becomes nonzero, the plug becomes connected and shall transition to the ACTIVE state.

State P2: Suspended. Upon entry to this state, the *online* bit in the plug control register shall be zeroed. The plug is connected and may transmit or receive isochronous data when a stream control ORB START function is executed.

Transition P2:P0. When both the *broadcast* bit and the *point_to_point* field are zero, the plug becomes unconnected and shall transition to the IDLE state.

Transition P2:P0. A Serial Bus reset shall cause the fields in the plug control register to be set to their bus reset values, as specified in 6.5. This causes both the *broadcast* bit and the *point_to_point* field to be zeroed; the plug becomes unconnected and shall transition to the IDLE state.

Transition P2:P3. The successful execution of a stream control ORB with a control function of START shall cause the *online* bit to be set to one. The plug shall transition to the ACTIVE state and may transmit or receive isochronous data.

State P3: Active. Upon entry to this state, the *online* bit in the plug control register shall be set to one. The plug is both connected and able to transmit or receive isochronous data.

Transition P3:P1. When both the *broadcast* bit and the *point_to_point* field are zero, the plug becomes unconnected and shall transition to the READY state.

Transition P3:P1. A Serial Bus reset shall cause the fields in the plug control register to be set to their bus reset values. This causes both the *broadcast* bit and the *point_to_point* field to be zeroed; the plug becomes unconnected and shall transition to the READY state. However, the target shall, for one second, continue to behave as if the plug control register retained its value prior to the bus reset.

NOTE – This one second isochronous delay permits isochronous operations to continue for a short period of time without disruption. During this time, the controlling application(s) are expected to reallocate the necessary isochronous channel(s) and bandwidth and then to reconfirm these by a write to the plug control register(s). In the event that the necessary resources cannot be reallocated, the expiration of the one second isochronous delay forces the plug to transition to the READY state and stops isochronous transmission or reception.

Transition P3:P2. The successful execution of a stream control ORB with a control function of PAUSE or STOP shall cause the *online* bit to be zeroed. The plug shall transition to the SUSPENDED state.

12.1.2 Establishing and breaking connections

A connection is established at a plug whenever the *broadcast* bit is modified from zero to one or whenever the *point_to_point* field is incremented. As described above, a plug makes a transition from an unconnected to a connected state upon the establishment of the first connection. A connection is broken at a plug whenever the *broadcast* bit is modified from one to zero or whenever the *point_to_point* field is decremented. In like fashion to the transition to connected, a plug makes a transition from a connected to an unconnected state upon the breaking of the last connection.

The following rules shall apply when an OUTPUT_PLUG or INPUT_PLUG register is modified, whether directly or indirectly:

- The channel number and necessary isochronous bandwidth shall be allocated in the isochronous resource manager's CHANNELS_AVAILABLE and BANDWIDTH_AVAILABLE registers before the corresponding output plug is connected. The channel and bandwidth shall not be deallocated, except as a result of a command or power reset at the isochronous resource manager or as a result of a Serial Bus reset, while the output plug remains connected;
- The *channel* and *spd* fields of an OUTPUT_PLUG register shall not be modified while the plug is connected;

- The *channel* field of an INPUT_PLUG register shall not be modified while the *point_to_point* field in the same register is nonzero;
- The *broadcast* bit shall not be set to one by a Serial Bus lock transaction; a lock transaction may zero the bit. A stream control ORB with a control function of CONFIGURE PLUG may zero the *broadcast* bit or set it to one;
- When an output plug transitions to a connected state, the *broadcast* bit and the *point_to_point*, *spd* and *overhead* fields shall be modified in a single operation, either one Serial Bus lock transaction or one CONFIGURE PLUG request. Additionally, if a CONFIGURE plug request is used the payload field shall also be modified in the same request; and
- If the *broadcast* bit of an OUTPUT_PLUG register is modified from zero to one at the same time that the *point_to_point* field remains zero, the *channel* field shall be modified in the same operation, either a Serial Bus lock transaction or a CONFIGURE PLUG request. The value for *channel* shall be determined from the OUTPUT_MASTER_PLUG register's *broadcast_base* field, as defined in 6.5.1.

When an entity attempts to establish a connection through an OUTPUT_PLUG or INPUT_PLUG register (or both), it shall retain sufficient prior state information to reverse the plug control register modifications in the event that the connection cannot be made.

12.2 Stream command block requests

With the notable exception of the lack of *data_descriptor* and *data_size* fields, stream command block requests are essentially similar to and operate in a like fashion as normal command block requests that transfer data to or from the medium.

The target agent that is responsible to fetch stream command block ORB's operates in the same way as the command block agent(s). The initiator may build linked lists of stream command block ORB's, signal them to the target and dynamically append new stream command block ORB's while the target is active. See section 9 for a more detailed description; the information is equally applicable to normal command block requests and stream command block ORB's.

Stream command block requests shall be executed in order. Each isochronous stream active at a logical unit shall have its own task set that is disjoint from all other task sets for the logical unit. The task set for an isochronous stream is instantiated when a successful isochronous login completes. The details of a task set for isochronous streams are explained at length in 10.1.

Unlike normal command block requests, the execution of stream command block ORB's may be temporarily blocked by the availability of target resources. The data provided to or returned from a stream command block ORB has no associated data buffer in system memory. Instead, the stream controller implemented within the target is the source or sink of the data for a stream command block ORB. A stream command block ORB that transfers data to the device medium shall await the delivery of *stream_length* bytes of data from the stream controller before the stream request shall complete. A stream command block ORB expected to transfer *stream_length* bytes of data to the stream controller shall suspend data transfer from the medium if the target's internal data path to the stream controller is full and shall resume data transfer from the medium as resource availability permits. These are not error conditions.

A stream command block ORB that transfers data from the device medium shall return completion status to the initiator when *stream_length* bytes of data have been successfully transferred to the stream controller or an error condition occurs. The return of completion status to the initiator does not provide any information as to how many of the *stream_length* bytes of data have been transmitted on Serial Bus.

A stream command block ORB that transfers data to the device medium shall return completion status to the initiator when *stream_length* bytes of data have been successfully transferred from the stream

controller or an error condition occurs. Alternatively, the target may be configured to delay the return of completion status until the data has been successfully written to the medium.

In both of these cases, note that no mechanism exists whereby the initiator may determine how much isochronous data is in the target's internal buffers that connect the stream controller with the stream command block ORB's.

12.3 Stream control

The stream controller is implemented within the target to mediate the transfer of data between Serial Bus and the stream command block ORB's that govern the transfer of data to or from the medium. The stream controller's responsibilities include:

- the synchronization, starting and stopping of either the reception or transmission of isochronous data from or to Serial Bus. The synchronization may occur at a specified time or may occur in response to a specified data patter in observed isochronous data;
- the selective enablement of specified isochronous channels. When listening, the stream controller selects which channels to receive and provide to the stream command block ORB's. When talking, the stream controller selects which channels to transmit from the data provided by stream command block ORB's; and
- the transformation of Serial Bus header and common isochronous packet (CIP) header information between the representation of the isochronous data on Serial Bus and on the device medium. This may include the channel number and time-stamp information and the generation or elimination of null packets, as appropriate.

The target agent that is responsible to fetch stream control ORB's supports a linked list of requests that the initiator may build in system memory. This capability is essential for stream control ORB's, since synchronization boundaries may occur with as little separation as 125 μs. See section 9 for a more detailed description; the information is as applicable to stream control ORB's as it is to normal command block requests and stream command block ORB's.

The clauses below describe the procedures that an initiator shall use to govern the actions of the stream controller. In addition, the stream controller shall transform isochronous header information independent of any stream control ORB's issued by the initiator.

12.3.1 Plug configuration

Before isochronous data can be transmitted or received by a target, the plug(s) that mediate the transfer shall be configured. Part of the configuration is the establishment of connections, as already described in 12.1.2. Although connection information may be stored in the plug control register(s) directly by means of Serial Bus lock transactions, it is expected that the CONFIGURE PLUG control function be used both to establish the connection(s) and to store other plug parameters.

In the case of an input plug, only the *channel* field and either the *broadcast* bit or the *point_to_point* field need to be set in the INPUT_PLUG register for the plug to be fully configured.

The configuration of an output plug is more complex. In addition to the connection information required for any plug (*broadcast*, *point_to_point* and *channel* fields), the output plug shall also be configured for transmission speed, the quantity of data (both header and payload) and the Serial Bus arbitration time required to transmit a packet each isochronous cycle. The *spd* field shall be set to the desired transmission speed, as encoded by Table 1. The *overhead* field shall be set to a value that reflects the Serial Bus topology, as described by the following table.

Note that *overhead* accounts for the time required to transmit the overhead components of an isochronous packet (a header quadlet, the header CRC and the data CRC) as well as for the time occupied by Serial Bus arbitration before the packet may be transmitted.

Lastly, the *payload* field shall be set to the maximum number of data quadlets that are present, within a single packet, in the previously recorded stream of isochronous data on the medium. The maximum applies only to the channel controlled by this plug; the sum of the values of *payload* for all plugs configured for a single stream shall be less than or equal to the value of *max_isochronous_payload* provided in the login request for the stream.

All of these values, whether for an input or an output plug, shall be established by means of a single stream control ORB with a CONFIGURE PLUG control function. The stream control ORB shall reference one of the plug control registers allocated in the login response for the stream.

12.3.2 Channel masks

Plugs are configured to receive or transmit isochronous channels independently of the selective enablement or disablement of the individual channels. The selection of an active set of channels shall be performed by the stream controller according to a channel mask maintained for each stream.

The channel mask may be updated by a stream control ORB with a SET CHANNEL MASK control function, as described in 5.1.3. The point in time at which the channel mask is updated may be synchronized with Serial Bus time or, in the case when the target is a listener, it may be synchronized with a data pattern in the isochronous data.

When the target is a talker, the channel mask refers to channel numbers as they are recorded on the medium—the same value to which the *int_channel* field of an OUTPUT_PLUG register refers. The channel mask is applied to the stream of isochronous data presented to the stream controller before any transformations are performed and before isochronous packets are transmitted on Serial Bus.

When the target is a listener, the channel mask refers to channels numbers as they are observed on Serial Bus—the same value to which the *ext_channel* field of an INPUT_PLUG register refers. The channel mask is applied to select isochronous packets from Serial Bus before any transformations are performed and before the data is recorded on the medium.

12.3.3 Synchronization

The execution of the stream control ORB functions START, STOP, PAUSE and UPDATE CHANNEL mask can be synchronized with external events, as described in 5.1.3. This section describes the effect of synchronization on the transfer of data between the stream controller and the stream command block ORB's.

When the target is a talker, data flows from the device medium (under the control of stream command block ORB's) to the stream controller. If the stream controller is in a state where isochronous data transmission is enabled (typically as the result of a stream control ORB with a START control function), isochronous data is selected according to the stream's channel mask, is then transformed as described in 12.3.4 and, dependent upon the state of the output plug, may be transmitted on Serial Bus. If the plug for a channel is not in the ACTIVE state, the isochronous data packets transformed by the stream controller shall be discarded. If the stream controller is in a state where isochronous data transmissions are 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.

When the target is a listener, data flows from Serial Bus to the stream controller. If the stream controller is in a state where isochronous data reception is enabled (typically as the result of a stream control ORB with a START control function) and if the input plug is in the ACTIVE state, isochronous data is selected

according to the stream's channel mask, is then transformed as described in 12.3.4 and is transferred to the device medium under the control of stream command block ORB's. If the plug for a channel is not in the ACTIVE state, isochronous packets for that channel are not received and the data recorded on the medium shall be the same as if no isochronous packets were observed for the channel. This shall not affect the reception of data for other channels within the stream.

12.3.4 Isochronous data transformation

The stream controller is responsible to both filter and transform isochronous data as it is recorded on or played from the device medium. Filtering is performed according to the channel number of the source, Serial Bus or device medium, before any transformations are applied. Once a set of enabled channel(s) is selected, the stream controller shall:

- replace the channel number found in the source isochronous data with the remapped channel number for the destination isochronous data. This may be the identity map;
- upon playback, if the isochronous data conforms to the common isochronous packet (CIP) format, replace the *sid* (or source ID) field in the CIP header with the target's own physical ID;
- when recording, transform all instances of time stamps in the CIP or source packet headers to reflect the delta, or time difference, between the time stamp and the Serial Bus isochronous cycle time at which the data is observed; and
- upon playback, add the Serial Bus isochronous cycle time at which the data is transmitted to all instances of time stamps in the CIP or source packet headers, in order to recreate an absolute time stamp.

The first two transformations are straightforward and require no additional explanation. The transformations to be applied to time stamps are more complex and are explained below.

The common isochronous packet (CIP) format, as currently standardized, contains isochronous time stamp information that is absolute rather than relative. That is, the time stamps contained within header data are a fixed offset ahead of, in the most significant bits, the isochronous cycle times contained in the cycle start packet that signals the cycle in which the packets are transmitted. Time stamps are found in two places in packets that conform to CIP format:

- the *syt* field of the second quadlet of a two-quadlet CIP header if the *fmt* field in that quadlet has a value between zero and $1F_{16}$, inclusive; and
- the *cycle_count* and *cycle_offset* fields of the isochronous source packet header.

See 11.4 for the exact specifications of these fields and the circumstances under which they are present in isochronous data.

In order to correctly generate absolute time stamps upon playback, the stream controller shall calculate a delta time constant for the first packet transmitted and then apply this delta time constant to all subsequent packets.

In order to permit the subsequent recreation of absolute time stamps upon playback, the stream controller shall calculate and store the delta, or time difference between the time stamp in the observed data and the CYCLE_TIME register when the data is observed.

For the *syt* field, the delta time shall be obtained by applying the following formula (shown in C code notation):

$$syt_{stored} = (syt_{observed} \text{ \& 0xF00}) - (\text{CYCLE\_TIME \& 0x0000F000})$$

For the *cycle_count* and *cycle_offset* fields of source packet headers, the procedure is essentially the same but the size of the fields permits 25 bits of cycle time to be expressed rather than the 16 bits that *syt* accommodates. If the *cycle_count* and *cycle_offset* fields are together considered as if they were one field, *sph_time*, the delta time shall be obtained by applying the following formula:

$$sph\_time_{stored} = (sph\_time_{observed} \text{ \& } 0x1FFF000) - (CYCLE\_TIME \text{ \& } 0x01FFF000)$$

The transformation upon playback is simpler. In each case, for either the *syt* field in a CIP header or a source packet header, the value of the CYCLE_TIME register for the isochronous cycle in which the packet is transmitted shall be masked with 01FF $F000_{16}$ and the result shall be added to the time stamp obtained from the medium. The resultant value, now an absolute time in terms of current isochronous cycle time, shall be transmitted in the outbound isochronous packets. The most significant bits of the sum shall be discarded so the result fits within the 16-bit *syt* field or the 25-bit field formed by the *cycle_count* and *cycle_offset* fields, as appropriate.

12.4 Error logs

When an isochronous stream is active, a stream controller may detect error conditions on Serial Bus or internally. Typical errors include but are not limited to:

- a missing isochronous packet or cycle start indication;
- an isochronous packet with a data CRC error;
- when the target is a talker, an underflow in the availability of data from the stream command block ORB's that causes no data to be transmitted for one or more channels during an isochronous cycle; or
- when the target is a listener, an overflow in which isochronous data from Serial Bus must be discarded because of an internal buffer overflow or a lack of stream command block ORB(s) to transfer the data to the medium.

An error log buffer provided in system memory by the initiator permits the stream controller to report these and other errors. The stream controller may be placed in one of three error reporting modes by means of a stream control ORB with a SET ERROR MODE control function:

- report the first error and stop execution of the current stream control ORB;
- report all errors but continue executing the current stream control ORB; or
- ignore all errors and continue executing the current stream control ORB.

The *rpt* field in the stream control ORB establishes one of the three error reporting modes and the *error_log* field is used in the same request to establish the address of the error log. See 5.1.3 for details of the SET ERROR MODE control function.

Error log entries identify the duration of a discontinuity in isochronous data. Typically there is one entry that specifies the time at which the discontinuity is detected and a complementary entry that specifies the time at which isochronous data flow resumes without errors. Insufficient buffer space for the error log or other errors may disrupt this symmetry. Each entry in the error log consists of two quadlets with the format shown below.

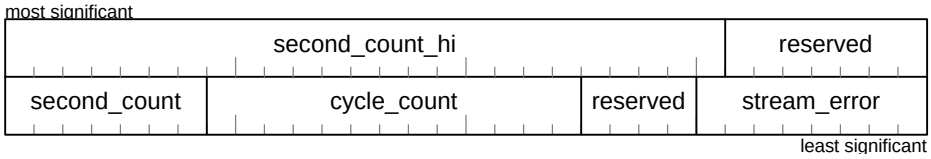

Figure 66 – Error log entry format

The *second_count_hi*, *second_count*, and *cycle_count* fields indicate the time of the error event. The value of the *second_count_hi* field shall be obtained from the target's BUS_TIME register for the isochronous cycle in which the error took place while the values of the *second_count* and *cycle_count* fields shall be obtained from the cycle start packet that initiated the isochronous cycle.

The *stream_error* field shall specify the nature of the event, as encoded by the table below.

Value	Stream error description
0	First valid isochronous data cycle following a discontinuity
1	Start of a discontinuity caused by an unspecified error
2	Start of a discontinuity caused by the target's inability to source or sink isochronous data
3	Start of a discontinuity caused by a missing isochronous packet (listener only)
4	Start of a discontinuity caused by a data CRC error in an isochronous packet (listener only)
5 – FF_{16}	Reserved for future standardization

Note that if a connection exists between two targets, one a talker and the other a listener, that the error logs created by each would not necessarily contain identical entries.

The contents of the system memory allocated to the error log are unspecified and should not be examined by the initiator until completion status has been returned for the stream control ORB that specified the error log address.

Annex A
(normative)

Minimum Serial Bus node capabilities

In addition to those minimum capabilities defined by IEEE Std 1394-1995, this annex specifies the minimum capabilities that an initiator or a target shall support in order to implement SBP-2.

A.1 Initiator capabilities

An initiator shall be capable of responding to block read or write requests with a *data_length* less than or equal to 32 bytes.

For the largest value of *max_payload* specified in any normal command block ORB signaled to the target, the initiator shall be capable of responding to block read and write requests with a *data_length* less than or equal to $2^{max\_payload + 2}$ bytes.

The initiator shall report the larger of these two possible *data_length* values by setting the value of the *max_rec* field in the bus information block in configuration ROM to a value of ($\log_2$ *data_length*) - 1.

A.1 Target capabilities

A target shall be capable of responding to block read or write requests with a *data_length* equal to eight bytes if the *destination_offset* specifies either the MANAGEMENT_AGENT or the ORB_POINTER register.

The target shall be capable of initiating block write requests with a *data_length* of at least eight bytes. Consequently, the *dreq* bit in the STATE_CLEAR and STATE_SET registers shall be implemented.

The target shall report this capability by setting the *drq* bit in the Node_Capabilities entry in configuration ROM to one.

NOTE – The value of STATE_CLEAR.*dreq* shall be unaffected by a Serial Bus reset. The target may automatically set *dreq* to zero (request initiation enabled) upon a power reset or a command reset.

The target shall report this capability by setting the value of the *max_rec* field in the bus information block in configuration ROM to a value of two.

Annex B
(informative)

Sample configuration ROM

Configuration ROM is located at a base address of FFFF F000 0400_{16} within a node's initial memory space. The requirements for general format configuration ROM for targets are specified in section 7. This annex contains an illustration of a typical configuration ROM for a simple target.

most significant

4	0014_{16}	ROM CRC (calculated)
3133 3934_{16} (ASCII "1394")		
node_options (00FF 2000_{16})		
node_vendor_ID		chip_ID_hi
chip_ID_lo		
4		Root directory CRC (calculated)
03_{16}	module_vendor_ID	
$0C_{16}$	node_capabilities (00 8380_{16})	
$8D_{16}$	2	
$D1_{16}$	4	
2		Leaf CRC (calculated)
node_vendor_ID		chip_ID_hi
chip_ID_lo		
7		Unit directory CRC (calculated)
12_{16}	unit_spec_ID (00 $609E_{16}$)	
13_{16}	unit_sw_version (01 0483_{16})	
38_{16}	command_set_spec_ID	
39_{16}	command_set_version	
54_{16}	csr_offset (00 4000_{16})	
$3A_{16}$	01 $0A08_{16}$	
14_{16}	00 0000_{16}	

least significant

Figure B.1 – Sample configuration ROM

The ROM CRC in the first quadlet is calculated on the twenty quadlets of ROM information that follow.

B.1 Root directory

The *node_options* field represents a collection of bits and fields specified in 7.1. The value shown, $00FF\ 2000_{16}$, represents basic characteristics of a device that is not isochronous capable. This value is composed of a *cyc_clk_acc* field with a value of FF_{16} and a *max_rec* value of two. The *max_rec* field encodes a maximum payload of eight bytes in block write requests addressed to the target.

The Node_Capabilities entry in the root directory, with *key_type* and *key_value* fields of $0C_{16}$, has a value where the *spt*, *64*, *fix*, *lst* and *drq* bits are all one. This is a minimum requirement for targets.

The Node_Unique_ID entry in the root directory, with *key_type* and *key_value* fields of $8D_{16}$, has an *indirect_offset* value of two that points to the node unique ID leaf.

The Unit_Directory entry in the root directory, with *key_type* and *key_value* fields of $D1_{16}$, has an *indirect_offset* value of four that points to the unit directory.

B.2 Unit directory

The Command_Set_Spec_ID and Command_Set_Version entries, with *key_type* and *key_value* fields of 38_{16} and 39_{16}, respectively, are expected to define the command set used by the target.

The Management_Agent entry in the unit directory, with *key_type* and *key_value* fields of 54_{16}, has a *csr_offset* value of $00\ 4000_{16}$ that indicates that the management agent CSR has a base address of $FFFF\ F001\ 0000_{16}$ within the node's initial memory space.

The Logical_Unit_Characteristics entry in the unit directory, with *key_type* and *key_value* fields of $3A_{16}$, has an immediate value of $01\ 0A08_{16}$ that indicates a target that implements the basic task management model, may reorder tasks without restriction and does not support isochronous operations. In addition, the target is expected to complete a login within five seconds and fetches 32-byte ORB's.

The Logical_Unit_Number entry in the unit directory, with *key_type* and *key_value* fields of 14_{16}, has an immediate value of zero that indicates a direct-access device with a logical unit number of zero.