

1394.3™

IEEE Standard for a High Performance Serial Bus Peer-to-Peer Data Transport Protocol (PPDT)

IEEE Computer Society

Sponsored by the
Microprocessor and Microcomputer Standards Committee



Published by
The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

14 April 2004

Print: SH95173
PDF: SS95173

Recognized as an
American National Standard (ANSI)

IEEE Std 1394.3™-2003 (R2008)

IEEE Standard for a High Performance Serial Bus Peer-to-Peer Data Transport Protocol (PPDT)

Sponsor

**Microprocessor and Microcomputer Standards Committee
of the
IEEE Computer Society**

Reaffirmed 10 December 2008

Approved 11 September 2003

IEEE-SA Standards Board

Approved 29 January 2004

American National Standards Institute

Abstract: This standard defines a peer-to-peer data transport (PPDT) protocol between Serial Bus devices that implement Serial Bus Protocol 2 (SBP-2). The facilities specified include device and service discovery, self-configurable (plug and play) binding, and connection management.

Keywords: computers, CSR architecture, connect, peer-to-peer, SBP-2, transport protocol

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2004 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 14 April 2004. Printed in the United States of America.

Print: ISBN 0-7381-3820-7 SH95173
PDF: ISBN 0-7381-3821-5 SS95173

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied “**AS IS**.”

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

[This introduction is not part of IEEE Std 1394.3-2003, IEEE Standard for a High Performance Serial Bus Peer-to-Peer Data Transport Protocol (PPDT).]

The IEEE approved P1394.3 on 30 January 2000 at the request of Gregory LeClair, former Chair of the 1394 Printer Working Group (PWG), an informal industry consortium. Although final approval of this standard has taken place under the aegis of the IEEE, most of the significant effort occurred prior to the formation of the P1394.3 working group.

The necessity for a Serial Bus transport protocol for printers and similar devices was first broached in hallway conversation at the quarterly meeting of the 1394 Trade Association in Redmond, WA, in October, 1996. This led to a “birds of a feather” assembly hosted at Adobe Systems in San Jose, CA, on 6 December 1996. The attendees at this crowded gathering discussed the suitability of existing protocols, particularly SBP-2, and the venue for the proposed standards work. The group concluded that no existing protocol suited their needs: one would have to be developed. Don Wright offered the services of the PWG, an ad hoc industry forum whose principal participants are printer vendors, and Greg LeClair wrote the initial charter for the group.

The first official meeting of the 1394 PWG was held in Albuquerque, NM, early in 1997. Consensus was quickly reached that the new protocol must be friendly to legacy data formats; this exposed the two major areas that occupied the working group for the next three years: a) device discovery and enumeration, and b) a data transport protocol.

The working group examined the facilities of IEEE Std 1394-1995 and the CSR architecture and concluded that configuration ROM might permit practical solutions for device discovery but that the current standard, ISO/IEC 13213:1994, did not adequately specify the necessary building blocks. The 1394 PWG applied to the IEEE Microprocessor and Microcomputer Standards Committee (MMSC) for a project authorization request (PAR) to commence new work. Simultaneously, the CSR architecture was due for revision and the MMSC approved a PAR for IEEE P1212 [now approved and published as IEEE Std 1212™-2001, IEEE Standard for a Control and Status Registers (CSR) Architecture for Microcomputer Buses] and suggested to the 1394 PWG that work be transferred to the IEEE. In fact, many participants became simultaneously active in the IEEE P1212 working group and provided strong continuity between the two complementary efforts.

With work on device discovery split off to IEEE P1212, the 1394 PWG was free to focus its efforts on the transport protocol. There were two principal candidates, SBP-2 and the “thin” protocol jointly developed by camera and printer companies in Japan. No matter which protocol (or adaptation thereof) was ultimately chosen, the 1394 PWG thought it was important to recognize the peer-to-peer nature of the desired operating environment. The working group concluded that a solution similar to the “sockets” API that supported confirmed, efficient bi-directional data delivery between peers would suit their needs. Close to a year was spent debating the pros and cons of different approaches. In the end, the group converged on Serial Bus Protocol 2 (SBP-2) as the underlying mechanism: the 1394 PWG would focus its efforts on enhancements to SBP-2 to render it more fully peer-to-peer.

Once the selection of an underlying protocol was agreed, work progressed fairly quickly on what was eventually dubbed peer-to-peer data transport (PPDT) protocol. The prototype efforts expended by some early adopters, both in Japan and the United States, have been invaluable in the resolution of detailed technical questions that arose as this standard was refined. In particular, the essential mechanisms first proposed as the Simple High Performance Transport (SHPT) were ultimately adopted. SHPT took advantage of the SBP-2 unordered execution model to provide independent, bi-directional data transport within the context of a single SBP-2 login. SHPT also optimized error recovery operations after bus reset by using execution context information known by both initiator and target.

Towards the end of summer 1999, it was apparent that the work neared conclusion; at this time the working group discussed possible homes for the draft standard and agreed upon the IEEE as the best choice.

Patent notice

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying all patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

The patent holder has, however, filed a statement of assurance that it will grant a license under these rights without compensation or under reasonable rates and nondiscriminatory, reasonable terms and conditions to all applicants desiring to obtain such a license. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreement offered by the patent holder. Contact information may be obtained from the IEEE Standards Department.

Participants

The following is a list of active participants in the IEEE 1394.3 working group (those who attended more than three meetings from original inception of the predecessor 1394 Printer Working Group activity to the time of publication).

Gregory LeClair, *Chair*
Alan Berkema, *Vice Chair*
Peter Johansson, *Editor*
Lee Farrell, *Co-secretary*
Larry Stein, *Co-secretary*

Brian Batchelder
Scott Bonar
Mike Fenelon
Doug Fritz
John Fuller
Osamu Hirata
Takashi Isoda
Laurie Lasslo

Bob Morford
Fumio Nagasaka
Brian Nagy
Atsushi Nakamura
Kazuo Nomura
Charles Rice
Fumio Samitsu
Yuji Sasaki
Hitoshi Sekine

Akihiro Shimura
Nobuhiko Shinoda
Greg Shue
Jerry Thrasher
Randy Turner
Shigeru Ueda
Rainier Wolfcastle
Don Wright

The following members of the balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Edmund H. Baulsir
Alan Berkema
Lon Canaday
Keith Chow
Sourav K. Dutta
Edwin Vivian El-Kareh
Lee Farrell
Michael A. Fischer
John Fuller
Clay E. Hudgins
Peter Johansson

Thomas M. Kurihara
Conrad A. Laurvick
Gregory LeClair
Joseph R. Marshall
Gene E. Milligan
Klaus-Dieter Mueller
Atsushi Nakamura
Robert O'Hara
Roman Orzol
Charles Rice

Gary S. Robinson
Jaideep Roy
Akihiro Shimura
Gary K. Sloane
Larry Stein
Michael Teener
Jerry Thrasher
Colin Whitby-Stevens
Don Wright
Oren Yuen
Janusz Zalewski

The following individuals served on the ballot response committee:

Brian Batchelder
Alan Berkema
Lon Canaday

Lee Farrell
Mike Fenelon
Peter Johansson
Fumio Nagasaka

Akihiro Shimura
Jerry Thrasher
Don Wright

When the IEEE-SA Standards Board approved this standard on 11 September 2003, it had the following membership:

Don Wright, *Chair*
Howard M. Frazier, *Vice Chair*
Judith Gorman, *Secretary*

H. Stephen Berger
Joe Bruder
Bob Davis
Richard DeBlasio
Julian Forster*
Toshio Fukuda
Arnold M. Greenspan
Raymond Hapeman

Donald M. Heirman
Laura Hitchcock
Richard H. Hulett
Anant Jain
Lowell G. Johnson
Joseph L. Koepfinger*
Tom McGean
Steve Mills

Daleep C. Mohla
William J. Moylan
Paul Nikolic
Gary Robinson
Malcolm V. Thaden
Geoffrey O. Thompson
Doug Topping
Howard L. Wolfman

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Alan Cookson, *NIST Representative*
Satish K. Aggarwal, *NRC Representative*

Andrew Ickowicz
IEEE Standards Project Editor

Contents

1.	Overview	1
1.1	Scope	1
1.2	Purpose	1
2.	References	2
3.	Definitions, abbreviations, and notation	2
3.1	Definitions	2
3.2	Abbreviations	6
3.3	Notation	6
4.	Model (informative)	8
4.1	Protocol stack and service model	8
4.2	Independent data paths for each service	9
4.3	Connection management	10
4.4	Data transfer between initiator and target	11
4.5	Control requests and responses	12
4.6	Unsolicited status	13
4.7	Reverse login and logout	13
5.	Data structures	14
5.1	Transport flow ORBs	14
5.2	Status block	15
5.3	Control information	17
5.4	Queue information	20
5.5	Reverse login request and response	21
6.	Control operations	23
6.1	Login, reverse login and queue zero	23
6.2	Autonomous response information	24
6.3	Connection management	25
6.4	Queue status information	29
6.5	Logout and reverse logout	30
7.	Transport flow operations	30
7.1	Data transfer to a target	32
7.2	Data transfer to an initiator	33
7.3	Completion status	34
7.4	Execution context for active ORBs	35
7.5	Error recovery	36
8.	Configuration ROM	37
8.1	Root directory	39
8.2	Instance directories	40

8.3 Feature directories	40
8.4 Keyword leaves	40
8.5 Initiator unit directory	41
8.6 Target unit directories	42
8.7 Private unit directory	43
8.8 Device ID	43
Annex A (normative) Minimum Serial Bus node capabilities	45
Annex B (normative) Compliance with ANSI INCITS 325-1998 (SBP-2)	46
Annex C (normative) Control request and response parameters	47
Annex D (normative) Control and status registers	48
Annex E (normative) Service ID registration	49
Annex F (informative) Configuration ROM	51
Annex G (informative) Bibliography	59

IEEE Standard for a High Performance Serial Bus Peer-to-Peer Data Transport Protocol (PPDT)

1. Overview

1.1 Scope

This is a full-use standard whose scope is the definition of a peer-to-peer data transport (PPDT) protocol between Serial Bus devices that implement ANSI INCITS 325-1998¹. The facilities specified include, but are not limited to, the following:

- Device and service discovery. PPDT devices may use uniform discovery procedures to locate other PPDT devices on the same bus. These procedures are extensible to an interconnected net of buses, when specified by IEEE P1394.1TM [B2]. Once other PPDT devices are identified, facilities are provided to permit client applications to discover services;
- Self-configurable (plug and play) binding of device drivers to PPDT devices in a dynamic environment where users are free to insert and remove devices at will; and
- Connection management. A PPDT device [either a Serial Bus Protocol 2 (SBP-2) initiator or target] may establish and manage uni- or bi-directional connections for data transfer with other PPDT devices. The connections may be blocking or nonblocking, dependent upon application requirements, and operate independently of each other.

Although this standard was developed by printer and scanner experts, it is relevant to any application that requires efficient, peer-to-peer transport of data between SBP-2 devices.

1.2 Purpose

Experience with SBP-2 has demonstrated its high efficiency for the confirmed transport of large quantities of data between two devices. For historical reasons, SBP-2 is tailored to an environment where one device is the client (initiator) and the other the server (target); this is not necessarily the most natural approach when client applications and their associated servers may be located within initiator, target or both. Because SBP-2 is already widely implemented in operating systems, this standard leverages that effort in order to enhance the value of Serial Bus to devices in a wider range of operational circumstances.

¹Information on references can be found in Clause 2.

This standard creates a new layer of protocol services based upon SBP-2, but that provides building blocks more suited to a peer-to-peer environment which includes printers, facsimile devices, and scanners (or multifunction devices that present some combination of these capabilities) when a computer is present—but it is also intended to address the peer-to-peer needs of devices to communicate with each other in the absence of a computer.

2. References

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

ANSI INCITS 325-1998, American National Standard for Information Technology—Serial Bus Protocol 2 (SBP-2).²

ANSI X3.4-1986, American National Standard for Information Systems—Coded Character Sets—7-Bit American National Standard Code for Information Interchange (7-Bit ASCII).

IEEE Std 1212™-2001, IEEE Standard for a Control and Status Registers (CSR) Architecture for Micro-computer Buses.^{3, 4}

IEEE Std 1284™-2000, IEEE Standard Signaling Method for a Bi-directional Parallel Peripheral Interface for Personal Computers.

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

IEEE Std 1394a™-2000, IEEE Standard for a High Performance Serial Bus—Amendment 1.

IEEE Std 1394b™-2002, IEEE Standard for a High Performance Serial Bus—Amendment 2.

3. Definitions, abbreviations, and notation

For the purposes of this standard, the following terms, definitions, abbreviations, and notational conventions apply. IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms* [B1], should be consulted for terms not defined in this clause.

3.1 Definitions

3.1.1 Conformance definitions

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.

²ANSI publications are available from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (<http://www.ansi.org/>).

³IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (<http://standards.ieee.org/>).

⁴The IEEE standards referred to in Clause 2 are trademarks belonging to the Institute of Electrical and Electronics Engineers, Inc.

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 ignore its value. The recipient of an object larger than a bit and defined by this standard as other than reserved shall inspect 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 Technical definitions

The following terms are used in this standard:

3.1.2.1 active ORB: From the perspective of an initiator, an operation request block (ORB) in a logical unit task set, i.e., an ORB for which completion status has yet to be stored at the initiator’s *status_FIFO*. From the perspective of a logical unit, an ORB for which the logical unit maintains execution context.

3.1.2.2 active queue: A queue, created as part of a connection, for which target resources are allocated and into which an initiator is permitted to signal ORBs. A queue remains active from its creation until it is shut down.

3.1.2.3 blocking connection: A bi-directional connection that utilizes a single queue for data transfer both to and from a logical unit. Because execution of transport flow ORBs within a queue is ordered, an uncompleted ORB for data transfer in one direction may block the execution of an ORB for data transfer in the other direction.

3.1.2.4 byte: Eight bits of data.

3.1.2.5 connection: A queue or a pair of queue(s) that affords access to a service. A connection may be uni-directional or bi-directional; in the latter case, a connection may be blocking or nonblocking. Two queues are necessary to implement a bi-directional, nonblocking connection.

3.1.2.6 control information: Information exchanged between an initiator and a logical unit whose format is defined by this standard. The format of control information is independent of the format of application data exchanged by client applications and services—but both control information and application data are transported by the same methods.

3.1.2.7 control ORB: A transport flow ORB whose *queue* field is zero and whose *end_of_message* and *notify* bits are one; it is used to transfer request or response control information between an initiator and a logical unit.

3.1.2.8 execution context: Information maintained by a logical unit for active ORBs so that data transfer may be resumed after a task set abort without loss of data or interruption perceivable to PPDT application clients or services.

3.1.2.9 final ORB: A transport flow ORB whose *final* and *notify* bits are both one. An initiator uses a final ORB to indicate to a logical unit that no subsequent operation request blocks with the same *queue* value will be signaled unless the queue number is reissued by the logical unit in a future CONNECT control request or response.

3.1.2.10 initiator: A node that originates SBP-2 management requests, control and transport flow ORBs and signals them to a logical unit for processing.

3.1.2.11 logical unit: The part of a target that provides access to one or more services. Targets compliant with this standard implement one logical unit with a LUN of zero.

3.1.2.12 login: The process by which an initiator obtains access to a logical unit fetch agent. The fetch agent and its CSRs provide a mechanism for an initiator to signal ORBs to the logical unit.

3.1.2.13 logout: The process that permits an initiator to relinquish its use of a logical unit fetch agent and all the resources associated with the login.

3.1.2.14 management service: A service provided by a peer unit to execute control requests that establish or terminate connections to other services of the peer unit. The connection to this service is implicitly established as the result of an SBP-2 login.

3.1.2.15 node: An addressable device attached to Serial Bus.

3.1.2.16 nonblocking connection: A bi-directional connection that utilizes two queues for data transfer to and from a logical unit, one for each direction. Since the execution of transport flow ORBs in each queue is unordered with respect to the other, an uncompleted ORB in one queue should not block execution of an ORB in the other so long as at least one of the connection's task slots is reserved for the sole use of each queue.

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

3.1.2.18 operation request block: A data structure fetched from system memory by a logical unit in order to execute the request encapsulated within it.

3.1.2.19 peer unit: A unit compliant with the requirements of this standard, either as an initiator or target. A device may be solely an initiator or may implement both architectures.

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

3.1.2.21 queue: An ordered set of operation request blocks within a task set that does not block with respect to other queues that are part of the same task set.

3.1.2.22 receive: When any form of this verb is used in the context of Serial Bus primary packets, it indicates that the packet is made available to the transaction or application layers, i.e., layers above the link layer. Neither a packet repeated by the PHY nor a packet examined by the link is "received" by the node unless the preceding is also true.

3.1.2.23 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.

3.1.2.24 request subaction: A packet transmitted by a node (the requester) that communicates a transaction code and optional data to another node (the responder) or nodes.

3.1.2.25 response subaction: A packet transmitted by a node (the responder) that communicates a response code and optional data to another node (the requester). A response subaction may consist of either an acknowledge packet or a response packet.

3.1.2.26 service: An independently operable component of a peer unit that processes requests and associated data from clients in other peer units.

3.1.2.27 service data unit: A set of data whose semantics are preserved when transferred between peers at the application layer (clients and services); it is not interpreted by the supporting transport layer (PPDT).

3.1.2.28 split transaction: A transaction that consists of a request subaction followed by a separate response subaction. Subactions are considered separate if ownership of the bus is relinquished between the two.

3.1.2.29 status block: A data structure that may be written to an initiator's *status_FIFO* by a logical unit when an operation request block has been completed.

3.1.2.30 store: When any form of this verb is used in the context of data transferred by the target to the system memory of either an initiator or other device, it indicates both the use of Serial Bus write request subaction(s), quadlet or block, to place the data in system memory and the corresponding response subaction(s) that complete the write(s).

3.1.2.31 subsequent ORB: ORB_Q is subsequent to ORB_P if it is signaled to a logical unit fetch agent after ORB_P , unless ORB_Q existed in the logical unit task set at the time ORB_P was first signaled.

3.1.2.32 system memory: The portions of any node's memory 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 might make system memory addressable from Serial Bus, but any node, including those usually thought of as peripheral devices, may have system memory.

3.1.2.33 target: A unit, compliant with SBP-2, that fetches SBP-2 management requests, control and transport flow ORBs from an initiator.

3.1.2.34 task: A task is an organizing concept that represents the work to be done by a logical unit to carry out a command encapsulated by an ORB. In order to perform a task, a logical unit 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 entered into the logical unit's task set, proceeds through a period of execution by the logical unit, and finishes either when completion status is stored at the initiator or when completion may be deduced from other information. While a task is active, it makes use of both target resources and initiator resources.

3.1.2.35 task set: A group of tasks available for execution by a logical unit. SBP-2 specifies some dependencies between individual tasks within the task set and this standard mandates others.

3.1.2.36 transaction: A Serial Bus request subaction and the corresponding response subaction. The request subaction transmits a transaction code (such as quadlet read, block write or lock); some request subactions include data as well as transaction codes. The response subaction is null for transactions with broadcast destination addresses or broadcast transaction codes; otherwise it returns completion status and possibly data.

3.1.2.37 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. A node may have multiple units, which normally operate independently of each other.

3.1.2.38 unit architecture: The specification of the interface to and the services provided by a unit implemented within a Serial Bus node. This standard extends the unit architecture defined by SBP-2 to include mechanisms for peer-to-peer data transport.

3.1.2.39 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.

3.1.2.40 unsolicited status block: A status block whose *src* field is two; the meaning of the *ORB_offset_hi* and *ORB_offset_lo* fields is unspecified and the status block does not pertain to any particular ORB.

3.1.2.41 working set: The part of a task set that has been fetched from the initiator by the logical unit and is available to the logical unit in its local storage.

3.2 Abbreviations

The following abbreviations are used in this standard:

CSR	control and status register
CRC	cyclical redundancy checksum
EUI-64	Extended Unique Identifier, 64-bits
LUN	logical unit number
ORB	operation request block
SDU	service data unit
SBP-2	Serial Bus Protocol 2 (see ANSI INCITS 325-1998)

3.3 Notation

The following conventions in 3.3.1 and 3.3.2 should be understood by the reader in order to comprehend this standard.

3.3.1 Numeric values

Decimal and hexadecimal 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, which are also used when the value represented has an underlying structure that is more apparent in a hexadecimal format than in a decimal format.

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. When the subscript is unnecessary to disambiguate the base of the number it may be omitted. For the sake of legibility, hexadecimal numbers are separated into groups of four digits separated by spaces.

As an example, 42 and 2A₁₆ both represent the same numeric value.

3.3.2 Bit, byte and quadlet ordering

Devices compliant with this standard use the facilities of Serial Bus—IEEE Std 1394-1995 as amended by IEEE Std 1394a-2000—therefore this standard 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 (from the perspective of Serial Bus) and not their physically addressed position (from the viewpoint of the node’s memory bus).

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 in Figure 1. The significance of the interior bits uniformly decreases in progression from *msb* to *lsb*.⁵

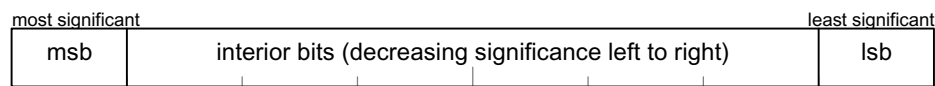


Figure 1—Bit ordering within a byte

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 in Figure 2.

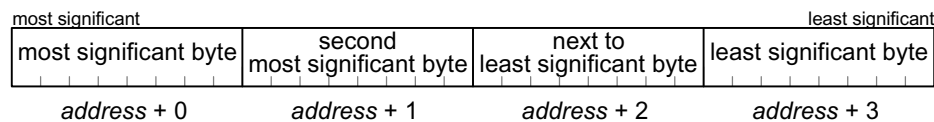


Figure 2—Byte ordering within a quadlet

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 Figure 3 indicates.

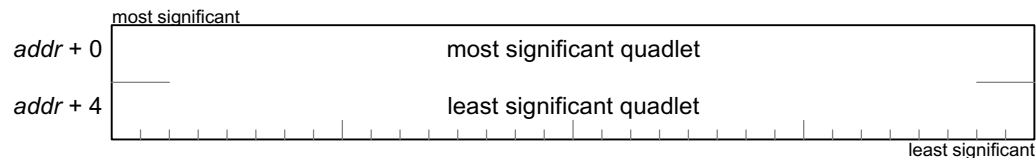


Figure 3—Quadlet ordering within an octlet

When block transfers take place that are neither quadlet aligned nor an integral number of quadlets, no assumptions shall 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 system memory.

⁵The format of the names *msb* and *lsb* reflects a convention: formal names of fields within data structures are italicized.

4. Model (informative)

This clause is informative.

This clause describes devices that conform to this standard and its normative references. It is intended to enhance the usefulness of the normative clauses of this standard. In addition to the information in this clause, users of this standard should also be familiar with the CSR architecture, Serial Bus standards, and SBP-2.

Examples of devices that come within the scope of this standard include (but are not limited to) copiers, printers, facsimile machines, scanners, and multifunction peripherals (MFPs) that combine two or more of these capabilities. These devices are characterized by high-volume transfers of application data; modest amounts of control information may be communicated in parallel with the application data transfers. These devices are used with diverse operating systems and application protocols; consequently any standard for their use with Serial Bus needs to hide many of the transport protocol details from the user applications. For example, a print driver that supports Postscript data formats should not be concerned with how data and control information are transported between it and the printer. This standard resolves those concerns.

4.1 Protocol stack and service model

The relationship between the initiator and target may be modeled as a software stack present in both devices, as shown by Figure 4 and Figure 5. The physical interconnection, via Serial Bus, exists at the lowest protocol level. Logical connections (shown by dashed lines) exist at the other protocol levels: an SBP-2 login between the initiator and target multiplexes queues (defined by this standard) that, in turn, support end-to-end connections (also defined by this standard) between client applications and services. This standard defines the data structures and methods necessary to implement the shaded levels in the protocol stacks, a peer-to-peer data transport (PPDT) based upon SBP-2. Note that client application(s) may reside at either the initiator or the target (they are commonly found at the initiator) and the service(s) at the complementary SBP-2 functional role, target or initiator.

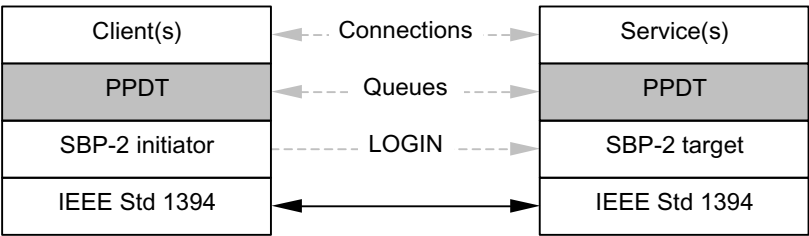


Figure 4—Protocol stack (service at target)

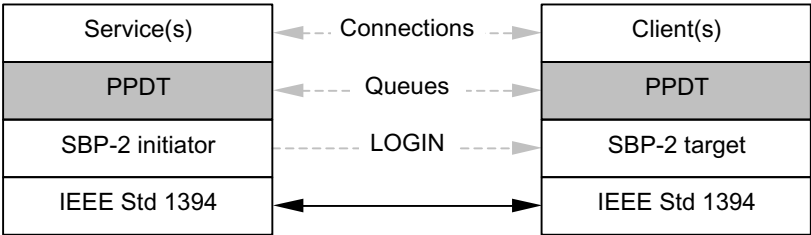


Figure 5—Protocol stack (service at initiator)

In order for the application(s) and service(s) to communicate in a peer-to-peer, transport-independent manner, this standard defines how SBP-2 may be used to implement uni- and bi-directional transport flows for both control information and application data. Key concepts introduced below are used to explain the details of the transport flow model:

- *peer unit*: A unit compliant with the requirements of this standard, either as an initiator or target. A device may implement both initiator and target unit architectures.
- *service*: An independently operable component of a peer unit that processes requests and associated data from clients in other peer units.
- *management service*: A service provided by a peer unit to execute control requests that establish or terminate connections to the other services of the peer unit. The connection to this service is implicitly established as the result of an SBP-2 login.
- *queue*: An ordered set of ORBs within a task set that does not block with respect to other queues that are part of the same task set.
- *connection*: A queue or a pair of queues that affords access to a service. A connection may be unidirectional or bi-directional; in the latter case, a connection may be blocking or nonblocking. Two queues are necessary to implement a bi-directional, nonblocking connection.

4.2 Independent data paths for each service

SBP-2 describes all the work to be performed by a particular logical unit as a task set, a collection of ORBs linked together as shown by Figure 6. Note that the queue labels shown for each ORB are not a feature of SBP-2; except for the distinction between ordered and unordered task sets, SBP-2 recognizes no relationship between tasks in a task set.

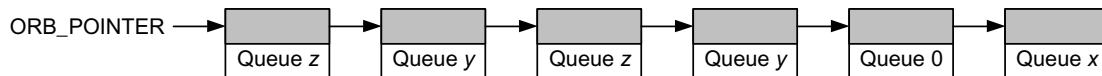


Figure 6—An SBP-2 task set

Because a single peer unit may implement one or more *services* (independently operable components of a peer unit), each of which may require an independent uni- or bi-directional transport flow, this standard augments SBP-2 to permit multiplexed *queues* within a single task set, as illustrated by Figure 7. A queue is a logical construct within a task set; each ORB in the task set is labeled to identify the logical queue to which it belongs. Queues operate independently of each other. Although a logical unit may in general reorder the execution of ORBs within the task set, all of the ORBs within a particular queue are executed in order. Within this framework, both the initiator and the logical unit manage the single task set illustrated in Figure 6 as the collection of logically independent queues illustrated in Figure 7. The dashed lines connecting ORBs represent the logical ordering of ORBs within each queue, not the actual pointers that link ORBs in the task set.

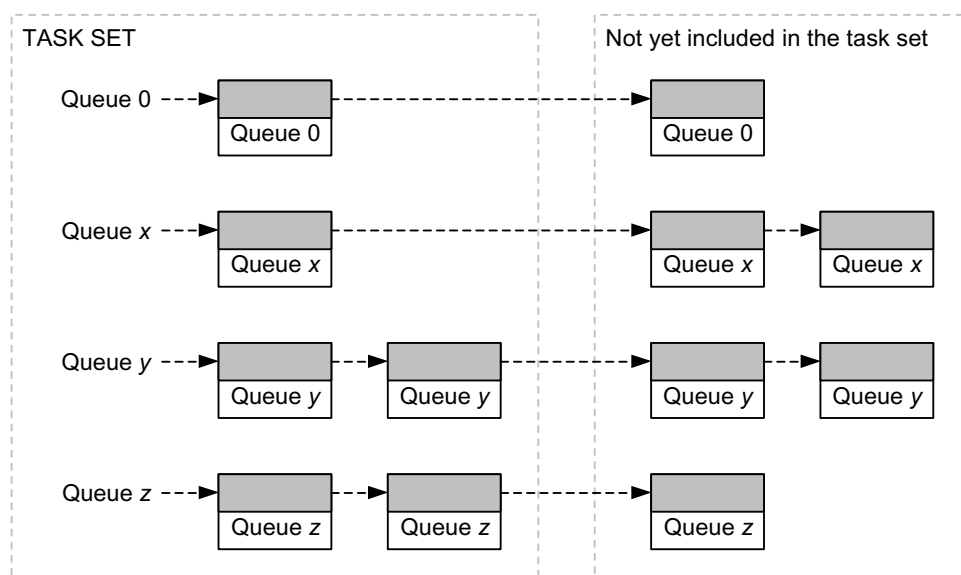


Figure 7—Independent queues (logical model)

In theory the size of an SBP-2 task set is bounded only by the amount of memory available to the initiator to store ORBs; in practice logical units have sufficient memory to fetch only a subset of each task set, the working set. Nonblocking behavior between the separate queues is achieved by restricting the size of the task set to that of each logical unit's working set. If the initiator never places more ORBs in a task set than the logical unit accommodates in its working set, all outstanding ORBs may be fetched by the logical unit and made available for execution. The initiator restricts the number of outstanding ORBs on a queue by queue basis so that a task slot in each working set is always available for each queue. Since the client application or service may initiate more data transfer requests than can be simultaneously active in a task set, the initiator maintains the ORBs (segregated by queue number) outside of the task set and signals them to the logical unit as task slots become available.

Because queues do not block with respect to each other, nonblocking bi-directional data transfer between initiator and target may be accomplished through the use of two queues, one for each direction.

4.3 Connection management

The multiplexed queue management scheme just described requires the allocation of target resources (queue numbers and task slots) before it may be used. Collectively these resources constitute a *connection* between a client and a service. This standard defines methods by which connection(s) are established and subsequently terminated and their resources freed.

Connections may be established by either an initiator or a target logical unit. Because of asymmetries in SBP-2, the connection parameters differ dependent upon the source of the connection request—but at the transport-independent level perceived by clients and services the connection mechanisms are peer-to-peer and symmetric. When a client wishes to establish a connection with a particular service in the other device, it provides a *service ID*, a unique string that specifies the desired service. Service IDs are maintained in a separate registry and are assumed by this standard to be well-known identifiers. The parameters of a particular service (e.g., the number of queues, their characteristics and the format of data exchanged between client and service) are fixed *a priori* and beyond the scope of this standard. If the specified service exists in the other device (along with sufficient resources for the connection), the connection is created and subsequently identified by the queue number(s) assigned to the connection.

Connections may be one of three different types:

- Unidirectional; the application data flow is one direction, either from the initiator to the target or vice versa;
- Bi-directional (nonblocking); the application data flows in both directions with a different queue used for each of the directions; or
- Bi-directional (blocking); the data flows in both directions via a single queue which has the potential to block. Nonblocking behavior is not guaranteed by the transport but should be a property of the application itself. The queue used by the management service is an example of a bi-directional blocking queue, but because both initiator and target logical unit restrict their usage such that only one request is outstanding until its corresponding response is transferred, the queue should not block.

Once a connection is established it persists across bus reset(s) until explicitly terminated or abandoned as a consequence of a logout.

Just as either initiator or target may establish a connection, either may terminate the connection regardless of which one created the connection. A disconnect may be synchronized with the transport flow in order to gracefully end the connection or it may preempt the transport flow if necessary. Once the disconnect is complete, the target resources (queue numbers and task slots) are available for reuse.

4.4 Data transfer between initiator and target

Once a connection is established, application data may be transferred between initiator and target. SBP-2 transport flow ORBs are used to regulate the data transfer: each ORB specifies the direction (from the target or to the target) and provides a buffer in system memory that is either the source or destination for the data. The target initiates all data transfer requests, which permits it to pace the data transfer rate according to the availability of its own resources. The initiator, on the other hand, makes all of the data (or a buffer to accommodate all of the data) accessible the whole time the ORB is active.

Data transfer from the initiator to the target is straightforward: the initiator signals an ORB with the appropriate *direction* bit to the target and the target issues read requests to access the data. Data transfer in the opposite direction, from the target to the initiator, is more complex. Because the target cannot signal an ORB to the initiator, it indicates to the initiator that data is available. Unless the initiator had anticipated data transfer from the target and already signaled an ORB, the initiator signals an ORB with an empty buffer to receive the data and the target issues write requests to store the data.

In both cases, ORB completion is indicated when the target stores a status block at the initiator. The status block specifies a residual count that indicates the quantity of data transferred. The status block also includes codes that describe successful or error completion of the data transfer described by the ORB. The status block does not indicate whether or not the data was successfully utilized by an application client or service—only whether or not the data was transferred across Serial Bus.

Data transfer between initiator and target is modeled either as a datagram or as a stream. When datagram mode is used, each service data unit (SDU) fits within a single buffer described by an ORB. If no SDU is available for transfer, the ORB may block and not be completed until an entire SDU is ready. Or, if the recipient is unable to accept the SDU (too small a buffer), no data is transferred and an error results. In contrast, stream mode permits data to flow as it becomes available. An ORB that transfers data to a target in stream mode cannot fail because the target's buffer is too small while an ORB that transfers stream data from a target may complete as soon as the first byte of data is available.⁶ In both cases—datagram and stream—circumstances may arise where ORBs are completed even though no data has been transferred.

⁶Applications may defer completion of an ORB until some minimum amount of data has been transferred. This “watermark” capability may improve performance.

4.5 Control requests and responses

In order to coordinate the flow of application data between initiator and target, a set of control operations are defined. These operations take the form of a control request and a corresponding response; their functions include establishment of a connection, notification that target data is available for transfer to the initiator and confirmed release of queue resources upon disconnection. Either initiator or target may originate a control request; neither should initiate a subsequent control request until a control response is returned for an outstanding request.

Control requests and responses are not encoded within ORBs themselves but are contained within data transferred between initiator and target buffers. ORBs that mediate the transfer of control information are no different from those used for application data except that they specify predefined queue zero. This bi-directional, blocking queue is reserved solely for control requests and responses and is not available for the transfer of application data between initiator and target. Queue zero (the control queue) is automatically allocated as a result of successful login and cannot be released by either initiator or target until logout.

The operational sequences for queue zero differ dependent upon the originator of the control request, initiator or target, as illustrated Figure 8.

When the initiator originates a control request, it signals an ORB to a logical unit that describes a buffer in system memory that contains the control request. The initiator subsequently signals another ORB to the logical unit that describes another buffer in system memory available to receive the response. For the sake of efficiency, both ORBs may be linked and signaled to the logical unit at the same time as illustrated by Figure 8.

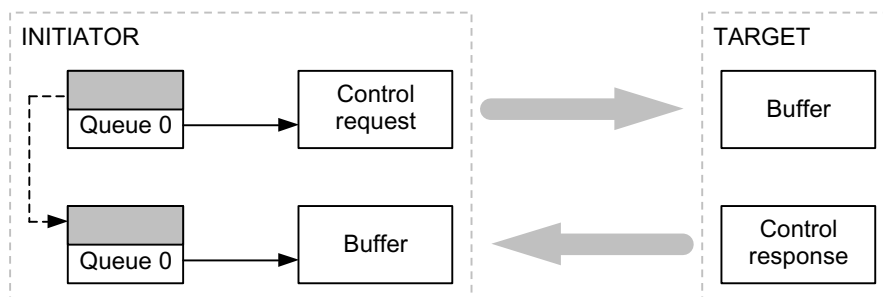


Figure 8—Control request originated by initiator

In Figure 8, a logical unit fetches the control request from the initiator's buffer and then stores completion status at the initiator's *status_FIFO*. The control request itself is not necessarily executed at this point but it has been securely read by the logical unit. After the request is executed, the logical unit stores a control response in the buffer provided by the second ORB and then stores completion status for that ORB at the initiator's *status_FIFO*. Successful receipt of the control response permits the initiator to examine the response data to determine whether or not the request succeeded.

In the case where a logical unit has a control request to make of the initiator, it first communicates to the initiator that a queue zero ORB is needed. This attention condition may be communicated by any status block, either one associated with the completion of application data transfer on a queue other than queue zero, a status block for a completed queue zero transfer or an unsolicited status transfer. The result is the same no matter what the source of the status block; this is illustrated by the upper portion of Figure 9.