

INTERNATIONAL STANDARD

ISO/IEC 14776-232

First edition
2001-11

Information technology – Small computer system interface (SCSI) – Part 232: Serial Bus Protocol 2 (SBP-2)



Reference number
ISO/IEC 14776-232:2001(E)

INTERNATIONAL STANDARD

ISO/IEC 14776-232

First edition
2001-11

Information technology – Small computer system interface (SCSI) – Part 232: Serial Bus Protocol 2 (SBP-2)

© ISO/IEC 2001

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland



PRICE CODE

X

For price, see current catalogue

CONTENTS

FOREWORD.....	6
1 Scope and object.....	7
1.1 Scope.....	7
1.2 Object	7
2 Normative references	8
3 Definitions and notation	9
3.1 Definitions	9
3.1.1 Conformance.....	9
3.1.2 Glossary.....	9
3.1.3 Abbreviations	12
3.2 Notation	12
3.2.1 Numeric values	12
3.2.2 Bit, byte and quadlet ordering.....	12
3.2.3 Register specifications	13
3.2.4 State machines	15
4 Model (informative).....	17
4.1 Unit architecture	17
4.2 Logical units	17
4.3 Requests and responses	17
4.4 Data buffers	18
4.5 Target agents	20
4.6 Ordered and unordered execution	21
5 Data structures.....	22
5.1 Operation request blocks (ORBs).....	23
5.1.1 Dummy ORB	24
5.1.2 Command block ORBs	24
5.1.3 Management ORBs	26
5.2 Page tables.....	33
5.2.1 Unrestricted page tables.....	34
5.2.2 Normalized page tables.....	34
5.3 Status block.....	35
5.3.1 Request status	37
5.3.2 Unsolicited device status.....	39
6 Control and status registers.....	40
6.1 Core registers	40
6.2 Serial Bus-dependent registers.....	40
6.3 MANAGEMENT_AGENT register.....	41
6.4 Command block agent registers	41
6.4.1 AGENT_STATE register	42
6.4.2 AGENT_RESET register.....	43
6.4.3 ORB_POINTER register	43
6.4.4 DOORBELL register	44
6.4.5 UNSOLICITED_STATUS_ENABLE register.....	45

7	Configuration ROM	46
7.1	Power reset initialization	46
7.2	Bus information block	47
7.3	Root directory.....	48
7.3.1	Module_Vendor_ID entry.....	48
7.3.2	Node_Capabilities entry	48
7.3.3	Unit_Directory entry	48
7.4	Unit directory.....	49
7.4.1	Unit_Spec_ID entry.	49
7.4.2	Unit_SW_Version entry	49
7.4.3	Command_Set_Spec_ID entry.....	50
7.4.4	Command_Set entry.....	50
7.4.5	Command_Set_Revision entry.....	50
7.4.6	Firmware_Revision entry.....	51
7.4.7	Management_Agent entry.....	51
7.4.8	Unit_Characteristics entry	51
7.4.9	Reconnect_Timeout entry.....	52
7.4.10	Logical_Unit_Directory entry.....	52
7.4.11	Logical_Unit_Number entry.....	52
7.4.12	Unit_Unique_ID entry	53
7.5	Logical unit directory	53
7.5.1	Command_Set_Spec_ID entry.....	54
7.5.2	Command_Set entry.....	54
7.5.3	Command_Set_Revision entry.....	54
7.5.4	Logical_Unit_Number entry	54
7.6	Unit unique ID leaf.....	54
8	Access	56
8.1	Access protocols	56
8.2	Login.....	56
8.3	Reconnection	57
8.4	Logout.....	58
9	Command execution	59
9.1	Requests and request lists	59
9.1.1	Fetch agent initialization (informative)	59
9.1.2	Dynamic appends to request lists (informative)	60
9.1.3	Fetch agent use by the BIOS (informative)	61
9.1.4	Fetch agent state machine	61
9.2	Data transfer	64
9.3	Completion status.....	64
9.4	Unsolicited status	65

10 Task management	66
10.1 Task sets	66
10.2 Basic task management model	66
10.3 Error conditions	67
10.4 Task management requests.....	67
10.4.1 Abort task.....	67
10.4.2 Abort task set	68
10.4.3 Logical unit reset	69
10.4.4 Target reset.....	69
10.5 Task management event matrix.....	70
 Annex A (normative) Minimum Serial Bus node capabilities	71
Annex B (normative) SCSI command and status encapsulation.....	73
Annex C (normative) Security extensions.....	78
Annex D (informative) Sample configuration ROM.....	81
Annex E (informative) Serial Bus transaction error recovery.....	85
Annex F (informative) SCSI Architecture Model conformance	87
 Figure 1 – Bit ordering within a byte.....	13
Figure 2 – Byte ordering within a quadlet	13
Figure 3 – Quadlet ordering within an octlet.....	13
Figure 4 – CSR specification example.....	14
Figure 5 – State machine example	16
Figure 6 – Linked list of ORBs	18
Figure 7 – Directly addressed data buffer	19
Figure 8 – Indirectly addressed data buffer (via page table)	19
Figure 9 – Address pointer.....	22
Figure 10 – ORB pointer	22
Figure 11 – ORB family tree.....	23
Figure 12 – ORB format	23
Figure 13 – Dummy ORB	24
Figure 14 – Command block ORB	25
Figure 15 – Management ORB.....	27
Figure 16 – Login ORB	28
Figure 17 – Login response	29
Figure 18 – Query logins ORB	30
Figure 19 – Query logins response format.....	30
Figure 20 – Reconnect ORB	31
Figure 21 – Logout ORB	32
Figure 22 – Task management ORB	33
Figure 23 – Page table element (unrestricted page table)	34

Figure 24 – Page table element (when page_size equals four).....	34
Figure 25 – Status block format	36
Figure 26 – TRANSPORT FAILURE format for sbp_status	38
Figure 27 – MANAGEMENT_AGENT format	41
Figure 28 – AGENT_STATE format.....	42
Figure 29 – AGENT_RESET format	43
Figure 30 – ORB_POINTER format.....	44
Figure 31 – DOORBELL format.....	45
Figure 32 – UNSOLICITED_STATUS_ENABLE format.....	45
Figure 33 – Configuration ROM hierarchy	46
Figure 34 – Bus information block format	47
Figure 35 – Module_Vendor_ID entry format.....	48
Figure 36 – Node_Capabilities entry format	48
Figure 37 – Unit_Directory entry format	49
Figure 38 – Unit_Spec_ID entry format	49
Figure 39 – Unit_SW_Version entry format	49
Figure 40 – Command_Set_Spec_ID entry format.....	50
Figure 41 – Command_Set entry format.....	50
Figure 42 – Command_Set_Revision entry format.....	50
Figure 43 – Firmware_Revision entry format.....	51
Figure 44 – Management_Agent entry format.....	51
Figure 45 – Unit_Characteristics entry format	51
Figure 46 – Reconnect_Timeout entry format.....	52
Figure 47 – Logical_Unit_Directory entry format	52
Figure 48 – Logical_Unit_Number entry format	52
Figure 49 – Unit_Unique_ID entry format	53
Figure 50 – Unit unique ID leaf format.....	54
Figure 51 – Fetch agent initialization with a dummy ORB.....	60
Figure 52 – Fetch agent state machine	62
Figure B.1 – SCSI command block ORB	73
Figure B.2 – SCSI control byte.....	73
Figure B.3 – Status block format for SCSI sense data.....	74
Figure C.1 – Set password ORB	79
Figure D.1 – Basic configuration ROM	81
Figure D.2 – SCSI configuration ROM.....	83
Table 1 – Data transfer speeds	26
Table 2 – Management request functions.....	27
Table F.1 – SAM-2 Service responses	89

INFORMATION TECHNOLOGY – SMALL COMPUTER SYSTEM INTERFACE (SCSI) –

Part 232: Serial bus protocol 2 (SBP-2)

FOREWORD

- 1) ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.
- 2) In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.
- 3) Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 14776-232 was prepared by subcommittee 25: Interconnection of information technology equipment, of ISO/IEC joint technical committee 1: Information technology.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 3.

Annexes A, B and C form an integral part of this International Standard.
Annexes D, E and F are for information only.

INFORMATION TECHNOLOGY – SMALL COMPUTER SYSTEM INTERFACE (SCSI) –

Part 232: Serial bus protocol 2 (SBP-2)

1 Scope and object

1.1 Scope

This part of ISO/IEC 14776 defines a protocol for the transport of commands and data over High Performance Serial Bus. The transport protocol, Serial Bus Protocol 2 or SBP-2, requires implementations to conform to the requirements of this standard as well as to ISO/IEC 13213:1994 and permits the exchange of commands, data and status between initiators and targets connected to Serial Bus.

1.2 Object

Original development work for Serial Bus Protocol (SBP) was initiated out of a desire to adapt SCSI capabilities and facilities to a particular serial environment IEEE 1394. 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, it became evident that 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. The enhanced goals set for the design of SBP-2 are ranked below:

- 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 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;
- 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 1394, it is believed that it will be appropriate for use with future extensions to Serial Bus as they are standardized.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 9899:1999, *Programming Languages – C*

ISO/IEC 13213:1994, *Information technology – Microprocessor systems – Control and Status Register (CSR) Architecture for Microcomputer Buses*

ANSI/IEEE 1394:1995, *IEEE Standard for High Performance Serial Bus*

IEEE P1394a, *Draft Standard for High Performance Serial Bus (Supplement)*¹⁾

BSR X3 PN 1157-D, *Information technology – SCSI Architecture Model 2 (SAM-2)*²⁾

BSR NCITS PN 1236-D, *Information technology – SCSI Primary Commands 2 (SPC-2)*²⁾

1) Under development. Available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331

2) Under development. Available from the National Committee for Information Technology Standards, 1250 Eye Street, NW, Suite 200, Washington, DC 20005-3922

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 an object defined by this standard other than reserved 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 command block: Space reserved within an ORB to describe a command intended for a logical unit that controls device functions or the transfer of data to or from device medium. The format and meaning of command blocks are outside the scope of SBP-2 and are command set- or device-dependent.

3.1.2.3 device server: A component of a logical unit responsible to execute tasks initiated by command blocks that specify data transfer or other device operations.

3.1.2.4 initial node space: The 256 terabytes of Serial Bus address space that may be 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 or ANSI/IEEE 1394 for more information on address spaces.

3.1.2.5 initial register space: A two kilobyte portion of initial node space with a base address of FFFF F000 0000₁₆. Core registers defined by ISO/IEC 13213 are located within initial register space as are Serial Bus-dependent registers defined by ANSI/IEEE 1394.

3.1.2.6 initial units space: A portion of initial node space with a base address of FFFF F000 0800₁₆. This places initial units space adjacent to and above initial register space. The CSRs and other facilities defined by unit architectures are expected to lie within this space.

3.1.2.7 initiator: A node that originates device service or management requests and signals these requests to a target for processing.

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

3.1.2.9 logical unit: The part of the unit architecture that is an instance of a device model, e.g., disk, CD-ROM or printer. Targets implement one or more logical units; the device type of the logical units may differ.

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 signal ORBs to the target.

3.1.2.11 login ID: A value assigned by the target during the login process. The login ID establishes a relationship between an initiator and a task set. 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: An addressable device attached to Serial Bus.

3.1.2.13 node ID: The 16-bit node identifier defined by ANSI/IEEE 1394 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.

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

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

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

3.1.2.17 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.

3.1.2.18 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.

3.1.2.19 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.

3.1.2.20 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.

3.1.2.21 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.

3.1.2.22 status block: A data structure that may be written to system memory by a target when an operation request block has been completed.

3.1.2.23 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).

3.1.2.24 system memory: The portions of any node's memory that are directly addressable by a Serial Bus address and that 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.

3.1.2.25 target: A node that receives device service or management requests from an initiator. In the case of device service requests, the commands are directed to one of the target's logical units to be executed. Management requests are serviced by the target. A CSR Architecture unit is synonymous with a target.

3.1.2.26 task: A task is an organizing concept that represents the work to be done by a target to carry out a command encapsulated by an ORB. 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 entered into the target's task set, proceeds through a period of execution by the target 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.

3.1.2.27 task set: A group of tasks available for execution by a logical unit of a target. This standard specifies some dependencies between individual tasks within the task set but there may be others not specified by this standard.

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

3.1.2.29 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.

3.1.2.30 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. Within this standard, a unit is equivalent to a target.

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

3.1.2.32 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.

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

3.1.3 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
SAM-2	SCSI Architecture Model 2
SBP-2	Serial Bus Protocol 2 (this standard itself)
SPC-2	SCSI Primary Commands 2

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

As an example, 42, 2A₁₆ and 0010 1010₂ 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, ANSI/IEEE 1394, 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 signifi-

cance 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*, see Figure 1.

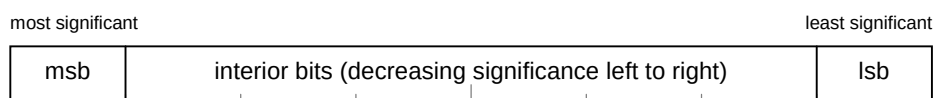


Figure 1 – Bit ordering within a byte

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, see Figure 2.

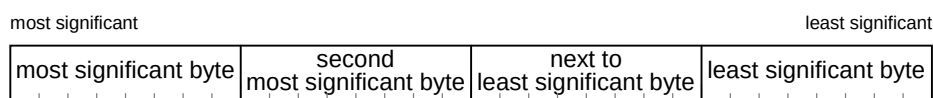


Figure 2 – Byte ordering within a quadlet

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, see Figure 3.

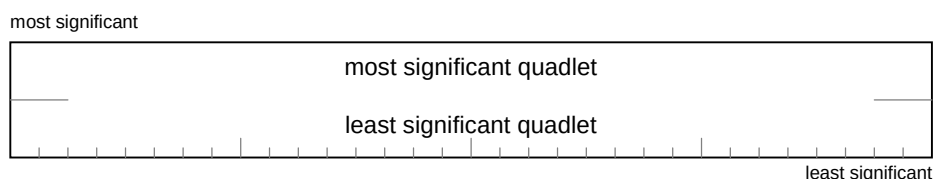


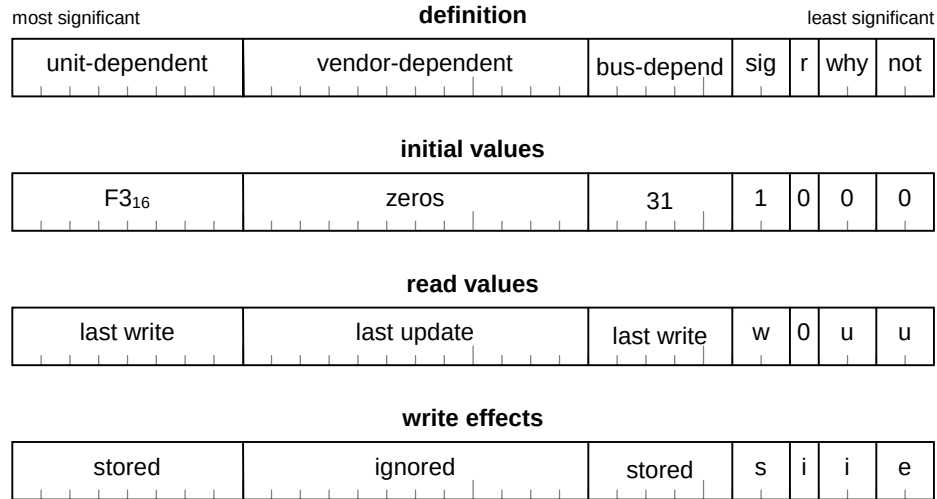
Figure 3 – Quadlet ordering within an octlet

When block transfers take place that are not quadlet aligned or not an integral number of quadlets, 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 defines the format and function of control and status registers, CSRs. 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 define CSRs, their bit fields, their initial values and the effects of read, write or other transactions, the format illustrated by Figure 4 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 field names have defined meanings.

Name	Abbreviation	Definition
bus-dependent	bus-depend	The meaning of the field is defined by the bus standard, in this case ANSI/IEEE Std 1394
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 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

CSRs 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 are 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. ⁴
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 are specified by the terms below.

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. CSRs 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 request.

3.2.4 State machines

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

⁴⁾ 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.

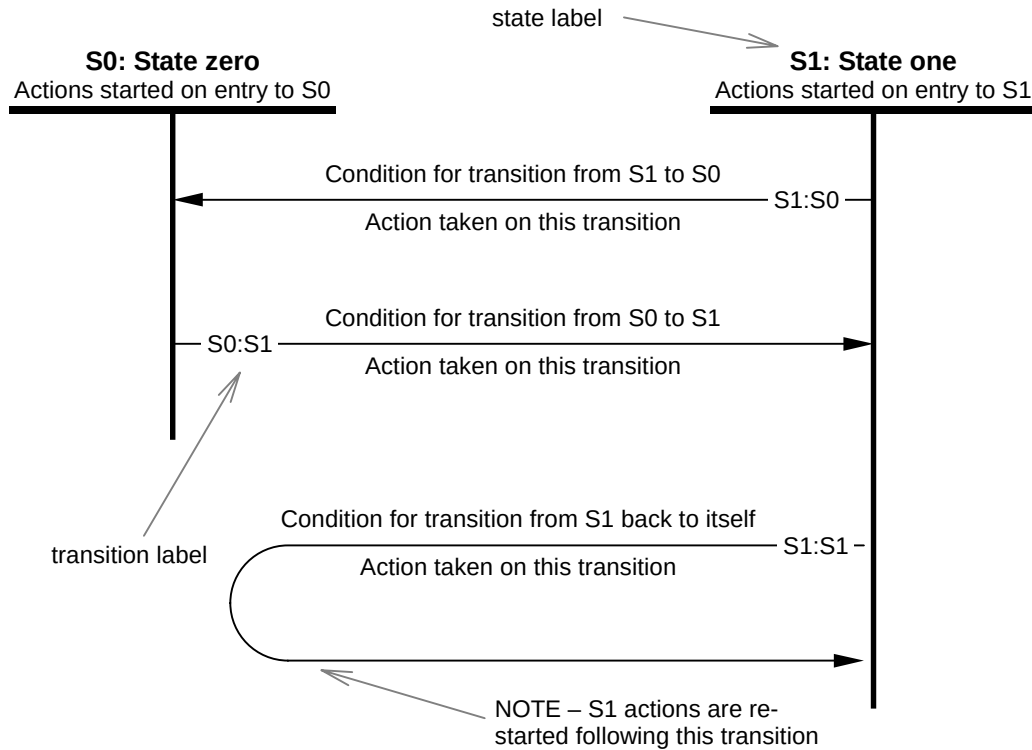


Figure 5 – State machine example

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)

This clause is informative and describes typical components and operation 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.

Serial Bus Protocol 2 (SBP-2) is a transport protocol defined for ANSI/IEEE 1394. It defines facilities for requests (commands) 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 associated with the commands. An SBP-2 device may assume the roles of initiator or target, either simultaneously or in succession. Commands and status may be transferred between the initiator and the target; data moves between the target and another device, which may be either the initiator or some other device.

4.1 Unit architecture

In CSR architecture and Serial Bus terminology, targets implemented to this standard are units. A Serial Bus node that implements a target has 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., disk, CD-ROM or printer. A logical unit consists of one or more device server(s) responsible to execute control or data transfer commands, one or more task sets that hold commands available for execution by the device server(s) and a logical unit number that is unique within the domain of the target.

Targets implement at least one logical unit, addressable as logical unit number (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 logical unit(s) are visible to the initiator, either as described by configuration ROM or as discoverable by command set-dependent requests directed to the target.

NOTE The structure of configuration ROM entries in the unit and logical unit directories permits considerable latitude to implementers in the description of target(s). For example, a device that implements multiple functions or instances of a function may be described either by multiple unit directories, each with a single logical unit, or by one unit directory that includes multiple logical units—or any combination in between, see clause 7.

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 (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 acquire or release target resources or to manage task sets (management requests—which include login requests); or
- to transport commands (command block requests).

Login and other management requests are directed to agents that can service only a single request at a time; there is no way to group these requests into a linked list. The ORBs for command block requests provide a field that contains the address of another ORB or a null pointer. This permits these requests to be in a linked list, see Figure 6.

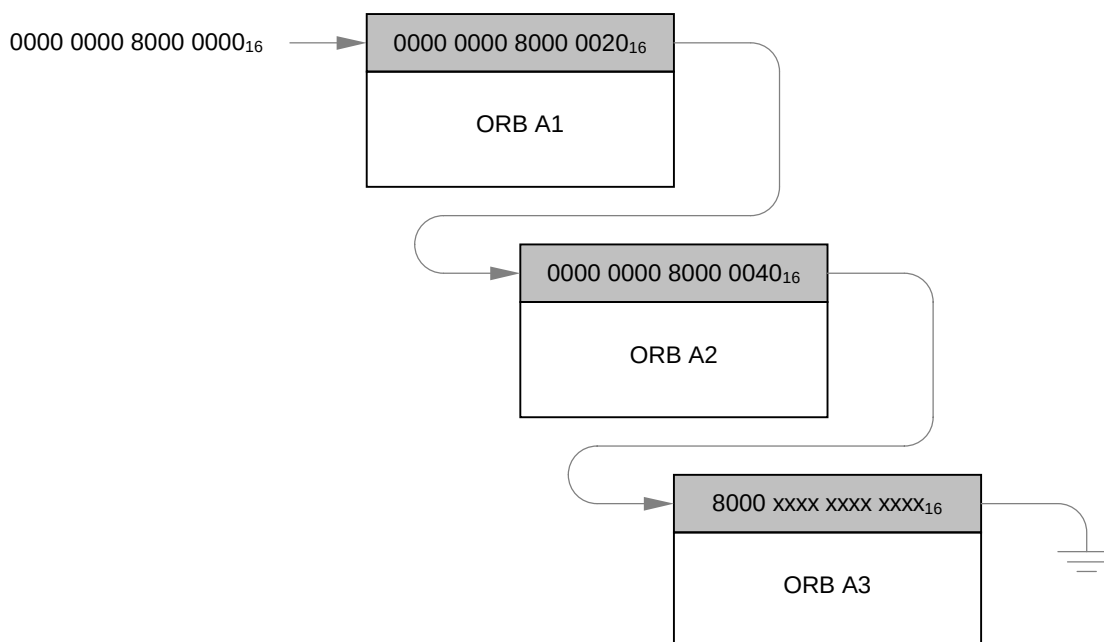


Figure 6 – Linked list of ORBs

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 may read ahead in the linked list; consequently the device server may reorder the execution of requests to improve performance.

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

4.4 Data buffers

The ORBs described in the preceding clause contain the device command and, for those commands which transfer data, the address of the data buffer for the command. The data buffer may be a single, contiguous buffer that is addressed directly by the ORB or it may be a collection of possibly disjoint segments that are addressed indirectly through a page table. The figures below illustrate both cases.

As an example, consider a command intended to transfer image data to a printer. If we assume that the image data is 3088₁₆ bytes long and the buffer starts at an address of 23 6174₁₆, the relationship between the ORB and the data buffer might appear as in Figure 7.

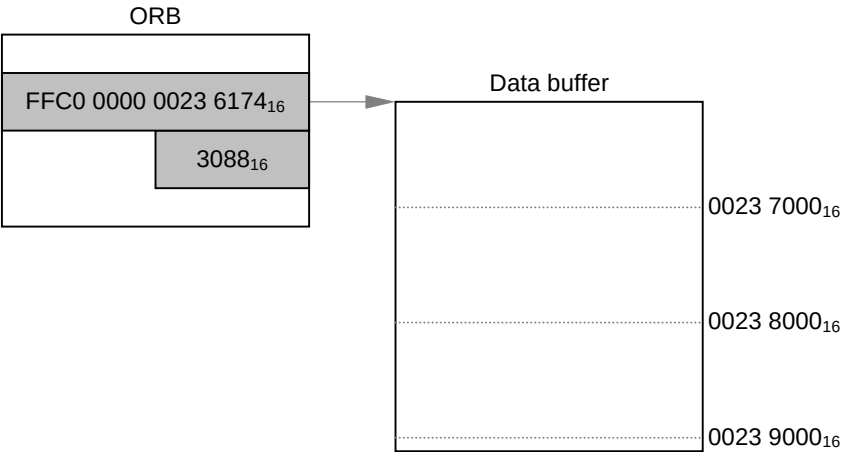


Figure 7 – Directly addressed data buffer

In the preceding example, two fields in the ORB specify the 64-bit address of the data buffer and its length, in bytes. The data buffer is shown with a node ID of FFC0₁₆, which is node zero on the local bus. The printer uses block read transactions to fetch data from the buffer before printing; the maximum size of the data payload for each request is controlled by a field in the ORB. The dotted lines within the data buffer indicate page boundaries. Although the data buffer is contiguous, the printer is not permitted to cross a page boundary in any one block read request.

When the data buffer consists of disjoint segments, it is necessary to indirectly address the data buffer through a page table, as shown in Figure 8. This figure could be an illustration of data read from a disk into various pages of an initiator's file system cache. In the example, assume that 2960₁₆ bytes of data are to be read from disk.

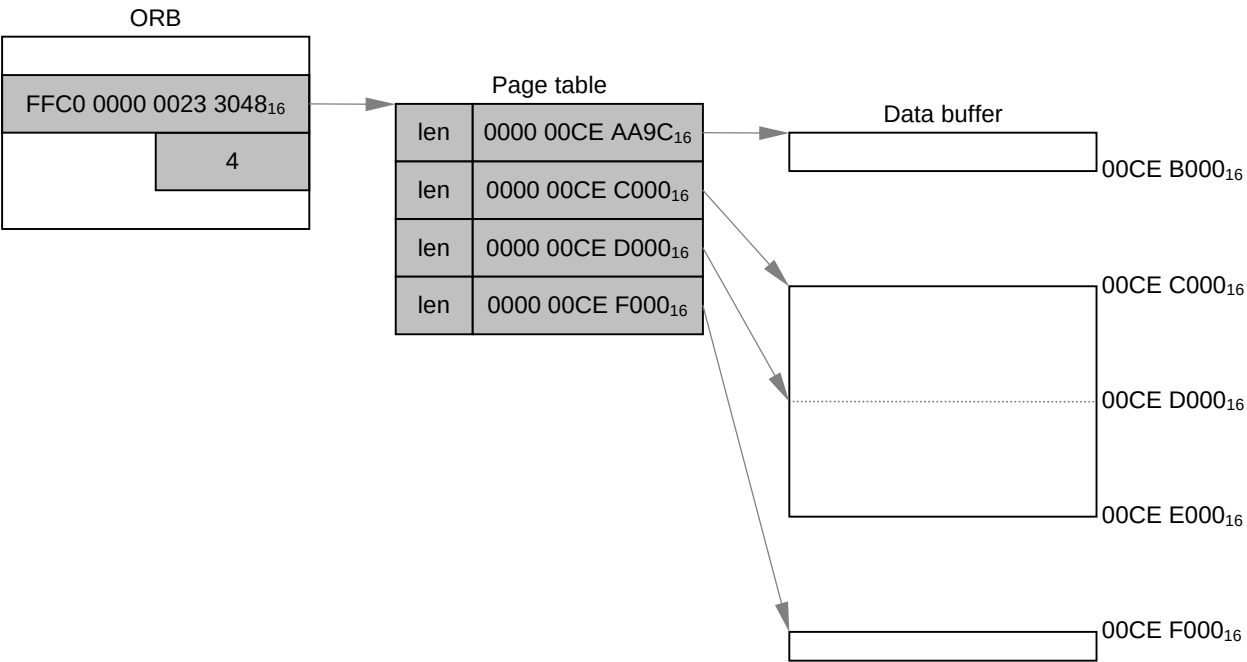


Figure 8 – Indirectly addressed data buffer (via page table)

The fields in the ORB that directly addressed a data buffer in the first example now point to a page table. Note that the ORB field that contains the data length when direct addressing is employed instead contains the number of elements in the page table—in this case, four. Each of the four page table elements points to the start of a segment of the data buffer. Each page table element also contains the length of the segment. The first segment ends on a page boundary, all other segments start on page boundaries (and the middle segments also end on page boundaries) while the last segment may end on any boundary. In this example, the segment lengths are 0564₁₆, 1000₁₆, 1000₁₆ and 03FC₁₆, respectively.

When a page table is used, both the page table and the data buffer it describes reside in the same node. The node ID of the page table, FFC0₁₆, is not repeated in the page table elements. The space that would have otherwise been occupied by the node ID instead is used to contain the length of each segment.

Another variant of page table format is permitted, called an unrestricted page table (or a scatter/gather list). In an unrestricted page table, data buffer segments may start on any boundary and may have arbitrary lengths: there is no underlying page size.

4.5 Target agents

A target agent is a facility that receives signals from the initiator that indicate the availability of requests. There are two 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. 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, rings a doorbell which causes the target agent to fetch the requests from system memory as target resources permit their execution.

Target agents that manage linked lists of requests utilize context maintained at both the initiator and target to fetch requests from memory. Once fetched, the request is locally available to the target for execution. The context consists of three elements:

- a linked list of ORBs at the initiator;
- a current ORB address at the target; and
- a doorbell at the target.

This standard defines procedures for both the initiator and the target that permits the addition of new requests to a linked list of ORBs while the target is actively fetching or executing previously queued requests. The procedures avoid the possibility of race conditions between the producer (initiator) and consumer (target) of the ORBs.

There are two types of target agents:

- management; and
- command block.

Management agents accept a variety of requests: login, task management and logout. Before making other requests, an initiator first completes a login via the management agent. Once this is done, the management agent will accept task management requests. 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.

A successful login returns the address of a command block agent. Command block agents service requests which are organized in linked lists. Individual linked list(s) are managed by (a) separate command block agent(s).

4.6 Ordered and unordered execution

Targets may implement either an ordered or unordered model of task execution. The ordered model is usually appropriate for devices where the context of a command affects its execution, i.e., the outcome of one command affects the subsequent command. A common example of a device with such command dependencies is a tape drive. The unordered model is usually appropriate for direct-access devices for which no positional or other context information is inherited from one command to the next.

The ordered model specifies both that tasks are executed in order and that completion status is returned in the same order. A consequence of ordering is that completion status for one task implicitly indicates successful completion status for all tasks that preceded it in the ordered list.

The unordered model permits the target to reorder active tasks without restriction. The actual execution sequence of tasks from any task set may bear no relationship to the order in which they were fetched. Unrestricted reordering leaves the responsibility for the assurance of data integrity with the initiator. If the integrity of data on the device medium could be compromised by unrestricted reordering involving a set of active tasks, $\{T_0, T_1, T_2, \dots, T_N\}$ and a new task T' , the initiator shall wait until $\{T_0, T_1, T_2, \dots, T_N\}$ 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 ORBs are signaled.

5 Data structures

There are three classes of data structures defined by this standard:

- operation request blocks (ORBs);
- page tables;
- status blocks.

These data structures may be allocated and initialized by an initiator in system memory at Serial Bus nodes. ORBs 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 64-bit address pointers within ORBs to conform to the format specified in Figure 9.

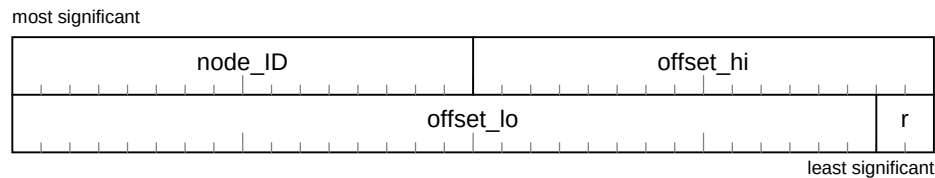


Figure 9 – Address pointer

The *node_ID* field shall identify the Serial Bus node for which the address pointer is valid, as defined by ANSI/IEEE 1394. 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.

The size of a data structure or buffer addressed by a pointer that conforms to Figure 9 is either explicitly specified by an associated length field or implicitly known from context. Whichever the case, the target shall not initiate any Serial Bus request subactions (read, write or lock) that reference system memory outside of the range determined by the address pointer and length.

ORBs shall be allocated at the initiator's node. Some types of ORBs contain a forward address pointer and may be organized as a linked list. Since the node ID is known for all ORBs in such a list, the address pointer format is redefined to reuse the *node_ID* field. An address pointer that references an ORB shall conform to the format in Figure 10.

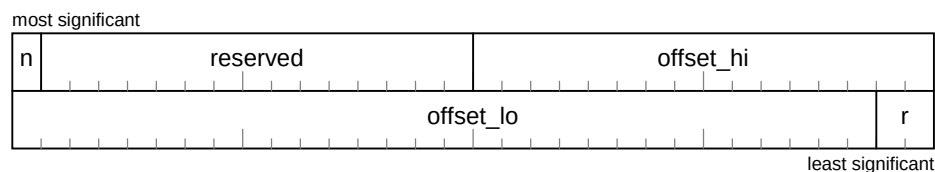


Figure 10 – ORB pointer

The *null* bit (abbreviated as *n* in Figure 10) indicates a null pointer when it is one. In this case the target shall ignore the *offset_hi* and the *offset_lo* fields.

5.1 Operation request blocks (ORBs)

All initiator requests for target actions are expressed within ORBs fetched by the target via Serial Bus read transaction(s). ORB formats vary according to use and may be viewed in hierarchical relationship to each other, as illustrated Figure 11.

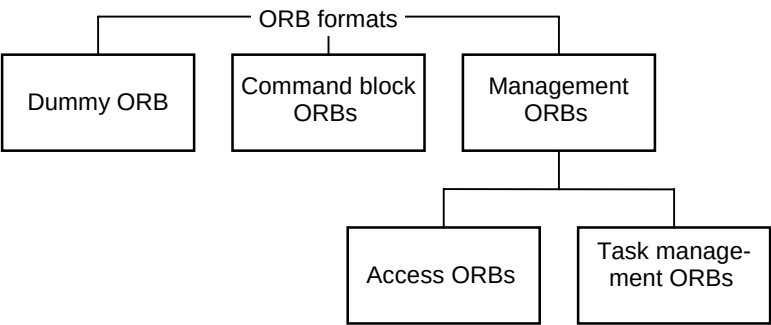


Figure 11 – ORB family tree

The formats of the ORBs are described in the subclauses that follow. This subclause specifies fields that are common to all ORBs, illustrated in Figure 12.

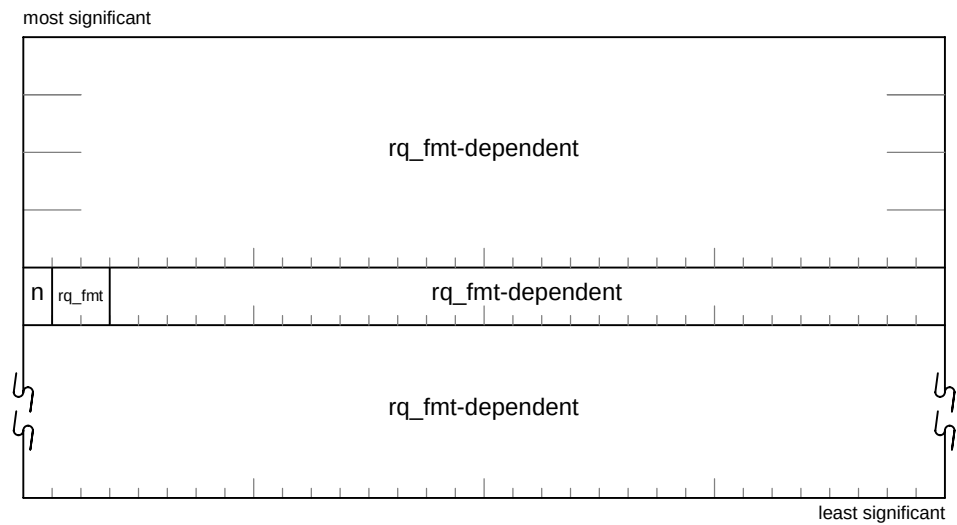


Figure 12 – ORB format

The *notify* bit (abbreviated as *n* in Figure 12) advises the target whether or not completion notification is required. When *notify* is zero, the target may elect to suppress completion notification except when there is an error, in which case the value of *notify* is ignored and a status block shall be stored. 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, at the address supplied in the login request.

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 target command sets and device types.

5.1.1 Dummy ORB

Dummy ORBs may be used as placeholders within linked lists of requests. An example is the use of a dummy ORB in the initialization of a target fetch agent (see 9.1.1). The initiator shall allocate at least the maximum ORB fetch size implemented by the target. The format of a dummy ORB is illustrated in Figure 13.

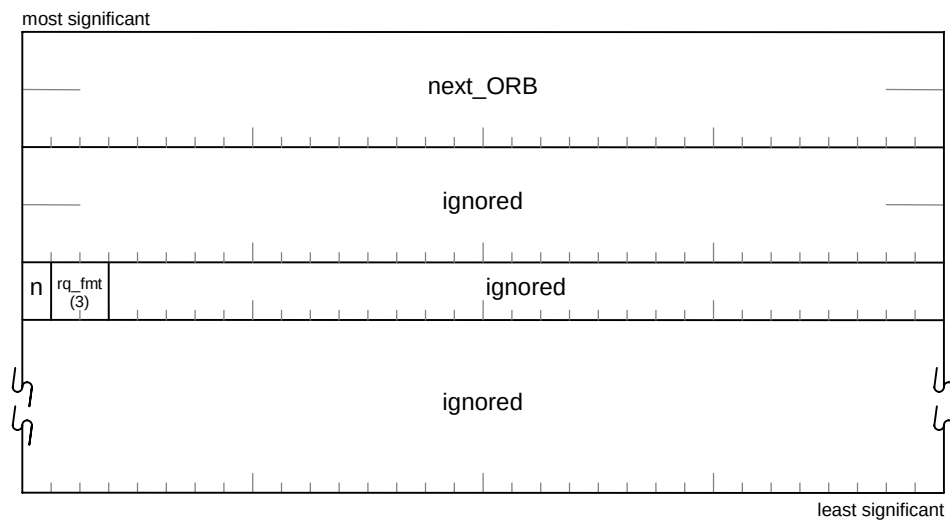


Figure 13 – Dummy ORB

The *next_ORB* field shall contain a null pointer or the address of an ORB and shall conform to the address pointer format illustrated by Figure 10.

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.

An *rq_fmt* value of three is also used to indicate an ABORT TASK request to a target. See 10.4.1 for details of ORB processing by the target and for permissible completion status values.

5.1.2 Command block ORBs

Command block ORBs 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 these ORBs, which shall be fixed for a particular command set and device type. A target reports this size in its configuration ROM (see 7.4.8).

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 well suited to many contemporary systems.

The format of the command block ORB is illustrated in Figure 14.

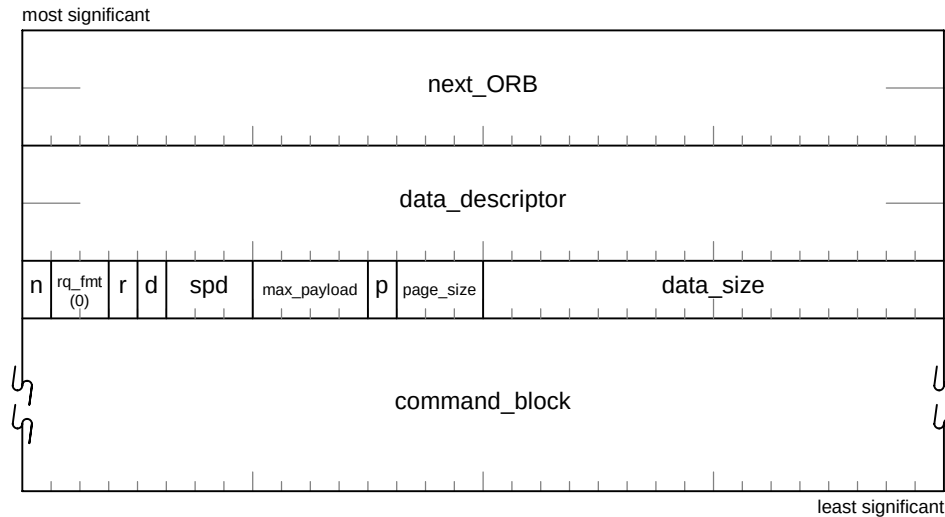


Figure 14 – Command block ORB

The *next_ORB* field shall contain a null pointer or the address of a dummy ORB or a command block ORB and shall conform to the address pointer format illustrated by Figure 10.

The value of the *data_descriptor* field is valid only when *data_size* is nonzero, in which case this field shall contain either the address of the data buffer or the address of a page table that describes the memory segments that make up the data buffer, dependent upon the value of *page_table_present* bit. The format of the *data_descriptor* field, when it directly addresses a data buffer, shall be a 64-bit Serial Bus address or, when it addresses a page table, shall be as specified by Figure 9. 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_fmt* field are as previously defined for all ORB formats. The *rq_fmt* field shall be zero.

The *direction* bit (abbreviated as *d* in the figure above) specifies direction of data transfer for the buffer. 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, as encoded by Table 1.

Table 1 – Data transfer speeds

Value	Speed
0	S100
1	S200
2	S400
3	S800
4	S1600
5	S3200
6 – 7	Reserved for future standardization

The maximum data transfer length is specified as $2^{\text{max_payload} + 2}$ bytes, which is the largest data transfer length that may be requested by the target in a single Serial Bus read or write transaction addressed to the data buffer. The *max_payload* field shall specify a maximum data transfer length less than or equal to the length 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. When *data_descriptor* addresses a page table, this bit shall be one.

The *page_size* field shall specify the underlying page size of the data buffer memory. A *page_size* value of zero indicates that the underlying page size is not specified. Otherwise, the page size is $2^{\text{page_size} + 8}$ bytes.

When *page_table_present* is one, the *page_size* field also specifies the format of the data structure that describes the data buffer. A *page_size* value of zero implies the unrestricted page table format (also known as a scatter/gather list). Otherwise, a nonzero *page_size* indicates a normalized page table.

If *page_table_present* is zero, the *data_size* field shall contain 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 contains information not specified by this standard.

5.1.3 Management ORBs

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 command block ORBs (which are implicitly associated with a particular task set by virtue of the fetch agent to which they are addressed), most management ORBs explicitly declare the task set for which they are intended.

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

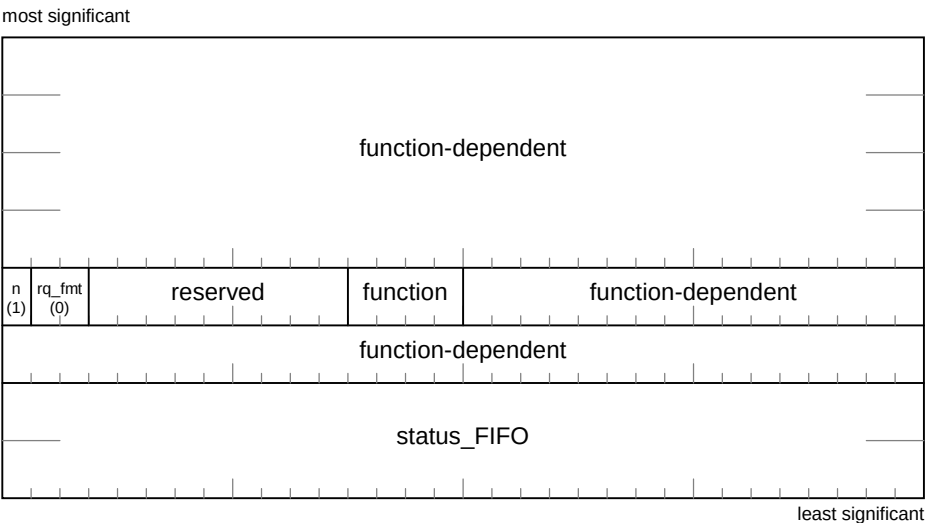


Figure 15 – Management ORB

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

The *function* field specifies the management function requested, as defined by Table 2.

Table 2 – Management request functions

Value	Management function
0	LOGIN
1	QUERY LOGINS
2	Reserved for future standardization
3	RECONNECT
4	SET PASSWORD (see Annex C)
5 – 6	Reserved for future standardization
7	LOGOUT
8 – A ₁₆	Reserved for future standardization
B ₁₆	ABORT TASK
C ₁₆	ABORT TASK SET
D ₁₆	Reserved for future standardization
E ₁₆	LOGICAL UNIT RESET
F ₁₆	TARGET RESET

The *status_FIFO* field shall contain an address allocated for the return of status information generated by the management request. The *status_FIFO* field shall conform to the format for address pointers specified by Figure 9 and shall address the same node as the initiator; consequently, the *node_ID* field of this address pointer is reserved.

NOTE – The *status_FIFO* address explicitly specified within a management ORB may differ from the status FIFO address implicitly associated with command block requests. The address in those cases is established by a LOGIN request and is not altered by other management requests.

5.1.3.1 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 in Figure 16.

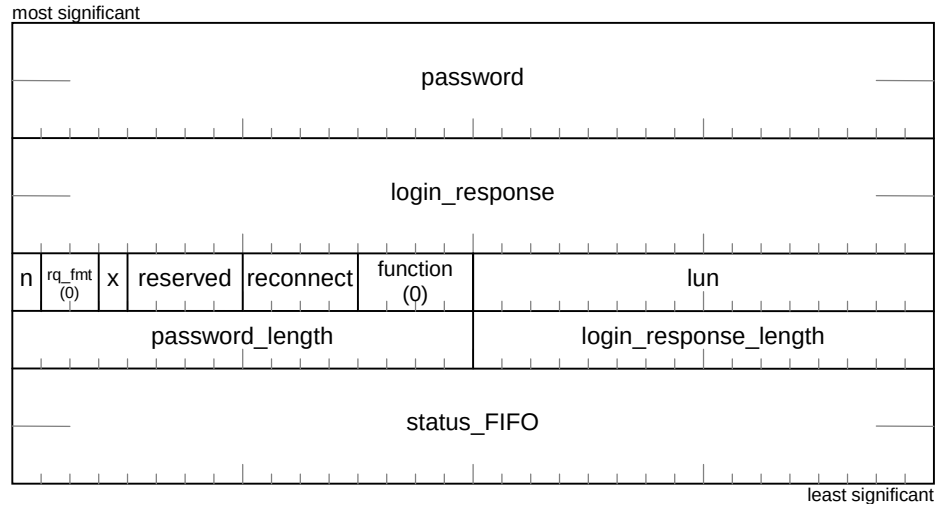


Figure 16 – Login ORB

The *password* and *password_length* fields may contain optional 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 conform to the format for address pointers specified by Figure 9 and shall contain the address of a buffer in the same node as the initiator; consequently, the *node_ID* field of this address pointer is reserved. 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 specified by Annex C.

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 *login_response* field shall conform to the format for address pointers specified by Figure 9 and shall address the same node as the initiator; consequently, the *node_ID* field of this address pointer is reserved. 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*. The initiator shall set *login_response_length* to a value of at least 12; the target may ignore this field if it stores no more than 12 bytes of login response data.

The *notify* bit and the *rq_fmt* field 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 for a description of target behavior.

The *reconnect* field shall specify the desired reconnect time-out as $2^{\text{reconnect}}$ seconds. The default reconnect time-out, when *reconnect* is zero, is one second. The target may not be able to support the requested value; see *reconnect_hold* in the login response data below.

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