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)

PRICE CODE X

For price, see current catalogue

CONTENTS

INFORMATION TECHNOLOGY – SMALL COMPUTER SYSTEM INTERFACE (SCSI) –

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

FOREWORD

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, date 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 Perfomance Serial Bus*

IEEE P1394a, *Draft Standard for High Perfomance 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 0000_{16}. 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 0800_{16}. 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, $2A_{16}$ and $0010\ 1010_2$ all represent the same numeric value.

3.2.2 Bit, byte and quadlet ordering

SBP-2 is defined to use the facilities of Serial Bus, 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 significance of bits within bytes, bytes within quadlets and quadlets within octlets in terms of their relative position and not their physically addressed position.

Within a byte, the most significant bit, *msb*, is that which is transmitted first and the least significant bit, *lsb*, is that which is transmitted last on Serial Bus, as illustrated below. The significance of the interior bits uniformly decreases in progression from *msb* to *lsb,* see Figure 1.

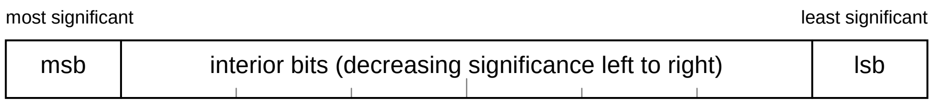

Figure 1 – Bit ordering within a byte

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.

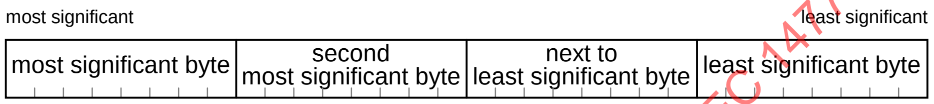

Figure 2 – Byte ordering within a quadlet

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.

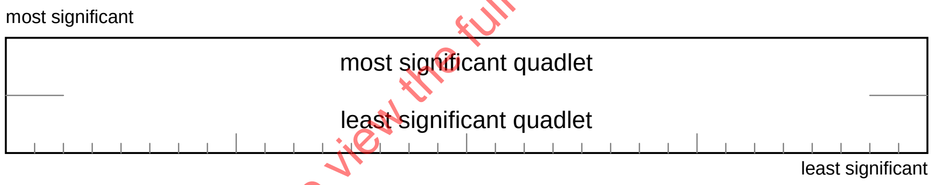

Figure 3 – Quadlet ordering within an octlet

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.

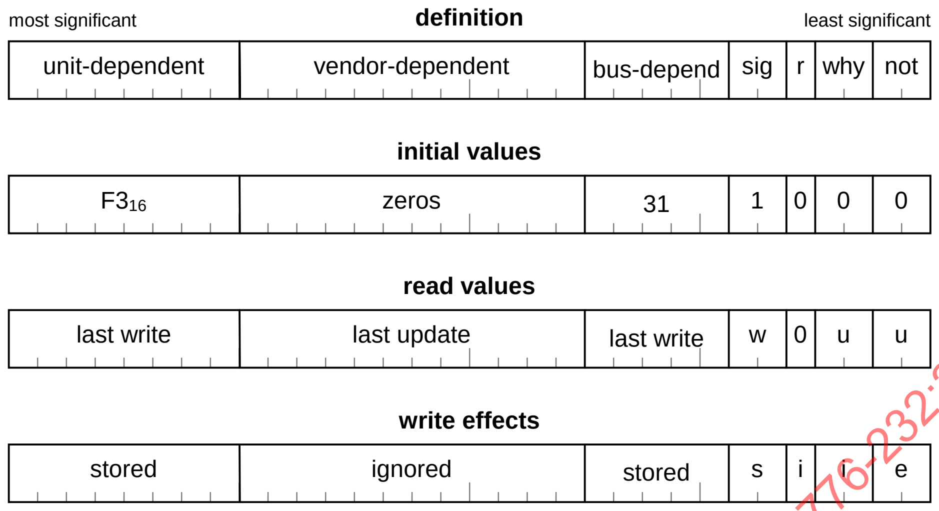

Figure 4 – CSR specification example

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.

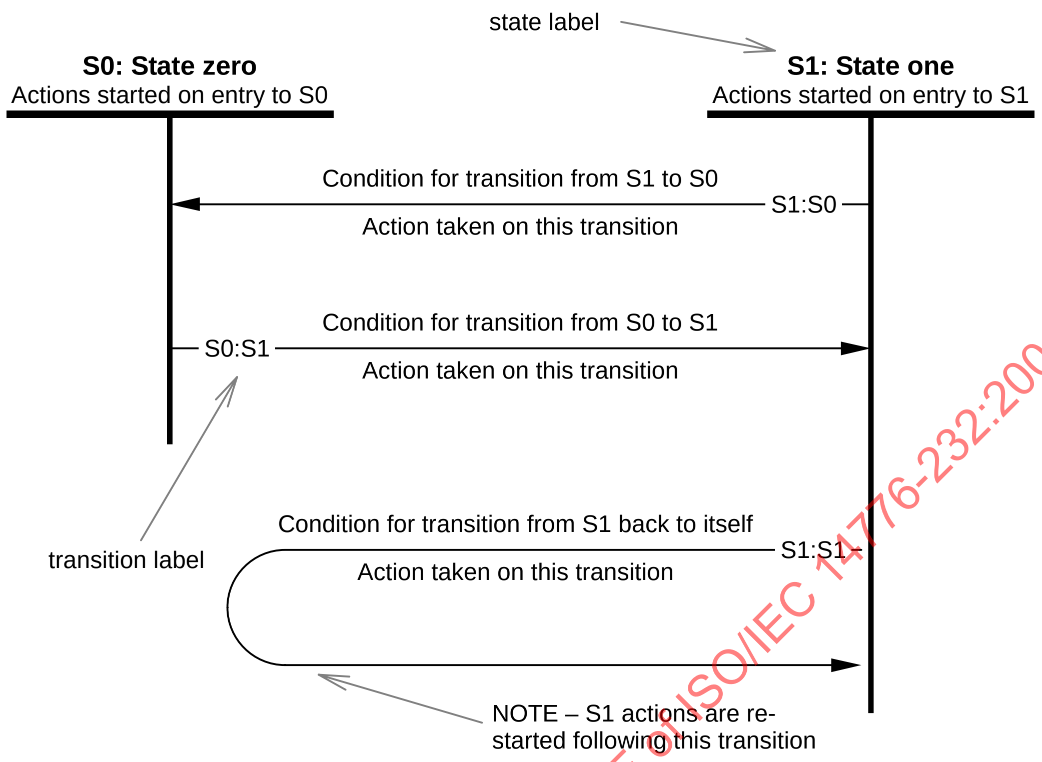
state label
S0: State zero
Actions started on entry to S0
S1: State one
Actions started on entry to S1
Condition for transition from S1 to S0
S1:S0
Action taken on this transition
Condition for transition from S0 to S1
S0:S1
Action taken on this transition
transition label
Condition for transition from S1 back to itself
S1:S1
Action taken on this transition
NOTE – S1 actions are re-started following this transition

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.

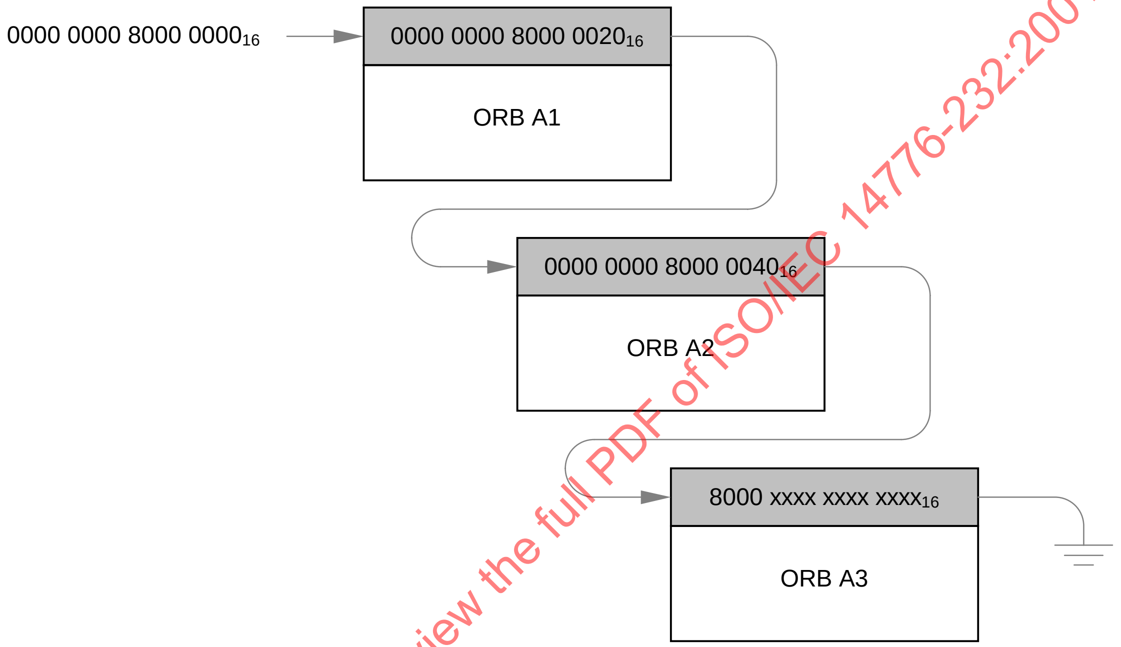

Figure 6 – Linked list of ORBs

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 3088_{16} bytes long and the buffer starts at an address of 23 6174_{16}, the relationship between the ORB and the data buffer might appear as in Figure 7.

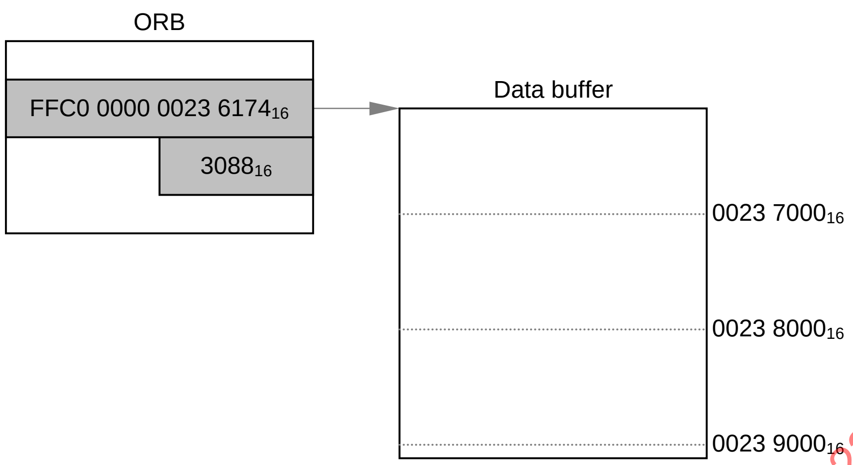

Figure 7 – Directly addressed data buffer

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 FFC0$_{16}$, 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 2960$_{16}$ bytes of data are to be read from disk.

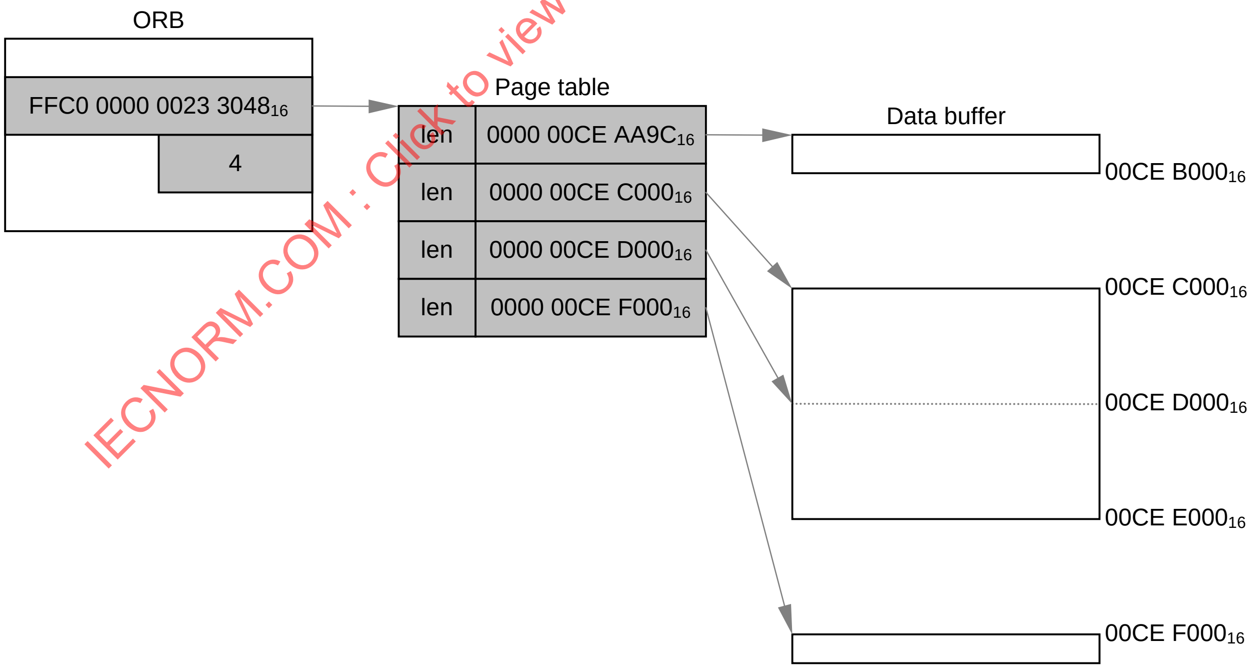

Figure 8 – Indirectly addressed data buffer (via page table)

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 0564_{16}, 1000_{16}, 1000_{16} and $03FC_{16}$, 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, $FFC0_{16}$, 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, $\{T_0, T_1, T_2, \ldots, T_N\}$ and a new task T´, the initiator shall wait until $\{T_0, T_1, T_2, \ldots, T_N\}$ 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.

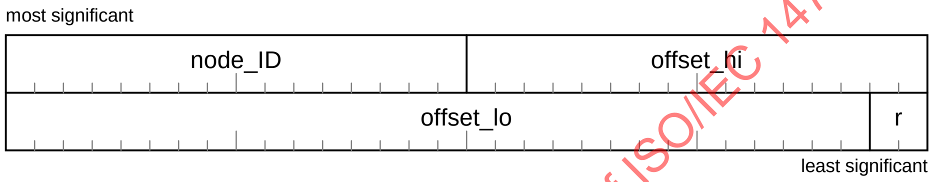

Figure 9 – Address pointer

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.

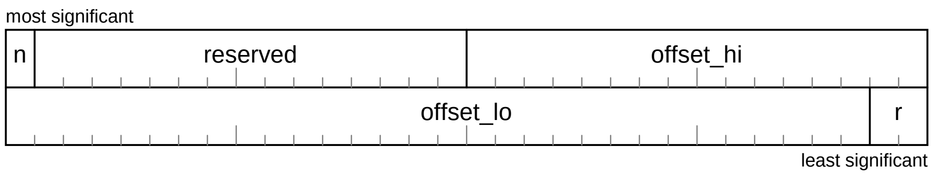

Figure 10 – ORB pointer

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.

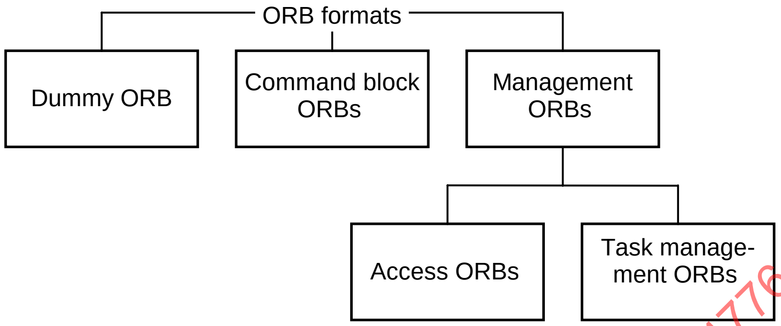

Figure 11 – ORB family tree

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.

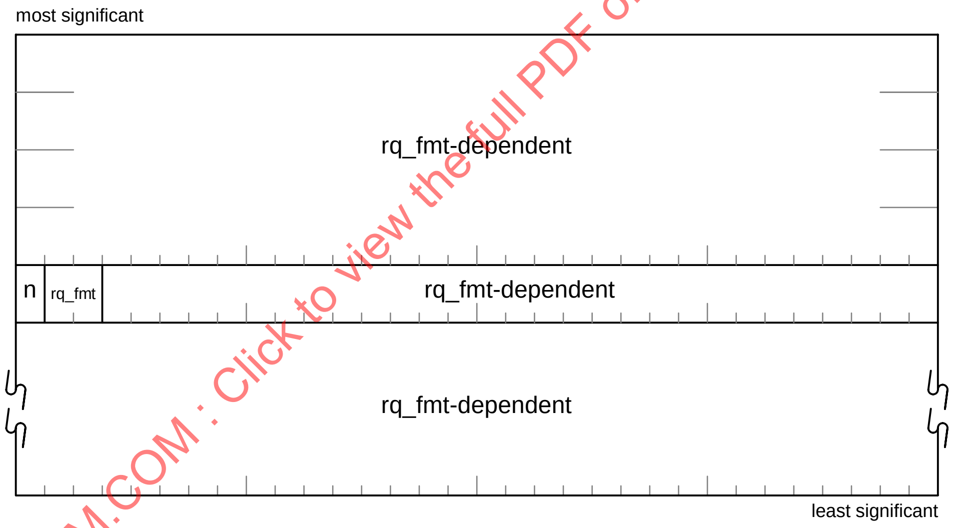

Figure 12 – ORB format

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.

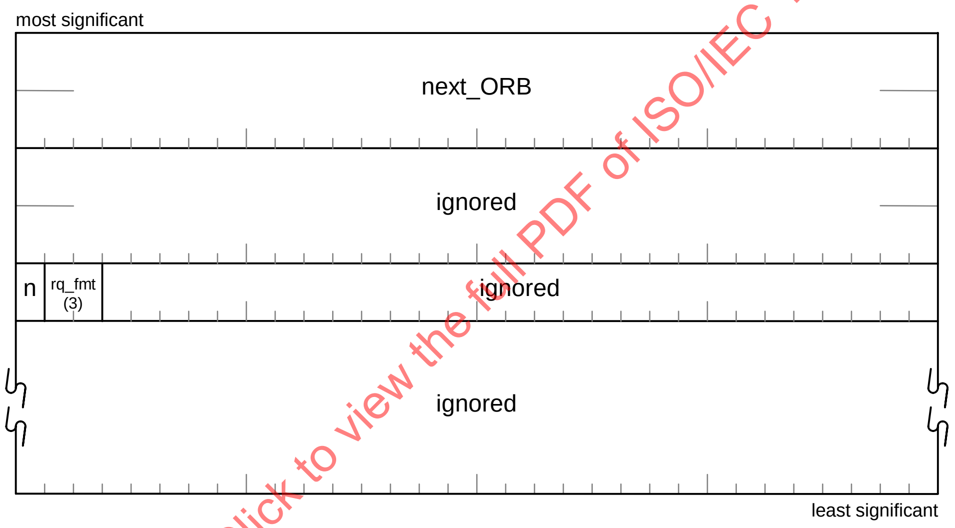

Figure 13 – Dummy ORB

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.

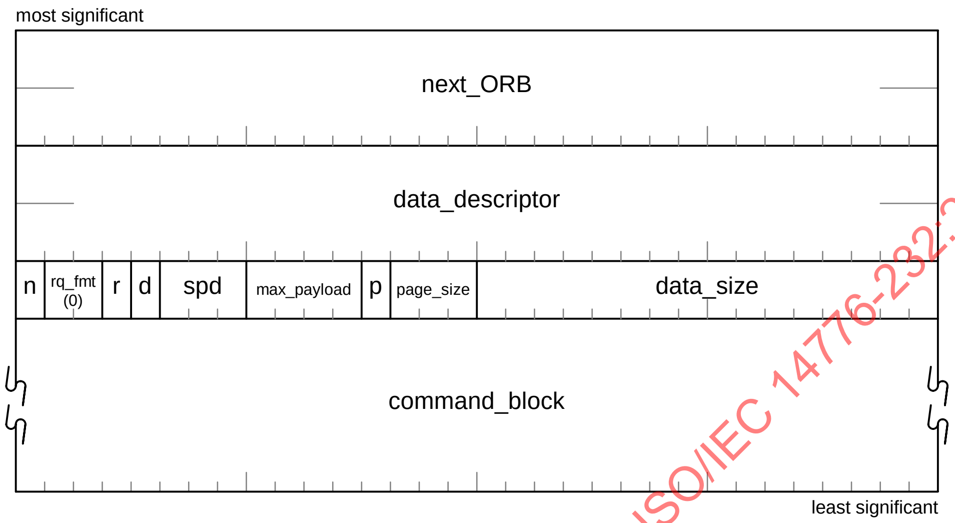

Figure 14 – Command block ORB

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 $2^{max\_payload + 2}$ 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 $2^{page\_size + 8}$ 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.

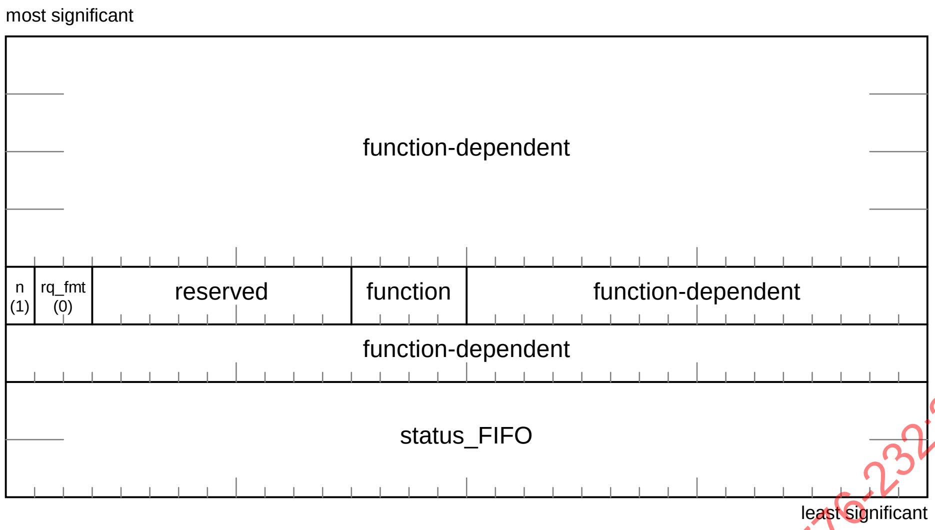

Figure 15 – Management ORB

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.

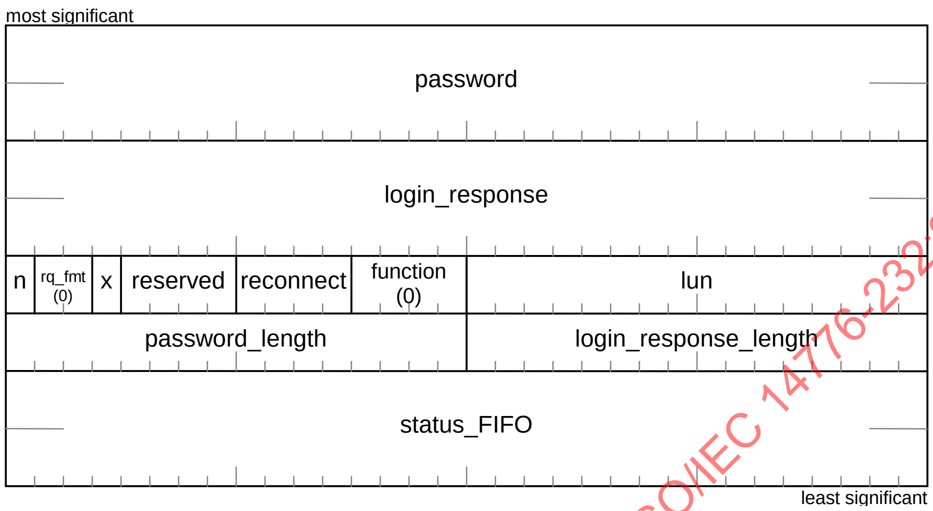

Figure 16 – Login ORB

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 $2^{reconnect}$ 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.

The *status_FIFO* field is as previously defined for management ORB formats and shall contain an address allocated for the return of status for the LOGIN request, status for all subsequent requests signaled to the *command_block_agent* allocated for this login and any unsolicited status generated by the logical unit.

If the login fails the contents of the response buffer are unspecified. Otherwise, upon successful completion of a login, the target shall store a minimum of 12 bytes of login response data and may store up to the entire 16 bytes illustrated in Figure 17 so long as the amount of data stored is an integral number of quadlets. Truncated login response data shall be interpreted as if the omitted fields had been stored as zeros.

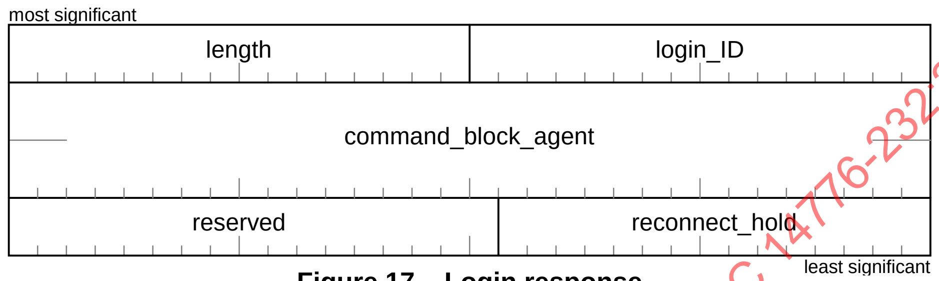

Figure 17 – Login response

The *length* field shall contain the length, in bytes, of the login response data.

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

The *command_block_agent* field specifies the base address of the agent's CSRs, which are defined in 6.4. This field shall conform to the format for address pointers specified by Figure 9. The *node_ID* portion of the field shall have a value equal to the most significant 16 bits of the target's NODE_IDS register.

The *reconnect_hold* field shall specify the time, in seconds less one, that the target will hold resources for a previously logged-in initiator subsequent to a bus reset. The value of *reconnect_hold* shall not be greater than $2^{reconnect}$ -1, where *reconnect* is obtained from the login request. If an initiator fails to complete a successful reconnect request within *reconnect_hold* + 1 seconds after a bus reset, the target will perform a logout and release all resources held by that initiator (see 8.3).

5.1.3.2 Query logins ORB

An initiator may determine the EUI-64 and node ID of all currently logged-in initiators by means of a query logins request, whose format is illustrated in Figure 18.

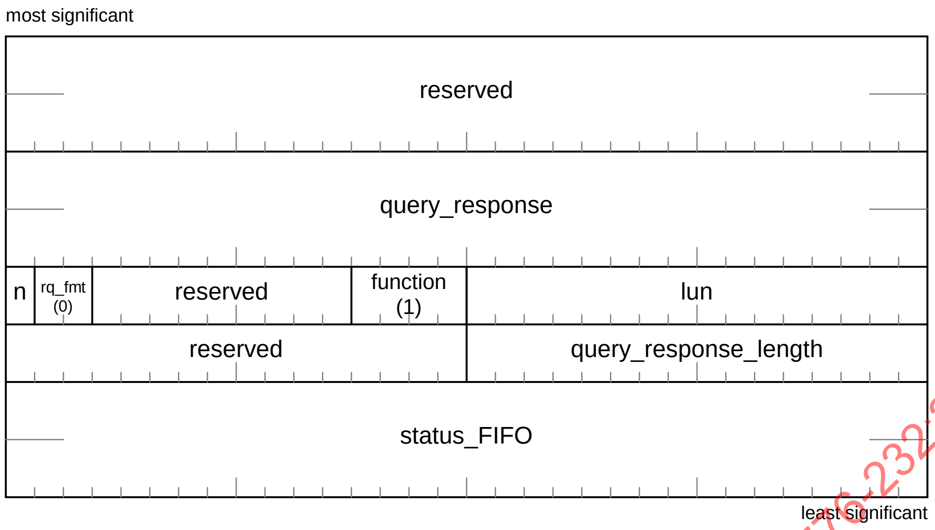

Figure 18 – Query logins ORB

The *query_response* and *query_response_length* fields specify the address and size of a buffer for the return of the query results. The *query_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 *query_response_length*.

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

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

The query response data returned shall have the following format, see Figure 19.

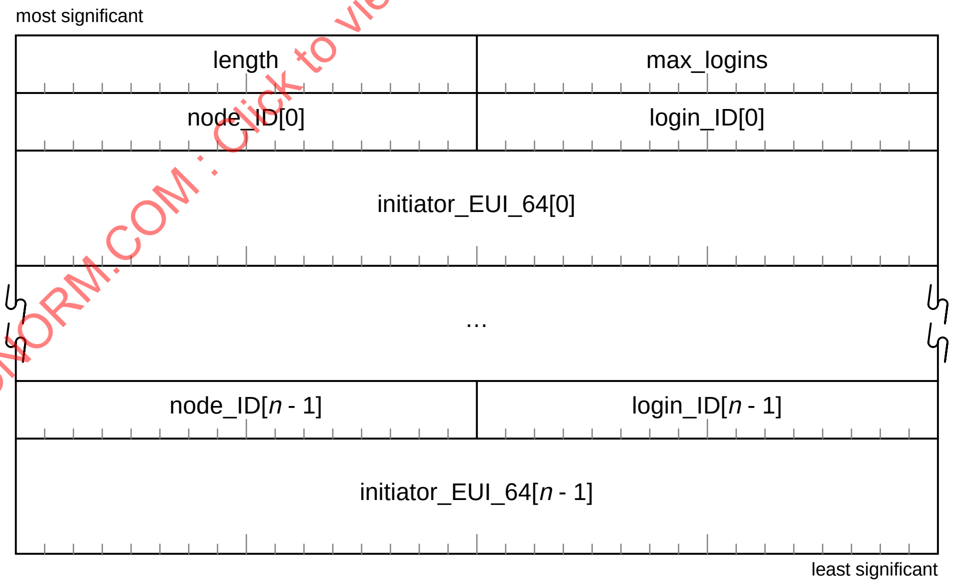

Figure 19 – Query logins response format

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

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

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

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

NOTE A *node_ID* value of $FFFF_{16}$ may be observed only in the reconnect interval that exists for *reconnect_hold* + 1 seconds after a Serial Bus reset because after this time the target performs an automatic logout for any initiator that has not reconnected.

If the *node_ID* field has a value of $FFFF_{16}$, the *login_ID* field shall contain the time remaining, in seconds less one, until the initiator is automatically logged-out by the target. Otherwise, the *login_ID* field of an entry shall contain the login ID provided to the initiator as a result of its successful login.

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

5.1.3.3 Reconnect ORB

After a Serial Bus reset an initiator shall reestablish access for a previously valid login before it signals new requests to the target for that login. This is accomplished by means of a reconnect request, with the format shown in Figure 20.

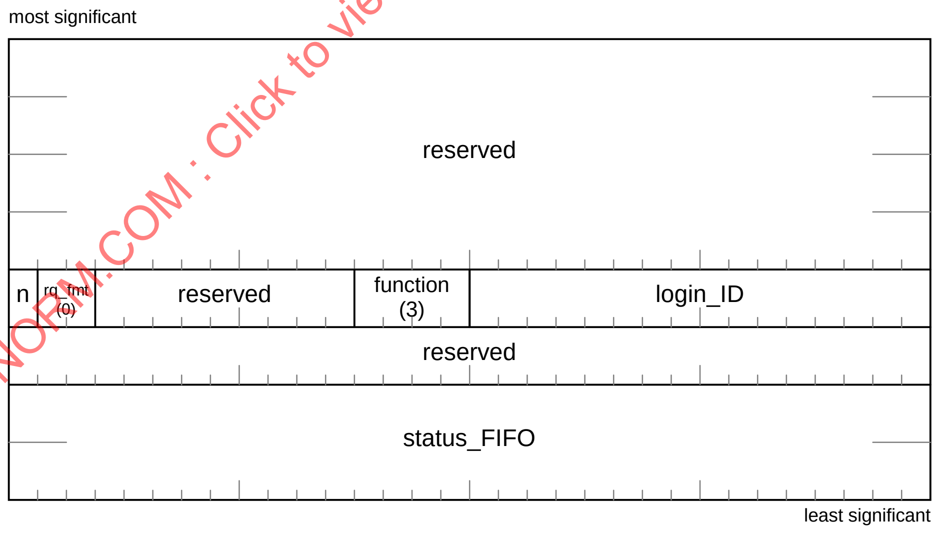

Figure 20 – Reconnect ORB

The *notify* bit and the *rq_fmt* field are as previously defined for management ORB formats.

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

The *status_FIFO* field is as previously defined for management ORB formats and shall contain an address allocated for the return of status for the RECONNECT request, only. The contents of this field shall not update the status FIFO address established by the successful login that returned *login_ID*.

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

5.1.3.4 Logout ORB

In order to relinquish its access privileges for a logical unit, an initiator shall perform a logout with the ORB format shown in Figure 21.

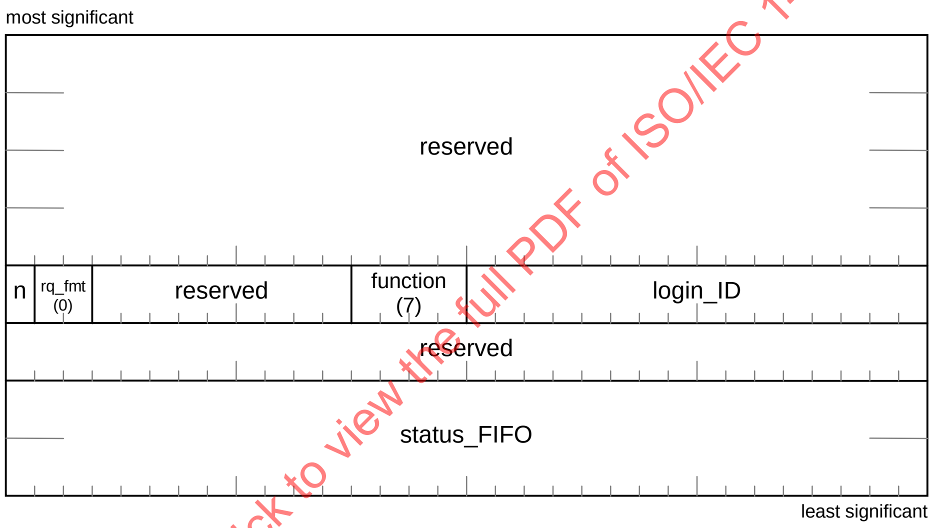

Figure 21 – Logout ORB

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

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

5.1.3.5 Task management ORB

The task management ORB is used to control task sets. This ORB shall have the format defined in Figure 22.

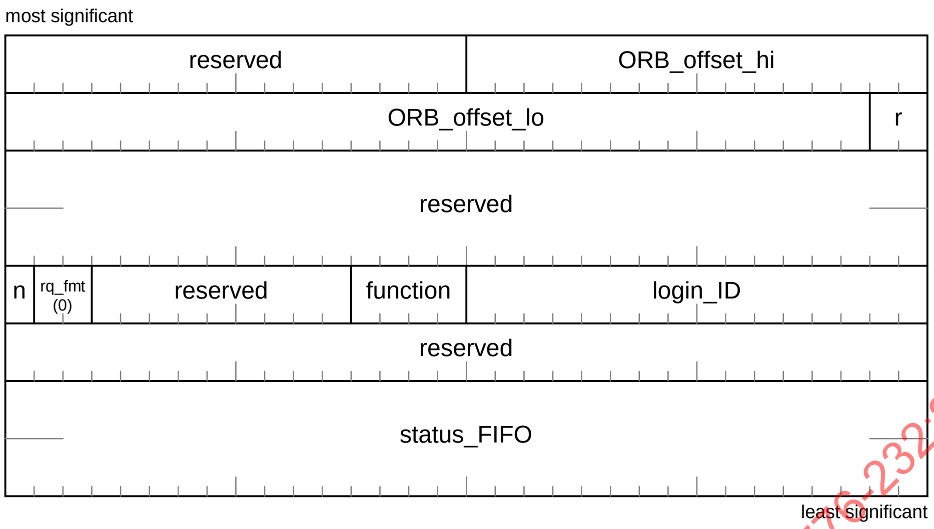

Figure 22 – Task management ORB

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

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

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

The *login_ID* shall be set to the value returned in login response data and identifies the task set to which the task management request is directed. In the case of TARGET RESET, which does not pertain to any one task set, *login_ID* shall be set to a value obtained as the result of any successful login completed by the initiator.

5.2 Page tables

The data buffer specified by a command block ORB is described by the *data_descriptor*, *page_table_present*, *page_size* and *data_size* fields. The data buffer is a logically contiguous area in system memory. As previously described, when *page_table_present* is zero, the data buffer is also contiguous within Serial Bus address space and no more than 65,535 bytes in length. In this case, *data_descriptor* contains the 64-bit address of the data buffer and *data_size* specifies its length, in bytes.

When the data buffer cannot be directly addressed (either because it is discontiguous or too large), it is necessary to describe it *via* a page table. A page table is a variable-length array of elements, each of which describes a segment that is contiguous within Serial Bus address space. Page table elements are eight bytes long and shall be octlet aligned.

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

Page tables may have one of two formats: an unrestricted page table or a normalized page table. The page table format is determined by *page_size*. When *page_size* is zero there are no underlying page boundaries to restrict the size or alignment of data buffer segments; this is the unrestricted format. Otherwise, the size and alignment of data buffer segments is determined by the nonzero *page_size*; this is the normalized format.

When a page table is used it shall be located in the same node as the data buffer it describes. The *spd* and *max_payload* fields of the ORB shall describe data transfer capabilities for both the data buffer and the page table. System memory addressed by a target request subaction that accesses the data buffer shall be entirely contained within a data buffer segment described by a single page table element.

5.2.1 Unrestricted page tables

An unrestricted page table shall be contiguous within Serial Bus address space and shall be accessible to block read requests with a *data_length* less than or equal to *data_size* * 8 bytes. The format of elements in an unrestricted page table is shown in Figure 23.

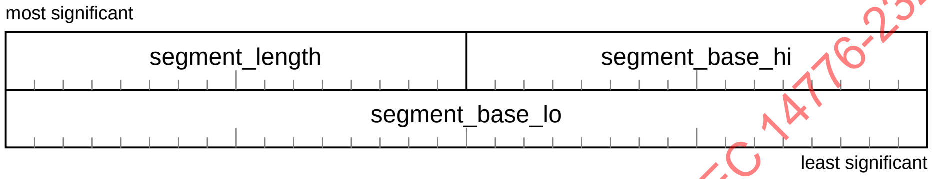

Figure 23 – Page table element (unrestricted page table)

The *segment_length* field shall contain the length, in bytes, of the portion of the data buffer (segment) described by the page table element. The value of *segment_length* shall be nonzero.

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

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

5.2.2 Normalized page tables

A normalized page table shall be contiguous within Serial Bus address space and shall be accessible to Serial Bus block read transactions with a *data_length* less than or equal to the smaller of *data_size* * 8 bytes or $2^{page\_size + 8}$ bytes so long as they do not cross Serial Bus address boundaries that occur every $2^{page\_size + 8}$ bytes, see Figure 24.

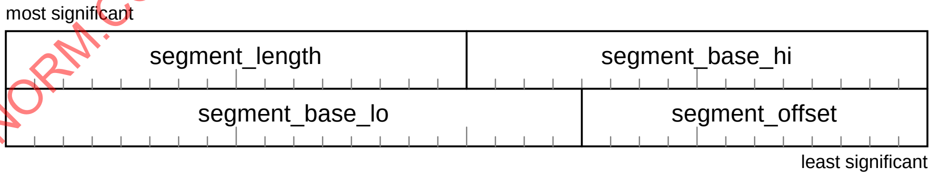

Figure 24 – Page table element (when *page_size* equals four)

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

The *segment_length* field shall contain the length, in bytes, of the portion of the data buffer (segment) described by the page table element. The value of *segment_length* shall be less than or equal to $2^{page\_size + 8}$.

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

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

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

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

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

Element Position	**Total page table elements**		
	1	**2**	***n* (where *n* >= 3)**
First	No additional restrictions	*segment_length* = $2^{page\_size + 8}$ - *segment_offset*	*segment_length* = $2^{page\_size + 8}$ - *segment_offset*
Middle	—	—	*segment_offset* = 0 *segment_length* = $2^{page\_size + 8}$
Last	—	*segment_offset* = 0	*segment_offset* = 0

5.3 Status block

A target may store status at an initiator *status_FIFO* address when a request completes (successfully or in error) or because of an unsolicited event (device status change). The *status_FIFO* address is obtained either explicitly from the ORB to which the status pertains or implicitly from the fetch agent context. Whenever the target has status to report and is enabled to do so, it shall store all or part of the status block shown in Figure 25.

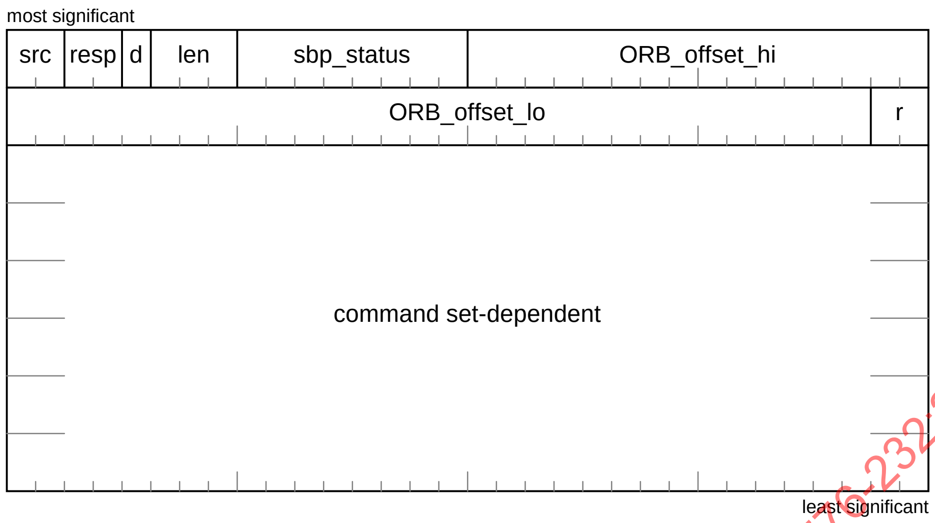

Figure 25 – Status block format

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

The *src* field indicates the origin of the status block, as specified by the table below.

Value	**Description**
0	The status block pertains to an ORB identified by *ORB_offset_hi* and *ORB_offset_lo*; at the time the ORB was most recently fetched by the target the *next_ORB* field did not contain a null pointer.
1	The status block pertains to an ORB identified by *ORB_offset_hi* and *ORB_offset_lo*; either the *next_ORB* field is absent or at the time the ORB was most recently fetched by the target the *next_ORB* field was null.
2	The status block is unsolicited and contains device status information; the contents of the *ORB_offset_hi* and *ORB_offset_lo* fields shall be ignored.
3	Reserved for future standardization.

The *resp* field shall contain a response status defined in the table below.

Value	**Name**	**Description**
0	REQUEST COMPLETE	The request completed without transport protocol error (Either *sbp_status* or command set-dependent status information may indicate the success or failure of the request)
1	TRANSPORT FAILURE	The target detected a nonrecoverable transport failure that prevented the completion of the request
2	ILLEGAL REQUEST	There is an unsupported field or bit value within the first 20 bytes of the ORB
3	VENDOR DEPENDENT	The meaning of *sbp_status* shall be specified by the vendor

The *dead* bit (abbreviated as *d* in Figure 25) shall indicate whether or not the target fetch agent transitioned to the dead state upon storing the status block. When *dead* is zero, the reported status has not affected the state of the fetch agent. If the *dead* bit is set to one, the fetch agent transitioned to the dead state as a consequence of the error condition reported by the status block.

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

The *sbp_status* field provides additional information that qualifies the response status in *resp*. The meanings assigned to *sbp_*status vary according to the value of *src* and *resp* and are described below.

When *src* is zero or one, the *ORB_offset_hi* and *ORB_offset_lo* fields together uniquely identify the ORB to which the status block pertains. For other values of *src*, the *ORB_offset_hi* and *ORB_offset_lo* fields are either ignored or redefined.

For all status block formats, the remainder of the status block after the first two quadlets, up to an overall maximum of 32 bytes, is command set-dependent.

5.3.1 Request status

Upon completion of a request, if the *notify* bit in the ORB is one or if there is exception status to report, the target shall store all or part of the status block shown in. For management ORBs (which explicitly provide the *status_FIFO* address as part of the ORB), the target shall store the status block at the address specified. Otherwise (for command block ORBs) the target shall store the status block at the *status_FIFO* determined by the fetch agent to which the ORB was signaled. In the case of command block ORBs the initiator provides the *status_FIFO* address as part of the login request.

When *resp* is equal to zero, REQUEST COMPLETE, the possible values for *sbp_status* are specified by the table below. Any value not enumerated is reserved for future standardization.

Value	Description
0	No additional information to report
1	Request type not supported
2	Speed not supported
3	Page size not supported
4	Access denied
5	Logical unit not supported
6	Maximum payload too small
7	Reserved for future standardization
8	Resources unavailable
9	Function rejected
10	Login ID not recognized
11	Dummy ORB completed
12	Request aborted
FF_{16}	Unspecified error

If a Serial Bus error occurs in the transport (*resp* is equal to one, TRANSPORT FAILURE), the *sbp_status* field either shall have a value of FF_{16}, unspecified error, or else the field shall be redefined as illustrated in Figure 26. This format provides for the return of additional information about the transport failure.

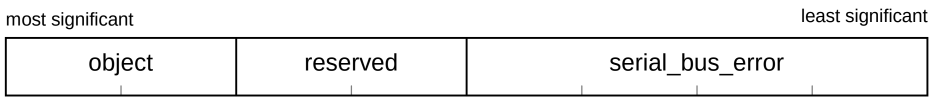

Figure 26 – TRANSPORT FAILURE format for *sbp_status*

The *object* field shall specify which component of an SBP-2 request, the ORB, the data buffer or the page table, was referenced by the target when the error occurred. The value of *object* shall be as defined by the following table.

Value	Referenced object
0	Operation request block (ORB)
1	Data buffer
2	Page table
3	Unable to specify

The *serial_bus_error* field shall contain the error response for the failed request, as encoded by the table below.

Value	Serial Bus error	Comment
0	Missing acknowledge	
1	Reserved; not to be used	
2	Time-out error	An *ack_pending* was received for the request but no response subaction was completed within the time-out limit
3	Reserved; not to be used	
4 – 6	Busy retry limit exceeded	The value reflects the last acknowledge, *ack_busy_X*, *ack_busy_A* or *ack_busy_B*
$7 – A_{16}$	Reserved for future standardization	
B_{16}	Tardy retry limit exceeded	An *ack_tardy* was received for the request and the vendor-dependent retry limit (which may be based upon either time or number of occurrences) for tardy responses has been exceeded
C_{16}	Conflict error	A resource conflict was detected by the addressed node
D_{16}	Data error	The data field failed the CRC check or the observed length of the payload did not match the data_length field
E_{16}	Type error	A field in the request was set to an unsupported value or an invalid transaction was attempted (e.g., a write to a read-only address)
F_{16}	Address error	The destination_offset field specified an inaccessible address in the addressed node

In the cases of conflict error and data error, these are errors that the target may retry up to an implementation-dependent limit before reporting TRANSPORT FAILURE.

No additional information is provided in *sbp_status* when *resp* equals two, ILLEGAL REQUEST. In this case, *sbp_status* shall be set to FF_{16}. An SBP-2 response code of ILLEGAL REQUEST shall not be used to indicate unsupported fields or bit values in the command set-dependent portion of the ORB. This response code shall be used only to indicate an error in the first 20 bytes of the ORB.

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

5.3.2 Unsolicited device status

When a change in device status occurs that affects a logical unit, the target may store the status block shown at the *status_FIFO* address provided by the initiator as part of a login request (see 5.1.3.1). If a target stores unsolicited status for any initiator logged-in to a logical unit it shall attempt to store status for all initiators logged-in to the same logical unit.

The *src* field shall be one to indicate unsolicited device status.

The *resp* field shall have a value of REQUEST COMPLETE or VENDOR DEPENDENT.

The *dead* bit and the *len* field are as previously defined for the status block.

If *resp* is equal to REQUEST COMPLETE, *sbp_status* shall be zero. Otherwise, the content and meaning of *sbp_status* shall be specified by the vendor.

The contents of the *ORB_offset_hi* and *ORB_offset_lo* fields are unspecified and shall be ignored by the initiator.

6 Control and status registers

The control and status registers (CSRs) implemented by a target shall conform to the requirements defined by this standard and its normative references. The CSRs are arranged in three principal categories:

- core registers required by ISO/IEC 13213;
- bus-dependent registers required by ANSI/IEEE 1394; and
- unit architecture registers required by this standard.

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

6.1 Core registers

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

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

The CSR architecture and ANSI/IEEE 1394 broadly define the effects of a write to the RESET_START register. In addition to those requirements, a write to RESET_START should cause all of a node’s SBP-2 units to reset in the same fashion as a power reset.

NOTE – Because of the potential for malicious interference in target operations by an unauthorized node, it is recommended that a write to RESET_START have no effect upon a target unless either a) there are no logged-in initiators or b) the *source_ID* of the write matches that of one of the currently logged-in initiators.

6.2 Serial Bus-dependent registers

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

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

6.3 MANAGEMENT_AGENT register

The MANAGEMENT_AGENT register permits the initiator to signal the address of a management ORB to the target. This register shall support 8-byte block read and block write requests whose *destination_offset* is equal to the address of the MANAGEMENT_AGENT register and shall reject quadlet write requests and all other block read and block write requests. The format of this register is illustrated in Figure 27.

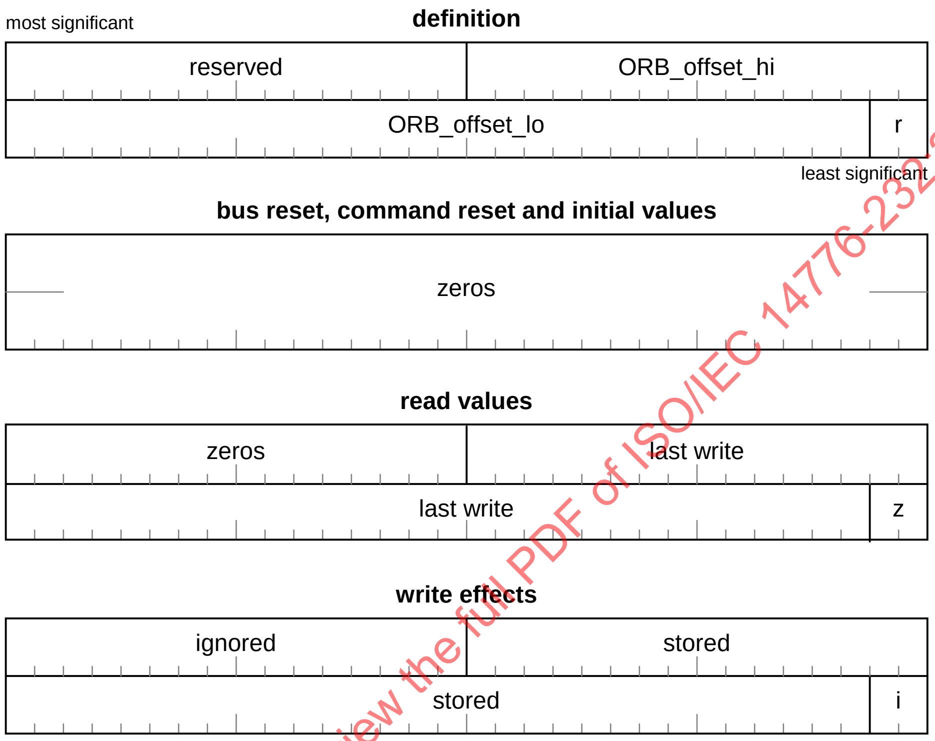

Figure 27 – MANAGEMENT_AGENT format

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

An initiator may signal a request by means of an 8-byte block write transaction that specifies the address of the request. If the management agent is busy with another request, the block write shall be rejected with a response of *resp_conflict_error*. If the write transaction is successful, the management agent shall fetch the request specified by *ORB_offset* and execute it. Unsuccessful write transactions shall not affect the execution of any request(s) in progress.

Because, in ANSI/IEEE 1394, a portion of initial units space is reserved for bus-dependent use, the MANAGEMENT_AGENT register shall be located at address FFFF F001 0000_{16} or above within the node's 48-bit address range. The address of the management agent is specified by the *csr_offset* field in the Management_Agent entry in configuration ROM (see 7.4.7).

6.4 Command block agent registers

Unlike the management agent, which services a single request at a time, the command block agents manage linked lists from which they fetch requests. For this reason, they are referred to as fetch agents.

Each target fetch agent has a set of control and status registers that lie within the target's initial units space; the fetch agent CSRs shall be located at or above address FFFF F001 0000_{16} within the node's 48-bit address range.

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

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

The base address of a fetch agent's CSRs is obtained from the *command_block_agent* field in the response returned by the target as part of a successful login.

A target shall ignore or reject Serial Bus request subactions addressed to any of a fetch agent's CSRs unless the *source_ID* matches the node ID of the initiator logged-in to that initiator.

6.4.1 AGENT_STATE register

The AGENT_STATE register is a read-only register that provides information about the current condition of the fetch agent. The definition is given in Figure 28.

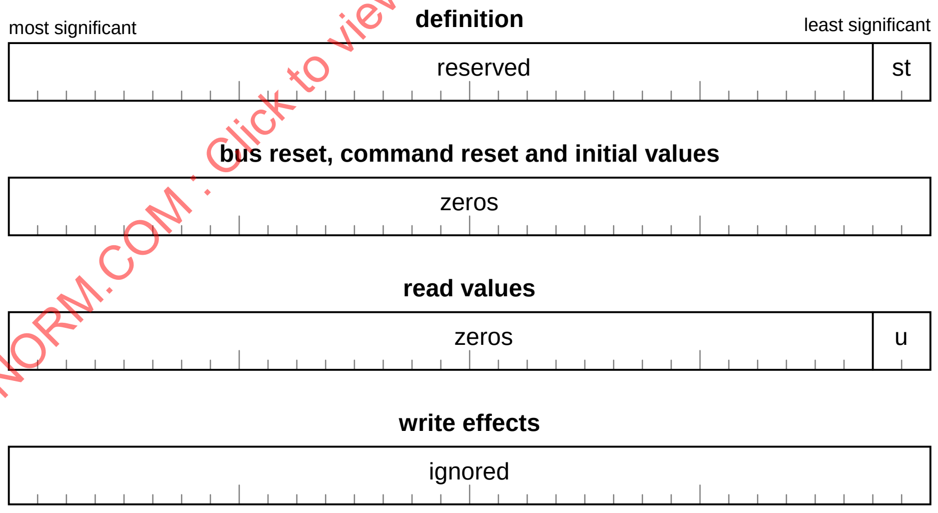

Figure 28 – AGENT_STATE format

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

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

6.4.2 AGENT_RESET register

The AGENT_RESET register permits an initiator to reset the operational state of a target fetch agent. The definition of this write-only register is given in Figure 29.

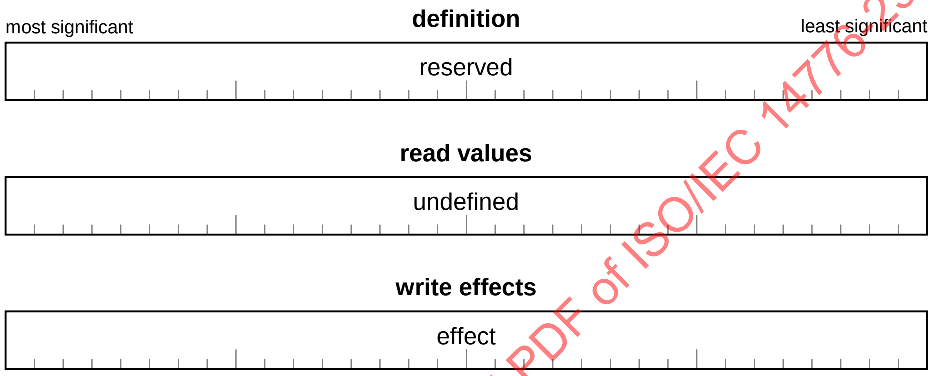

Figure 29 – AGENT_RESET format

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

6.4.3 ORB_POINTER register

The ORB_POINTER register contains the address of an ORB in system memory. This register shall support 8-byte block read and block write requests whose *destination_offset* is equal to the address of the ORB_POINTER register and shall reject quadlet write requests and all other block read and block write requests. The definition is given in Figure 30.

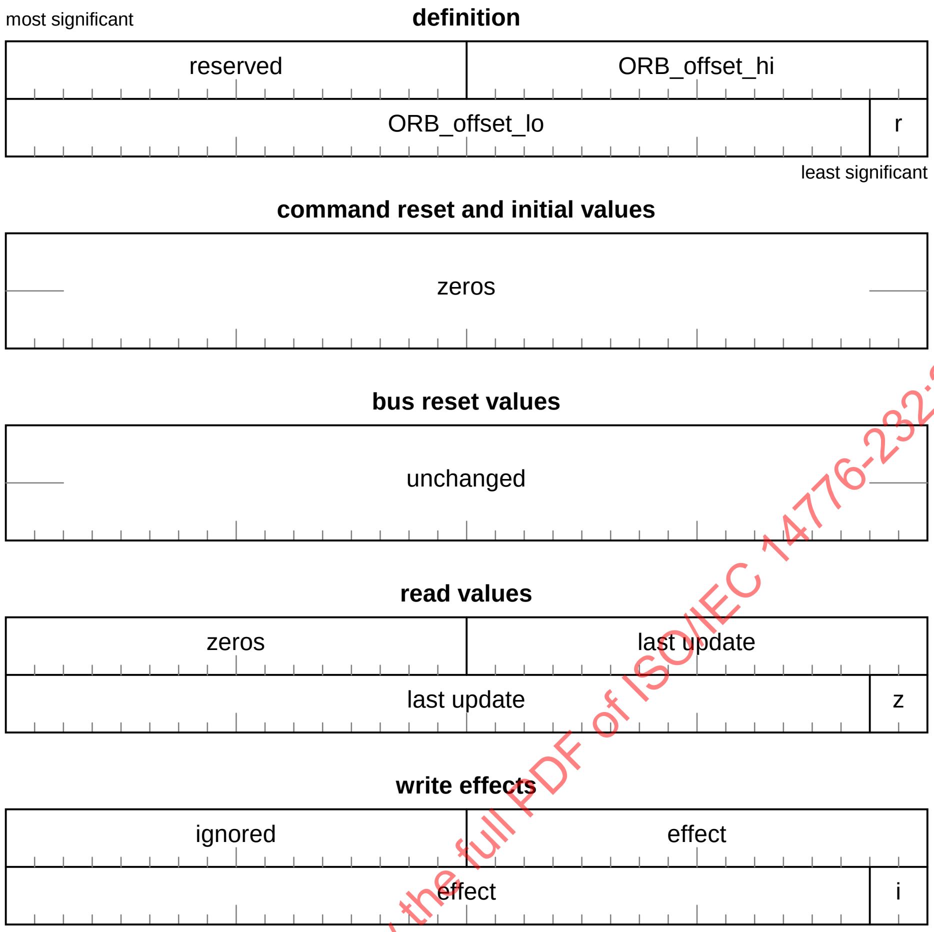

Figure 30 – ORB_POINTER format

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

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

6.4.4 DOORBELL register

The DOORBELL register provides a means by which the initiator signals the target that a linked list of requests has been updated. The definition of this write-only register is given in Figure 31.

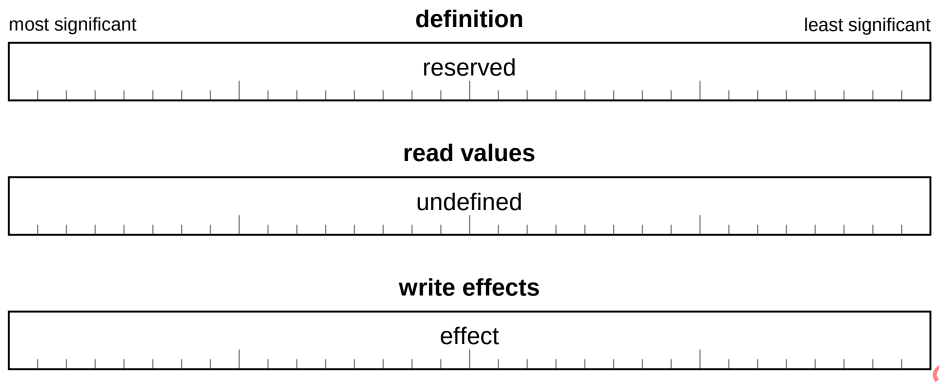

Figure 31 – DOORBELL format

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

6.4.5 UNSOLICITED_STATUS_ENABLE register

The UNSOLICITED_STATUS_ENABLE register provides a means by which the initiator may grant the target permission to store an unsolicited status block. The definition of this write-only register is given in Figure 32.

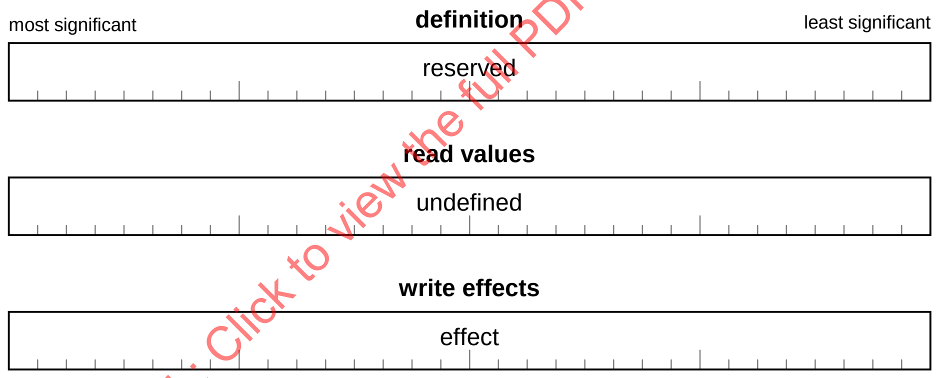

Figure 32 – UNSOLICITED_STATUS_ENABLE format

A quadlet write of any value to this register shall cause the fetch agent's *unsolicited status enabled* variable to be set to one. A successful login shall zero the *unsolicited status enabled* variable. As described in 9.4, any time a target stores an unsolicited status block it shall zero the *unsolicited status enabled* variable for that login. Before the target may store a subsequent unsolicited status block, it is necessary for the initiator to write to the UNSOLICITED_STATUS_ENABLE register.

7 Configuration ROM

All nodes that implement SBP-2 targets shall implement general format configuration ROM in accordance with ISO/IEC 13213, ANSI/IEEE 1394 and this standard. General format configuration ROM is a self-descriptive structure as illustrated in Figure 33. The bus information block and root directory are at fixed locations; all other directories and leaves are addressed by entries in their parent directory.

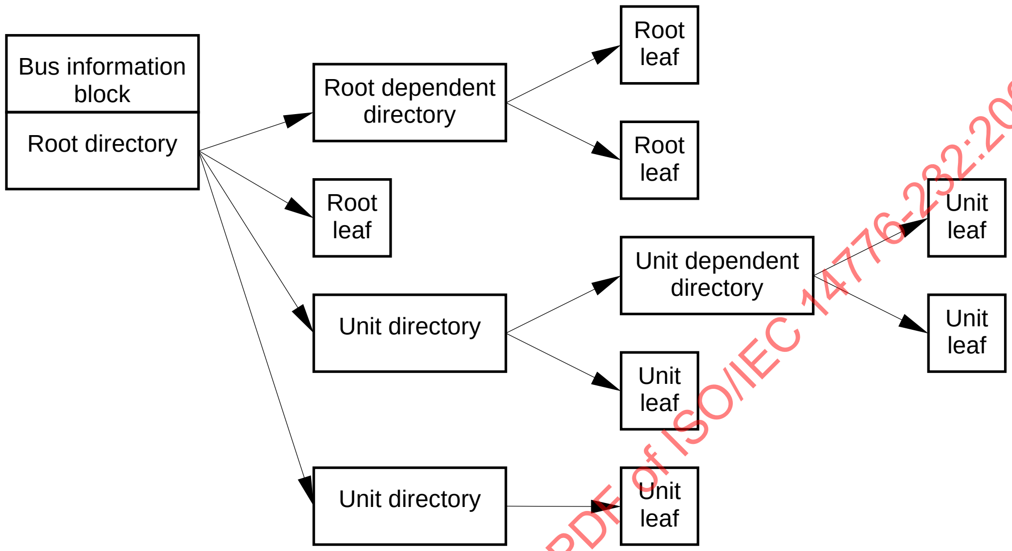

Figure 33 – Configuration ROM hierarchy

The figure above shows the potential of the general ROM format to accommodate a diversity of directory and leaf entries in a tree structure. In practice a target need implement only a portion of the entries shown above.

7.1 Power reset initialization

During the initialization process that follows a power reset, a target may not be able to respond to Serial Bus request subactions addressed to parts of configuration ROM. When the target has insufficient information to make more than the first quadlet of configuration ROM accessible, it shall return a data value of zero in the response to any read request addressed to FFFF F000 0400_{16} or acknowledge the request subaction with *ack_tardy*, as specified by IEEE P1394a. Until the initialization process completes, responses to requests addressed to other parts of configuration are unspecified.

Targets shall complete initialization within five seconds of a power reset. Once power reset initialization completes, the target shall make all mandatory configuration ROM entries available. The target should not initiate a Serial Bus reset solely as a consequence of the completion of power reset initialization.

Optional configuration ROM information, such as textual descriptor leaves that identify the target vendor and model, may not be available when power reset initialization completes. The target may add this information to configuration ROM as it becomes available and may initiate a Serial Bus reset to alert other nodes to the changed configuration ROM. The target should initiate a Serial Bus reset if there is no expectation that other nodes would otherwise become aware of changed configuration ROM.

7.2 Bus information block

All targets shall implement a bus information block at a base address of FFFF F000 0404_{16}. For convenience of reference, the format of the bus information block defined by ANSI/IEEE 1394 is reproduced in Figure 34. The current version of the referenced standard or its supplements shall be consulted for the most recent information.

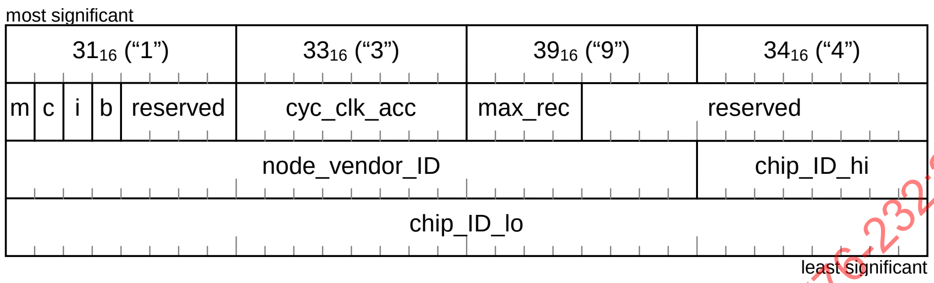

Figure 34 – Bus information block format

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

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

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

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

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

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

The *max_rec* field defines the maximum data payload size that the target supports. The data payload size applies to block write requests addressed to the target and to block read responses transmitted by the target. The maximum data payload is equal to $2^{max\_rec + 1}$ bytes. The *max_rec* field does not place any limits on the maximum payload size of block write requests or block read responses that the target may transmit or receive, respectively.

The *node_vendor_ID* field shall be uniquely assigned by the IEEE/RAC, as specified by ISO/IEC 13213. Unique identifiers for a company or organization may be obtained from:

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

The *chip_ID_hi* and *chip_ID_lo* fields are concatenated to form a 40-bit chip ID value. The vendor specified by *node_vendor_ID* shall administer the chip ID values. When appended to the *node_vendor_ID* value, these shall form a unique 64-bit value called the EUI-64 (Extended Unique Identifier, 64 bits). The EUI-64 is also referred to as the node unique ID. Because physical addresses on Serial Bus may change after a bus reset, this unique identifier is the only reliable method of node identification.

7.3 Root directory

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

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

7.3.1 Module_Vendor_ID entry

The Module_Vendor_ID entry is an immediate entry in the root directory that provides the company ID of the vendor that manufactured the module. Figure 35 shows the format of this entry.

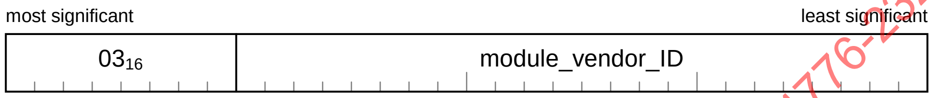

Figure 35 – Module_Vendor_ID entry format

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

The IEEE/RAC uniquely assigns the *module_vendor_ID* to each module vendor, as specified by ISO/IEC 13213. There is no requirement that the values of *module_vendor_ID* and *node_vendor_ID* be equal.

NOTE – A recommended convention to provide vendor identification in displayable form is to immediately follow the Module_Vendor_ID entry with a textual descriptor leaf entry. This associates an ASCII string with the module vendor. See ISO/IEC 13213 for the specification of textual descriptor leaves; examples are given in Annex D.

7.3.2 Node_Capabilities entry

The Node_Capabilities entry is an immediate entry in the root directory that describes node capabilities. Figure 36 shows the format of this entry.

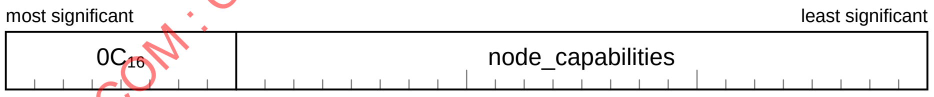

Figure 36 – Node_Capabilities entry format

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

The *node_capabilities* field contains subfields, specified by ISO/IEC 13213. Targets shall implement the SPLIT_TIMEOUT register, the 64-bit fixed addressing scheme, the STATE_CLEAR.*lost* bit and the STATE_CLEAR.*dreq* bit and indicate this by setting the *spt*, *64*, *fix*, *lst* and *drq* bits to one. If no other *node_capabilities* bits are one, this results in a value of 00 $83C0_{16}$.

7.3.3 Unit_Directory entry

The Unit_Directory entry is a directory entry in the root directory that describes the location of a unit directory within configuration ROM. There may be more than one unit directory; each unit directory shall be located by a separate Unit_Directory entry. Figure 37 shows the format of this entry.

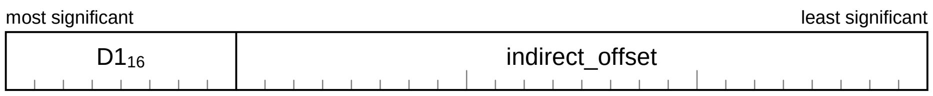

Figure 37 – Unit_Directory entry format

D1$_{16}$ is the concatenation of *key_type* and *key_value* for the Unit_Directory entry.

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

7.4 Unit directory

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

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

The unit directory may also contain a Unit_Unique_ID entry.

7.4.1 Unit_Spec_ID entry

The Unit_Spec_ID entry is an immediate entry in the unit directory that specifies the organization responsible for the architectural definition of the target. Figure 38 shows the format of this entry.

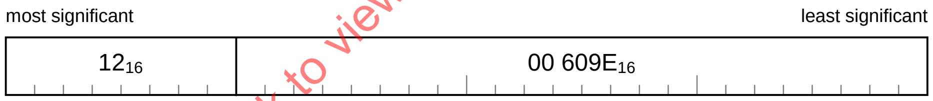

Figure 38 – Unit_Spec_ID entry format

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

00 609E$_{16}$ is the *unit_spec_ID* obtained by NCITS from the IEEE/RAC. The value indicates that the NCITS Secretariat and its Technical Committee T10 are responsible for the maintenance of this standard.

7.4.2 Unit_SW_Version entry

The Unit_SW_Version entry is an immediate entry in the unit directory that, in combination with the *unit_spec_ID*, specifies the software interface of the target. Figure 39 shows the format of this entry.

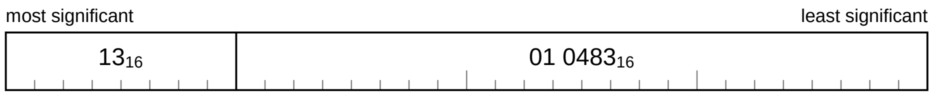

Figure 39 – Unit_SW_Version entry format

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

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

7.4.3 Command_Set_Spec_ID entry

The Command_Set_Spec_ID entry is an immediate entry that, when present in the unit directory, specifies the organization responsible for the command set definition for the target. Figure 40 shows the format of this entry.

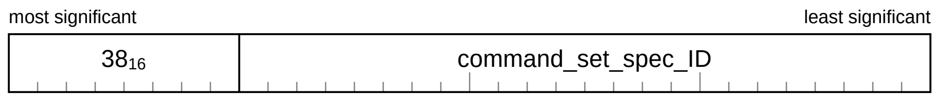

Figure 40 – Command_Set_Spec_ID entry format

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

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

7.4.4 Command_Set entry

The Command_Set entry is an immediate entry that, when present in the unit directory, in combination with the *command_set_spec_ID* specifies the command set implemented by the target. Figure 41 shows the format of this entry.

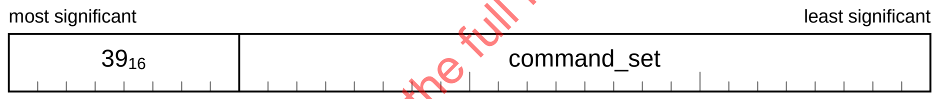

Figure 41 – Command_Set entry format

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

The value of *command_set* shall be specified by the owner of *command_set_spec_ID*.

7.4.5 Command_Set_Revision entry

The Command_Set_Revision entry is an immediate entry that, when present in the unit directory, specifies the revision level of the command set implemented by the target. Figure 42 shows the format of this entry.

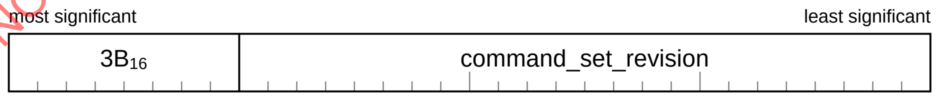

Figure 42 – Command_Set_Revision entry format

$3B_{16}$ is the concatenation of *key_type* and *key_value* for the Command_Set_Revision entry.

The value of *command_set_revision* shall be specified by the owner of *command_set_spec_ID*.

7.4.6 Firmware_Revision entry

The Firmware_Revision entry is an immediate entry that, when present in the unit directory, specifies the firmware revision level implemented by the target. Figure 43 shows the format of this entry.

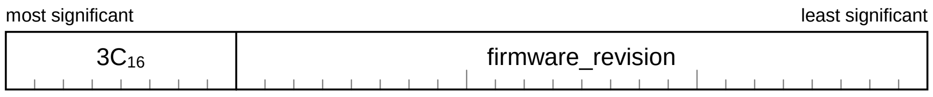

Figure 43 – Firmware_Revision entry format

$3C_{16}$ is the concatenation of *key_type* and *key_value* for the Firmware_Revision entry.

The value of *firmware_revision* shall be specified by the manufacturer of the target.

7.4.7 Management_Agent entry

The Management_Agent entry is an immediate entry in the unit directory that specifies the base address of the target's MANAGEMENT_AGENT register. Figure 44 shows the format of this entry.

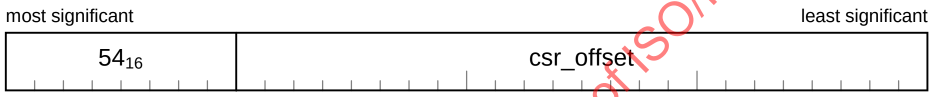

Figure 44 – Management_Agent entry format

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

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

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

7.4.8 Unit_Characteristics entry

The Unit_Characteristics entry is an immediate entry in the unit directory which specifies characteristics of the target implementation. Figure 45 shows the format of this entry.

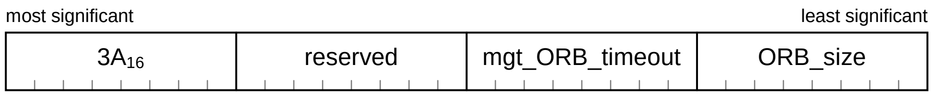

Figure 45 – Unit_Characteristics entry format

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

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

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

7.4.9 Reconnect_Timeout entry

The Reconnect_Timeout entry is an optional entry in the unit directory that describes the maximum reconnect timeout supported by a logical unit. Figure 46 shows the format of this entry.

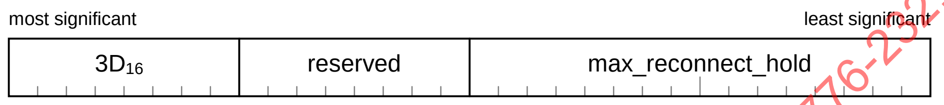

Figure 46 – Reconnect_Timeout entry format

$3D_{16}$ is the concatenation of *key_type* and *key_value* for the Reconnect_Timeout entry.

The *max_reconnect_hold* field specifies the maximum value of *reconnect_hold* that the target may return in login response data (see 5.1.3.1). If this entry is not present in configuration ROM, either the target does not include *reconnect_hold* in login response data or the value returned is always zero.

7.4.10 Logical_Unit_Directory entry

The Logical_Unit_Directory entry is an optional directory entry in the unit directory that describes the location of the logical unit directory within configuration ROM. Figure 47 shows the format of this entry.

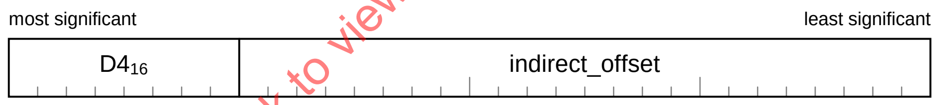

Figure 47 – Logical_Unit_Directory entry format

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

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

7.4.11 Logical_Unit_Number entry

The Logical_Unit_Number entry is an immediate entry that, when present in the unit directory, specifies the characteristics, peripheral device type and logical unit number of a logical unit implemented by the target. Figure 48 shows the format of this entry.

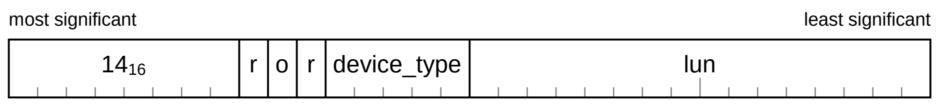

Figure 48 – Logical_Unit_Number entry format

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

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

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

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

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

7.4.12 Unit_Unique_ID entry

The Unit_Unique_ID entry is an optional leaf entry in the unit directory that describes the location of the unit unique ID leaf within configuration ROM. If a vendor implements a device with multiple Serial Bus access paths, i.e., multiple links to Serial Bus each of which receives a distinct *node_ID* as the result of Serial Bus initialization or bus enumeration, the Unit_Unique_ID entry shall be implemented. Figure 49 shows the format of this entry.

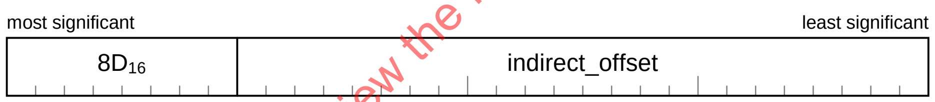

Figure 49 – Unit_Unique_ID entry format

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

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

7.5 Logical unit directory

The logical unit directory provides one of two methods by which a logical unit implemented by the target may be described (the other is a Logical_Unit_Number entry in the unit directory, already described in 7.4.11). The entries permitted in a logical unit directory are summarized in the table below.

Entry	Required
Command_Set_Spec_ID	Optional
Command_Set	Optional
Command_Set_Revision	Optional
Logical_Unit_Number	Mandatory

7.5.1 Command_Set_Spec_ID entry

The Command_Set_Spec_ID entry is an immediate entry that, when present in a logical unit directory, specifies the organization responsible for the command set definition for the logical unit. If there is no Command_Set_Spec_ID entry in the logical unit directory, the Command_Set_Spec_ID entry in the unit directory shall apply; otherwise, the entry in the logical unit directory shall take precedence. Figure 40 shows the format of this entry; the fields are defined in 7.4.3.

7.5.2 Command_Set entry

The Command_Set entry is an immediate entry that, when present in a logical unit directory and in combination with the *command_set_spec_ID*, specifies the command set implemented by the logical unit. If there is no Command_Set entry in the logical unit directory, the Command_Set entry in the unit directory shall apply; otherwise, the entry in the logical unit directory shall take precedence. Figure 41 shows the format of this entry; the fields are defined in 7.4.4.

7.5.3 Command_Set_Revision entry

The Command_Set_Revision entry is an immediate entry that, when present in a logical unit directory, specifies the revision level of the command set implemented by the logical unit. If there is no Command_Set_Revision entry in the logical unit directory, the Command_Set_Revision entry in the unit directory shall apply; otherwise, the entry in the logical unit directory shall take precedence. Figure 42 shows the format of this entry; the fields are defined in 7.4.5.

7.5.4 Logical_Unit_Number entry

The Logical_Unit_Number entry is an immediate entry in a logical unit directory that specifies peripheral device type and logical unit number of the logical unit implementation. Figure 48 shows the format of this entry; the fields are defined in 7.4.11.

7.6 Unit unique ID leaf

Although the node unique ID (EUI-64) present in the bus information block is sufficient to uniquely identify nodes attached to Serial Bus, it is insufficient to identify a target when a vendor implements a device with multiple Serial Bus node connections. In this case initiator software requires information by which a particular target may be uniquely identified, regardless of the Serial Bus access path used. Figure 50 shows the format of the unit unique ID leaf.

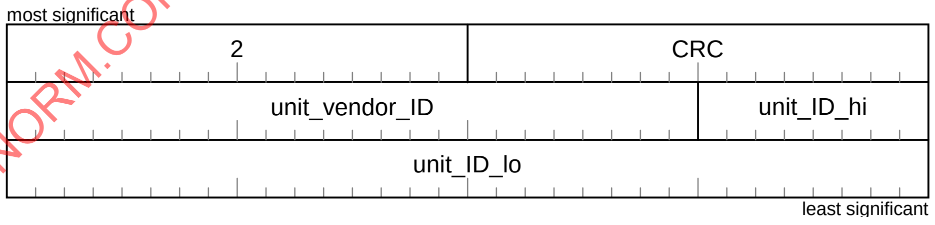

Figure 50 – Unit unique ID leaf format

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

The *unit_vendor_ID* value shall be uniquely assigned by the IEEE/RAC, as specified by ISO/IEC 13213.

The *unit_ID_hi* and *unit_ID_lo* fields are concatenated to form a 40-bit unit ID value. The vendor specified by *unit_vendor_ID* shall administer the unit ID values. When appended to the *unit_vendor_ID* value, these shall form a unique 64-bit value referred to as the unit unique ID.

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

8 Access

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

8.1 Access protocols

Targets shall implement a logical unit reservation protocol that may be used to guarantee single initiator access to the logical unit and to preserve that initiator's access rights across a Serial Bus reset. Targets may optionally implement the extensions to the logical unit reservation protocol specified by Annex C, which support both passwords and persistent reservations. Neither of these mechanisms preclude additional, command-set-dependent methods that control access to a target.

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

– The target implements a set of one or more *login_descriptors* that are used to hold context for logins. The context of a login stored in a *login_descriptor* consists of the *lun*, the *login_owner_ID*, the *login_owner_EUI_64*, the *status_FIFO* address, an *exclusive* variable, the base addresses of the fetch agent CSRs, the *login_ID* to be used by the initiator to identify the login and the *reconnect_hold* period guaranteed by the target — these last three are returned to the initiator in the *login_response* data.

– The *login_owner_ID* is the 16-bit node ID of the current owner of a login. Upon either a Serial Bus reset or a power reset, the *login_owner_ID* for all *login_descriptors* shall be reset to all ones. The target shall use the *login_owner_ID* to qualify all write requests addressed to the *login_descriptor* fetch agent CSRs.

– The *login_owner_EUI_64* is the unique 64-bit identifier of the current owner of a login. Upon a power reset, the *login_owner_EUI_64* for all *login_descriptors* shall be reset to all ones. Upon a Serial Bus reset, the *login_owner_EUI_64* persists for *reconnect_hold* + 1 seconds and shall then be reset to all ones unless it has been reestablished.

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

8.2 Login

Before an initiator may signal any other requests to a target, it shall first perform a login. The login request, whose format is specified in 5.1.3.1, shall be signaled to the target's MANAGEMENT_AGENT register by means of an 8-byte block write transaction that specifies the Serial Bus address of the login request. The address of the management agent shall be obtained from configuration ROM.

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

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

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

- the target shall read the initiator's unique ID, EUI-64, from the bus information block by means of two quadlet read transactions. The *source_ID* from the write transaction used to signal the login ORB to the target's MANAGEMENT_AGENT register shall be used as the *destination_ID* in the quadlet read transactions;

  the target shall determine whether or not the initiator already owns a login by comparing the EUI-64 just obtained against the *login_owner_EUI_64* for all *login_descriptors*. If the initiator is currently logged-in to the same logical unit, the login request shall be rejected with an *sbp_status* of access denied;
- if the *exclusive* bit is set in the login ORB and there are any active *login_descriptors* for the logical unit, the target shall reject the login request with an *sbp_status* of access denied;
- if an active *login_descriptor* with the *exclusive* attribute exists for the *lun* specified in the login ORB, the target shall reject the login request with an *sbp_status* of access denied; else
- the target shall determine if a free *login_descriptor* is available and, if none are available, reject the login request with an *sbp_status* of resources unavailable.

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

If the target is able to satisfy the login request, it shall return a login response as specified in 5.1.3.1. A critical component of a login response returned to the initiator is the base address of the target agent that the initiator shall use to signal any subsequent requests to the target for the indicated *login_ID*.

8.3 Reconnection

Upon a Serial Bus reset, the target shall abort all task sets for all command block agents created as the result of login request(s).

For each login, the target shall retain, for at least *reconnect_hold* + 1 seconds subsequent to a bus reset, sufficient information to permit an initiator to reconnect its login. After this time, but within *reconnect_hold* + 2 seconds, the target shall perform an implicit logout for each login ID that has not been successfully reconnected to its original initiator. The *reconnect_hold* parameter is communicated from the target to the initiator as part of the login response data.

After a Serial Bus reset, an initiator should not signal requests for an otherwise valid login until it first performs a reconnect. The reconnect request, whose format is specified in 5.1.3.3, shall be signaled to the target's MANAGEMENT_AGENT register by means of an 8-byte block write transaction that specifies the Serial Bus address of the reconnect ORB. The address of the management agent is that previously obtained by the initiator from the target's configuration ROM.

The speed at which the block write request to the MANAGEMENT_AGENT register is received shall determine the speed used by the target for all subsequent requests to read the initiator's configuration ROM, fetch ORB's from initiator memory or store status at the initiator's *status_FIFO*. This replaces the speed most recently obtained from the prior login or reconnect request.

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

– the target shall read the initiator's unique ID, EUI-64, from the bus information block by means of two quadlet read transactions. The *source_ID* from the write transaction used to signal the reconnect ORB to the target's MANAGEMENT_AGENT register shall be used as the *destination_ID* in the quadlet read transactions;

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

If the *login_ID* is valid, the target shall update *login_owner_ID* in the referenced *login_descriptor* with the initiator's *source_ID*.

Upon successful completion of a reconnect request, the fetch agent associated with *login_ID* shall be in the reset state. No *login_response* data is stored for a reconnect request; the completion status is indicated by the status block stored at the *status_FIFO* address.

8.4 Logout

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

9 Command execution

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

9.1 Requests and request lists

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

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

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

The target is responsible to fetch ORB's from system memory, as described in more detail in 9.1.4. The remainder of this subclause describes what the initiator shall do to:

– initialize a target fetch agent;

– dynamically append new requests to an active list and notify a target fetch agent of the new requests; and

– notify a target fetch agent of a single new request.

9.1.1 Fetch agent initialization (informative)

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

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

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

c) the initiator writes the address of the dummy ORB to the fetch agent's ORB_POINTER register by means of an 8-byte block write request. In the example in Figure 51, this is the value 0000 0000 8004 00C0$_{16}$. This causes the fetch agent to transition to the ACTIVE state.

The figure below illustrates the result of these actions:

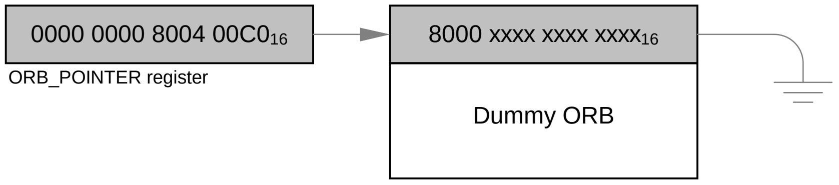

Figure 51 – Fetch agent initialization with a dummy ORB

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

NOTE – Initialization does not require that the first command block signaled to a fetch agent be a dummy ORB nor that the list contain only one ORB. A linked list with more than one command may be used both to initialize the fetch engine and to execute the commands.

9.1.2 Dynamic appends to request lists (informative)

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

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

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

b) the initiator updates the *next_ORB* field of what had been the last ORB, in this example the dummy ORB in Figure 51, with the address of the first request in the new request list, in this example 0000 0000 8000 0000$_{16}$; and

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

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

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

9.1.3 Fetch agent use by the BIOS (informative)

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

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

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

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

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

9.1.4 Fetch agent state machine

The operations of a target fetch agent are specified in Figure 52. The state of a fetch agent is visible in the context displayed by the AGENT_STATE and ORB_POINTER registers described in 6.4. The state machine diagram and accompanying text explicitly specify the conditions for transition from one state to another and the actions taken within states.

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

– an acknowledgment of *ack_type_error*;

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

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

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

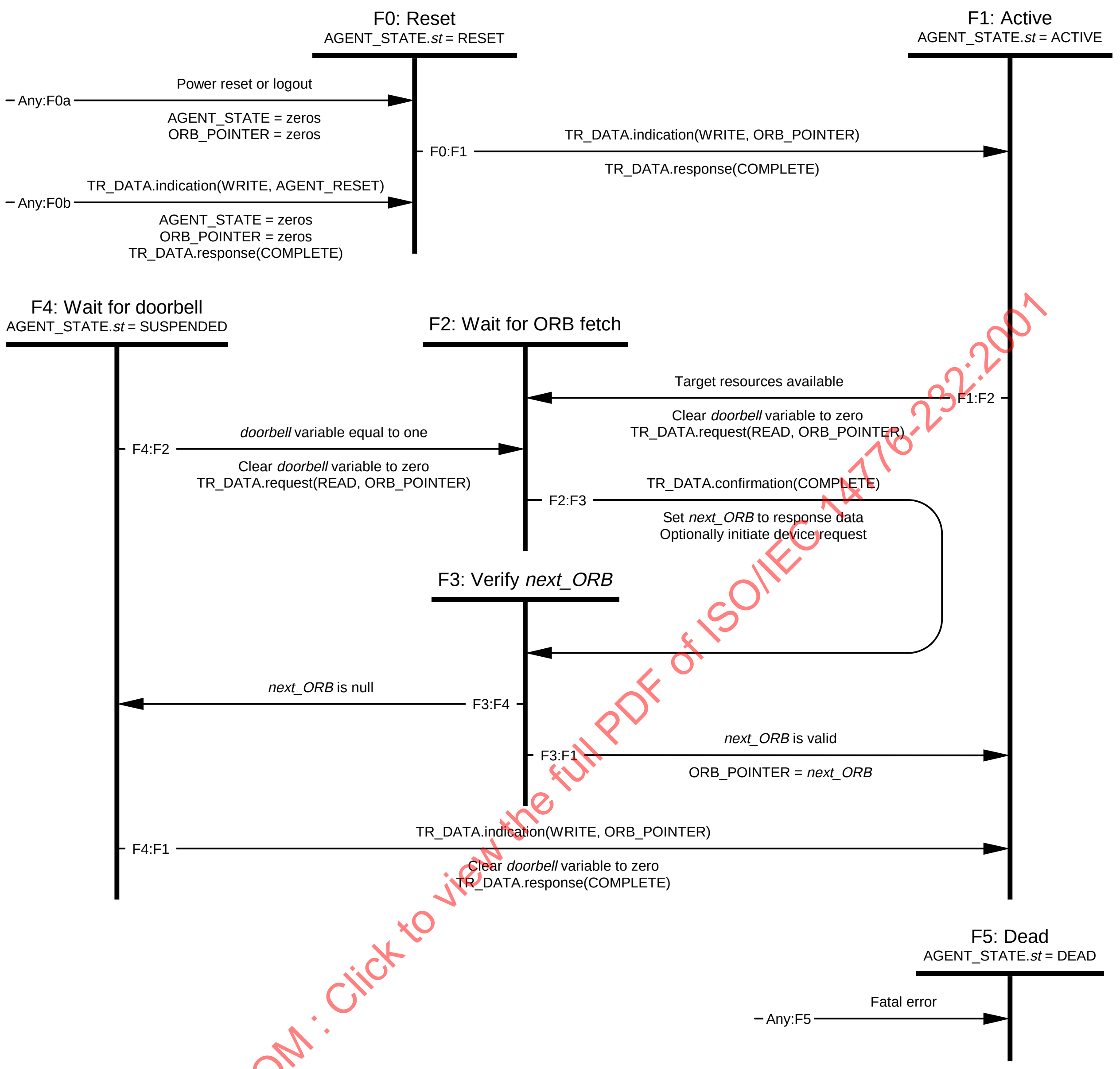
F0: Reset
AGENT_STATE.st = RESET
F1: Active
AGENT_STATE.st = ACTIVE
Any:F0a
Power reset or logout
AGENT_STATE = zeros
ORB_POINTER = zeros
F0:F1
TR_DATA.indication(WRITE, ORB_POINTER)
TR_DATA.response(COMPLETE)
Any:F0b
TR_DATA.indication(WRITE, AGENT_RESET)
AGENT_STATE = zeros
ORB_POINTER = zeros
TR_DATA.response(COMPLETE)
F4: Wait for doorbell
AGENT_STATE.st = SUSPENDED
F2: Wait for ORB fetch
Target resources available
F1:F2
Clear doorbell variable to zero
TR_DATA.request(READ, ORB_POINTER)
F4:F2
doorbell variable equal to one
Clear doorbell variable to zero
TR_DATA.request(READ, ORB_POINTER)
F2:F3
TR_DATA.confirmation(COMPLETE)
Set next_ORB to response data
Optionally initiate device request
F3: Verify next_ORB
F3:F4
next_ORB is null
F3:F1
next_ORB is valid
ORB_POINTER = next_ORB
F4:F1
TR_DATA.indication(WRITE, ORB_POINTER)
Clear doorbell variable to zero
TR_DATA.response(COMPLETE)
F5: Dead
AGENT_STATE.st = DEAD
Any:F5
Fatal error

Figure 52 – Fetch agent state machine

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

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

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

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

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

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

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

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

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

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

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

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

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

Transition F4:F1. If an indication of a write to the ORB_POINTER register is received, the fetch agent shall clear the *doorbell* variable to zero and confirm the write transaction with a response subaction of COMPLETE. After the confirmation, the fetch agent shall transition to state F1.

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

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

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

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

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

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

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

9.2 Data transfer

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

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

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

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

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

9.3 Completion status

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

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

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

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

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

The return of completion status indicates to the initiator that the task commenced by the ORB is no longer part of the task set. The status block also specifies whether or not the system memory allocated to the ORB may be released. If the *src* field has a value of zero, the initiator may reuse or deallocate the system memory occupied by an ORB. When *src* has a value of one, the system memory shall not be reused or deallocated until the target stores completion status for some subsequent ORB in the linked list.

NOTE – For targets that support the ordered model of task execution, the return of completion status for an ORB implicitly indicates that all preceding ORB's in the linked list have completed successfully, are no longer part of the task set and that the initiator may reuse or deallocate their system memory.

9.4 Unsolicited status

In addition to status associated with a particular ORB, described in the preceding subclause, a fetch agent may store unsolicited status at the address specified by *status_FIFO*. A status block that contains unsolicited status shall be identified by setting the *src* field to a value of two to indicate the unsolicited information is device status (see 5.3).

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

NOTE – One use for unsolicited status is to report progress of lengthy operations such as a disk format or tape rewind. Device implementers that use unsolicited status for this purpose should pick an appropriate interval for the reports. Frequent unsolicited status transfers reduce available Serial Bus bandwidth and may increase processing overhead at the initiator without any perceivable benefits.

The action taken by a target when unsolicited status is generated but cannot be stored because the *unsolicited status enable* variable is zero depends upon the nature of the status. If the status is for a unit attention condition, the target shall retain the information with the intent to store it as soon as the *unsolicited status enable* variable is set to one. The unit attention condition shall persist until the corresponding status block is stored at the initiator's *status_FIFO*. The definition of unit attention conditions is beyond the scope of SBP-2 and is usually the province of the command-set standard for the target. Other status information, that does not constitute a unit attention, may be discarded by the target, queued for future delivery or replace an existing, pending status of the same type. Which of these behaviors is appropriate is determined by the command set standard.

10 Task management

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

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

10.1 Task sets

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

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

10.2 Basic task management model

Targets shall support, at a minimum, a basic task management model. Targets may implement other task management models so long as they support all of the features of the basic model. Within the basic model, the following rules apply:

– all tasks within a task set share the same execution characteristics: either they are all reorderable or else they are all ordered;

– the reorderable or ordered execution characteristics of a task set are implicit in the target implementation and are not subject to control by the initiator. Configuration ROM shall specify whether the target may reorder task execution or not;

– all tasks within a task set are uniquely identified by the Serial Bus address of the ORB that initiated the task. This address shall be unique for the life of the task;

– the abort task, abort task set and target reset task management functions, described later in this clause, shall be implemented.

An element of choice in the implementation of a task set under the basic model is whether or not the target may reorder task execution. An 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. An ordered model may be more appropriate for devices, such as sequential storage, where the outcome of one command affects the next.

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

The ordered model requires both that tasks shall be executed in order and that completion status shall be returned in order. Because Serial Bus transactions may complete out of order, the target shall single-thread the return of completion status. Once the target has transmitted a Serial Bus block write request to store completion status in system memory, it shall not attempt to store completion status for any other task in the task set until *ack_complete* or *resp_complete* is received.

10.3 Error conditions

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

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

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

c) finally, the target shall return error completion status for the faulted task. The *dead* bit shall be one in the status block.

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

10.4 Task management requests

The subclauses that follow describe the use of the *rq_fmt* field in ORBs and the task management ORBs defined in 5.1.3.5.

10.4.1 Abort task

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

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

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

b) the initiator may construct a management ORB in system memory for the abort task function. The initiator shall set the appropriate values in the *rq_fmt*, *login_ID* and *ORB_offset* fields of the ORB, as described in 5.1.3.5. The *function* field shall be set to ABORT TASK; *ORB_offset* shall contain the Serial Bus address of the ORB for the task to be aborted. The initiator then signals the abort task management ORB to the management agent.

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

– if the ORB to be aborted has already been fetched by the target, the task may be completed by the target without recognition of the abort task request; otherwise

– when the ORB is first fetched, the target shall recognize the *rq_fmt* field value of three and shall not execute the command. That target shall store completion status for the aborted ORB; the request status shall be REQUEST COMPLETE and the *sbp_status* field shall indicate dummy ORB completed.

A second method to abort task(s) may be available by means of task management ORB's with a *function* of ABORT TASK. Target support for this method of abort task is optional. Targets that implement this method shall store a completion status of REQUEST COMPLETE for the abort task request in the status buffer specified by the ORB.

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

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

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

b) the target shall wait for response subactions to pending data transfer requests;

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

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

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

10.4.2 Abort task set

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

To abort a task set, the initiator shall construct a management ORB in system memory for the abort task set function. The initiator shall set the appropriate values in the *rq_fmt* and *login_ID* fields of the ORB, as described in 5.1.3.5. The *function* field shall be set to ABORT TASK SET.

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

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

a) the target shall halt the operation of the fetch agent associated with the *login_ID* by making a transition to the DEAD state;

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

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

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

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

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

10.4.3 Logical unit reset

Logical unit reset is a task management function that causes a logical unit to perform the actions described below and to create unit attention conditions for all initiators logged-in to the logical unit. Support for logical unit reset is a target option.

To reset a logical unit, the initiator shall construct a management ORB in system memory for the logical unit reset function. The initiator shall set the appropriate values in the *rq_fmt* and *login_ID* fields of the ORB, as described in 5.1.3.5. The *function* field shall be set to LOGICAL UNIT RESET.

The initiator shall signal the logical unit reset ORB to the management agent.

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

a) the target shall halt the operation of all of the logical unit's fetch agents by making transitions to the DEAD state;

b) the target shall not issue data transfer requests for any task in any of the logical unit's task sets;

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

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

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

10.4.4 Target reset

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

To reset a target, the initiator shall construct a management ORB in system memory for the target reset function. The initiator shall set the appropriate values in the *rq_fmt* and *login_ID* fields of the ORB, as described in 5.1.3.5. The *function* field shall be set to TARGET RESET.

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

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

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

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

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

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

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

10.5 Task management event matrix

Common events that affect the state of target fetch agents and their associated task set(s) are summarized below. Refer to the governing subclauses in clauses 8 and 9 as well as this clause for detailed information.

Event	AGENT_STATE.*st*	Task set(s)
Power reset	RESET	Clear all task sets
Command reset (write to RESET_START)	RESET	Clear all task sets
Bus reset (immediate)	RESET	Clear all task sets
Bus reset (after *reconnect_hold* + 1 seconds)	—	Logout any initiator that has failed to successfully reconnect
Login	—	—
Reconnect	—	—
Logout	RESET	Abort initiator's task set
Fetch agent reset (write to AGENT_RESET)	RESET	Abort initiator's task set
Faulted command (CHECK CONDITION)	DEAD	Abort faulted initiator's task set
ABORT TASK	—	—
ABORT TASK SET	DEAD	Abort initiator's task set
LOGICAL UNIT RESET	DEAD	Abort all the logical unit's task sets
TARGET RESET	DEAD	Clear all task sets

When an event affects more than one task set, all of the associated fetch agents transition to the state indicated by the table. With respect to events supported by the target's management agent, e.g., logout, there is an assumption of successful completion. In the case of a function rejected response or other indication of failure, the preceding table does not apply.

Immediately upon detection of a bus reset, all command block fetch agents transition to the reset state and their associated task sets are cleared without the return of completion status.

For *reconnect_hold* + 1 seconds subsequent to a bus reset, targets save state information for initiators that were logged-in at the time of the bus reset. The timer commences when the target observes the first subaction gap subsequent to a bus reset; if a bus reset occurs before the timer expires, the timer is reset. If an initiator successfully completes a reconnect request during this period, the actions described in 8.3 occur. The task set is empty and, once the fetch agent is initialized, the initiator may signal new ORBs to the target.

Once *reconnect_hold* + 1 seconds have elapsed after a bus reset, the target shall automatically perform a logout operation for all login IDs that have not been reconnected with their initiator. This returns all the affected fetch agents to the reset state.

Annex A
(normative)

Minimum Serial Bus node capabilities

This annex specifies capabilities (in addition to the minimum defined by ANSI/IEEE 1394) that an initiator or target shall support in order to implement SBP-2.

Once a node that implements one or more initiator(s) or target(s) completes its power reset initialization sequence, it shall acknowledge, and subsequently respond to, Serial Bus transaction request subactions within the time limits specified by ANSI/IEEE 1394 and draft standard IEEE P1394a. A Serial Bus reset shall not alter a node's responsiveness to request subactions.

A.1 Initiator capabilities

With the exception of configuration ROM and control and status registers, an initiator shall be capable of responding to block read or write requests with a *data_length* less than or equal to 32 bytes.

An initiator shall also be capable of responding to block read requests with a *data_length* less than or equal to 4 * *ORB_size*, where *ORB_size* is obtained from the Unit_Characteristics entry in the target's configuration ROM.

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

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

A.2 Target capabilities

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

A target shall be capable of initiating write requests and shall report this by setting the *drq* bit in the Node_Capabilities entry in configuration ROM to one. Consequently, the target shall implement the *dreq* bit in the STATE_CLEAR and STATE_SET registers. The value of STATE_CLEAR.*dreq* shall be unaffected by a Serial Bus reset. The target may automatically set *dreq* to zero (request initiation enabled) upon a power reset or a command reset.

A target shall be capable of initiating block write requests with a *data_length* of at least eight bytes and shall report this by setting the value of the *max_rec* field in the bus information block in configuration ROM to a value of two.

While initializing after a power reset, a target shall respond to quadlet read requests addressed to FFFF F000 0400_{16} with either a response data value of zero or acknowledge the request subaction with *ack_tardy*, as specified by draft standard IEEE P1394a. This indicates that the remainder of configuration ROM, as well as other target CSRs, are not accessible.

A.3 Target security

As specified by draft standard IEEE P1394a, a target shall abide by the following restrictions:

- if a target’s unique ID, EUI-64, is read from the configuration ROM bus information block by quadlet read requests, the value returned shall be the EUI-64 assigned by the manufacturer;
- the target shall not originate a request or response subaction with a *source_ID* field that is not equal to either a) the most significant 16 bits of the target’s NODE_IDs register or b) the concatenation of $3FF_{16}$ and the physical ID assigned to the target’s PHY during the self-identify process; and
- the target shall not receive a request or response subaction that specifies *destination_ID* unless that field is equal to either a) the concatenation of the most significant 10 bits of the target’s NODE_IDs register and either the physical ID assigned to the target’s PHY during the self-identify process or $3F_{16}$, or b) the concatenation of $3FF_{16}$ and either the physical ID assigned to the target’s PHY during the self-identify process or $3F_{16}$.

Annex B
(normative)

SCSI command and status encapsulation

SBP-2 defines a protocol that permits initiator(s) to control the operation of devices (disks, tapes, printers, *etc.*), but it does not specify the command sets used by the devices—only the mechanisms by which commands, data and status are transported. This annex specifies how SBP-2 may be used for devices that use the SCSI command set(s). This encompasses encapsulation of SCSI command descriptor blocks (CDBs), a standard format for SCSI status and sense data and the necessary configuration ROM entries.

B.1 SCSI command descriptor block

SBP-2 provides for the transport of 6-, 10- and 12-byte SCSI CDBs within a command block ORB, as illustrated by Figure B.1. When 6- and 10-byte CDBs are encapsulated within an ORB, the least significant (unused) bytes of the ORB shall be zero.

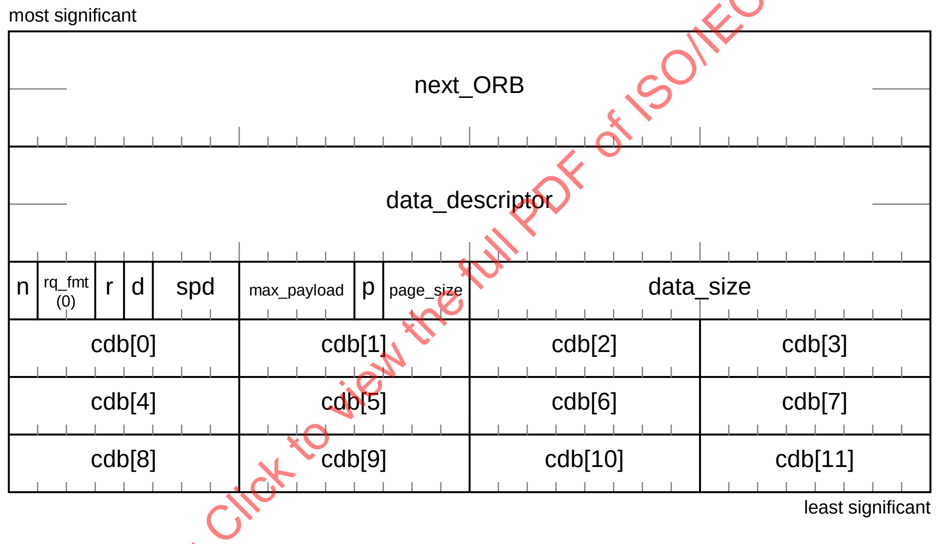

Figure B.1 – SCSI command block ORB

Parts of the control byte (the last byte of a SCSI command descriptor block) are constrained to values illustrated by Figure B.2.

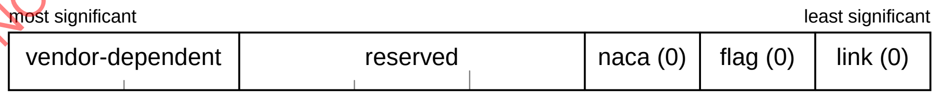

Figure B.2 – SCSI control byte

The *naca*, or normal ACA, bit shall be zero; SBP-2 supports SCSI-2 contingent allegiance.

The *flag* and *link* bits shall be zero; SBP-2 does not support linked commands.

B.2 SCSI status and sense data

Upon completion of a command, if the *notify* bit in the ORB is one or if there is exception status to report, the target shall signal the initiator by storing all or part of the status block shown by Figure B.3 at the *status_FIFO* address provided by the initiator as part of the login request.

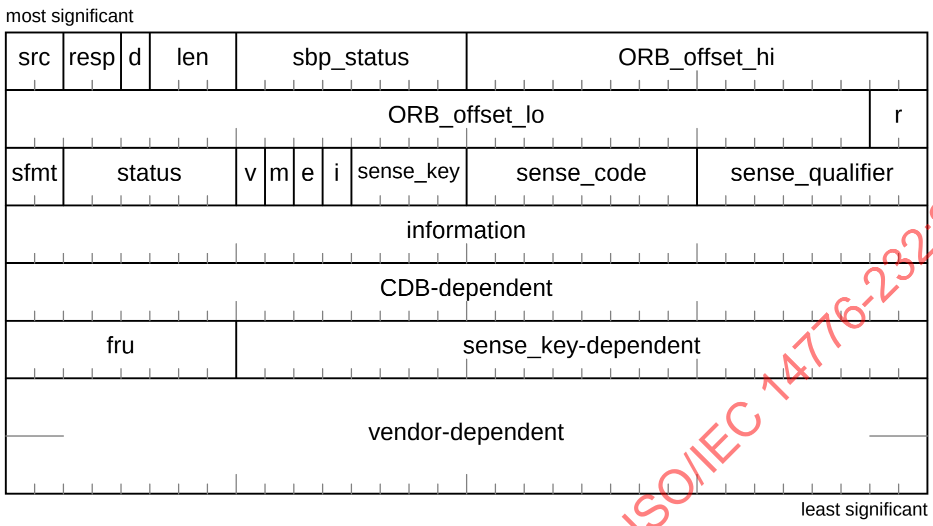

Figure B.3 – Status block format for SCSI sense data

When a command completes with GOOD status, only the first two quadlets of the status block shall be stored at the *status_FIFO* address; the *len* field shall be one. Otherwise, both SCSI status and sense data shall be stored in a status block that conforms to the format illustrated above.

SBP-2 permits the return of a status block between two and eight quadlets in length. When a truncated status block is stored, the omitted quadlets shall be interpreted as if zero values were stored.

The *src*, *resp*, *len*, *sbp_status*, *ORB_offset_hi* and *ORB_offset_lo* fields, as well as the *dead* bit (abbreviated as *d* in the figure above), are as previously described in 5.3.

The *sfmt* field shall specify the format of the status block and shall additionally indicate whether the error condition associated with *sense_key* is current or deferred. The table below defines permissible values for *sfmt*.

Value	Description
0	Current error; status block format defined by this standard
1	Deferred error; status block format defined by this standard
2	Reserved for future standardization
3	Status block format vendor-dependent

The *status* field shall contain SCSI status information as defined by SAM-2, with the exceptions noted in the table below.

Value	Description
0	GOOD
2	CHECK CONDITION
4	CONDITION MET
8	BUSY
10_{16}	Not supported by SBP-2
14_{16}	Not supported by SBP-2
18_{16}	RESERVATION CONFLICT
22_{16}	COMMAND TERMINATED
28_{16}	Not supported by SBP-2
30_{16}	Not supported by SBP-2
All other values	Reserved for future standardization

The *valid* bit (abbreviated as *v* in the figure above) shall specify the content of the *information* field. When the *valid* bit is zero, the contents of the information field are not specified. When the *sfmt* field has a value of zero or one and the *valid* bit is one, the contents of the information field shall be as defined by SPC-2 or the relevant command set standard.

The meanings of the *mark*, *eom* and *illegal_length_indicator* bits (abbreviated as *m*, *e* and *i*, respectively, in figure B.3) are defined by SPC-2 or the relevant command set standard. These bits correspond to the filemark, EOM and ILI bits defined by SPC-2 for sense data.

The *sense_key*, *sense_code* and *sense_qualifier* fields shall contain command completion information defined by SPC-2 or the relevant command set standard. These fields correspond to the sense key, additional sense code and additional sense code qualifier fields defined by SPC-2 for sense data.

The contents of the *information* field are unspecified if either the *valid* bit is zero or the *sfmt* field has a value of three. For *sfmt* values of one or two, the contents of the *information* field are device-type or command dependent and, if the *valid* bit is one, are defined within SPC-2 or the appropriate standard for the command. Characteristic uses of the *information* field are for:

– the unsigned logical block address associated with *sense_key* and the command; or

– the least significant 32-bits of the unsigned logical block address associated with *sense_key* and the command; or

– the residue of the requested data transfer length minus the actual data transfer length, in either bytes or blocks as determined by the command. Negative values are indicated in two's complement notation.

The contents of the *CDB*-dependent field are device-type or command dependent and are defined within SPC-2 or the appropriate standard for the command.

Nonzero values in the *fru* field may be used to identify a device-dependent, field replaceable mechanism or unit that has failed. A value of zero in this field shall indicate that no specific mechanism or unit has been identified to have failed or that the data is unavailable. When *fru* is nonzero, the format of the information is not specified by this standard.

When *sfmt* has a value of zero or one, the contents of the *sense_key*-dependent field are defined by SPC-2 or the relevant the command set standard. In this case, the most significant bit of the *sense_key*-dependent field is the SKSV bit defined by SPC-2. When *sfmt* is equal to three, the contents of *sense_key*-dependent are unspecified.

B.3 Configuration ROM

SCSI targets shall implement configuration ROM in accordance with clause 7 and this annex. At least one logical unit, logical unit zero, shall be implemented; additional logical units may be implemented. A logical unit is described by entries in a unit directory or by entries in a logical unit directory dependent upon the unit directory or by entries taken in combination from both places.

Mandatory and optional configuration ROM entries for SCSI targets, and the directories in which they may occur, are summarized by the table below.

Entry	Unit directory	Logical unit directory	Comments
Unit_Spec_ID	Mandatory	—	
Unit_SW_Version	Mandatory	—	
Command_Set_Spec_ID	Mandatory (in at least one directory)		See precedence rules below.
Command_Set	Mandatory (in at least one directory)		See precedence rules below.
Command_Set_Revision	Optional	Optional	See precedence rules below.
Firmware_Revision	Optional	—	
Management_Agent	Mandatory	—	
Unit_Characteristics	Mandatory	—	
Logical_Unit_Directory	Optional	—	
Logical_Unit_Number	Mandatory (in at least one directory)		See precedence rules below.
Unit_Unique_ID	Optional	—	

For the entries that may be present in either a unit directory or a logical unit directory, the entry in the logical unit directory takes precedence over that in the parent unit directory (if any). If there is no entry in the logical unit directory, the value is inherited from the corresponding entry in the parent unit directory.

The presence of one or more Logical_Unit_Number entries in a unit directory makes the Command_Set_Spec_ID and Command_Set entries mandatory in the unit directory.

Within a logical unit directory, at least one Logical_Unit_Number entry is required and the Command_Set_Spec_ID and Command_Set entries are both mandatory unless the corresponding entry is present in the parent unit directory.

B.3.1 Command_Set_Spec_ID entry

The Command_Set_Spec_ID entry is an immediate entry in either a unit or logical unit directory that specifies the organization responsible for the command set definition for the target. The format of this entry is specified by 7.4.3.

SCSI targets shall have a *command_set_spec_ID* value of 00 609E$_{16}$, which indicates that NCITS is responsible for the command set definition.

B.3.2 Command_Set entry

The Command_Set entry is an immediate entry in either a unit or logical unit directory that, in combination with the *command_set_spec_ID*, specifies the command set implemented by the target. The format of this entry is specified by 7.4.4.

SCSI targets shall have a *command_set* value of 01 $04D8_{16}$, which indicates that the target's command set is specified by SCSI Primary Commands 2 (SPC-2) and related command set standard(s)—as determined by the target's peripheral device type(s). In addition, this *command_set* value specifies that the target conforms to all requirements of this annex.

B.3.3 Logical_Unit_Number entry

The Logical_Unit_Number entry is an immediate entry in either a unit or logical unit directory that specifies the peripheral device type and logical unit number of a logical unit implemented by the SCSI target. The format of this entry is specified by 7.4.11.

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

Value	Peripheral device type
0 – $1E_{16}$	The value of *device_type* shall have the same meaning as the peripheral device type field returned in INQUIRY data as specified by SPC-2
$1F_{16}$	Unknown device type

Annex C
(normative)

Security extensions

SBP-2 specifies an access protocol, in clause 8, that by itself makes no provisions for security. This annex defines extensions to SBP-2 that may be implemented by targets to provide some measure of security. Targets that implement these security extensions shall conform to all provisions of this annex.

Conformance to this annex does not preclude additional, command-set-dependent security facilities.

C.1 Passwords

A target shall implement two passwords:

– the master password, which shall be unchangeable and equal to the target serial number. The target serial number should be in a humanly readable form affixed to the target. The master password shall not be readable *via* the target's Serial Bus interface except by a logged-in initiator; and

– the current password, which shall accommodate 28 bytes of password data and shall be alterable only by the set password function (see clause C.3).

All password values shall be unchanged by power reset, bus reset or command reset.

The value of the master password shall be obtainable by command set-dependent means.

A target may be manufactured with a current password of all zeros, with the expectation that the user will assign a nonzero current password as part of target initialization. If a target is manufactured with a nonzero current password, the target shall be shipped with the current password in a humanly readable form.

C.2 Login

The description of the login protocol below reproduces that specified by clause 8 and adds validation of cumulative login attempts and the *password* field from the login request. The target shall implement an internal counter, *login_attempts*, which shall be zeroed upon a power reset or upon a successful login or logout request. The target shall perform the following (in any order) to validate a login request:

– the target shall reject the login request if *login_attempts* is equal to three;

– the target shall read the initiator's unique ID, EUI-64, from the bus information block by means of two quadlet read transactions. The *source_ID* from the write transaction used to signal the login ORB to the target's MANAGEMENT_AGENT register shall be used as the *destination_ID* in the quadlet read transactions;

  the target shall determine whether or not the initiator already owns a login by comparing the EUI-64 just obtained against the *login_owner_EUI_64* for all *login_descriptors*. If the initiator is currently logged-in to the same logical unit, the login request shall be rejected with an *sbp_status* of access denied but *login_attempts* shall not be incremented;

– the target shall validate the password provided by the login request. If *password_length* is zero, the password is eight bytes of immediate data present in the *password* field. Otherwise, *password_length* specifies the size of the password addressed by *password*. If *password_length* is greater than 28 the target shall increment *login_attempts* and reject the login request with an *sbp_status* of access denied. When *password_length* is valid, the password provided is extended to