CONGRUENT SOFTWARE, INC.
98 Colorado Avenue
Berkeley, CA 94707
(510) 527-3926
(510) 527-3856 FAX

FROM: Peter Johansson

TO: 1394 Printer Working Group

DATE: February 28, 1999

RE: Merged documents for proposed draft

The updated draft that follows is an attempt to merge many of the contributions made to the 1394 PWG to date and to place them in a form close to an eventual standards document. It also reflects some useful brainstorming that has occurred in private.

Although this is the first revision of a previously issued document, there has been no opportunity for review of its progenitor and it is reasonable to assume that review will commence with this document. Consequently, I have not used change bars in this draft.

Contents

Page

Tables

Figures

Annexes

1 Scope and purpose

1.1 Scope

1.2 Purpose

2 Normative references

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

Copies of the following documents can be obtained from ANSI:

Approved ANSI standards;

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

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

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

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

2.1 Approved references

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

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

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

ISO/IEC 9899:1990, Programming Languages—C

2.2 References under development

At the time of publication, the following referenced standards were still under development.

IEEE P1212r, Draft Standard (Reaffirmation) for a Control and Status Register (CSR) Architecture for Microcomputer Buses

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

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 as 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 connection: A queue or a pair of queue(s) that affords access to a service.

3.1.2.3 function: A capability of the device expressed as a unit architecture (unit directory) with a single logical unit (LU zero).

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

3.1.2.5 initiator control request ORB: An ORB whose *queue* field and *direction* bit are zero, whose *control* and *end_of_message* bits are one and whose buffer contains control information with the *rq* bit set to one.

3.1.2.6 initiator control response ORB: An ORB whose *queue* field and *direction* bit are zero, whose *control* and *end_of_message* bits are one and whose buffer contains control information with the *rq* bit zeroed.

3.1.2.7 logical unit: The part of the unit architecture that provides access to one or more services. Devices compliant with this standard implement one logical unit with a LUN of zero.

3.1.2.8 login: The process by which an initiator obtains access to a target fetch agent. The target fetch agent and its control and status registers provide a mechanism for an initiator to signal ORBs to the target.

3.1.2.9 node: An addressable device attached to Serial Bus.

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

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

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

3.1.2.13 queue: An ordered set of ORBs within a task set that does not block with respect to other queues that are part of the same task set.

3.1.2.14 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.

3.1.2.15 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.

3.1.2.16 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.

3.1.2.17 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.

3.1.2.18 service: A protocol used to control an independently operable component of a function.

3.1.2.19 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.

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

3.1.2.21 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).

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

3.1.2.23 target: A node that receives device service or management requests from an initiator. In the case of control operation or transport flow requests, the ORBs are directed to the target's logical unit zero to be executed. Management requests are serviced by the target. A CSR Architecture unit is synonymous with a target.

3.1.2.24 target control request ORB: An ORB whose *queue* field is zero, whose *direction*, *control* and *end_of_message* bits are one and whose buffer contains control information with the *rq* bit set to one.

3.1.2.25 target control response ORB: An ORB whose *queue* field is zero, whose *direction*, *control* and *end_of_message* bits are one and whose buffer contains control information with the *rq* bit zeroed.

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. ANSI NCTIS.325-1998 specifies some dependencies between individual tasks within the task set and this standard mandates others.

3.1.2.28 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.

3.1.2.29 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.

3.1.2.30 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 image devices (*e.g.*, printers, scanners or multifunction peripherals) intended to be used with the unit architecture for SBP-2 targets.

3.1.2.31 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.

3.1.2.32 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 are abbreviations that 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
SBP-2	ANSI NCITS.325-1998

3.2 Notation

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

3.2.1 Numeric values

Decimal and hexadecimal numbers are used within this standard. By editorial convention, decimal numbers are most frequently used to represent quantities or counts. Addresses are uniformly represented by hexadecimal numbers, which are also used when the value represented has an underlying structure that is more apparent in a hexadecimal format than in a decimal format.

Decimal numbers are represented by Arabic numerals without subscripts or by their English names. Hexadecimal numbers are represented by digits from the character set 0 – 9 and A – F followed by the subscript 16. When the subscript is unnecessary to disambiguate the base of the number it may be omitted. For the sake of legibility, hexadecimal numbers are separated into groups of four digits separated by spaces.

As an example, 42 and $2A_{16}$ both represent the same numeric value.

3.2.2 Bit, byte and quadlet ordering

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

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

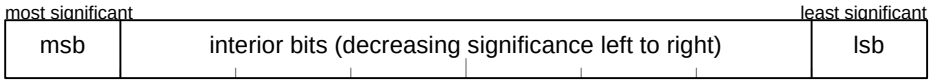

Figure 1 – Bit ordering within a byte

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

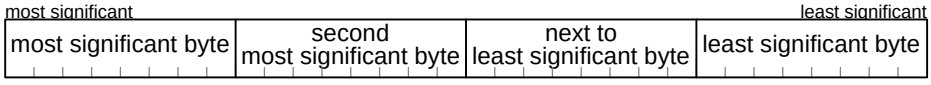

Figure 2 – Byte ordering within a quadlet

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

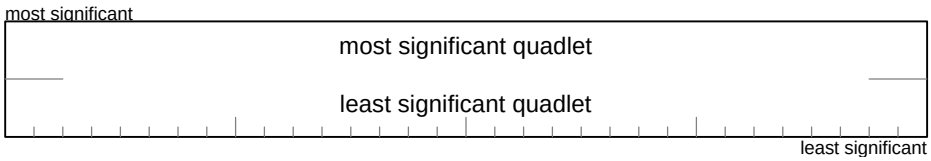

Figure 3 – Quadlet ordering within an octlet

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.

4 Model (informative)

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

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

4.1 Protocol stack and service model

The relationship between the initiator and target may be modeled as a software stack present in both devices, as shown by Figure 4 and Figure 5 below. The physical interconnection, *via* Serial Bus, exists at the lowest protocol level. This document defines the data structures and methods necessary to implement the shaded levels in the protocol stacks. Note that client application(s) may reside at either the initiator or the target (they are commonly found at the initiator) and the service(s) at the corresponding SBP-2 functional role, target or initiator.

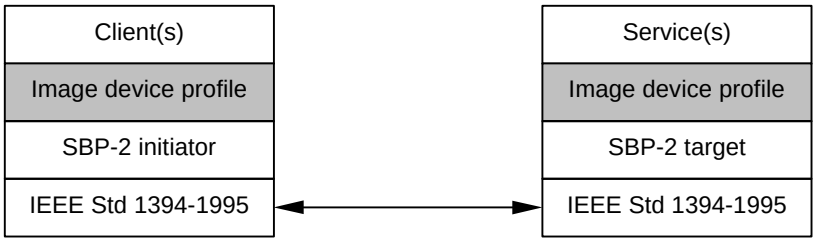

Figure 4 – Protocol stack (service at target)

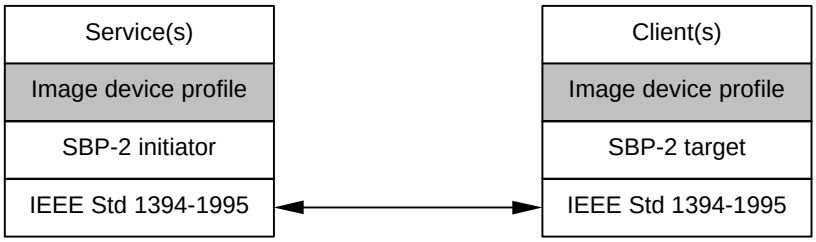

Figure 5 – Protocol stack (service at initiator)

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

function: A capability of the device expressed as a unit architecture (unit directory) that contains a single logical unit (LU zero);

service: A protocol used to control an independently operable component of a function;

management service: A mandatory service provided for each function; it executes control requests to establish or terminate connections to the other services of the function. The connection to this function is implicitly established as the result of an SBP-2 login;

queue: An ordered set of ORBs within a task set that does not block with respect to other queues that are part of the same task set; and

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

4.2 Independent data paths for each service

SBP-2 describes all the work to be performed by a particular logical unit as a *task set*, a collection of ORBs linked together as shown by Figure 6.

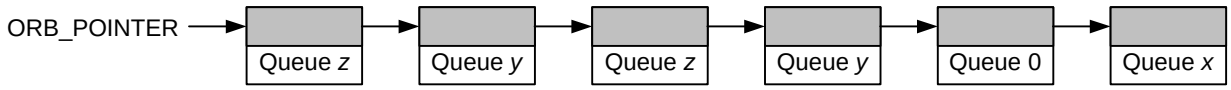

Figure 6 – Multiplexed queues in an SBP-2 task set

Because a single device function (logical unit) may be implemented as one or more services, each of which may require an independent uni- or bi-directional transport flow, this document extends SBP-2 to permit multiplexed *queues* within a single task set; each ORB in the task set is tagged to identify the logical queue to which it belongs. Although the target may in general reorder the execution of ORBs within the task set, all of the ORBs within a particular queue are executed in order. Within this framework, both the initiator and the target manage the single task set illustrated above as the collection of logically independent queues illustrated below.

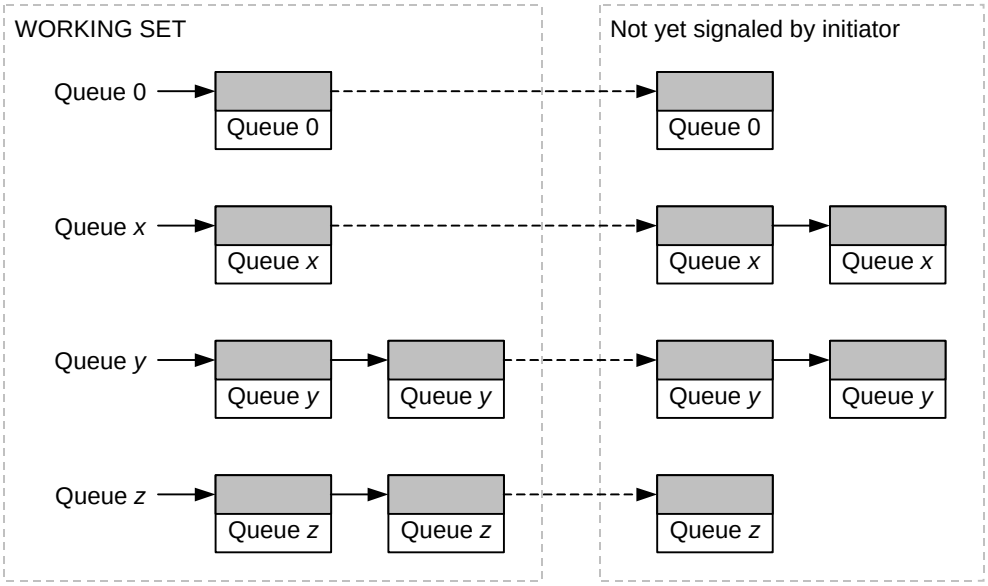

Figure 7 – Independent queues (logical model)

In theory the size of an SBP-2 task set is bounded only by the amount of memory available to the initiator to store ORBs; in practice targets have sufficient memory to fetch only a subset of the task set, the *working set*. Nonblocking behavior between the separate queues is achieved through cooperative use of the target's working set. If the initiator never places more ORBs in the task set than the target can accommodate in its working set, all outstanding ORBs may be fetched by the target and made available for execution. Equipped with this knowledge, the initiator may restrict the number of outstanding ORBs on a queue by queue basis so that a *task slot* in the working set is always available for each queue. Nonblocking bi-directional data transfer between initiator and target may be accomplished through the use of two queues, one for each direction.

4.3 Connection management

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

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

Connections may be one of three different types:

- Unidirectional; the application data flow is one direction, either from the initiator to the target or *vice versa*;
- Bi-directional (nonblocking); the application data flows in both directions with one queue used for each of the directions; or
- Bi-directional (blocking); the data flows in both directions *via* a single queue which has the potential to block. Nonblocking behavior is not guaranteed by the transport but must be a property of the application itself. The queue used for management services is an example of a bi-directional, blocking queue, but because it is restricted to single-threaded, serialized use it cannot block.

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

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

4.4 Data transfer between initiator and target

4.5 Control requests and responses

4.6 Unsolicited status

5 Data structures

This document defines the format of those parts of the SBP-2 ORB and status block reserved by ANSI NCITS.325-1998 for specification by command set standards. It also defines a format for control information transferred between initiator and target. All data structures defined in the following clauses shall be aligned on quadlet boundaries.

5.1 Transport flow ORBs

ANSI NCITS.325-1998 defines command block ORBs for SBP-2 devices; these have a common 20-byte header and leave the definition of the subsequent quadlets to individual command set standards. Image devices compliant with this standard shall use 32-byte command block ORBs (renamed transport flow ORBs to emphasize their function) whose format is illustrated by Figure 8. Transport flow ORBs are used to regulate the transfer of application data or control information between initiator and target.

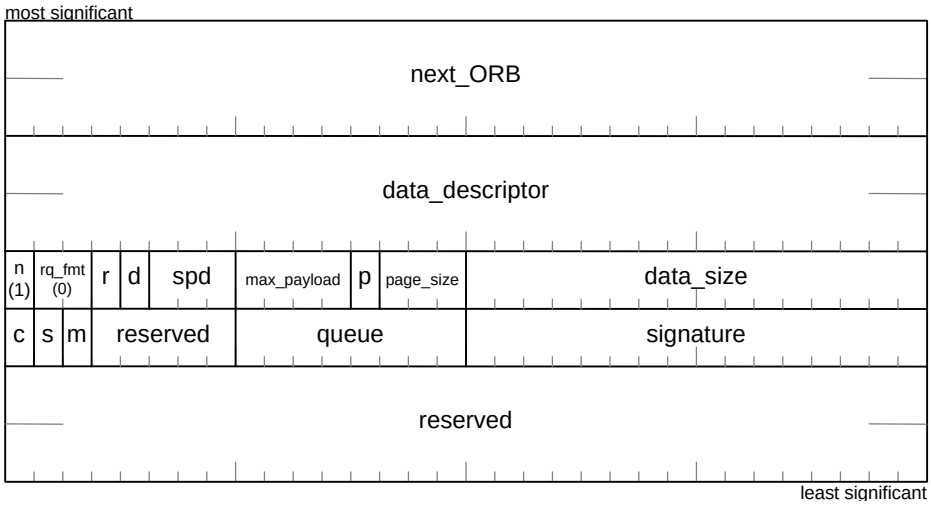

Figure 8 – Transport flow ORB

The usage of the *next_ORB*, *data_descriptor*, *rq_fmt*, *spd*, *max_payload*, *page_size* and *data_size* fields and the *notify* and *page_table_present* bits (abbreviated as *n* and *p*, respectively, in the figure above) is defined by ANSI NCITS.325-1998. The *notify* bit shall be one and the *rq_fmt* field shall be zero.

The *direction* bit (abbreviated as *d* in the figure above) shall specify the direction of data transfer for the buffer described by *data_descriptor*. If the *direction* bit is zero, the target shall use Serial Bus read transactions to fetch data from the buffer (the flow direction is from the initiator to the target). Otherwise, when the *direction* bit is one, the target shall use Serial Bus write transactions to store data in the buffer (the flow direction is from the target to the initiator).

The *control* bit (abbreviated as *c* in the figure above) shall specify the usage of the buffer described by *data_descriptor*. If the *control* bit is zero, the buffer shall be used for application data exchanged between the client and service. Otherwise, when the *control* bit is one the buffer shall be used for transport control information.

The *special* bit (abbreviated as *s* in the figure above) provides additional information pertinent to application data transferred from the initiator to the target. The meaning of the *special* bit is unspecified when either of the *direction* or *control* bits are one. Otherwise the meaning and usage of the *special* bit are

application-dependent and shall apply to all of the application data contained within the buffer described by the ORB.

NOTE – Stream socket abstractions include the notion of *out of band* data, as some transport protocols allow portions of incoming data to be marked as "special" in some way. These special data blocks may be delivered to the user out of the normal sequence—for example, expedited data in X.25 and other OSI protocols or the use of urgent data in TCP by BSD Unix. The *special* bit enables such usage to be mapped to a transport protocol based on SBP-2.

The *end_of_message* bit (abbreviated as *m* in the figure above) shall indicate whether or not a boundary exists in the application data or control information transferred from the initiator to the target. The meaning of the *end_of_message* bit is unspecified when the *direction* bit is one. Otherwise, when *end_of_message* is one, a boundary exists after the last byte of application data or control information described by the ORB. In the case of application data, the nature of the boundary and its interpretation shall be specified by the service definition. When the *control* bit is one, the *end_of_message* bit shall also be one; all control information for a single request or response shall be contained within one buffer.

NOTE – Need an example illustrative of the use of the *end_of_message* bit.

The *queue* field shall specify a queue number assigned the target in either a CONNECT request or response.

The *signature* field shall contain an identifying number assigned by the initiator and shall be unique within the context of a queue. This field is used to facilitate the resumption of data transfer after a bus reset or other transient interruption while minimizing retransmission of data securely stored prior to the interruption (see X).

5.2 Control information

Control information, both requests and their corresponding responses, may be exchanged between initiator and target *via* command ORBs whose *control* bit is set to one. This indicates that the data in the buffer (or the data to be stored in the buffer) associated with the ORB is control information rather than application data. Only one control request or response shall be transferred by an ORB; the format of the control information in the buffer is illustrated by Figure 9.

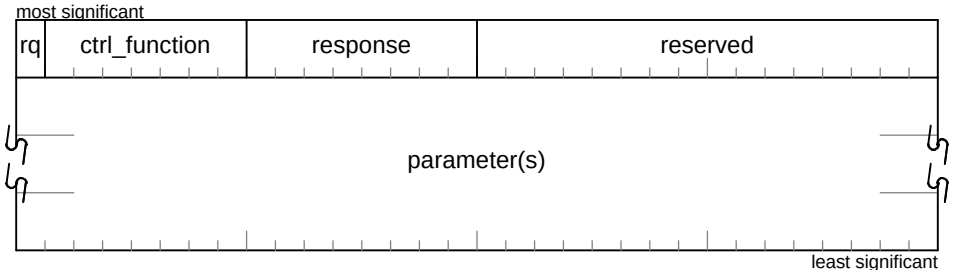

Figure 9 – Control information format

The *rq* bit shall specify whether the control function is a request or a response. A value of one indicates a request.

The *ctrl_function* field shall specify the control function, as defined by the table below.

ctrl_function	Name	Comment
0		Reserved for future standardization
1	CONNECT	Establish a connection with a particular service
2	ABORT CONNECTION	Immediately terminate a connection without regard to its current state
2	RESET CONNECTION	Resynchronize a connection between a client application and the service
4	DISCONNECT	Terminate a connection once it has quiesced
5	SERVICE DIRECTORY	Query all the services implemented (identified by a list of service Ids)
$6 – 7F_{16}$		Reserved for future standardization

The *response* field is valid only when the *rq* bit is zero. In this case, it encodes a response indication for the corresponding control function, as defined by the table below.

response	Definition
0	Request completed OK; response parameters are meaningful.
1	Unknown control function.
2	Insufficient resources are available to complete the request; the same request may succeed if resubmitted later.
3	The service identified by the SERVICE_ID parameter does not exist.
4	Mismatch between actual and expected queue number parameter(s).
5	The connection request is refused.
6	The connection identified by the queue number parameter(s) does not exist.
FF_{16}	Unspecified error.

The remainder of the control information, up to the maximum size specified by *data_size* in the ORB that references the control information buffer, shall consist of one or more parameters. Each parameter shall be quadlet-aligned and occupy an integral number of quadlets. The first parameter shall start in the second quadlet of control information and subsequent parameters, if any, shall immediately follow the preceding parameter. The order in which parameters appear is unimportant. Any quadlets that follow the last parameter, up to the end of the control information, shall be cleared to zero.

The parameter ID shall specify the parameter format, either immediate or variable-length. The most significant bit of the parameter ID determines the format; parameters whose ID values are in the range zero to $7F_{16}$, inclusive, shall conform to the format specified by Figure 10 while those in the range 80_{16} – FF_{16}, inclusive, shall conform to the format specified by Figure 11. Defined values for parameter ID are given in Table 1; all values not specified are reserved for future standardization.

Table 1 – Parameter ID values

Parameter ID	Parameter name	Description
1	TASK_SLOTS	The maximum size of the working set. Dependent upon context, this is either the maximum supported by the target or the maximum utilized by the initiator.
82_{16}	SERVICE_ID	An ASCII string that uniquely identifies a service.
3	I2T_QUEUE	The queue number (within the context of a connection) used for the transport of application data from the initiator to the target
4	T2I_QUEUE	The queue number (within the context of a connection) used for the transport of application data from the target to the initiator

The format of immediate parameters is shown below.

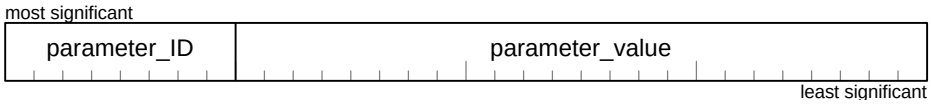

Figure 10 – Immediate parameter format

The *parameter_ID* field shall specify the parameter, as encoded by Table 1.

The *parameter_value* field shall specify the immediate value of the parameter. Unless otherwise specified for a particular value of *parameter_ID*, the *value* field shall contain an unsigned 24-bit number.

The format of variable-length parameters (which are usually ASCII text strings) is shown below.

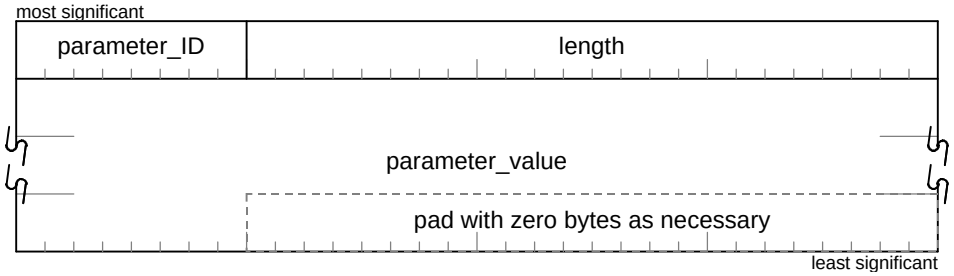

Figure 11 – Variable-length parameter format

The *parameter_ID* field shall specify the parameter, as encoded by Table 1.

The *length* field shall specify the parameter length, in bytes.

The *parameter_value* field shall contain the value of the parameter and shall commence with the most significant byte of the parameter value. If the length of the parameter is not a multiple of four, the parameter value shall be padded with trailing bytes of zero. Unless otherwise specified for a particular value of *parameter_ID*, the *value* field shall contain an ASCII text string without leading or trailing blank characters.

5.3 Status block

As described by ANSI NCITS.325-1998, 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 the status block shown below.

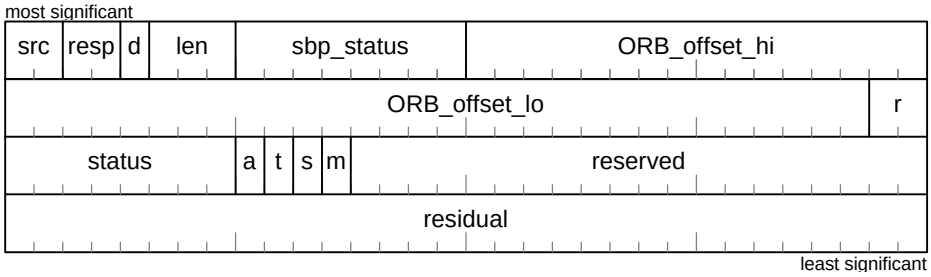

Figure 12 – Status block format

The definition and usage of 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 specified by ANSI NCITS.325-1998.

The *len* field shall have a value of three to indicate that the length of the status block is four quadlets.

The *status* field shall specify the completion status of the transport flow requested by the ORB, as encoded by the table below.

status	**Description**
0	The application data or control information has been successfully transferred; consult the *residual* field for details of the actual transfer length.
1	Invalid queue; the queue identified in the ORB has not been allocated to an active connection.
2	Target reset by another initiator; all tasks aborted.

The *attention* bit (abbreviated as *a* in the figure above) indicates the availability of target control information. When the *attention* bit is one, the initiator should post an ORB for queue zero to retrieve the control information. Once set to one by the target, this bit shall remain set until the initiator successfully retrieves the control information.

The *target_data_pending* bit (abbreviated as *t* in the figure above) indicates the availability of target application data for the queue specified by the ORB identified by *ORB_offset_hi* and *ORB_offset_lo*. When the *target_data_pending* bit is one, the initiator should post an ORB for the specified queue to retrieve the application data. The target shall zero this bit when there is no pending application data awaiting transfer to the initiator. The meaning of *target_data_pending* is unspecified for an unsolicited status block.

The *special* bit (abbreviated as *s* in the figure above) provides additional information pertinent to application data transferred from the target to the initiator. The meaning of the *special* bit is unspecified when the value of the *src* field is two or when (in the ORB identified by *ORB_offset_hi* and *ORB_offset_lo*) either the *direction* bit is zero or the *control* bit is one. The meaning and usage of the *special* bit are application-dependent and shall apply to all of the application data contained within the buffer described by the ORB.

The *end_of_message* bit (abbreviated as *m* in the figure above) shall indicate whether or not a boundary exists in the application data or control information transferred from the target to the initiator. The meaning of the *end_of_message* bit is unspecified when the value of the *src* field is two or when the *direction* bit in the ORB identified by *ORB_offset_hi* and *ORB_offset_lo* is zero. Otherwise, when *end_of_message* is one, a boundary exists after the last byte of application data or control information described by the ORB. In the case of application data, the nature of the boundary and its interpretation shall be specified by the service definition. When the *control* bit in the ORB identified by *ORB_offset_hi* and *ORB_offset_lo* is one, the *end_of_message* bit in the associated status block shall also be one; all control information for a single request or response shall be contained within one buffer.

The *residual* field shall specify the difference between the requested and actual data transfer lengths, in bytes. The target shall calculate *residual* by subtracting the actual data transfer length from the size of the buffer provided by the initiator; negative values are indicated in two's complement notation. A nonzero *residual* value is not necessarily indicative of an error.

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 may be arranged in three principal categories:

- core registers defined by draft standard IEEE P1212r and required by either that standard or this document;
- bus-dependent registers required by IEEE Std 1394-1995; and
- unit architecture registers required by ANSI NCITS 325.1998.

The relevant standard shall be consulted for details of register definition and usage; the table below provides a quick reference that summarizes all CSRs used by this document. Except for the optional MESSAGE_REQUEST and MESSAGE_RESPONSE registers, all of the CSRs are mandatory.

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
C_{16}	RESET_START	Resets the node's state
18_{16} – $1C_{16}$	SPLIT_TIMEOUT	Time limit for split transactions
80_{16} – BC_{16}	MESSAGE_REQUEST	Message area for target requests when no login exists
$C0_{16}$ – FC_{16}	MESSAGE_RESPONSE	Message area for initiator responses to target requests addressed to MESSAGE_REQUEST.
210_{16}	BUSY_TIMEOUT	Controls transaction layer retry protocols
specified by configuration ROM	MANAGEMENT_AGENT	Login and other SBP-2 task management requests
specified by login response data	AGENT_STATE	Reports SBP-2 fetch agent state
	AGENT_RESET	Resets SBP-2 fetch agent
	ORB_POINTER	Address of current ORB
	DOORBELL	Signals SBP-2 fetch agent to refetch an address pointer
	UNSOLICITED_STATUS_ENABLE	Acknowledges the SBP-2 initiator's receipt of unsolicited status

7 Configuration ROM

All devices compliant with this standard shall implement general format configuration ROM in accordance with IEEE Std 1394-1995, ANSI NCITS.325-1998, draft standards IEEE P1394a and IEEE P1212r and the additional requirements of this document. General format configuration ROM is a self-descriptive structure, an example of which is illustrated below.

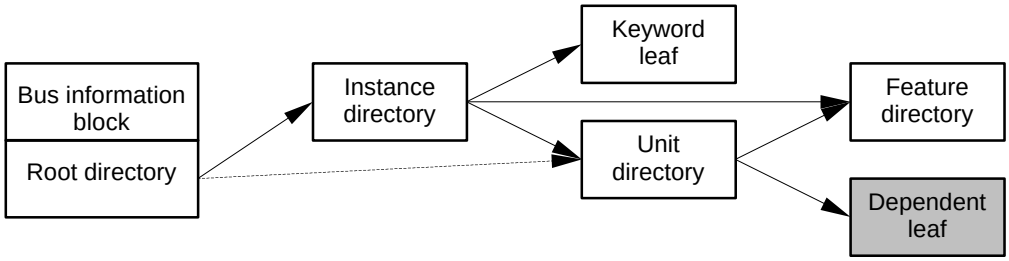

Figure 13 – Example configuration ROM hierarchy

With the exception of the dependent leaf (shown shaded), all of the configuration ROM components shown above are required for devices compliant with this standard. The connection from the root directory to the unit directory (shown by a dashed line) is optional; instance directories are the preferred access routes for unit directories.

The location of the bus information block is fixed at FFFF F000 0400_{16}; the size of the bus information block predetermines the location of the immediately following root directory. The locations of all other configuration ROM data structures are determined indirectly by pointers.

During the initialization process that follows a power reset, the first quadlet of configuration ROM shall be zero; ANSI NCTIS.325-1998 and draft standard IEEE P1212r specify target behavior during initialization.

Once initialization completes, general format configuration shall be accessible to read requests. In addition to the requirements of the referenced standards and draft standards, the first five quadlets of configuration shall conform to the format illustrated by Figure 14.

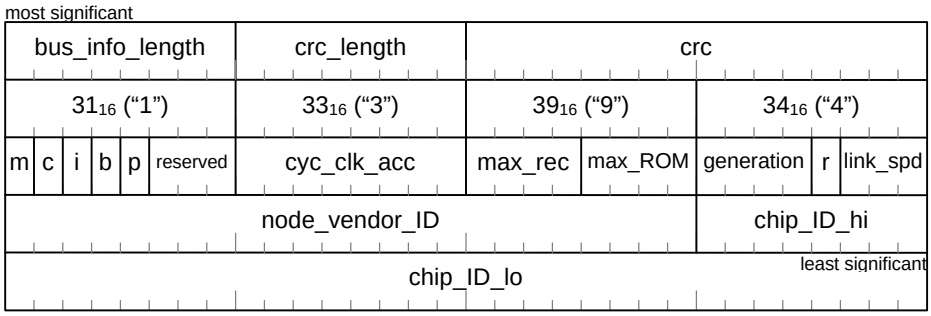

Figure 14 – First five quadlets of configuration ROM

The *bus_info_length* field shall have a value of four.

The *crc_length* field shall have a value of four plus the size, in quadlets, of the root directory. This indicates that the *crc* field is calculated for both the bus information block and the root directory—but not

for any of the other configuration ROM data structures. The value of the *crc* field shall be calculated in accordance with draft standard IEEE P1212r.

The second quadlet shall contain the string “1394” in ASCII characters as specified by draft standard IEEE P1394a.

The meaning and usage of the *irmc*, *cmc*, *isc*, *bmc* and *pmc* bits (abbreviated as *m*, *c*, *i*, *b* and *p*, respectively, in the figure above) and the *cyc_clk_acc*, *max_rec*, *max_ROM*, *generation*, *link_spd*, *node_vendor_ID*, *chip_ID_hi* and *chip_ID_lo* fields are specified by draft standard IEEE P1394a.

The *max_rec* field shall have a minimum value of TBD.

The *max_ROM* field shall have a minimum value of one.

TO BE DETERMINED – The minimum value of one requires image devices to support block read requests aligned on 64-byte addresses with a data length of 64 bytes. I think it would be preferable to require a *max_ROM* value of two to indicate support for block read requests aligned on quadlet addresses with a data length less than or equal to 1024 bytes. The 1394 PWG has yet to make a decision.

7.1 Root directory

Configuration ROM for devices compliant with this standard shall contain a root directory. The root directory immediately follows the bus information block and has an address of FFFF F000 0414_{16}. Relevant mandatory and optional entries for the root directory are summarized by the table below; unless explicitly excluded, any optional root directory entries permitted by draft standard IEEE P1212r are also permitted by this document.

Table 2 – Root directory entries

Directory entry			
Name	**Type**	**Mandatory**	**Description**
Vendor_ID	I	Y	24-bit RAC ID of the vendor that manufactured the device. This entry shall be immediately followed by a Textual_Descriptor entry. The addressed textual descriptor leaf (or leaves, if an intermediate textual descriptor directory exists) should contain an informal form of the vendor name easily recognizable by users.
Node_Capabilities	I	Y	Identifies which options of the CSR architecture are implemented.
Node_Unique_ID	L		Although permitted by draft standard IEEE P1212r, devices compliant with this standard shall not include a Node_Unique_ID entry in the root directory.
Keyword_Leaf	L		"Thumbnail" description of the characteristics of the all instances implemented by the device.
Instance_Directory	D	Y	The instance directories provide a method to group unit architectures (software protocols) to identify shared physical components.
Unit_Directory	D		The use of Unit_Directory entries in the root directory is discouraged; designers should consult draft standard IEEE P1212r for more information.

The Vendor_ID entry shall contain the RAC ID of the vendor that manufactured the device and shall be immediately followed by a Textual_Descriptor entry that specifies the location of either a textual descriptor directory or leaf. The referenced textual descriptor leaf or leaves should contain an informal (short) form of the company name of the vendor.

A Keyword_Leaf entry is optional within the root directory and, if present, shall specify the location of a keyword leaf in configuration ROM. The keywords included in the keyword leaf shall be the union of all keywords from all keyword leaves in the device's configuration ROM.

At least one Instance_Directory entry is required in the root directory; each shall specify the location of an instance directory in configuration ROM.

7.2 Instance directories

Configuration ROM for devices compliant with this standard shall contain one or more instance directories, each of which describes the function(s) implemented by a particular instantiation within the device. The mandatory and optional directory entries for an instance directory are specified by draft standard IEEE P1212r.

All instance directories shall contain a Keyword_Leaf entry.

7.3 Feature directories

All unit directories compliant with the requirements of clause 7.4 shall contain a Feature_Directory entry that specifies the location of a feature directory whose content and meaning are compliant with this clause. Configuration ROM may contain feature directories whose content and meaning are specified either by this standard, another organization or vendor. Relevant mandatory and optional entries for feature directories compliant with this document are summarized by the table below; unless explicitly excluded, any optional feature directory entries permitted by draft standard IEEE P1212r are also permitted by this document.

Table 3 – Feature directory entries

Directory entry Name	Type	Mandatory	Description
Specifier_ID	I	Y	24-bit RAC ID of the directory specifier, 00 5029_{16}.
Software_Version	I	Y	In combination with the directory specifier ID, it identifies the software interface for the unit.
Service_ID	L	?	Collection of service ID text strings for all services implemented for the instance or unit.
Device_ID	D	Y	Device identifier commonly used for plug and play device enumeration.

The Specifier_ID entry, whose 24-bit immediate value shall be 00 5029_{16}, and the Software_Version entry, whose 24-bit immediate value shall be TBD, identify this document as the specification of the feature directory.

The Service_ID entry shall specify the location of a leaf in configuration ROM that contains text strings, each of which is the service ID of a service implemented by the instance or unit. The format of the leaf shall be identical to that specified by draft standard IEEE P1212r for keyword leaves.

The Device_ID entry shall specify the location of a textual descriptor leaf in configuration ROM that contains a device identifying string in the format specified by IEEE Std 1284-1994 clause 6.6.

TO BE DETERMINED – Is this the right way to do this? Or should a textual descriptor leaf be associated with some other entry? Is the device ID string correlated with a device driver? If so, it probably doesn't belong in the feature directory but in the unit directory.

7.4 Keyword leaves

Each instance directory shall be characterized by a set of appropriate keywords selected from Table 4 and placed in a keyword leaf referenced by a Keyword_Leaf entry in the instance directory. Additional keywords may be present in any keyword leaf, but their meaning and usage are beyond the scope of this standard. Instances that share exactly the same set of keywords may reference the same keyword leaf.

Table 4 – Recommended keywords

Keyword	Recommended usage
CAMERA	
COLOR	
DISK	
FAX	
IMAGE	
MFP	
MODEM	
PHOTO	
PRINTER	
RECEIVE	
SCANNER	
SEND	

7.5 Unit directories

Configuration ROM for devices compliant with this standard shall contain one or more unit directories, each of which specifies a software interface (unit architecture) for a device function. Relevant mandatory and optional entries for unit directories are summarized by the table below; unless explicitly excluded, any optional unit directory entries permitted by draft standard IEEE P1212r or ANSI NCITS.325-1998 are also permitted by this document.

Table 5 – Unit directory entries

Directory entry Name	Type	Mandatory	Description
Specifier_ID	I	Y	24-bit RAC ID of the directory specifier.
Software_Version	I	Y	In combination with the directory specifier ID, it identifies the software interface for the unit.
Command_Set_Spec_ID	I	Y	24-bit RAC ID of the command set specifier, 00 5029_{16}.
Command_Set	I	Y	In combination with the command set specifier ID, it identifies the command set for the unit.
Management_Agent	I	Y	Provides the address of the SBP-2 MANAGEMENT_AGENT register for login to the device.
Unit_Characteristics	I	Y	
Logical_Unit_Number	I	Y	
Reconnect_Timeout	I		TBD—Is this entry mandatory and if so what is the minimum value for *max_reconnect_hold*?
Feature_Directory	D	Y	Additional information that describes features (usually independent of the software interface and command set) of the unit. At least one of the feature directories shall be specified by this standard.

The Specifier_ID entry, whose 24-bit immediate value shall be 00 $609E_{16}$, and the Software_Version entry, whose 24-bit immediate value shall be 01 0483_{16}, identify the device as compliant with ANSI NCITS.325-1998, SBP-2.[1]

The Command_Set_Spec_ID entry, whose 24-bit immediate value shall be 00 5029_{16}, and the Command_Set entry, whose 24-bit immediate value shall be TBD, identify the device as compliant with this document. The optional Command_Set_Revision entry, if present, shall have a 24-bit immediate value of zero.

The Unit_Characteristics entry shall specify a vendor-dependent *mgt_ORB_timeout* and an ORB size of eight quadlets (32 bytes). Consult ANSI NCITS.325-1998 for details.

Devices compliant with this standard shall contain a single Logical_Unit_Number entry for logical unit zero in each unit directory. The entry shall specify an unordered execution model (the *ordered* bit shall be zero). The *device_type* field shall contain a value specified by the table below.

[1] The names given are those used by draft standard IEEE P1212r; they correspond to the names Unit_Spec_ID and Unit_SW_Version, respectively, in both ISO/IEC 13213:1994 and ANSI NCTIS.325-1998.

device_type	**Peripheral device type**
2	Printer
3	Processor
6	Scanner
9	Communications
$1F_{16}$	Unspecified device type; command set-dependent means are necessary to determine the peripheral device type

TO BE DETERMINED – Is service ID discovery a sufficient "command set-dependent" method of peripheral device type discovery? Or do we need a new control request and response? Alternatively, should the *device_type* field in all the Logical_Unit_Number entries be $1F_{16}$? I think the latter; user service ID discovery in all cases.

There shall be at least one Feature_Directory entry that specifies the location of a feature directory whose content and meaning are specified by this standard. There may be additional Feature_Directory entries that reference feature directories whose content and meaning are specified either by this standard, another organization or vendor.

8 Control operations

Before application client(s) and service(s) may exchange data in uni- or bi-directional transport flows (explained in detail in section 9), control operations are necessary to set up the communication paths. This section specifies the methods used by both initiator and target to establish and manage connections for these transport flows.

8.1 Login and queue zero

Access to a target compliant with this standard commences with an SBP-2 login request by the initiator. Upon successful completion of the login request, the target has reserved resources for the use of the initiator:

- SBP-2 registers unique to the login (the AGENT_STATE, AGENT_RESET, ORB_POINTER, DOORBELL and UNSOLICITED_STATUS_ENABLE registers);
- queue zero, the control operations queue; and
- two task slots for use by queue zero ORBs.

Once queue zero exists, either initiator or target may use it in a peer to peer fashion to communicate control information, requests or responses, to the other. At no time shall the task set contain more than two ORBs whose *queue* field is zero.

The completion of a request requires two ORBs, one, which describes the control information buffer that contains the request and a complementary ORB which describes the control information buffer for the response. Although queue zero provides full peer to peer functionality between initiator and target, the details of its use are asymmetric and vary according to whether the initiator or the target is the requester.

When an initiator issues a request to a target, they shall perform the following operations:

a) The initiator shall store the request and its associated parameters (if any) in a buffer in its own system memory and signal to the target fetch agent an ORB, whose *queue* field and *direction* bit are zero and *control* and *end_of_message* bits are one, that describes the control information buffer;

b) The target shall fetch the ORB and read the control information buffer. The status block stored by the target to complete the ORB shall have its *attention* bit set to one to indicate that the target intends to transfer control information (the response) to the initiator;

c) At any time the initiator receives a status block whose *attention* bit is one and there is no ORB in the task set whose *queue* field is zero and *direction*, *control* and *end_of_message* bits are one, the initiator shall create such an ORB and place it in the task set; and

d) Once the target has executed the indicated request and there is an ORB in the working set whose *queue* field is zero and *direction*, *control* and *end_of_message* bits are one, the target shall store the response data in the buffer described by the ORB and then store completion status for the ORB. So long as the target has pending control information to transfer to the initiator, it shall continue to set the *attention* bit to one in any status block (including unsolicited status) stored into the initiator *status_FIFO*.

NOTE – In order to reduce ORB fetch latency, the initiator may place two control information ORBs in the task set at the same time, the first for the request (with a *direction* bit of zero) and the second for the response (with a *direction* bit of one). Although the algorithm described above works correctly even if the initiator awaits a status block whose *attention* bit is one before signaling a target response ORB to receive the response data, it is more efficient to post both ORBs at the same time.

When a target issues a request to an initiator, they shall perform the following operations:

a) The target shall set the *attention* bit to one in a status block stored into the initiator *status_FIFO*. Either unsolicited status or completion status associated with an ORB may be used. So long as the target has pending control information to transfer to the initiator, it shall continue to set the *attention* bit to one in any status block stored into the initiator *status_FIFO*.

b) At any time the initiator receives a status block whose *attention* bit is one and there is no ORB in the task set whose *queue* field is zero and *direction*, *control* and *end_of_message* bits are one, the initiator shall create such an ORB and place it in the task set;

c) Once there is an ORB in the working set whose *queue* field is zero and *direction*, *control* and *end_of_message* bits are one, the target shall store the control information data (request) in the buffer described by the ORB and then store completion status for the ORB. The *attention* bit shall be zero in the status block associated with the ORB;

d) When the initiator has executed the indicated request, it shall store the response and its associated parameters (if any) in a buffer in its own system memory and signal to the target fetch agent an ORB that describes the control information buffer. The ORB's *queue* field and *direction* bit shall be zero and the *control* and *end_of_message* bits shall be one;

e) The target shall fetch the ORB and read the response from the control information buffer. The status block stored by the target to complete the ORB may have its *attention* bit set to one if the target intends to transfer other control information (request or autonomous response) to the initiator.

It is possible for both initiator and target to initiate requests at roughly the same time. In this case the working set contains an ORB for transfer of the request from initiator to target while the status block *attention* condition is simultaneously asserted by the target. The ordered execution properties of queue zero give a natural precedence to initiator requests over target requests, as follows. When a target fetches an ORB whose *queue* field and *direction* bit are zero and whose *control* and *end_of_message* bits are one, the request contained in the control information shall be processed before a request is transferred to the initiator. Consequently, if a target has an uncompleted initiator request when it fetches an ORB whose *queue* field is zero and whose *direction*, *control* and *end_of_message* bits are one it shall not store any control information except the response that completes the request.

When neither initiator nor target have outstanding requests or responses, the control queue (queue zero) is idle and there shall be no ORBs in the task set whose *queue* field is zero.

8.2 Autonomous response information

The preceding clause describes the use of queue zero for request / response pairs between initiator and target. It is also possible for either initiator or target to autonomously transfer response information to the other. Autonomous response information is typically status information and does not necessarily require any additional action on the part of the recipient.

Autonomous response information may be sent for any of the *ctrl_function* values enumerated in the table below.

ctrl_function	**Name**
4	SERVICE DIRECTORY
5	STATUS

The *response* code in autonomous response information shall be zero.

Autonomous response information shall not be transferred while there is an uncompleted control request. A target requests the transfer of autonomous response information by means of the status block *attention* bit. If a target asserts *attention* and subsequently fetches an initiator request ORB, it shall first complete the initiator's control request and transfer the corresponding response information to the initiator before

transferring the autonomous response information. The *attention* bit shall remain asserted in any status blocked stored in the initiator *status_FIFO* while the transfer of the autonomous response information is pending.

8.3 Service discovery

Services implemented by either initiator or target are uniquely identified by their service ID, an ASCII string registered with TBD. A client application that wishes to establish a connection with a particular service may attempt the connection without *a priori* knowledge that the service is implemented or the client application may request service directory information.

A service discovery request shall have a *ctrl_function* code of SERVICE DIRECTORY and no parameters. The response information shall contain zero or more SERVICE_ID parameters that identify all of the services implemented. The order of the SERVICE_ID parameters in the response is unspecified.

TO BE DETERMINED – Some method of “paging” through large quantities of service ID information needs to be agreed.

8.4 Connection management

Table 6 – Connection type encoded by queue ID parameters

Connection type	I2T_QUEUE value	T2I_QUEUE value
Unidirectional	unrestricted	—
	—	unrestricted
Bi-directional (nonblocking)	not equal to T2I_QUEUE	not equal to I2T_QUEUE
Bi-directional (blocking)	equal to T2I_QUEUE	equal to I2T_QUEUE

8.4.1 Connection establishment

8.4.1.1 Connection established by an initiator

8.4.1.2 Connection established by a target

8.4.2 Aborting a connection

8.4.3 Resetting a connection

8.4.4 Disconnection

8.5 Queue status information

9 Transport flow operations

Once a connection is established between a client application and a service, work is accomplished by the flow of application-dependent data between the two, whether it is uni- or bi-directional. This section describes normative data transfer scenarios as well as error recovery procedures. Transport flow is described from the viewpoint of a queue instead of from a connection; it is straightforward to generalize the details of a single queue's operation to that of two queues operating together to provide a bi-directional connection.

Services may be implemented to utilize either a datagram or stream model for transport flow. The datagram model is the simplest: there is a one-to-one relationship between ORBs, buffers and messages, as illustrated by Figure 15.

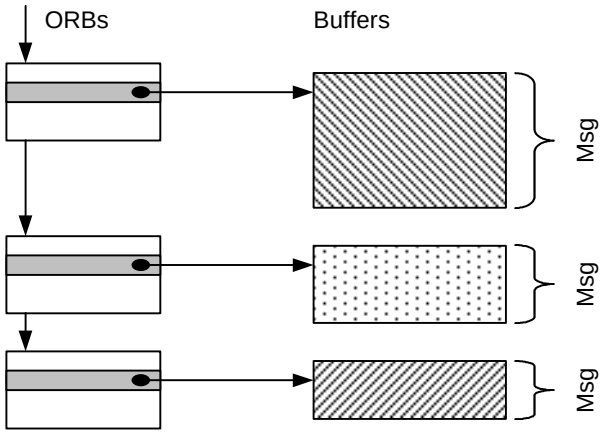

Figure 15 – Transport flow (datagram model)

When the stream model is used for the application data, the message boundaries (*e.g.*, the separation between pages or print jobs for a printer) may occur without regard for the boundaries between data buffers specified by different ORBs. Figure 16 illustrates the relationship between stream data, the ORBs that describe its buffers and the messages that may span the buffers.

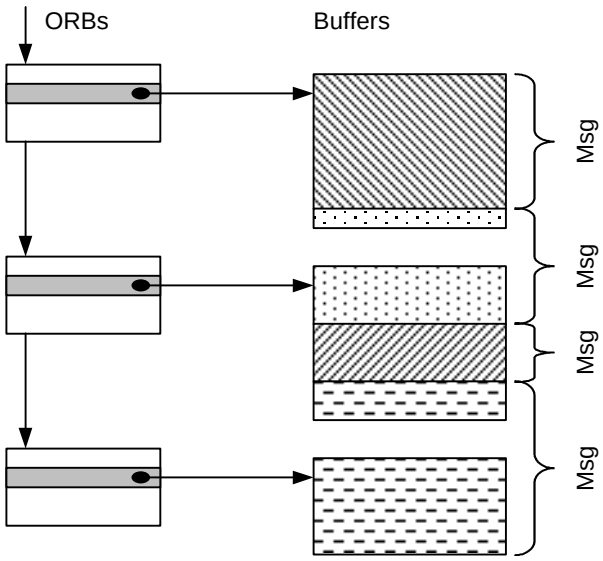

Figure 16 – Transport flow (stream model)

9.1 Data transfer to a target

Application data is transferred to a target by means of transport flow ORBs placed in an I2T_QUEUE and whose *direction* and *control* bits are zero. Within the limits of TASK_SLOTS allocated by the target at the time the connection (to which the I2T_QUEUE pertains) was established, the initiator may post more than one such outstanding transport flow ORB to the task set at a time. ORB fetch latency is reduced if the initiator is permitted to have at least two such outstanding ORBs in the task set.

9.2 Data transfer to an initiator

Application data is transferred to a target by means of transport flow ORBs placed in a T2I_QUEUE and whose *direction* bit is one and *control* bit is zero. Within the limits of TASK_SLOTS allocated by the target at the time the connection (to which the T2I_QUEUE pertains) was established, the initiator may post more than one such outstanding transport flow ORB to the task set at a time. ORB fetch latency is reduced if the initiator is permitted to have at least two such outstanding ORBs in the task set.

9.3 Completion status

The target shall signal completion status for a transport flow ORB by storing a status block to the initiator *status_FIFO* active for the login. Unless a nonrecoverable error occurs, the target shall transfer all the data specified by the ORB and receive a response subaction of *resp_complete* for each data transfer request subaction before it stores completion status to the initiator *status_FIFO*. In the case of a nonrecoverable error, all pending request subactions for the data transfer specified by the ORB shall either be completed or timed out before the target stores a status block for the ORB to the initiator *status_FIFO*.

9.4 Error recovery

9.5 Bus reset

Upon a bus reset, the target aborts all task sets and awaits reconnection from initiator(s) active prior to the reset. Once an initiator successfully completes a RECONNECT with the target, its client application(s) and

service(s) may resume data transfer with target service(s) or client application(s) and service(s) on a connection by connection basis.

Data transfer between a client application and a service may have caused device operations to commence even if not all the data had been transferred before the bus reset. For this reason, it is essential for each connection to be resynchronized by one of two methods. The simplest case is to abandon any operations in progress, flush initiator and target buffers as necessary and return both endpoints of the connection to a known state—at which point the abandoned operation(s) may be reinitiated.

Although this method of recovery from a bus reset is robust, it may be improved upon. If the client application and service can reliably resume data transfer from the point it was interrupted, it may be unnecessary to cancel operations and flush buffers. In order for this method to work, the transport must be able to recognize resumption of an ORB in progress at the time of the bus reset. The *signature* field in a transport ORB provides a method by which identical ORBs may be recognized if they are resubmitted after a reset.

An initiator may implement the simpler recovery procedure by issuing a RESET CONNECTION control operation to the target. The connection to be reset shall be identified by the same I2T_QUEUE and T2I_QUEUE parameters provided by the target when the connection was established. Until the RESET CONNECTION control operation completes successfully, the initiator shall not signal any ORBs to the target whose *queue* field is equal to either the I2T_QUEUE or T2I_QUEUE parameter for the connection. Once a connection has been reset, ORBs may be signaled on the connection's queue(s) independent of the status of other connections for the same login.

If the initiator elects not to reset the connection, data transfer may be safely resumed if initiator and target can identify, for each queue, the ORB active at the time of the bus reset. For a particular queue, the active ORB is the oldest outstanding ORB when the bus reset occurred. When an initiator does not reset a connection, it shall perform the following steps for each queue that forms the connection:

a) If there were no uncompleted ORBs in the task set whose *queue* field identifies one of the queue(s) that form the connection, no action is necessary and the initiator may resume data transfer for the connection;

b) Otherwise, for the oldest uncompleted ORB for each of the connection's queues, the initiator shall signal an equivalent ORB to the target fetch agent. Certain parts of the ORB shall remain unchanged: the *direction*, *control*, *special* and *end_of_message* bits and the *queue* and *signature* fields shall have the same values both before and after the bus reset. The *data_descriptor*, and *data_size* fields and the *page_table_present* may have different values but they shall describe a buffer of the same size and whose contents are identical to the buffer described by the ORB aborted by the bus reset. The *spd* and *max_payload* bits may differ as a result of a different topology between the initiator and target after the bus reset.

c) ORBs for a particular queue (other than the oldest ORB) shall be interpreted by the target as if they are new; there are no restrictions on their field values. Nevertheless, the most straightforward implementation for the initiator transport is to signal equivalent ORBs (subject to the requirements in the preceding paragraph) in exactly the same order that they were signaled prior to the bus reset. This strategy is known to work but other implementations are possible.

The target shall detect an error if the initiator does not reset a connection and subsequently, for either of the connection's queues, signals an ORB to the target whose *signature* field is not equal to the signature of the active ORB for that queue at the time of the bus reset. If this occurs, the target shall abort the ORB with a *status* of TBD and reset the connection. The target shall communicate an autonomous RESET CONNECTION response to the initiator and continue to abort all ORBs signaled for one of the conection's queues until the autonomous response has been successfully transferred to the initiator.

Annex A
(normative)

Minimum Serial Bus node capabilities

In addition to the minimum capabilities defined by IEEE Std 1394-1995, ANSI NCITS.325-1998 and draft standard IEEE P1394a, this annex specifies other capabilities or restrictions mandated by this standard.

A.1 Initiator capabilities

TO BE DETERMINED – Review all of these (from SBP-2) and determine if this profile requires any GREATER 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

TO BE DETERMINED – Review all of these (from SBP-2) and determine if this profile requires any GREATER 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.

Targets shall support management request functions addressed to the MANAGEMENT_AGENT register as specified by the table below.

function	**Support**	**Description**
0	Mandatory	LOGIN
1	Mandatory	QUERY LOGINS
2	—	Reserved for future standardization
3	Mandatory	RECONNECT
4	Optional	SET PASSWORD (see ANSI NCITS.325-1998 Annex C)
5 – 6	—	Reserved for future standardization
7	Mandatory	LOGOUT
8 – A_{16}	—	Reserved for future standardization
B_{16}	Not supported	ABORT TASK
C_{16}	Mandatory	ABORT TASK SET
D_{16}	—	Reserved for future standardization
E_{16}	Not supported	LOGICAL UNIT RESET
F_{16}	Mandatory	TARGET RESET

Annex B
(normative)

Control request and response parameters

The table below provides a quick reference to the parameters associated with particular control requests and responses; consult section 8 for details for a particular request or response. Optional parameters are shown by parentheses; the last column indicates whether or not the response information may be sent autonomously.

ctrl_function	**Name**	**Requester**	**Request parameters**	**Response parameters**	**Autonomous response**
1	CONNECT	Initiator	SERVICE_ID (TASK_SLOTS)	Queue ID(s)[2] TASK_SLOTS	No
		Target	Queue ID(s)[2] SERVICE_ID TASK_SLOTS	(TASK_SLOTS)	No
2	DISCONNECT	—	Queue ID(s)[3]	—	No
3	ABORT CONNECTION	—	Queue ID(s)[3]	—	No
4	SERVICE DIRECTORY	—	—	SERVICE_ID(s)	Permitted
5	STATUS	Initiator	—	QUEUE_INFO	Target only

[2] At least one queue ID parameter shall be present, either I2T_QUEUE or T2I_QUEUE, and both may be present. In the latter case the two queue ID parameters may identify different queues or the same queue.

[3] The queue ID parameter(s) shall be the same originally provided by the target when the connection was established.

Annex C
(informative)

Configuration ROM

Configuration ROM is located at a base address of FFFF F000 0400_{16} within a node's address space. The requirements for general format configuration ROM for devices compliant with this standard are specified in section 7. This annex contains illustrations of typical configuration ROM for a variety of devices.

C.1 Bus information block and root directory

Error! Reference source not found. below shows a typical bus information block, root directory and textual descriptor leaves for devices compliant with this standard. Not shown are the instance, feature and unit directories themselves; these may vary according to the complexity of the device and its supported software interfaces. Consult other clauses in this annex for examples of printers, scanners and other, multifunction devices.

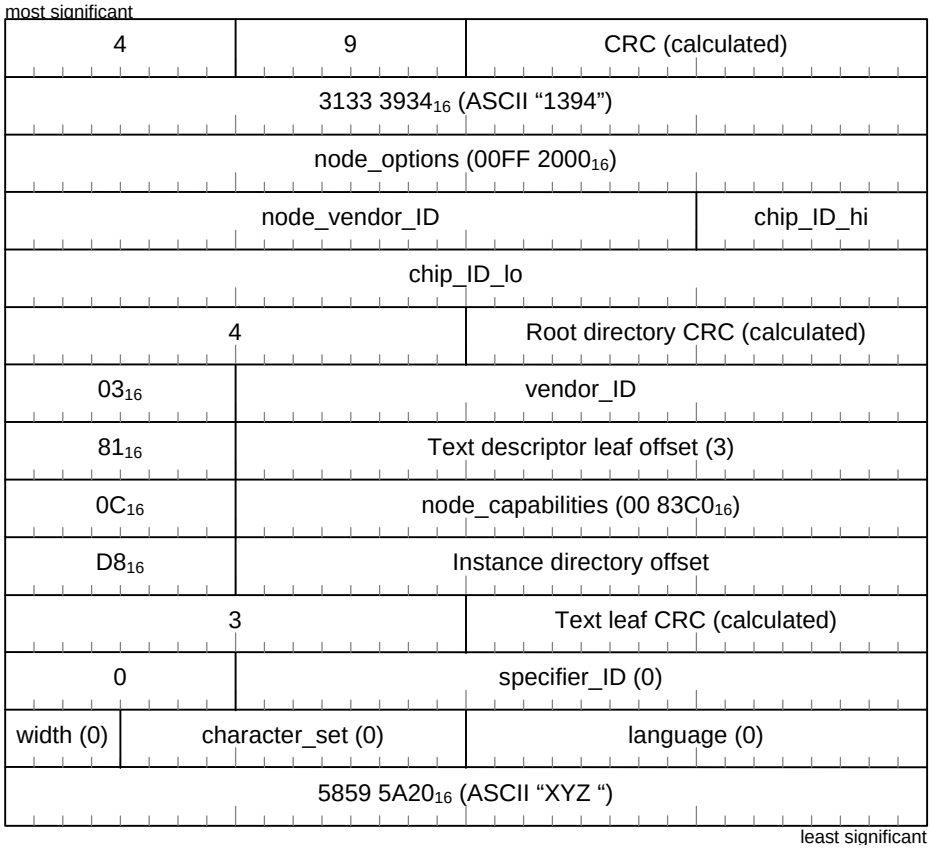

Figure C-1 – Example bus information block and root directory

The CRC in the first quadlet is calculated on following nine quadlets of configuration ROM, the bus information block and the root directory. Devices should not include all of configuration ROM within the coverage provided by this CRC; the other directories and leaves each contain their own CRC.

The *node_options* field represents a collection of bits and fields specified draft standard IEEE P1212r. The value shown, 00FF 2000_{16}, represents basic characteristics of a device that is not isochronous capable. This value is composed of a *cyc_clk_acc* field with a value of FF_{16} and a *max_rec* value of two. The *max_rec* field encodes a maximum payload of eight bytes in block write requests addressed to the target.

The Node_Capabilities entry in the root directory, with a *key* field of $0C_{16}$, has a value where the *spt*, *64*, *fix*, *lst* and *drq* bits are all one. This is a minimum requirement for devices compliant with this standard.

The Vendor_ID entry in the root directory, with a *key* field of 03_{16}, is immediately followed by a textual descriptor leaf entry, with a *key* field of 81_{16}, whose *indirect_offset* value points to a leaf that contains an ASCII string that identifies the vendor (the XYZ company). Although the textual descriptor leaf utilizes minimal ASCII, a permissible variant might include a textual descriptor directory in order to provide multiple language support.

The Instance_Directory entry in the root directory, with a *key* field of $D8_{16}$, is the starting point for device discovery (enumeration) software to search configuration ROM for particular function instances.

EDITOR's NOTE – Incorporate sample configuration ROM from the CSR and configuration ROM profile for image devices.

C.2 Scanner with a single unit architecture

C.3 Printer with multiple unit architectures

C.4 Multifunction device with uniform unit architectures