1394 PWG

1394 PRINTER WORKING GROUP

IEEE 1394 HIGH SPEED BUS IMAGING DEVICE COMMUNICATIONS SPECIFICATION

*****PRELIMINARY DRAFT PROPOSAL *****

Revision 0.05 - December 18, 1997

Editor - Alan Berkema

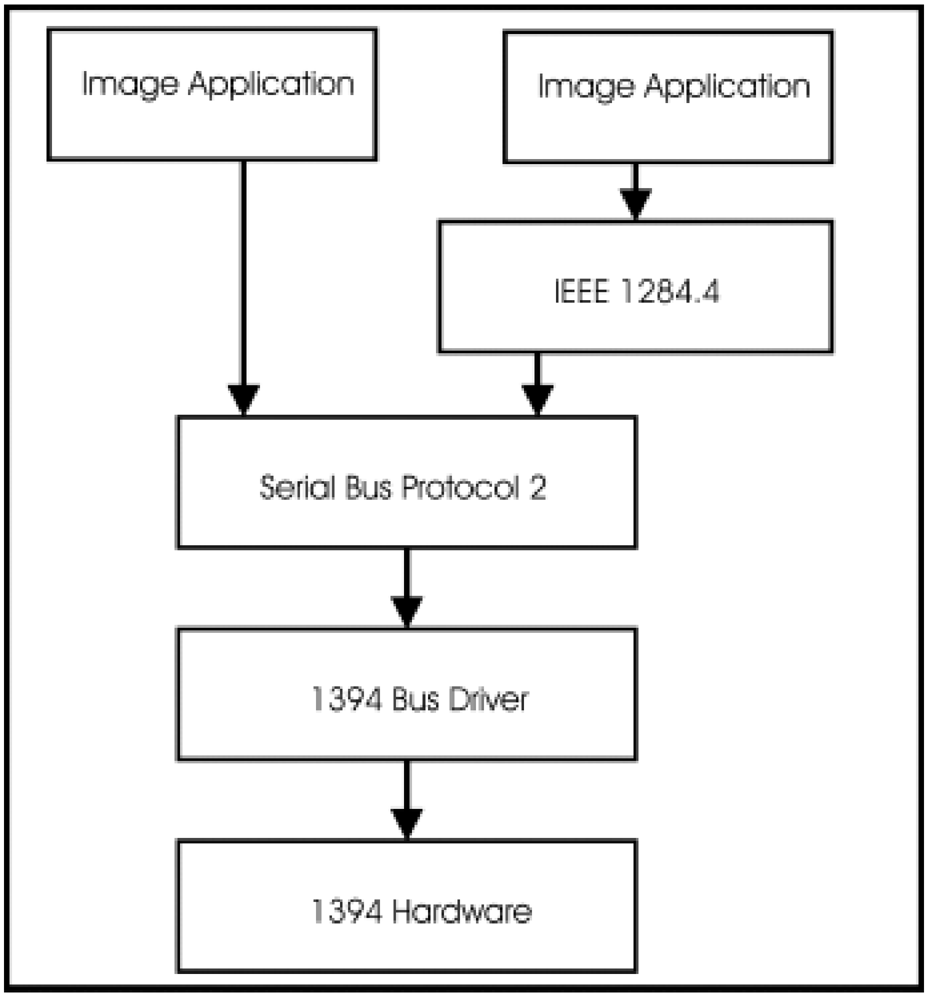
1. Overview 2
2. IEEE 1394 Abstract 3
3. References 3
4. Control & Status Registers (CSR) 4
5. Requirements For Isochronous Data Transmission 4
6. 1394 Bus Management 5
7. Bit, Byte and Quadlet ordering 6
8. Control & Status Register (CSR) Summary 7
9. Configuration ROM 8
10. 1394PWG Requirements for Our Thick Transport Stack (ROTTS?) 13
    10.1. Musts 13
    10.2. Wants 13
11. Discovery 14
12. SBP-2 Communication Protocol 15
13. Why SBP-2 18
14. ORB List Processing 18
15. Multiple Logical Channels 19
16. Bi-Directional Communication 19
17. Out of Order ORB Processing 21
18. PEER to PEER 21
19. Multiple Host and/or Multiple Device 21
20. Command Block ORBs 22
21. Login & Login Response 23
22. Unsolicited Status 25
23. Status Block 26
24. Reconnection 28
25. Query Login 29
26. Logout 29
27. Isochronous Data Transmission 29
28. 1394 Bus Reset Behavior 29
29. Error Recovery 30

1. Overview

The purpose of this document is to define the communications specification for IEEE 1394 printers, scanners, digital still cameras and other imaging devices. This specification will include traditional computer host communication to these devices as well as direct peer to peer communication.

The term “image device” is used throughout the remainder of this document to refer to any of the devices listed above.

The primary focus of this document is related to the SBP-2 protocol and how it can be used for image device communication. This document will also address the use of the IEEE 1284.4 transport over SBP-2. The goal is for this specification to be flexible enough to allow either implementation.



2. IEEE 1394 Abstract

IEEE 1394-1995 was ratified in December of 1995. This standard describes a high speed serial bus that has a 64 bit address space, control registers, and read/write/lock operations that conform to the ISO 13213/IEEE 1212, Command and Status Register (CSR) standard.

In addition to the standard read/write/lock transactions used for Asynchronous communications the Serial Bus provides an Isochronous data transport that guarantees latency and bandwidth.

Data transmission uses two low-voltage differential signals to connect devices at 98.304 Mbit/s, 196.608 Mbit/s, and 393.216 Mbit/s speeds.

The cable medium allows up to 16 cable hops between any two device, each hop up to 4.5 meters long, giving a total cable distance of 72 meters. Bus management recognizes smaller configurations to optimize performance. The physical topology for the cable environment is a non-cyclic network (no loops) with the only limitation being the number of cable hops between any two devices (called "nodes") and the length of a cable hop. The cables consist of 2 shielded twisted pairs for signals and one pair for power and ground. These cables connect to "ports" on the nodes. Each port consists of terminators, transceivers and simple logic. The cable and ports act as bus repeaters between the nodes to simulate a single logical bus. The Serial Bus also uses a fair bus access mechanism that guarantees all nodes equal access.
[5 Teener].

Plug and play is supported through automatic Node Identification without the need for switches or terminators.

The Serial Bus is not a network or an I/O channel, it is a shared memory architecture. The 64 bit address is divided into 16 bits for the Node ID (node number and bus number) and 48 bits (256 terabytes) for the memory space and CSR registers.

The 1394 standardization effort continues with p1394.a and p1394.b. P1394.a is focused on correcting ambiguities and problems in 1394-1995 as well as new enhancements such as Arbitrated Reset, Fly-By-Arbitration and Ack Acceleration. P1394.b is focused on higher speeds (800 Mbit/s) as well as long distance considerations using Plastic Optical Fiber.

3. References

1. ISO/IEC 13213 ANSI/IEEE 1212:1994, Control and Status Register Architecture for Microcomputer Buses.
2. IEEE Std 1394-1995, Standard for High Performance Serial Bus.
3. Serial Bus Protocol 2, Revision T10/1155x.
4. Software Design for IEEE 1394 Peripherals, Peter Johansson.
5. New Technology in the IEEE P1394 Serial Bus, Michael Teener, March 29, 1994.
6. IEEE P1284.4 Standard for Data Delivery and Logical Channels for Std. 1284 Interfaces.
7. P1394a Draft Standard for a High Performance Serial Bus (Supplement).

4. Control & Status Registers (CSR)

The basic functionality of a Transaction Capable 1394 node includes fundamental PHY repeater operation as well as the implementation of certain core CSR's. The CSRs are defined within the initial register space beginning at offset 0xFFFF F000 0000. This allows the node to participate in Asynchronous read/write/lock transactions. The core CSRs are specified below:

Note:
Whether these registers are actually present as part of 1394 silicon or whether it is actually Random Access Memory is implementation dependent.

STATE_CLEAR and STATE_SET. All bits within these registers are optional but the registers themselves must be present.

NODE_IDS. Used to identify the 16-bit address of the node. In the cable environment, the LINK and PHY must together initialize this register during the self identify process that follows a bus reset.

RESET_START. This is a write-only register available to force a command reset of the node, as defined by ISO/IEC 13213:1994.

SPLIT_TIMEOUT In order to make requests, the node must implement a transaction time-out capability and make it configurable (or at least readable) through the SPLIT_TIMEOUT register.

5. Requirements For Isochronous Data Transmission

Serial Bus nodes that can participate in Isochronous operations, either as a talker or a listener, must have all of the features of transaction capable nodes. Additional requirements support the timing and detection of Isochronous operations. A key element in Serial Bus is that all Isochronous nodes share the same, coordinated time. Because Serial Bus is a distributed environment, each node must have its own 24.576 MHz cycle clock which runs freely when the node's link layer is active. This clock must be visible through the CYCLE_TIME register. Jitter in these clocks is eliminated every 125 usecs by the appearance of a cycle start packet on Serial Bus. A cycle start packet is essentially a write to the CYCLE_TIME register with a resynchronization value that eliminates jitter.

An Isochronous node must implement its resynchronization logic such that Serial Bus time, as observed by the values of the CYCLE_TIME register, can never give the appearance of moving backward. Although optional, Isochronous operations are expected to be inexpensive to implement within a node.

In addition to the requirements to talk or listen during Isochronous cycles, at least one node on a Serial Bus must be able to be the source of the synchronized cycle clock. This node is referred to as the **cycle master**. The cycle master must be able to use its 24.576 MHz clock to trigger cycle start events 8,000 times a second. As soon after a cycle start that the cycle master is able to arbitrate for the bus, it transmits a cycle start packet to resynchronize the clocks at all Isochronous nodes.

In addition to acting as the source for the Serial Bus cycle clock, a cycle master also has to implement the BUS_TIME register. This register is needed to extend the range of the Serial Bus clock from the limit permitted by the CYCLE_TIME register (about two minutes) to approximately 136 years.

An **Isochronous resource manager** is not actually an active manager of resources so much as it is an agreed upon location where all the Serial Bus nodes can cooperatively record their use of Isochronous resources, channel and bandwidth. BANDWIDTH_AVAILABLE is a register that stores the amount of Isochronous cycle time available to transmit data. Before any bandwidth is allocated, the register is initialized to a value that represents approximately 100 usecs. This permits some time to be implicitly reserved for asynchronous operations. CHANNELS_AVAILABLE is a register that is a bit map of all 64 possible Isochronous channels, free or in use. The Isochronous resource manager has another function, and that is to point to the bus manager, if any. Like the two CSR's just described, the BUS_MANAGER_ID register is a known location that is initialized by the node that assumes the role of the bus manager. It's important to note that a node that is Isochronous resource manager capable must be able to not only participate in the self identify process but to also analyze all of the self-ID packets observed. This is because if there are more than one Isochronous resource manager capable nodes on Serial Bus, only one becomes the Isochronous resource manager — the one with the largest 6-bit physical ID.
[4 Johansson]

It is recommended that all Isochronous nodes provide Isochronous Resource Manager functions. as well as cycle master capability.

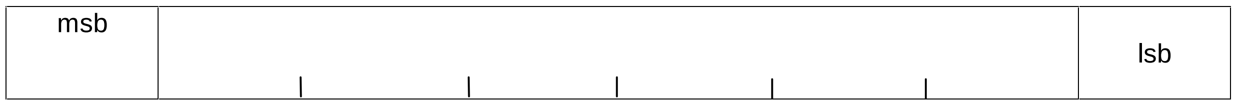
6. 1394 Bus Management

The highest level of functionality available to a Serial Bus node is that of the **bus manager**. Bus managers need all of the capabilities discussed in the preceding sections plus additional intelligence to analyze the Serial Bus configuration and optimize it. It is likely that bus manager capabilities are implemented in firmware. They do not require additional hardware support as much as some sort of processor to perform the analysis. This does not, however, preclude the design of a bus manager entirely in logic. The key functions of a bus manager are: If Isochronous operations are desired but the current root node is not cycle master capable, the bus manager must perform a bus reset and insure that the new root can be the cycle master. The bus manager must collect and analyze the self-ID packets in order to make the TOPOLOGY_MAP and SPEED_MAP registers available. Serial Bus management applications need to communicate this information to system administrators. Serial Bus has a configurable parameter, the gap count, which is initially set to a default value. The bus manager can significantly improve Serial Bus performance by setting the gap count to a smaller value, the minimum which can be determined from maximum hop count of the current topology. Power management permits the bus manager to intelligently enable and disable selected nodes if the aggregate power demands are greater than the power available. An important distinction to remember about the bus manager is that it is an autonomous entity that has functions to perform even if no API is provided to a hypothetical bus management application. Also keep in mind that the costs associated with implementing a bus manager are not likely to be silicon costs, but testing costs. There can be a complex matrix of test cases to cover.
[4 Johansson]

7. Bit, Byte and Quadlet ordering

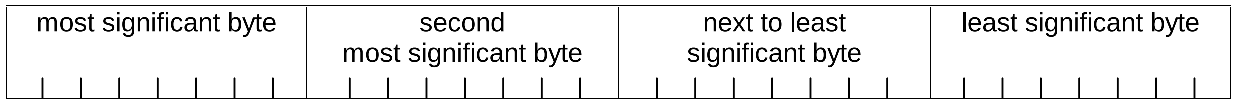
This profile defines the order and significance of bits within bytes, bytes within quadlets and quadlets within octlets in terms of their relative position and not their physically addressed position. Within a byte, the most significant bit, msb, is that which is transmitted first and the least significant 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.

Bit ordering within a byte



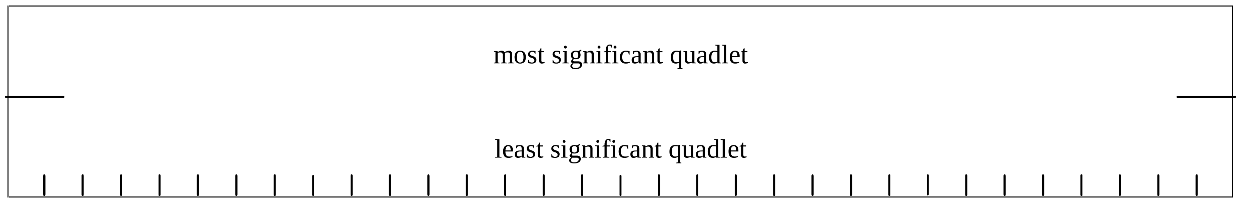
Byte ordering within a quadlet

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.



Quadlet ordering within an octlet

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.



8. Control & Status Register (CSR) Summary

All 1394 PWG devices shall implement the CSRs as defined in ISO 13213/IEEE 1212:1994 and IEEE Std 1394-1995. Note that only the core and BUSY_TIMEOUT would be needed if Isochronous operations are not included. BUSY_TIMEOUT is needed for Asynchronous retry transactions.

Core Registers

Offset	Register	Initial Value
0x000	STATE_CLEAR	
0x004	STATE_SET	
0x008	NODE_IDS	
0c00C	RESET_START	
0x018-01C	SPLIT_TIME_OUT	

Serial Bus Dependent

Cycle Master

Offset	Register	Initial Value
0x200	CYCLE_TIME	
0x204	BUS_TIME	

Other Serial Bus Dependent

Offset	Register	Initial Value
0x210	BUSY_TIMEOUT	

Isochronous Resource Manager

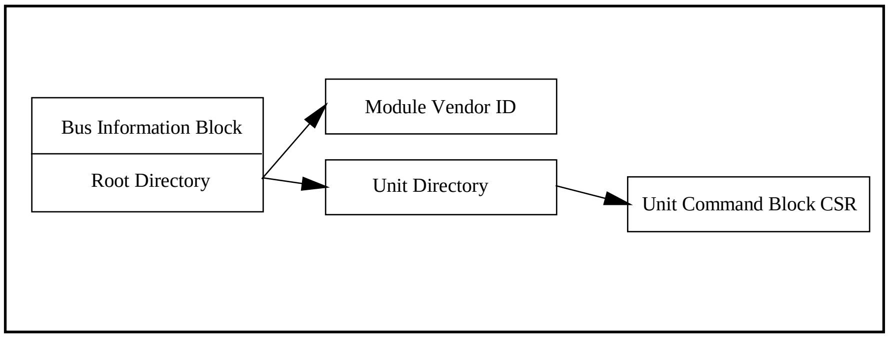
Offset	Register	Initial Value
0x21C	BUS_MANAGER_ID	
0x220	BANDWIDTH_AVAILABLE	4915
0x224-228	CHANNELS_AVAILABLE	All Ones

See the IEEE 1394-1995 specification for detailed information about each register.

9. Configuration ROM

All 1394 PWG devices shall implement configuration ROM as defined in ISO 13213/IEEE 1212:1994 and IEEE Std 1394-1995. The ROM directory structure is a hierarchy of information blocks with the blocks higher in the structure pointing to the blocks beneath them. The locations of the initial blocks, *Bus_Info_Block* and *Root_Directory*, are fixed. The locations of the other entries are specified in the *Root_Directory* and its associated directories.

The block diagram below illustrates device Configuration ROM relationships. Additional directories are defined in following sections.



Note:
Reserved fields shall be set to zero.
Length values in the Configuration ROM specify the number of Quadlets.
There are two types of offsets specified by ISO 13213/IEEEE 1212.
1) Initial register space offset which is an offset in quadlets from the initial register space base address of 0xFFFF F000 0000. Value contained in the register multiplied by 4 plus base address.
2) Indirect space offset, which is an offset in quadlets from the current register address. Value contained in the register multiplied by 4 plus address of register.
Number 1 above has a key_type of 0x1. Number 2 above has a key_type of 0x2 or 0x3, see ISO 13213/IEEEE 1212 section 8.2.4 table 21 for all key_type definitions.

First Quadlet
Offset: 0x400

bus_info_length	CRC_length	ROM_CRC_value
0x04	0x04	(calculated)

Bus Information Block
Offset: 0x404

0x31 “1”						0x33 “3”	0x39 “9”		0x34 “4”		
IRC	CMC	ISC	BMC	PMC	resv.	Cyc_Clk_Acc	Max_Rec	reserved 0x00	g	resv.	link_ spd
Node_Vendor_ID									Chip_ID_High		
Chip_ID_Low											

Taken together the Node_Vendor_ID, Chip_ID_High and Chip_ID_Low are the EUI-64 also known as the Global Unique Identifier (GUID).

Upon detection of a bus reset the generate bit abbreviated as “g” in the bus info block shall be modified if any portion of the configuration ROM has changed since the prior bus reset. The CRC in the first quadlet will be recalculated each time the generate bit is modified.

Root Directory
Offset: 0x414

Directory Length 0x04	Directory CRC (calculated)
vendor ID key 0x03	Module_Vendor_ID (can be the same as Node_Vendor_ID)
Module_Vendor_ID _Key 0x81	Module_Vendor_ID_Textual_Descriptor_Offset (indirect offset) 0x03
Node_Capabilities_Key 0x0C	Node_Capabilities 0x0083E0
Unit_Directory_Key 0xD1	Unit_Directory_Offset (indirect offset)

Module_Vendor_ID_Textual_Descriptor
Offset: 0x428

Leaf Length		Leaf CRC (calculated)	
0x41 “A”	0x42 “B”	0x43 “C”	0x08 “ ”
string continued			
			0x00

Unit Directory
Offset: 0x43C

Unit Directory Length					Directory CRC (calculated)	
Unit_Spec_ID key 0x12	Unit_Spec_ID					
Unit_SW_Version key 0x13	Unit_SW_Version					
Cmd_Set_Spec_ID key 0x38	Cmd_Set_Spec_ID					
Command_Set key 0x39	Command_Set					
Command_Set_Rev key 0x3B	Command_Set_Rev ision					
Management_Agent key 0x54	Management_Agent_Offset (initial register space offset) 0x4000 (example)					
LU_Characteristics key 0x3A	q	o	I	reserved 0x00	Mgt_ORB_Timeout	ORB_size
Logical_Unit_Number 0x14	reserved 0x00			device_type	Logical_Unit_number 0x00	
Logical_Unit_Model_ID 0x17	Logical_Unit_Model_ID					
LU_Model_ID leaf 0x81	Logical_Unit_Model_ID_Textual_Descriptor Leaf offset (indirect offset)					

Leaf Length 0x04		Leaf CRC (calculated)	
0x41 “A”	0x42 “B”	0x43 “C”	0x08 “ ”
string continued			

Management_Agent_Register
Address: 0xFFFF F001 0000 (example)

reserved 0x00	ORB_offset_hi
ORB_offset_lo	

The Management Agent Register shall be written using a 1394 Block write with 8 bytes of payload. Even though SBP-2 uses the word offset as part of the ORB address names in ORB_offset_hi and ORB_offset_lo, these values combined with the node ID form an actual IEEE 1394 64-bit address.

Unit Command_Block_Agent CSRs (address is returned in the Login response)
Address: 0xFFFF F001 0008 (example for single Login device)

Relative Offset	Name	Description
0x00	Agent_State	Reports fetch Agent State
0x04	Agent_Reset	Resets fetch agent
0x08	ORB_Pointer	Address of ORB
0x10	Doorbell	Signals fetch agent to re-fetch an address pointer
0x14	Unsolicited_Status_Enable	Acknowledges the Initiator's receipt of unsolicited status
0x18 - 0x1C		Reserved

The Unit Command_Block_Agent CSRs are provided in the Target's memory space. The Address is returned in the Login Response.

The Isochronous_Plug_Register address shall directly follow the Command_Block_Agent CSR address.

Isochronous_Plug_Register
Address: 0xFFFF F001 0020 (example)

reserved 0x00	Isochronous Channel
Bandwidth (total number of bytes - includes 1394 header and payload CRC)	

10. 1394PWG Requirements for Our Thick Transport Stack (ROTTS?)

The requirements are split into two sections: musts and wants. They are intentionally brief, with definitions of terms following each requirement.

10.1. Musts

- Support multiple, concurrent, independent, symmetrical connections

  Multiple, concurrent - Allows for more than one connection at a time. Independent - Activity on one connection has no effect on other connections.
  Symmetrical - Either endpoint can open and close the connection, and send data.
  Connection - well-bounded communication path between two endpoints. The endpoints can be on the same device or on different devices.

- Provide in-order, byte-stream and buffer (datagram?) services In-order - Data is delivered to the receiving endpoint in the same order as it was presented by the sending endpoint.

  Byte-stream - Data is delivered as a stream of bytes. The stream of bytes is not guaranteed to be delivered to the receiving endpoint in the same form as it was presented by the sending endpoint. For example, a stream of 80 bytes of data may be presented as 4-20 byte buffers, but delivered as 2-40 byte buffers. Buffer (datagram?) - Data is guaranteed to be delivered to the receiving endpoint in the same form as it was presented by the sending endpoint.
  For example, if data is presented in a buffer of 30 bytes followed by a buffer of 10 bytes, it must be delivered in a buffer of 30 bytes followed by a buffer of 10 bytes.

- Provide a directory service
  Endpoints on a specific device may be referenced by their service name. This allows connections to be opened without any knowledge of the underlying layer's implementation of sockets, etc.

- Be data, application and O/S independent
  The transport stack shall not put any requirements on the format of the data, nor shall it interpret the data in any way. The transport stack shall work with any application that correctly uses the appropriate interfaces. The transport shall be implementable under any operating system.

- Do not preclude concurrent operation of other protocol stacks
  Devices may implement and use other protocol stacks concurrently with this transport stack.

- Transparently handle transient link interruptions
  The transport stack shall handle transient link interruptions without affecting the endpoints. These link interruptions include: temporary cable disconnect, 1394 bus reset, etc.

10.2. Wants

- Connectionless service
  A non-bounded communication path between two endpoints. Data may be sent without "opening" a connection.

- Multi-casting

Simultaneously sending data from one endpoint to multiple endpoints. Does this need to be bi-directional? Does it need to be reliable?

- Bus-independent transport layer
  The transport layer may be used on other busses.

- Data tagging
  Data can be tagged as “special data” by the sending endpoint. The transport will indicate to the receiving endpoint that the data is tagged. This is also known as out-of-band data.

- Provide endpoints with fair access to other endpoints
  The transport will prevent endpoints from monopolizing the link and preventing other endpoints from access.

- Selectable quality of service
  The ability to adjust various quality of service parameters, including:
  Isochronous delivery
  Priority
  Propagation Delay
  Rate of transfer (bandwidth)

11. Discovery

The primary method for discovering devices on the Serial Bus is through information read from the Configuration ROM. Continued with information supplied by PWG and IEEE 1212 when this settles.

The Function Discovery Service is an initial attempt at performing high level 'Function' discovery.

[The Unit directory section should contain more info about the Service Discovery.]

12. SBP-2 Communication Protocol

This section is provided to get a general idea of how SBP-2 works. SBP-2 defines a method to exchange commands, data, and status between devices connected to the Serial Bus.

The terms Initiator and Target have a specific meaning derived from SBP-2 and do not imply the direction of data transfer.

Initiator: Originates management & command functions such as Login, data transfer and Reconnect

Target: Responds to management & command functions and generates Unsolicited status.

SBP-2 requires that an Initiator login to a Target to begin communication. The basic building blocks of SBP-2 include Operation Request Block (ORB) data structures. The two main types of ORBs are the Command Block ORB and the Management ORB. SBP-2 describes the services that operate on these two types of ORBs as agents.

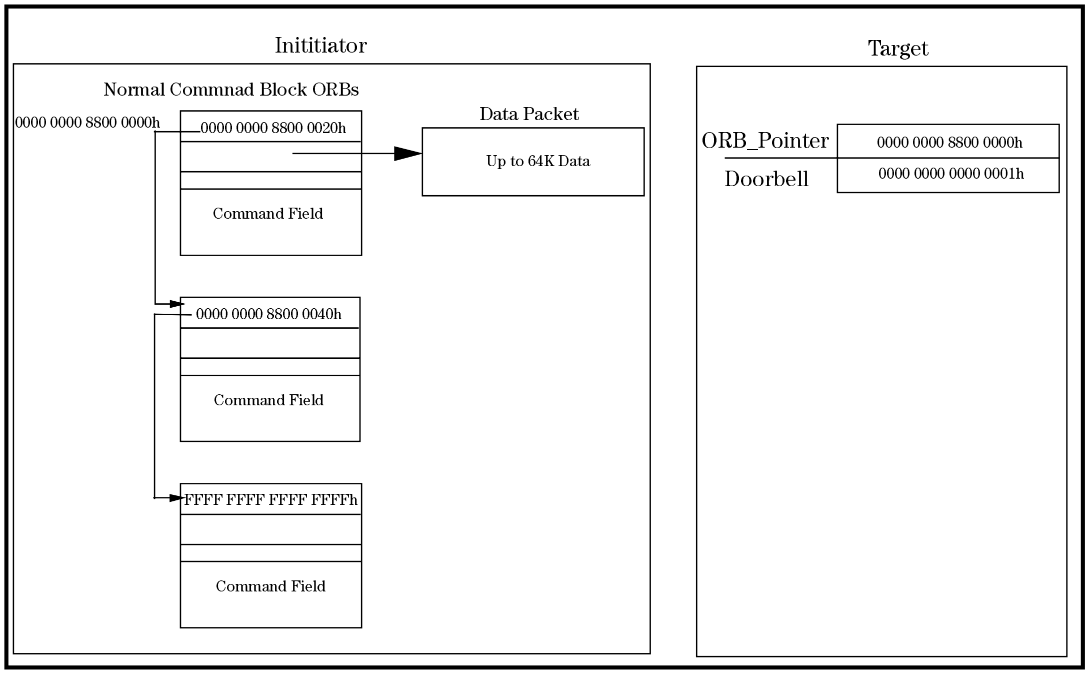
After power-on or bus reset, the Command_Agent and Management_Agent engines are in the Reset state.

The initiator reads the device’s Configuration ROM data in order to determine 1394 capabilities, SBP-2 capabilities, EUI-64 (GUID) value, command set identifiers, software versions, and Management_Agent CSR address.

The initiator performs a Login operation prior to any request to the device. To perform a Login, the initiator writes its Login ORB address to the Management_Agent register.

The device returns the Login response to the bus address specified in the Login ORB. One field of the Login response contains the Command_Block_Agent CSR base address.

Prior to initiating command transfers, the initiator builds a list of Command_Block ORBs in system memory. The list may be as short as one ORB, but this example assumes a list length of more than one. The last ORB in the list contains a NULL Next_ORB pointer, which indicates the end of the list to the device's Command_Agent fetch engine. A NULL address has the n bit (most significant bit) set to a one.



To transition the Command_Agent state from Reset to Active the initiator writes the offset of the first ORB in the ORB list to the device's ORB_Pointer. The ORB_Pointer was discovered through the Command_Block_Agent CSR. This allows the Command_Agent fetch engine to begin fetching ORBs from initiator memory. If the initiator writes to the Doorbell CSR, the device will ignore the Doorbell at this time.

The device may optionally fetch ORBs until its ORB space is full or until an ORB containing a NULL Next_ORB pointer is fetched. Fetched ORBs are routed to the Execution engine. The Execution engine may reorder the commands contained in the ORBs as long as it can guarantee in order data delivery (see section on Out Of Order ORB Processing).

The Direction bit (d) in the ORB determines the direction of data transfer from the Target's point of view. If the direction bit is zero the Target will use serial bus read transactions to fetch the data from the Initiator. If the direction bit is one the Target will use serial bus write transactions to transfer data to the Initiator. As each ORB is executed the device transfers data in the appropriate direction using serial bus block transactions.

Following the data transfer portion of each ORB the Target writes a Status Block to the Initiator's Status_FIFO address. The Status_FIFO address is the address that was obtained in the Login process. The status block contains SBP-2 specific status information as device-dependant status information.

If an ORB containing a Null Next_ORB pointer is fetched the Execution engine completes all fetched commands, including the one in the just fetched ORB, before the Command_Agent transitions to the Suspended state.

If additional commands are to be executed, the initiator creates a new list of Command_Block ORBs; changes the Next_ORB pointer in the last ORB of the old list from NULL to the offset of the first ORB in the new list and then writes to the device's Doorbell CSR address. This transitions the Command_Agent to the Active state.

The device fetches the new Next_ORB pointer value from the last ORB of the old list and begins fetching ORBS from the new list at that offset.

If the Command_Agent fetch engine has not reached the ORB containing a Null Next_ORB pointer, (and is still in the Active state) the device ignores any writes to the Doorbell CSR address.

This sequence may continue until the device is reset, power is removed, or an error occurs.

13. Why SBP-2

The 1394 PWG has examined several Protocols since its formation in early 1997 (see Appendix A) and measured them with respect to the requirements. From the beginning SBP-2 has been a leading contender and some of the primary reasons are listed below:

- Existing standard being used for other devices.
- SBP-2 is an efficient transport protocol and is optimized for 1394 DMA shared memory access.

Perceptions about its lack of bi-directional functionality caused concern. The specific test case, which SBP-2 does not address directly, occurs when a Target is obligated to send an undetermined quantity of data back to the Initiator. In this case the Initiator cannot predicate how much data the Target will send and therefore the Command Block ORBs required to receive this data may not have been built and provided in advance.

Several options to modify SBP-2 were proposed to work around this perceived deficiency. Some of these are listed below as an aid to understanding the events that lead to the current proposal.

- Multiple Fetch Agents In a Target
- Targitator - Dual Logins
- Abort Task List
- Fetch Ahead
- Bi-directional ORB (a.k.a. Request/Reply ORB)
- Single Command Block ORB

[Add: Explanation of each option above]

The main draw back to each of the options above is that they call for a significant modification of the SBP-2 spec or to use SBP-2 in a way that a standard driver might not support. The proposal, which follows in the next section attempts to enhance standard SBP-2, perhaps in a way which is unique to image devices, that could be supported by a standard SBP-2 driver.

Another major point is that since SBP-2 works well for the “normal” case of sending print data or receiving scan data the, communications protocol should not be totally distorted for the unique non performance path case.

14. ORB List Processing

This specification defines the basic communication path as a Login from an Initiator to a Target. The device that an Initiator logs into is represented by a Unit Directory and a Target may have more than on Unit Directory. A specific Initiator is only allowed one Login per Unit Directory on a specific Target

For a given Login the
Initiator provides a single linked list of ORBs called the task list and the Target fetches ORBs from this task list

15. Multiple Logical Channels

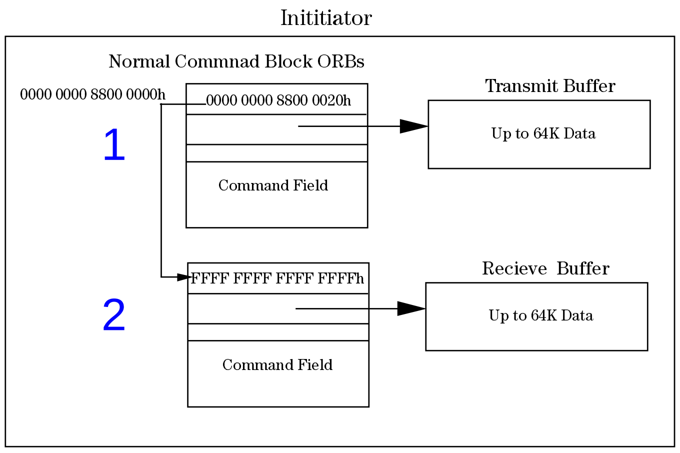
IEEE 1284.4 defines a logical channel as “An independently flow-controlled connection between a primary socket and a secondary socket. It provides a logical conduit for moving data between the two endpoints. SBP-2 neither aids nor impedes multiple logical channels.

16. Bi-Directional Communication

The basic provision for communication in a particular direction is the direction bit in a Command Block ORB. This requires that the Initiator can predict how much data will be transmitted or received.

Applications which want to transmit print data or receive scan data shall send a command and a data buffer to the SBP-2 layer. SBP-2 will build a Command Block ORB with the data_descriptor pointing to the buffer, with the direction bit set to the appropriate direction and will copy the command into the command_block field.

Applications that require a Request & Reply shall send two transactions to the SBP-2 layer. One for the transmit and one for the receive. SBP-2 will build two separate Command Block ORBs.

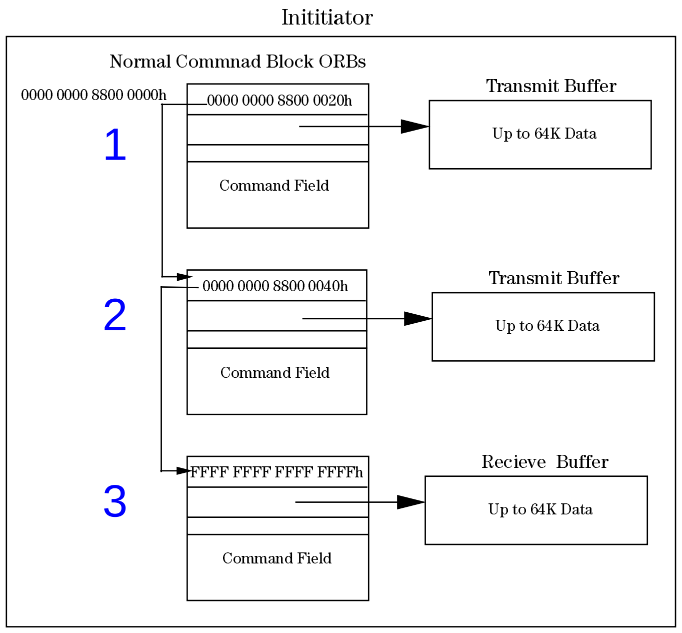


Applications that transmit data with embedded requests in the image description language shall send two transactions as described above.

1. If the application does not provide a buffer for the data in the receive (reverse) direction the Target will send unsolicited status indicating that it has data intended for the Initiator.

2. If the resulting reply is larger than the data buffer provided, the Target shall send unsolicited status indicating that it has data intended for the Initiator.

3. The application will provide an additional transaction to the SBP-2 layer with a new receive buffer. SBP-2 will build and link the new ORB at the end of the list. The result is illustrated below. ORB number 3 is the new ORB added to the list and there may or may not be other ORBs such as ORB number 2 on the list. After sending the unsolicited status as in step 1 or 2 above, the Target will finish the transmit data associated with ORB number 1. It will then search the ORB list for an ORB that points to a receive buffer. After sending the reverse data associated with ORB 3 the Target will continue processing the ORB list. If ORB number 2 exists it will be next, otherwise ORB processing will continue in the normal manner. Status Blocks will be posted according to the notify bit.



4. If the Application chooses to ignore the receive data, the Target will time out on the reverse communication attempt and continue accepting transmit data. The timeout value will be the Mgt_ORB_Timeout period specified in the Target's Unit directory.

Another case is Target generated unsolicited data. The basic method to communicate unsolicited data is similar to the scenario above. The Target shall send unsolicited status indicating that it has data intended for the Initiator. The Application will provide an additional transaction to the SBP-2 layer with a new receive buffer. SBP-2 will build and link the new ORB at the end of the list. ORB processing is the same as case 3 above.

17. Out of Order ORB Processing

In general an Application should attempt to allow for data that will flow in the reverse direction by sending the appropriate transactions to the SBP-2 layer. Exceptions are outlined in the previous section. The added complexity of processing Command Block ORBs out of order is implementation dependent and is left to the Target. The rules for in order data delivery apply across a given channel. Even though it is admissible to process ORBs out of order the data in a particular direction shall be delivered in order. The Initiator shall build ORBs to deliver transmit data in order. The Target shall send data to the Initiator in order. On a given channel all outbound data shall be sent in order and all inbound data shall be received in order. The Target is allowed to process an ORB for inbound data ahead of other ORBs that contain outbound data. Information for channel B may be processed ahead of channel A.

If Target cannot do out of order processing should the Target return status of number of bytes transferred and return status of no bytes transferred for other ORBs in the list until it gets to the one that has the receive buffer?

18. PEER to PEER

The peer to peer requirement states that any device can initiate communication. Since SBP-2 is architected around an Initiator and a Target model true peer to peer is not a natural fit. If peer to peer is really needed it can be accomplished at least two different solutions may be considered.

The Imaging Device Profile provides a mechanism for two nodes to communicate with one another in a peer to peer fashion. The mechanism requires each node to implement target and initiator functionality as defined in the SBP-2 specification. The command set used by such a device must be able to adapt to the differences of being either the initiator or target as defined by SBP-2. The details of such an implementation are beyond the scope of this specification.

Another possibility is to define a “Login Solicitation Register”. This register would allow a Target to request that an Initiator perform a Login to the Target.

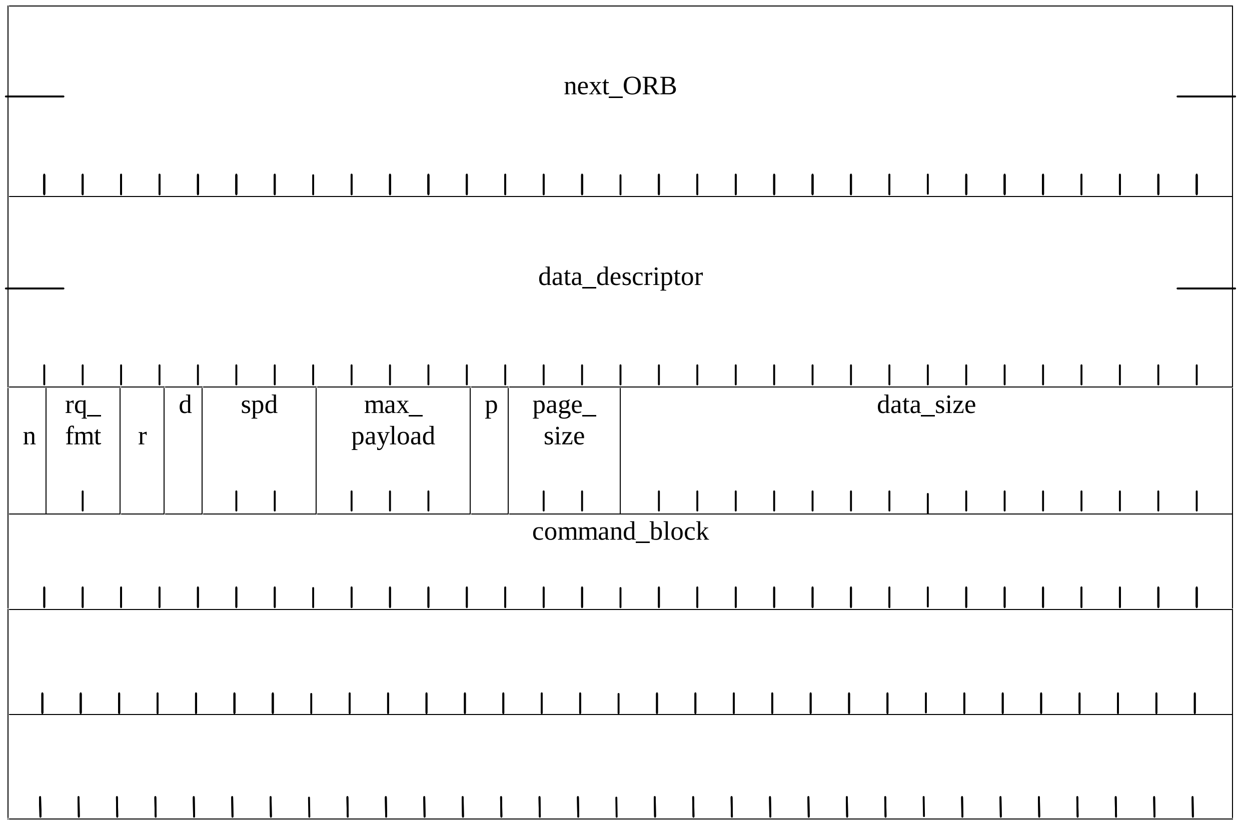
19. Multiple Host and/or Multiple Device

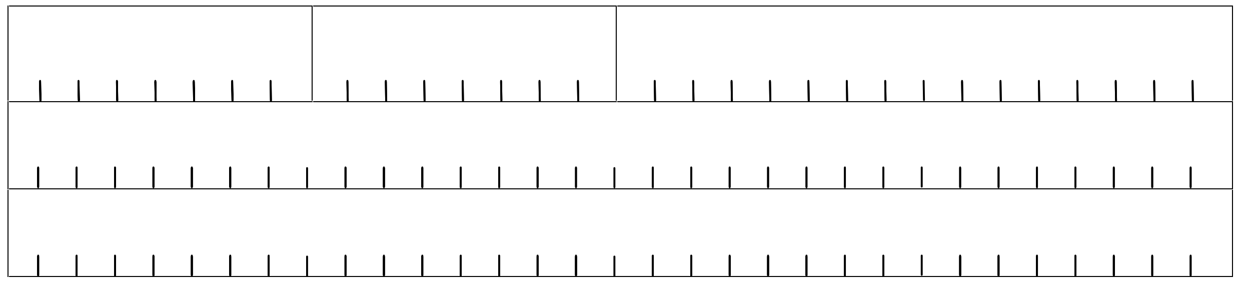
There is a need to provide fair access to devices on a 1394 bus. Each node is accessible from any of the other nodes on the bus and possibly nodes outside of the bus in a bridged environment. It is reasonable to expect that more than one node may attempt to communicate with a target node at the same time.

SBP-2 provides a Login function. A successful Login creates a connection between two nodes. This creates the required instance data within the target memory. Devices that conform to this profile are required to support a minimum of a single Login. A specific implementation may support multiple logins and arbitrate between them. The details of such an implementation are beyond the scope of this specification.

20. Command Block ORBs

Command Block ORB





21. Login & Login Response

This section will specify the details of the Login Process.

The primary reasons for Login are access control, unsolicited status and the simple SBP-2 reconnect scheme.

1) The Initiator will discover a device by reading the CSR & Configuration ROM space of all devices on the 1394 bus.
2) The Management_Agent_Rregister is also discovered at this time.
3) The Initiator will record the Target's GUID.

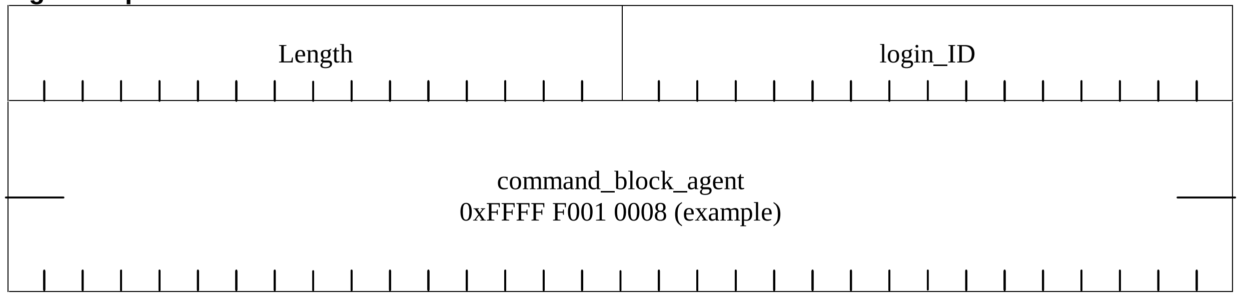
Login ORB

password					
login_response					
n	rq_ fmt	x	reserved 0x00	Function 0x00	lun
password_length 0x00				login_response_length 0x05	
Status_FIFO					

4) The Initiator will build the Login ORB with password.
5) The login_response address is the temporary memory address that the Target will use to send its response to the Login.
6) The Status_FIFO address will remain static during the life of the Login.
7) The Initiator will write the address of the Login ORB to the Target's Management_Agent register. [Target shall monitor writes to this address or set up an Interrupt]
8) The 1394 speed (s100, s200, s400) of the write to the Management Agent register will determine the speed used for communication.
9) The Target will read the Login ORB.
10) The Target will read the Initiator's bus information block to discover the GUID.
11) The Target will validate this new Login by comparing the GUID against current login_descriptors. If this Initiator is already logged in the Login shall be rejected. If the Target only supports one Login and another device is logged in, the Login shall be rejected.

12) The Target will build a login_descriptor data structure that will be associated with this specific login.
13) The Target will store the Initiators GUID in the login_descriptor login_owner field.

Login Response ORB



14) The Target will build the Login Response ORB and fill in the login_ID. The login_ID is like a connection identifier that is unique across active Logins.
15) The Target will store the login_ID in the login_descriptor.
16) The command_block_agent address points to the Unit Command_Block_Agent CSRs in the Configuration ROM.
17) Finally the Target writes the Login Response ORB to the login_response address.

Writing “resources_unavailable”, in the sbp_status field of the status block, to the Login’s Status_FIFO address will reject a Login.

22. Unsolicited Status

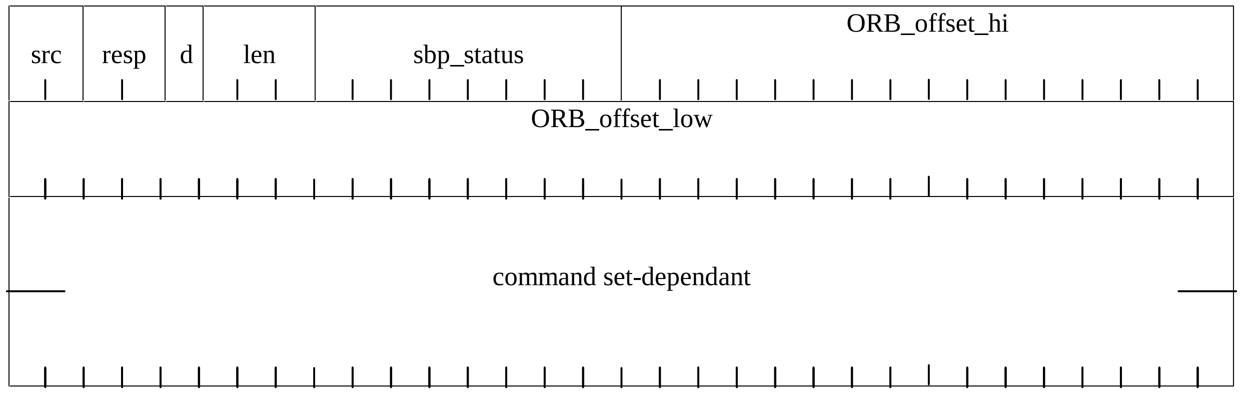
1. The Target shall clear it’s Unsolicited_Status_Enable register after a successful Login.
2. The Initiator shall write a one to the Target’s Unsolicited_Status_Enable to allow Unsolicited Status.
3. The Target may send Unsolicited status, using a Status Block, to the Initiator using the Status_FIFO address that the Target received during Login.
4. The Target shall use its Unsolicited_Status_Enable register to handshake this status block.
5. The Target can only store status when the Unsolicited_Status_Enable register is set to one.
6. After writing the status, the Target shall clear this register.
7. The Initiator shall write a one to the Target’s Unsolicited_Status_Enable to allow subsequent Unsolicited Status.

The reason for the handshake for the Unsolicited status is because of it's unsolicited nature. The initiator when preparing a FIFO to receive status knows how many ORB's it has given or will give to the target. The Initiator can allocate enough FIFO for those status reports. Since the Initiator does not know how many Unsolicited reports it may receive it is required to allocate at least one FIFO location and use the handshake with Unsolicited_Status_Enable when that one is available.

<u>23. Status Block</u>

If status is sent to the Status_FIFO in response to a management ORB the ORB_offset fields will contain the appropriate address. ORB_offsets for Unsolicited Status will be set to zero.

Status Block



Implementations are not required to use all of the status information specified in the tables that follow. Could add information that states which codes are recommended in an appendix ?

src field

Value	Description
0	The status block pertains to an ORB identified by ORB_offset; 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; 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 field shall be ignored.
3	The status block is unsolicited and contains isochronous error report information as specified by 12.3.

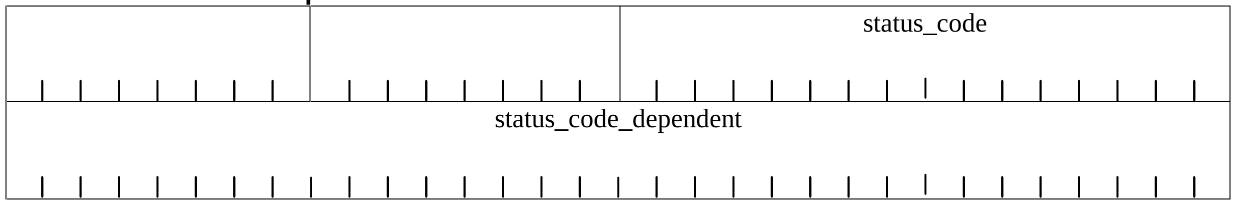
resp field

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 in the ORB; the sbp_status field may provide additional information
3	VENDOR DEPENDENT	The meaning of sbp_status shall be specified by this specification

sbp_status field

Value	Description
0	No additional sense 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	Too many channels
8	Resources unavailable
9	Function rejected
10	Login ID not recognized
11	Dummy ORB completed
12	Request aborted
0xFF	Unspecified error

1284.4 Command Set-Dependent Status

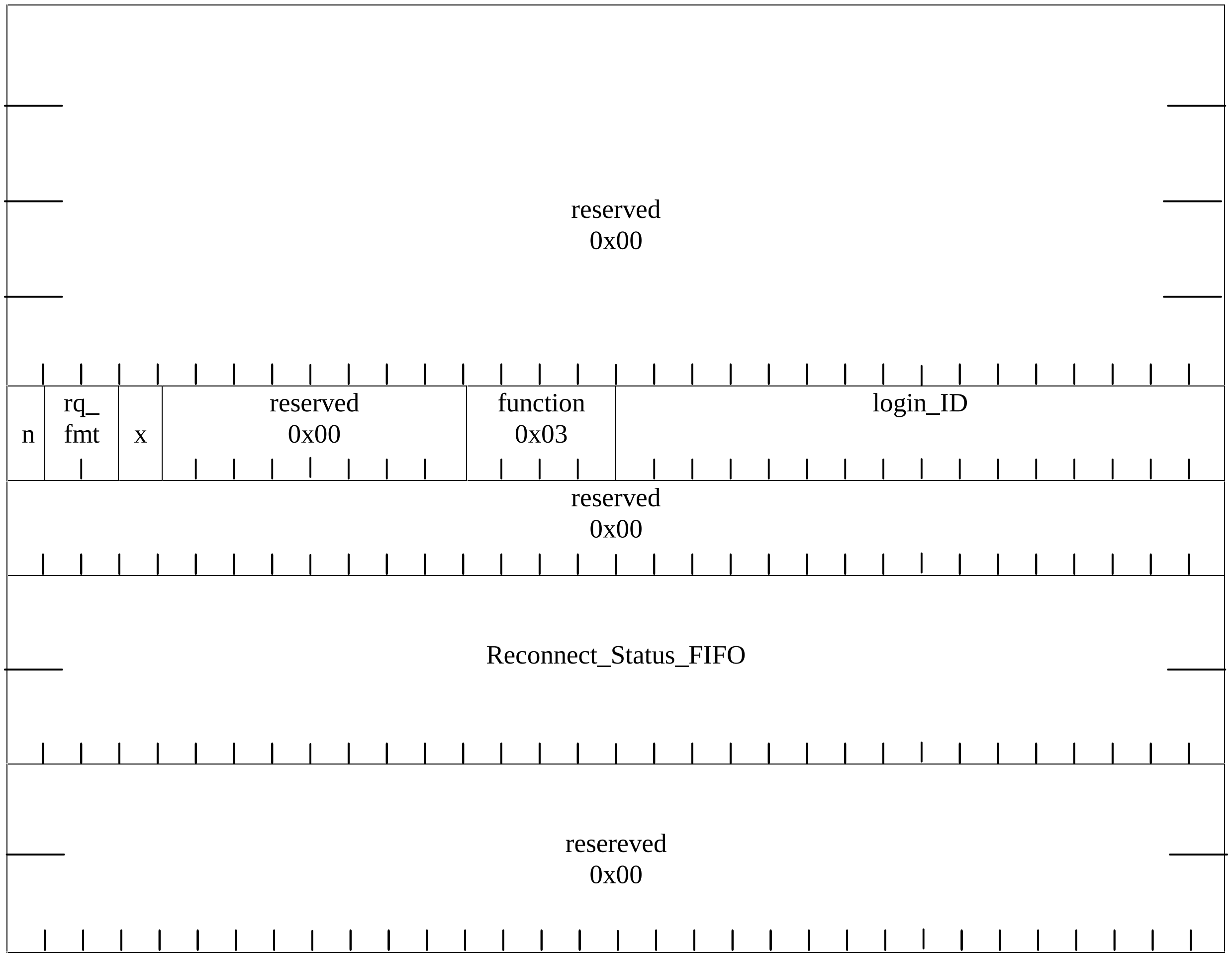


status_code	description	status_code_dependent
0x01	data transfer complete	actual number of bytes transferred
0x02	Target to Initiator transfer request	number of bytes to transfer

24. Reconnection

Reconnection after a Bus Reset will be accomplished using the Reconnect ORB.

Reconnect ORB



1. After a bus reset, the Initiator is required to re-discover the Target that it was Logged into by reading the Bus Information Blocks of nodes on the bus searching for a matching GUID.
2. After a bus reset, if the Target was connected by a Login the, Target will start a timer. The SBP-2 specification suggests 2 seconds, we may want to tune this value.
3. Once the Target is found the Initiator can write the address of the reconnect ORB to the Target's Management_Agent register.
4. The 1394 speed (s100, s200, s400) of this write will determine the speed used for communication.
5. The Initiator shall use the same login_ID that the Target provided at Login. The Target will fetch the reconnect ORB and read the Initiators Bus Information Block to verify that the Initiators GUID match the GUID established at Login.
6. The reconnect is completed when the Target writes status to the Reconnect_Status_FIFO. Note that this is a new separate Reconnect_Status_FIFO, which is not the same Status_FIFO established at Login.
7. After reconnect the 48 bits of the Login Status_FIFO is used for unsolicited status.

8. The Login Status_FIFO address may have to be patched with the Initiators new node number and bus number.
9. The Data_FIFO_Addresses may also have to be patched with the new Node number and Bus number.
10. If the Target's timer expires before a reconnect ORB is provided the Target will perform an automatic Logout. Logout consists of resetting the login descriptor variables to their initial values. The Unit Command_Block_Agent CSRs should also be reset to initial values.

A special case occurs when an active Login exists between an Initiator and a Target if the Initiator is power cycled independent of the Target. After the bus reset the Target is expecting a Reconnect and the Initiator will attempt a new Login. The Target shall refuse the Login if it happens before the Targets timer has expired. Initiators shall retry the Login after waiting a timeout period.

25. Query Login

26. Logout

27. Isochronous Data Transmission

28. 1394 Bus Reset Behavior

After a Bus Reset:

During idle state - no data transactions pending

Reconnect as described in preceding section.

During Asynchronous data transmission

Reconnect as described in preceding section and continue data phase.

During Isochronous data transmission

Continue with Isochronous data transmission.

29. Error Recovery

Asynchronous Data Transmission Error Recovery

Any packet, which contains a CRC error, shall be re-transmitted when the error_ACK is returned to the sender.

Appendix A - Protocol Proposal Comparison

The 1394 PWG has explored several options for peripheral communications protocols. In general proposals have gravitated towards SBP-2 and IEC 61883 (FCP). As these were examined with respect to the list of 1394 PWG requirements both of these seem to …. to be continued