1394 PRINTER WORKING GROUP

IEEE 1394 HIGH SPEED BUS

IMAGING DEVICE

COMMUNICATIONS SPECIFICATION

***PRELIMINARY DRAFT PROPOSAL ***

Revision 0.55 - February 23, 1999

Editor - Alan Berkema
8000 Foothills Blvd. MS #5558
Roseville Ca, 95746
916 785-5605
Fax 916 785-1195

1. Contents

Page

2. Figure

Page

3. Scope

This document specifies the communications profile and device communications command set for imaging devices attached to the IEEE 1394 High Speed Serial Bus.

SBP-2 provides multiple, concurrent, independent connections which do not preclude concurrent operation of other protocol stacks; is data, application, and OS independent. In order to meet the requirements for imaging devices, SBP-2 must be supplemented by this a device profile.

Information supplemental to the IEEE 1394 and SBP-2 specifications are provided in this document.

- a policy to be used for maintaining device access across “transient” link interruptions
- a model of use for maintaining guaranteed, in-order data deliver across “transient” link interruptions
- a command set which supports
  - independent, bi-directional, half-duplex communication
  - data tagging of an associated data payload
  - ability for either end of a connection to close the connection at any time

This specification does not address:

- Isochronous communication
- Use with 1394.1 bridges.
- Security.

4. Purpose

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 image devices in general including 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. Requirements are specified to allow imaging device communication conformance to SBP-2. In all areas that concern transport protocol, SBP-2 should be followed. Where SBP-2 allows more than one choice of implementation, this profile defines the choice for imaging devices.

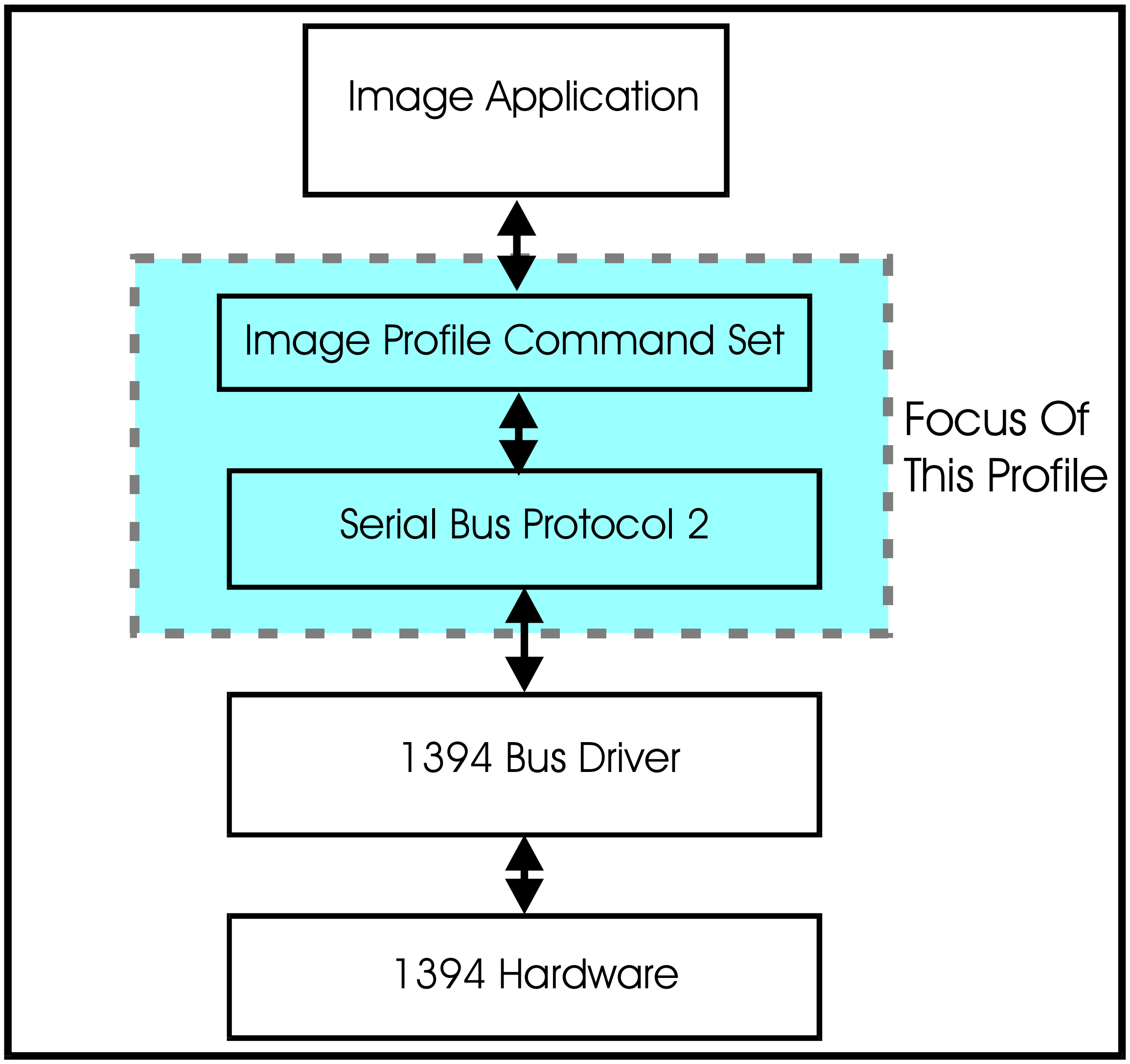

Figure 1 - Focus Block Diagram

5. References

This document makes reference to and contains excerpts from several industry standards. The revisions of those standards listed are current at the time of this document's release. However, each standard referenced is subject to change. More recent revisions may or may not support the information contained in this document:

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

5.1. Bit, Byte and Quadlet ordering

See IEEE std. 1394-1995.

5.2. Control & Status Registers (CSR)

All 1394 PWG devices shall implement the CSRs as defined in ISO 13213/IEEE 1212:1994 and IEEE Std 1394-1995. Required registers are the same as for SBP-2.

6. Configuration ROM

7. Function Discovery

The primary method for discovering devices on the Serial Bus is through information read from the Configuration ROM.

This section will be continued with the Function Discovery Service information supplied by the PWG and IEEE 1212r when this is specified.

8. Communication Model

This specification defines the basic communication path as a Login from an Initiator to a 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.

For bi-directional communication, devices may use the out of order ORB processing model described in the next section.

8.1. Uni-Directional Communication

The initiator may configure the Target for data transfer in a single direction. Either from the Initiator to the Target or from the Target to the Initiator. See the command set parameters for details.

8.2. Bi-Directional Communication Model

This specification recommends that the Initiator configure the Target for data transfer in both directions. From the Initiator to the Target and from the Target to the Initiator. See the command set parameters for details. This profile provides full duplex communication capability between an initiator device and a target device using an unordered ORB processing model. Data transfer within a specific direction is accomplished in order with respect to that direction. If the Initiator configures the device for bi-directional communication it shall insure that a sufficient number of ORBs are available for communication in each direction. The total number of ORBs on the task list which includes ORBs for each direction can be configured using the set parameters command with the max task set size parameter.

8.3. Target Model

Figure 2 illustrates an example block diagram of a command block agent for the target. The command block agent contains one command fetch agent, two command pre-fetch queues, called **Write queue** and **Read queue**, and two execution agents, called **Write execution agent** and **Read execution agent** connected to the Write queue and the Read queue respectively.

The command fetch agent fetches the normal command block ORB's in order. When the command fetch agent fetches the normal command block ORB, the command fetch agent examines the command specified in the *command_block* field of the command block ORB. The fetch agent dispatches the command block ORB to either the Write queue or Read queue according to the parameter. All Write commands are dispatched to the Write queue, and all Read commands are dispatched to the Read queue. The Write execution agent and Read execution agent, execute the commands queued in the Write queue and Read queue respectively.

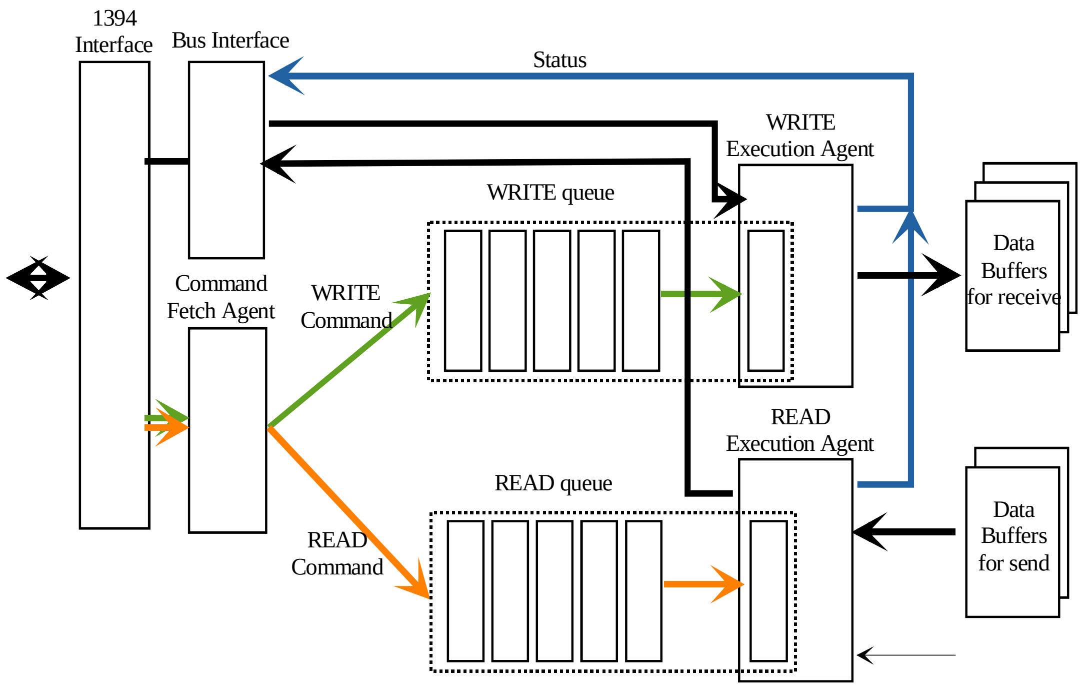

Figure 2 - Target Model

Each execution agent executes the dispatched command in the connected queue in order.

Both execution agents are independent of each other. Each execution agent executes the data transfer associated with the command according to the parameters specified in the command.

The target stores a status block in the initiator's memory according to the value of the *notify* bit of the command block ORB after executing the command as specified by SBP-2. Each execution agent shall store the status_block in order of execution for that particular agent.

While fetching ORBs the fetch agent is free to process the task list according to the values in the next_ORB field. After an ORB has been fetched and placed in the Read or Write queue, fetch agents (and execution agents) shall not refer to Initiator memory addressed by the *next_ORB* field. The ORBs addressed by the *next_ORB* field in the initiator's memory may not contain a valid pointer since the addressed ORB may already be completed due to unordered execution.

8.4. Initiator Model

The initiator may have two **i/o request queues** as illustrated below.

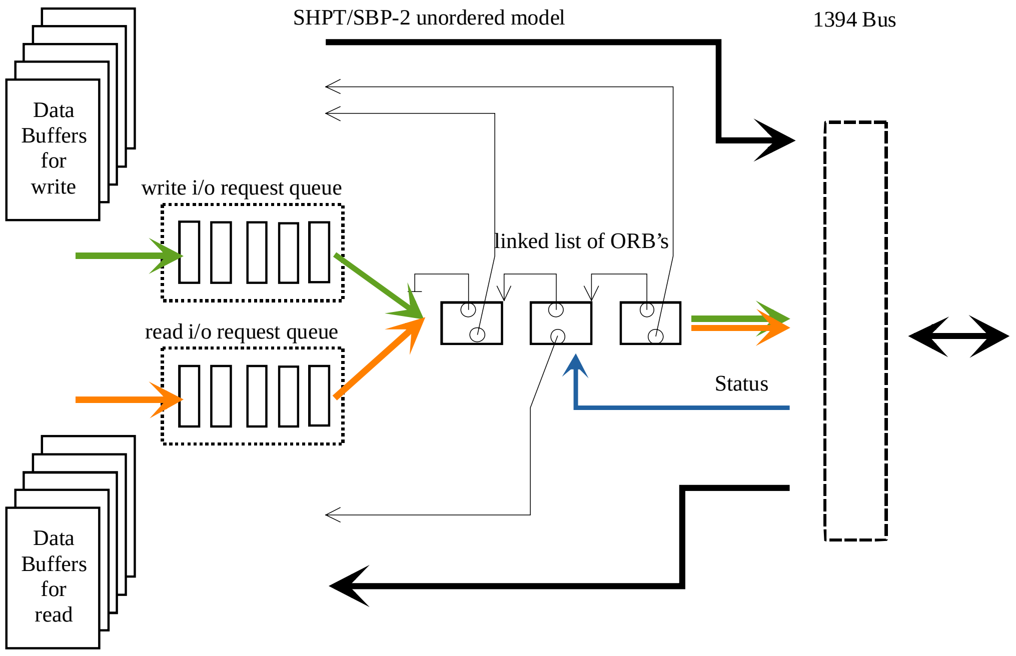

Figure 3 - Initiator Model

The **Write queue** and **Read queue** in the target queue the command ORB's destined to the **Write execution agent** and **Read execution agent** respectively. The initiator may only append a new **task** to a current **task list** when there is space in the list according to the combined depth count for the task list. In order to manage this constraint, the initiator retrieves the depth of the task list from the target before starting communication.

The SBP-2 status block notifies the Initiator that an item on the task list has been completed. The Initiator may release the memory associated with the ORB and the request can be removed from the appropriate queue according to the command.

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

NOTE: The initiator does not need to update the *next_ORB* field of the completed ORB in the current **task set**, since the target never refers to this field retroactively.

9. 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 initiator node may attempt to communicate with a target service 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.

10. Management ORBs

The following table specifies support for Management ORB functions.

Table 12 - Management ORB Functions Support List

Function Value$_{16}$	Management Function	Support Level
0	Login	Mandatory
1	Query Login	Mandatory
3	Reconnect	Mandatory
4	Set Password	Optional
7	Logout	Mandatory
B	Abort Task	Not supported
C	Abort Task Set	Mandatory
E	Logical Unit Reset	Not supported
F	Target Reset	Mandatory

11. Login & Login Response

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

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

The *login_response_length* shall accommodate the size, in bytes, the Login Response specified in this standard.

12. Unsolicited Status

As of the May 1998 PWG meeting a need for Unsolicited Status has not been identified.

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.

If a Target wants to send unsolicited status and the Unsolicited_Status_Enable regsiter is not set the Target shall wait at least a reconnect time out period before asserting a 1394 bus reset.

13. Command

13.1. Status FIFO write request recovery

SBP-2 identifies that a target should take no error recovery actions when it detects a missing acknowledgement after a write to an initiator’s status FIFO. It also states that an initiator is expected to discover this error by a higher-level mechanism and to initiate appropriate error recovery actions, though the details are beyond the scope of SBP-2.

When a target detects a missing acknowledgement after a write to an initiator’s status FIFO, it does not know which of the following applies:

- The information was correctly received in the status FIFO (and the initiator knows that the command has been completed); or
- The information had a CRC or other error, has not been correctly received by the initiator and the command is still uncompleted.

To recover from this condition, the initiator must determine when to take error recovery actions, and the target must be able to tolerate executed, but uncompleted non-idempotent commands. (Idempotent commands have indistinguishable effect whether they are executed once or many times in succession.)

NOTE – Whatever mechanism is defined needs to be broad enough to cover standard commands, vendor-specific commands, and unsolicited status notifications.

When this condition occurs, the target shall proceed as follows:

a) It shall stop execution of all commands.

b) It shall remember the specific command and queue which had the error.

c) It shall release all resources allocated by the target which are associated with the ORB.

d) It shall update the ORB_POINTER register to reference an offending ORB.

e) It shall transition the Fetch Agent to the DEAD state.

This allows the initiator to detect the erroneous condition by reading the AGENT_STATE register and recover from it by re-queuing the uncompleted task list.

Alternatively, the condition would automatically be corrected as a consequence of the next Bus Reset event. Upon completion of a successful reconnection, the initiator must restore TRANSPORT_I2T_DATA and TRANSPORT_T2I_DATA commands and associated data to the task set in the order previously queued within the target. No assumptions shall be made about ORB or buffer locations remaining the same within the 1394 shared memory space, or about the same physical buffers being used.

NOTE – For this device model, this case is assumed to be extremely infrequent, so error recovery actions with a high impact on the bus environment are assumed to be tolerable. It is anticipated to be acceptable for the target’s node to generate a Bus Reset when this condition occurs.

13.2. Queue Management

To simplify queue management and error recovery, the ABORT TASK and LOGICAL UNIT RESET management function shall not be supported.

For stream services, aborted ORBs must be requeued in the same order as previously existed, and must retain the sequence numbers previously assigned.

For message services, re-queued ORBs must retain the sequence numbers previously assigned, and must be restored to the same queue order as before.

13.3. Sequence Number Management

In order for the target to distinguish between completed and acknowledged, executed but uncompleted, and partially or unexecuted commands within a data stream, each command shall contain a queue identifier and associated sequence number. The sequence number shall be unique to the queue.

The sequence number found in the first command placed on each queue shall be used as the initial sequence number value. At any point in time all data block sequence numbers on a queue must lie within one quarter of the size of the sequence number space. Sequence numbers are assigned in ascending order. Once a sequence number has been assigned to a data block, it may not be reused for a different data block until it falls outside the sliding window of usable sequence numbers. Commands, which do not transfer application data still implicitly have a (possibly empty) data block, and so still require a sequence number.

NOTE – For the following discussion Sna represents the sequence number of the last completed and acknowledged command on this queue. Sne represents the sequence number of the last fully executed, but uncompleted (or unacknowledged) command on this queue. Snl represents the sequence number of the command with a lost Ack on a status FIFO write. Sn represents the sequence number of the command being examined. The function dist(A,B) is evaluated as ((A + sequence_number_range – B) % sequence_number_range) and will produces only positive distances.

Whenever a target begins processing a command, it shall examine the sequence number. If the target is trying to recover from a lost status FIFO write Ack:

a) If (dist(Sn, Snl) >= sequence_number_range / 2) then the command had been successfully executed. If the notification bit is set, the command completion notification is immediately given.

b) Else if (dist(Sn, Snl) == 0) then the command has been requeued. The target shall only (re)send the previous completion notification. Upon receipt of the Ack on the status FIFO write, the target shall be synchronized with the initiator.

c) Else if (dist(Sn, Snl) > 0) then the initiator received the previously attempted (successful) completion notification, and the target is resynchronized with the initiator. The sequence number has been implicitly acknowledged.

When the target is not trying to recover from a lost status FIFO write Ack, it shall proceed as follows:

d) If (dist(Sn, Sna) == 0 || dist(Sn, Sna) >= sequence_number_range / 2), then the command has already been executed. If the notification bit is set, the previous completion notification is immediately given.

e) Else the command is executed.

When an initiator queues a command, it may only do so if:

a) The size of the active task set is less than the size of the Maximum active task set supported by the target

b) And, a unused sequence number is available < (sequence_number_range / 2) distance away from the last acknowledged sequence number.

In this case the initiator shall use the lowest available sequence number.