# Simple High Performance Transport - SHPT (Draft)

Revision 0.4d

March 31, 1998

Shigeru Ueda

Takashi Isoda

Akihiro Shimura

Contact e-mail address: oid3-1394@pure.cpdc.canon.co.jp

CANON INC.

## Contents



## Tables

## Figures

# 1. Overview

## 1.1 Scope

This document defines the command set protocol for transporting data between computer system and peripheral device on top of Serial Bus Protocol 2 (SBP-2). The SBP-2 is a transport protocol defined for IEEE Std 1394-1995, Standard for a High Performance Serial Bus.

This document defines the following attributes and features, required to interface devices via the standard IEEE Std 1394/SBP-2 mechanisms:

Command block and status block formats

Task management and behavior model

## 1.2 Purpose

The design of the SHPT was intended to meet the following objectives:

a) *High Performance and low overhead*. The protocol should efficiently utilize the high bandwidth of IEEE 1394 and processing resources on devices.

b) *Flexible bi-directional data transport*. The protocol should enable full duplex communication between the devices.

c) *Multiple service channels*. The protocol should provide multiple communication channels between the devices.

d) *Application independence*. The protocol should be independent of application and should not depend on a particular device class or control set.

e) *Backward compatibility with bus environment support.* The protocol should cover the application that conventional point-to-point interface like a parallel port covers. The protocol should also provide additional support to take the advantage of the bus environment features like device sharing for such application.

Note: To focus on the bi-directional communication, this document does not include above "c)" and "e)" parts.

# 2. References

This document shall be used in conjunction with the following publications. When they are superseded by an approved revision, the revision shall apply:

ISO/IEC 13213:1994, Control and Status Register (CSR) Architecture for Microcomputer Buses

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

ANSI X3T10 1155D, Serial Bus Protocol 2 (SBP-2)

# 3. Definitions

## 3.1 Conformance

Several keywords are used to differentiate levels of requirements and optimality, as follows:

3.1.1 expected: A keyword used to describe the behavior of the hardware or software in the design models assumed by this document. Other hardware and software design models may also be implemented.

3.1.2 may: A keyword that indicates flexibility of choice with no implied preference.

3.1.3 shall: A keyword indicating a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other products conforming to this document.

## 3.2 Glossary

The following terms are used in this document:
(Omitted at this point.)

# 4. Acronyms and abbreviations

The following are abbreviations that are used in this document.

| | |
|---|---|
| SHPT | Simple High Performance transport (this document itself) |
| ORB | Operation request block |
| SBP-2 | Serial Bus Protocol 2 |

# 5. Simple High Performance Transport(SHPT) Model (informative)

Note 1: This document focuses how the command-set dependent *unordered model* under the queuing model defined by the SBP-2 provides efficient bi-directional communication. This document also describes a simple method to manage the linked list of ORB in the initiator that utilizes the model defined in this document.

Note 2: Multiple service channels are assumed to be provided via utilizing multiple WRITE/READ queue pairs or some other means. Extended reconnect time out is also assumed to be provided via the mechanism introduced by the imaging profile or some other means.

Note 3: The command and status related to device control are not included in this document to focus on the bi-directional communication.

Simple High Performance Transport (SHPT) defines a small command set and a behavior model on top of the Serial Bus Protocol 2 (SBP-2). The SBP-2 is a transport protocol defined for IEEE Std 1394-1995, Standard for a High Performance Serial Bus. The SHPT provides full duplex communication capability between an initiator device and a target device.

This clause describes components of the SHPT model. In addition to the information in this clause, users of this document should also be familiar with the SBP-2 and its normative references.

The SHPT uses the data exchange mechanism provided by the SBP-2. The command block ORB (operation request block) works as a data transfer request for both direction as specified by the SBP-2. The SHPT introduces a queuing model on the target to queue those requests. In order to achieve full duplex communication, the SHPT defines a task management model under the queuing model defined by the SBP-2, and controls the flow of those requests by using each queue. The unsolicited status defined by the SBP-2 may be used as a request indication for an asynchronous data transfer from the target to the initiator.

The SHPT provides full duplex communication capability over single login with small additional resources by adopting the flow control based on the requests rather than on the transporting data itself. The benefits of the SBP-2 like high performance and low overhead are achieved via the features of SBP-2 like shared memory model and listed execution. The SHPT also inherits these benefits by relaxing the synchronization between both ends with the queuing model.

## 5.1 Target Model

Figure 1 illustrates an example block diagram of a command block agent for the SHPT 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 parameter specified in the *command_block* field of the command block ORB. The fetch agent dispatches the command block ORB to either of 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.

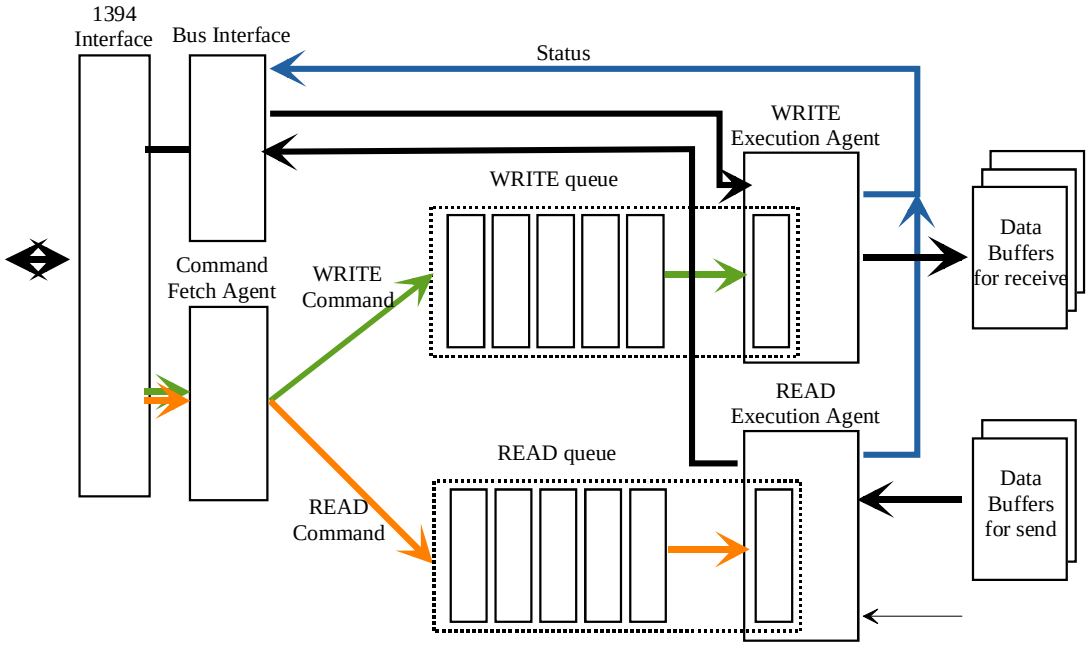


**Figure 1 - Target Model**

Each execution agent executes the dispatched command in the connected queue in order and independently of other agent. 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 the SBP-2. Each execution agent shall store status_block in order within each execution agent.

The fetch agent (and execution agents) shall not refer to the *next_ORB* field of the ORB's in the

initiator's memory that is already fetched and already in either pre-fetch queues except for the ORB contains null *next_ORB* pointer.

Note: The *next_ORB* field of the backward ORB's in the initiator's memory may not be valid pointer any longer since the pointed ORB may already be completed by the unordered execution.

When the target has data to be sent to the initiator and no READ command is available from the initiator, the target may store a **data available status** in the initiator memory. This status block may be either a normal status_block or an unsolicited status defined by the SBP-2. This status may be used as an data indication from the target by the initiator.

## 5.2 Initiator Model

The initiator of the SHPT has two **i/o request queues** as illustrated below.

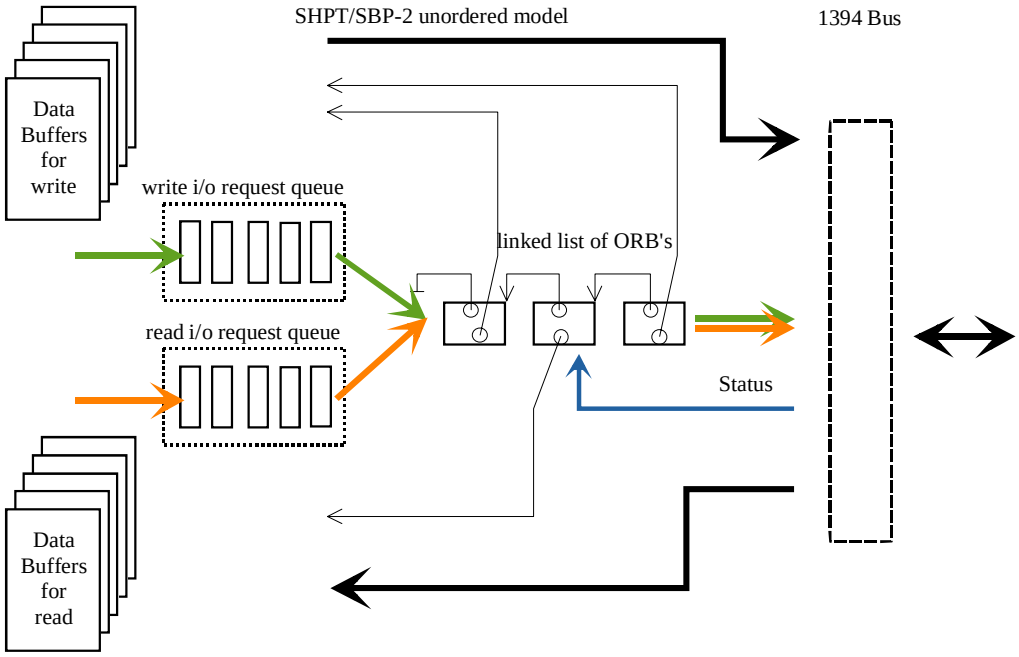


**Figure 2 - Initiator Model**

The initiator manages a constraint on appending a new **task** to a current **task set**.

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 restricts to append a new **task** destined to the each execution agent in the manner that the number of the commands destined to each execution agent in the **task set** does not exceed the available depth of each queue in the target.

In order to manage this constraint, the initiator retrieves the depth of the each queue from the target before starting a communication.

The initiator creates a command that specifies the **WRITE execution agent** as a destination in case of the data transfer from the initiator to the target. The initiator creates a command that specifies the **READ execution agent** as a destination in case of the data transfer from the target to the initiator.

The initiator becomes aware that the target has consumed the content of each queue by receiving the status block specified by the SBP-2 corresponding to the command destined to each queue.

The initiator may free the completed ORB indicated by the status block, and complete i/o request in the head of the **write i/o request queue** or the **read i/o request queue** depending on the information in the status and remove the i/o request from the queue.

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

## 5.3 Error Recovery

The initiator may detect that the target has aborted the execution of a certain task and stopped processing of succeeding tasks in the list via status block or agent state register. When the initiator detects this case, the initiator shall discard all ORB's in the current task set and re-initiate fetch agent with recreated linked list of ORB from the contents of **write i/o request queue** and **read i/o request queue**. The initiator shall maintain relationship between i/o request in each queue and corresponding sequence identifier. The initiator shall also maintain the contents of the buffer associated with each request.

The target shall be responsible to prohibit to duplicate processing of the content of each i/o request. In order to do this, the target maintains the sequence identifier and buffer offset currently processing. After the target aborted the execution of a certain task and re-initiated by the initiator, the target shall examine the sequence identifier in new ORB. If the target finds from the sequence identifier that the request is already executed, the target may complete the request without execution. If the target finds from the sequence identifier that the request was processed intermediately, the target may continue processing from the point indicated by the buffer offset.

## 6. Data Structure (tentative)

There are two classes of data structures defined by SHPT:

- operation request blocks (ORB's);
- status blocks.

Note: The data structures described in this clause may be neither concrete nor consistent at this point. The description intends to help understanding how SHPT works and what SHPT appears.

### 6.1 Normal command block operation request blocks (ORB's) (tentative)

The format of the normal command block ORB is illustrated by the figure below.

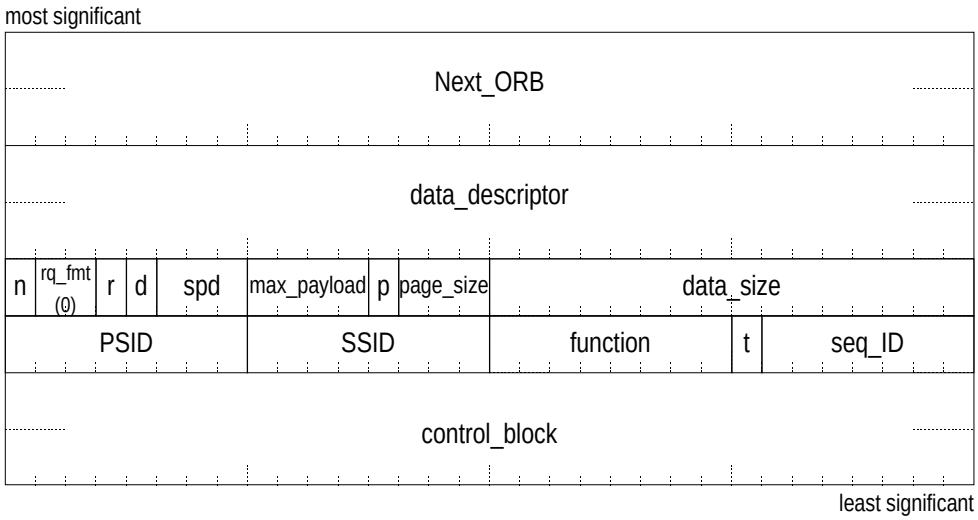


**Figure 3 - Normal command block ORB**

The *PSID, SSID* field specifies the communication channel identifier.

The *function* field specifies the SHPT command requested, as defined by the tables below.

| Value | SHPT command |
|---|---|
| 0 | WRITE |
| 1- $3F_{16}$ | reserved for future standardization |
| $40_{16}$ | READ |
| $41_{16}$- $FF_{16}$ | reserved for future standardization |

**Table 1 - SHPT commands**

The *tag* bit (abbreviated as *t* in the figure above) specifies data tag.

The *seq_ID* field specifies the sequence identifier.

The format of the *control_block* field is uniquely determined by a combination of *function* field, the control set implemented by the target. This document specifies those parts of the ORB that are invariant across target control sets.

All other fields are described by the SBP-2 standard.

### 6.1.1 WRITE command ORB (tentative)

The WRITE command requests the target to fetch data in the referenced data buffer destined for the target device with Serial Bus read transactions.

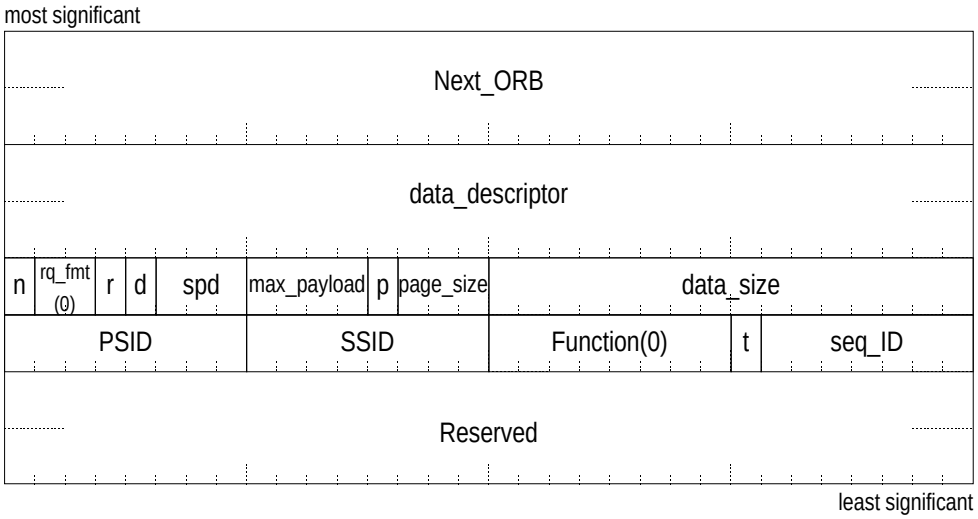


**Figure 4 - WRITE command ORB**

The *data_descriptor* field, *page_table_present* bit (abbreviated as *p* in the figure above), *page_size* field and *data_size* shall specify the data buffer as specified in SBP-2.

The *direction* bit (abbreviated as *d* in the figure above) specifies direction of data transfer for the buffer as specified in SBP-2 and shall be zero.

The *tag* bit (abbreviated as *t* in the figure above) shall be one if the data is tagged as out-of-band, and shall be zero otherwise.

The *seq_ID* field specifies the sequence identifier for the data buffer, and shall be assigned in ascending order within each WRITE command for a channel.

### 6.1.2 READ command ORB (tentative)

The READ command requests the target to store data to the referenced data buffer destined for the

initiator device with Serial Bus write transactions.

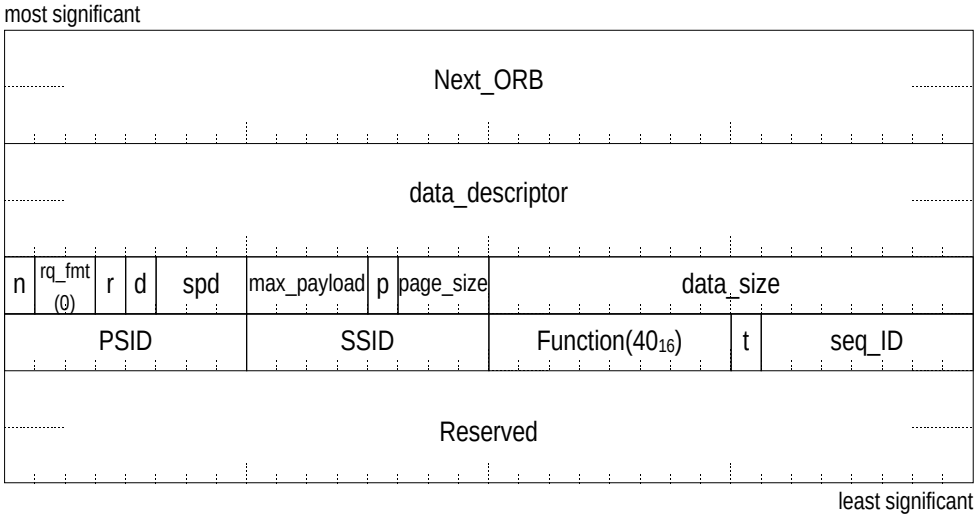


**Figure 5 - READ command ORB**

The *data_descriptor* field, *page_table_present* bit (abbreviated as *p* in the figure above), *page_size* field and *data_size* shall specify the data buffer as specified in SBP-2.

The *direction* bit (abbreviated as *d* in the figure above) specifies direction of data transfer for the buffer as specified in SBP-2 and shall be one.

The *tag* bit (abbreviated as *t* in the figure above) shall be zero and ignored by the target.

The *seq_ID* field specifies the sequence identifier for the data buffer, and shall be assigned in ascending order within each READ command for a channel.

## 6.2 Status block (tentative)

The format of the status block is illustrated by the figure below.

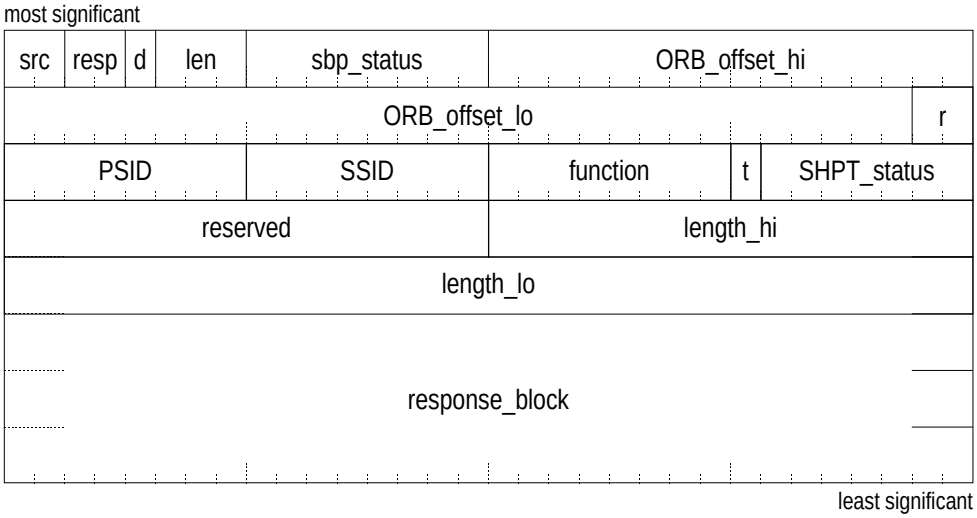


**Figure 6 - Status block**

The *PSID, SSID* field specifies the communication channel identifier.

The *function* field specifies the SHPT function, as defined by the tables below.

| Value | SHPT function |
|---|---|
| 0 | WRITE |
| 1- $3F_{16}$ | reserved for future standardization |
| $40_{16}$ | READ |
| $41_{16}$ | Data available |
| $42_{16}$- $FF_{16}$ | reserved for future standardization |

**Table 2 - SHPT function**

The *tag* bit (abbreviated as *t* in the figure above) specifies data tag.

The *SHPT_status* field specifies the result status, as defined by the tables below.

| Value | SHPT_status |
|---|---|
| 0 | Fully Completed |
| 1 | Partially Completed |
| 2 | Error |
| 3- $7F_{16}$ | reserved for future standardization |

**Table 3 - SHPT_status**

The *length_hi, length_lo* field specifies the length of processed data in byte.

All other fields are described by the SBP-2 standard.

### 6.2.1 WRITE status (tentative)

The WRITE status is a status_block for the WRITE command ORB.

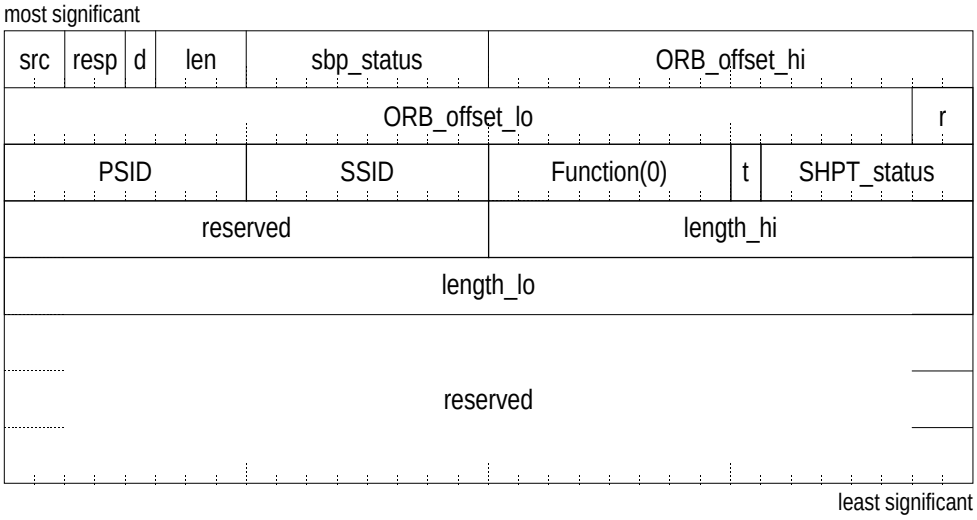


**Figure 7 - WRITE status**

The *tag* bit (abbreviated as *t* in the figure above) shall be zero and ignored by the initiator.

### 6.2.2 READ status (tentative)

The READ status_block is a status for the READ command ORB.

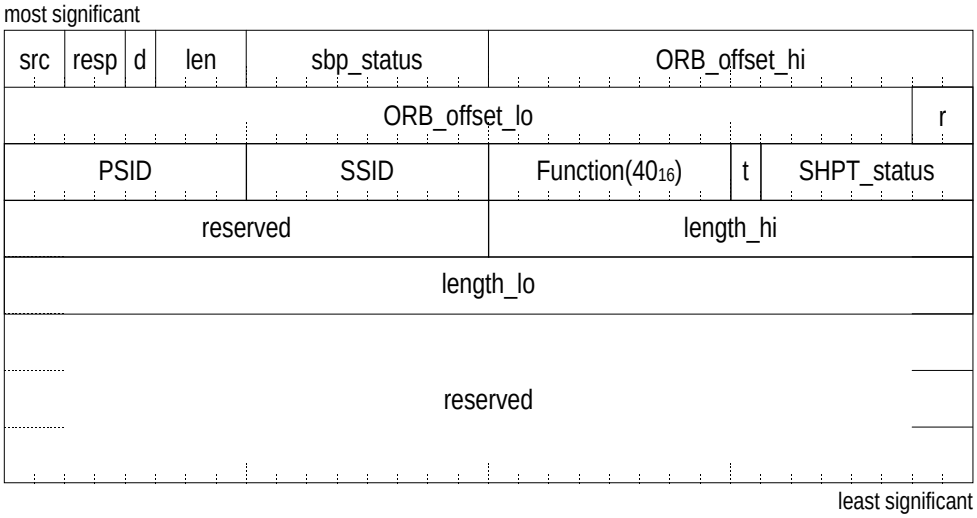


**Figure 8 - READ status**

The *tag* bit (abbreviated as *t* in the figure above) shall be one if the data transferred has been identified as out-of-band data, and shall be zero otherwise.

### 6.2.3 Data available status (tentative)

(To be described...)

## 7. Task Management Model

Targets shall support a task management model described in this document.

The task management model in this document is characterized by restricted reordering of the active tasks.

The target shall not reorder the actual execution sequence within the tasks those have same command function(WRITE or READ). The target may reorder the actual execution sequence between READ tasks and WRITE tasks. The responsibility for the assurance of data integrity is placed on the target. The target shall be responsible to avoid duplicated execution of a task that is identified as a duplicated task from the sequence identifier. The responsibility to keep up the correspondence between the sequence identifier and the buffer contents for a task is placed on the initiator.

The initiator may discard the completed ORB's from the middle of task set. The target is responsible to fetch and execute tasks as initially listed by the initiator.

The target is responsible to be able to reorder the actual execution sequence up to the number of tasks for each function reported to the initiator.

# 8. Control and Status Registers

The control and status registers (CSR's) implemented by a target shall conform to the requirements defined by the SBP-2 and its normative references.

# 9. Configuration ROM

The configuration ROM implemented by a target shall conform to the requirements defined by the SBP-2 and its normative references. A target shall implement entries defined by the SBP-2 as described in the clauses that follow.

## 9.1 Command_Set_Spec_ID entry

The Command_Set_Spec_ID entry is an immediate entry in the unit directory or the logical unit directory, that specifies the organization responsible for the command set definition for the target. Figure 9 shows the format of this entry.

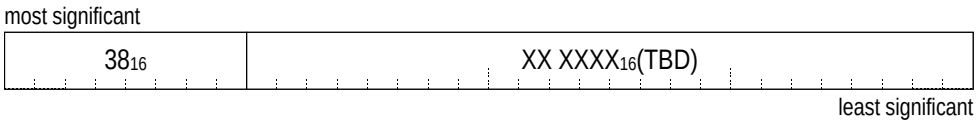


**Figure 9 - Command_Set_Spec_ID entry format**

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

XX $XXXX_{16}$(TBD) is the *command_set_spec_ID* value, an organizationally unique identifier, obtained from the IEEE/RAC by the organization responsible for the command set definition in this document.

## 9.2 Command_Set entry

The Command_Set entry is an immediate entry in the unit directory or the logical unit directory that, in combination with the *command_set_spec_ID*, specifies the command set implemented by the target. Figure 10 shows the format of this entry.

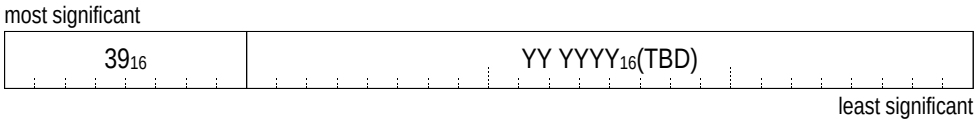


**Figure 10 - Command_Set entry format**

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

YY $YYYY_{16}$(TBD) is the *command_set* value that indicates that the target conforms to this document.

## 9.3 Command_Set_Revision entry

The Command_Set_Revision entry is an immediate entry in the unit directory or the logical unit directory that specifies the revision level of the command set implemented by the target. Figure 11 shows the format of this entry.

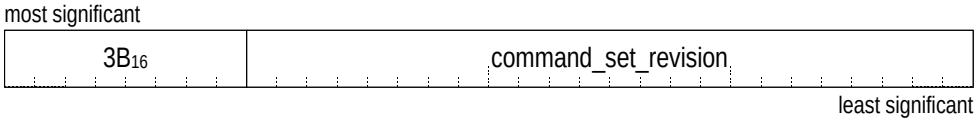


**Figure 11 - Command_Set_Revision entry format**

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

The meaning of *command_set_revision* is TBD.

## 9.4 Logical_Unit_Characteristics entry

The Logical_Unit_Characteristics entry is an immediate entry in the unit directory or the logical unit directory that specifies characteristics of the target implementation. Figure 12 shows the format of this entry.

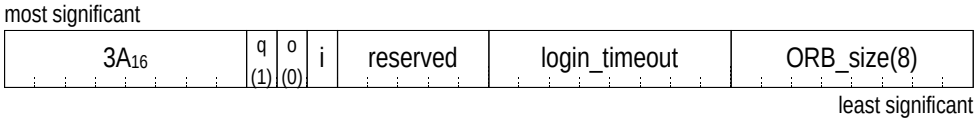


**Figure 12 - Logical_Unit_Characteristics entry format**

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

The target implements the task management (queuing) model described in this document. The target executes and reports completion status without any ordering constraints.

The *isochronous* bit (abbreviated as *i* in the figure above) specifies whether or not the target supports isochronous operations as specified by SBP-2.

The *login_timeout* field shall specify the maximum time an initiator allows for a target to store a status block in response to the initiator's login request. The setting of the *login_timeout* field is implementation specific.

The target uses the 8 quadlets (32 bytes) fetch size to obtain ORB's from initiator memory.

## 9.5 Logical-Unit-Number entry

The Logical_Unit_Number entry is an immediate entry in the unit directory or the logical unit directory that specifies the peripheral device type and number of logical unit implemented by the target. Figure 13

shows the format of this entry.

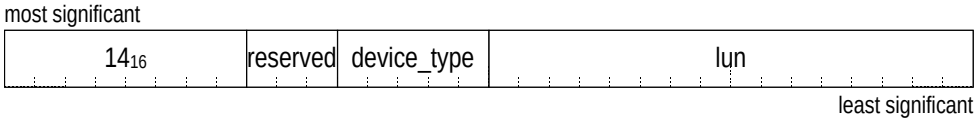


**Figure 13 - Logical_Unit_Number entry format**

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

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

| Value | Peripheral device type |
|---|---|
| 0-1 | Reserved for future standardization |
| 2 | Printer device |
| 3-$1E_{16}$ | Reserved for future standardization |
| $1F_{16}$ | Unknown device type; function discovery mean will be used to determine the peripheral device type. |

**Table 4 - Peripheral device type**

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

## 10. Function Discovery

Function Discovery implemented by a target shall conform to the requirements defined by the Function Discovery specification.

Note: the Function Discovery is under work at this point. The definitions required by the SHPT are TBD.