1394 PRINTER WORKING GROUP

MODEL & COMMAND SET

***PRELIMINARY DRAFT PROPOSAL ***

Revision 0.49 - February 21, 1999

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

1. Contents

2. Figures

3. PWG Function/Service Model

Function (Instance) = unit directory with Logical Unit Number (LUN) 0

Service = Control protocol for Independently operable component of a Function

Queue = Ordered set of ORB’s that does not block with respect to other queues. ORBS on a queue shall be executed in order.

Connection = Set of Queues that affords access to a service

Mixed Queue = A queue which may have ORBs for either direction. ORBs on a mixed queue shall be executed in order.

Action = Instructions defined in this specification that are delivered in the data buffer referenced by an ORB’s data_descriptor

The fundamental communication mechanism is a bi-directional non blocking connection. This connection is realized by ORB's on an Initiator's Task Set which are fetched and organized into two queues by the Target. Uni-directional and mixed queues are also supported.

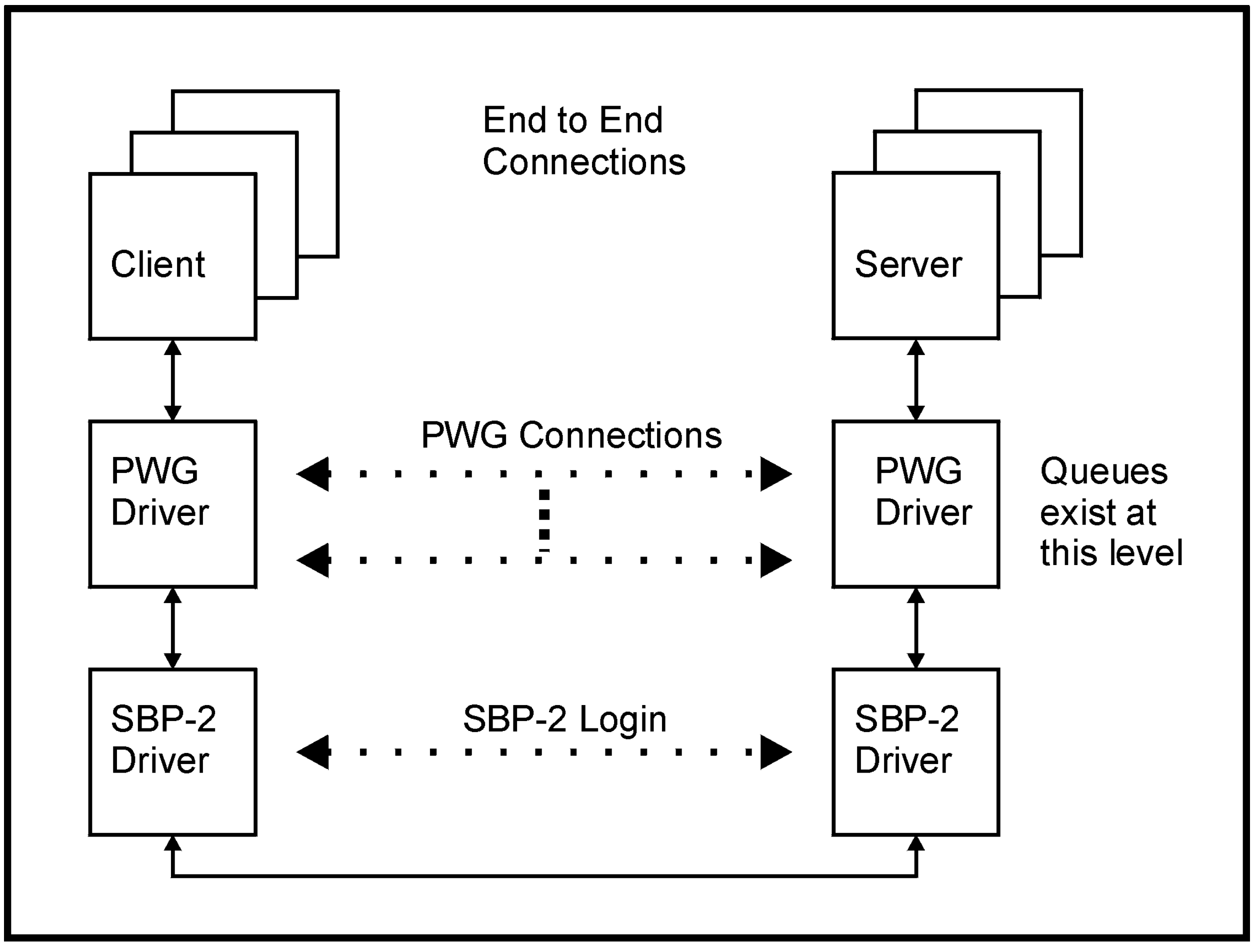
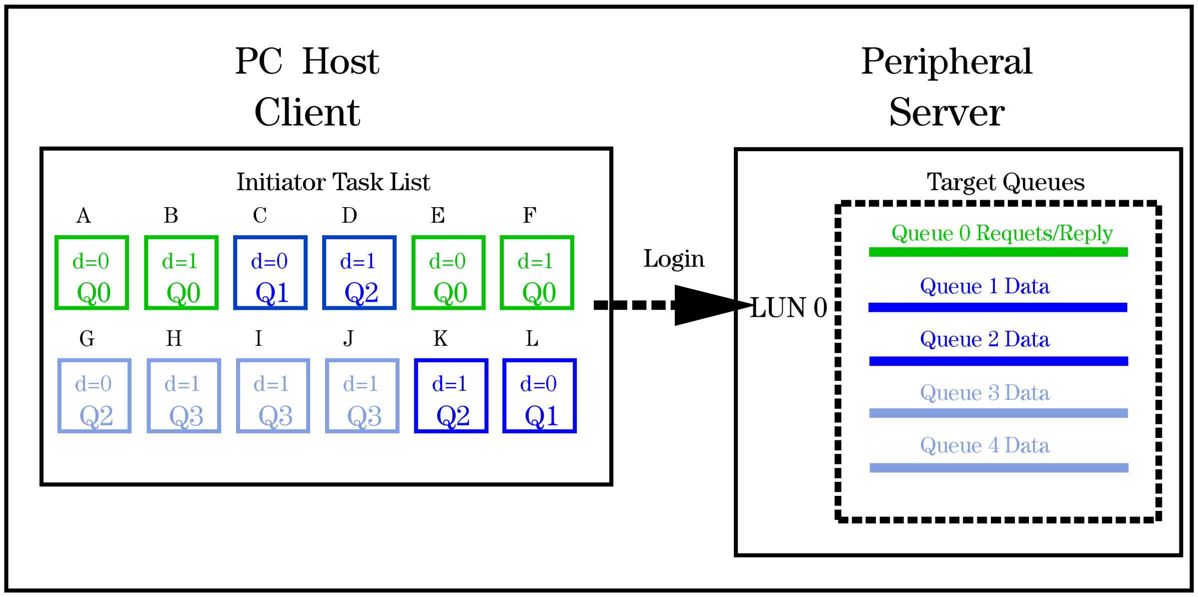


Figure 1 - High Level Model

A unit directory represents the function and has exactly one Logical Unit Number, LUN zero. To establish communication to the Function, an Initiator shall Login to LUN zero. At successful Login, Queue zero is automatically established as the Control queue. Data connections are created by using Connect Actions over queue zero.

Since the PWG Transport needs to concurrently process data transfers in both directions, the target implementing the PWG Transport must report itself to SBP-2 specifying the UNORDERED execution model. For bi-directional communication the PWG Transport target shall use two execution queues. Each queue maintains an ordered execution sequence of Tasks for a single direction of data flow.

Queue numbers are assigned by the Target and are scoped to be unique for a particular Login. A queue number is determined when a connection is established and cannot be assigned to more then one connection. The Target shall only reuse queue numbers when a connection is disconnected or aborted.



4. Data Transfer Communication Model

The basic data structure associated with data transfer is the SBP-2 ORB. An ORB allows for data_descriptor fields which provide for data buffers that allow data to flow in either direction according to a direction bit. The type of data is further qualified by PWG defined bits in the command dependent portion of the ORB.

5. Control Queue Zero

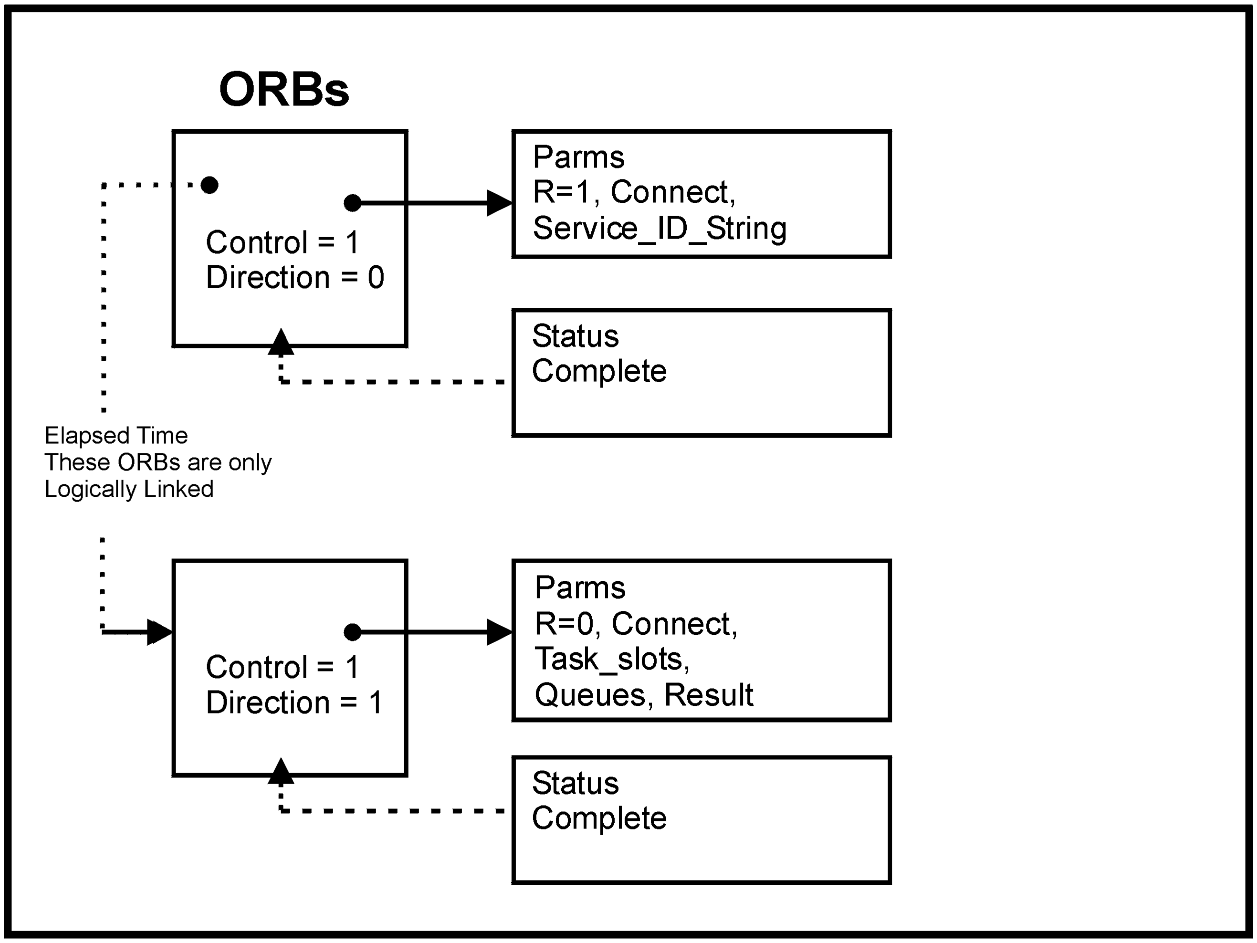
Queue zero is used to communicate control functions which manage connections. Actions are encoded as parameters and are transferred along with parameters as part of the data associated with an ORB. Actions are located in the Data portion to also allow a Target to deliver Actions. The SBP-2 status block contains the outcome of the data transfer, it does not mean that any Actions were successfully accomplished. The outcome of an Action is communicated in the Result parameter.

Queue zero is automatically allocated two Task_slots. Queue zero is defined as a mixed queue and ORBs placed on this queue should complete in a reasonable time. Queue zero can contain request and reply ORBs. The Initiator may only place one request ORB on queue zero for a particular direction.

5.1. Control Communication

When an Initiator wants to send Actions to the Target it places a Control ORB on queue zero with direction equal to Initiator to Target data transfer. The data buffer associated with the ORB contains the Request bit, Action and parameters. To receive the Reply Action, return parameters and result the Initiator Places a Control ORB with direction equal to Target to Initiator data transfer on queue zero. The Target transfers the Reply bit, Action, parameters and result of the request operation into the buffer associated with the Control ORB.

Example Commands and Actions to request a connection.



In general Target originated Actions are similar to Initiator generated Actions. When the Target wants to send Actions it makes a request for the Initiator to place an ORB containing a Control ORB on queue zero. Since the Initiator does not always know when a Target wants to send an Action the Target may use the status block of a completing ORB or the Target may use unsolicited status. The Target may also use this mechanism to request data ORBs. Depending on the what type of ORB is requested the Initiator shall place an ORB on the Control queue or on the appropriate Data queue of a connection.

How dose the Target communicate the size of the buffer it needs???

6. Data structures

There are three data structures described by this standard:

- SBP-2 Operation request blocks (ORBs);
- SBP-2 Status blocks;
- Action and Parameter encoding.

6.1. Operation Request Blocks (ORBs)

All ORB formats described in SBP-2, rev 4, shall be supported.

6.1.1. Command Block ORB Format

Command block ORBs are used to encapsulate data transfer or device control commands for transport to the target.

The general format of the command block ORB is illustrated by the following figure.

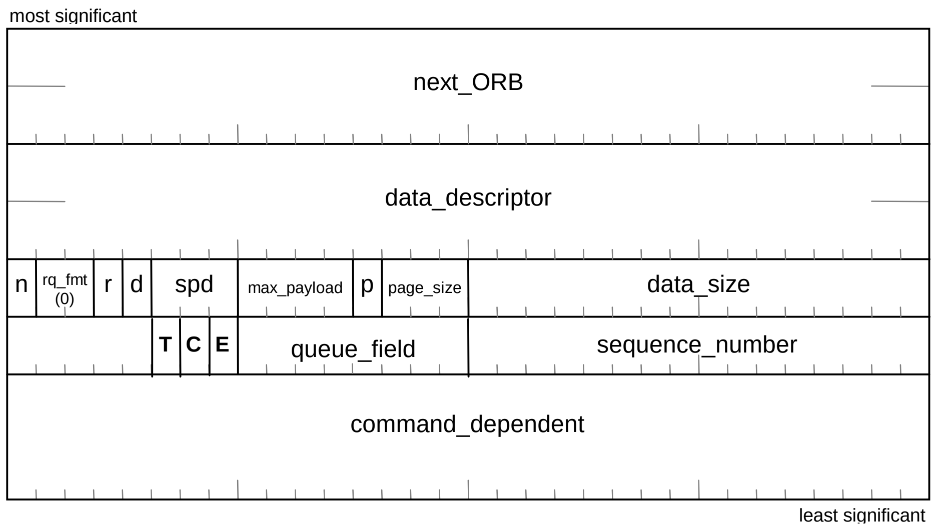


Figure 2 – Command block ORB

The *next_ORB*, *data_descriptor*, *rq_fmt*, *spd*, *max_payload*, *page_size* and *data_size* fields and the *notify*, *direction* and *page_table_present* bits (abbreviated as *n*, *d* and *p*, respectively, in the figure above) shall be as specified by SBP-2.

The *tag* bit (abbreviated as T in the figure above) shall specify that the data is special.

The Control bit (abbreviated as C in the figure above) is defined according to the following table.

Control bit (C)	Meaning
1	The Data associated with the ORB specifies a Control Function
0	The Data associated with the ORB is ordinary data

The *E* bit (abbreviated as E in the figure above) shall specify that this is the end of a message.

The *queue_field* shall identify the execution queue to which the ORB belongs.

The *sequence_number* field carries the initiator-assigned sequence number for this command instance. The sequence numbers are private to each queue.

6.1.2. Status Block

A target shall store status at the initiator *status_FIFO* address when a request completes and either the notification bit is set, an error occurred, or the completion notification contains other than zero bits beyond the first two quadlets ???. Status only pertains to the data transfer operation of the transport layer. It does not imply that any Action in the data portion completed successfully.

The *status_FIFO* address is obtained implicitly from the fetch agent context. Whenever the target has status to report, it shall store all or part of the status block shown below.

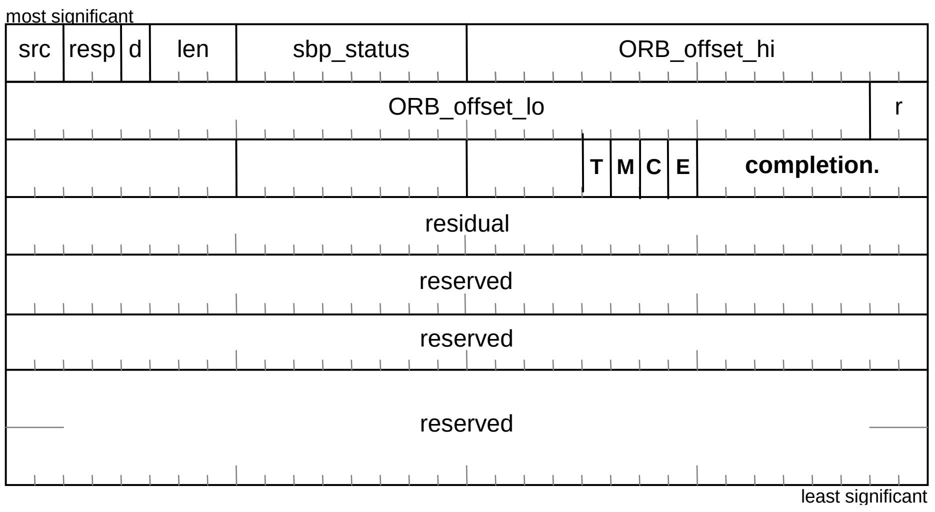


Figure 3 – Status block format

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

The *src*, *resp*, *len*, *sbp_status*, *ORB_offset_hi*, and *ORB_offset_lo* fields and the *dead* bit (abbreviated as *d* in the figure above) shall be as specified by SBP-2.

The *completion* field shall contain status information as defined by this standard. The completion field pertains to general data transfer. The receipt of any status shall indicate that the associated task has ended. The following table defines permissible values for *status*.

status value	Description
0	Good
2	Invalid queue number
3	Invalid Control ORB
4	Invalid Data ORB
$3F_{16}$	Unspecified error
All other values	reserved

Definitions for each status code are given below:

Good. This status indicates that the target has successfully completed the task.

Unspecified error. This status indicates that the target has detected an error condition not specified in this standard. (E.g. an internal error condition which prevents the target from continuing without a power cycle.)

Invalid queue number. The queue number specified in the ORB does not exist.

Invalid Control ORB. The specified queue does not support Control ORBs.

Invalid Data ORB. The specified queue does not support Data ORBs.

Invalid queue number. The queue number specified in the ORB does not exist.

Unspecified error. This status indicates that the target has detected an error condition not specified in this standard. (E.g. an internal error condition which prevents the target from continuing without a power cycle.)

The *tag* bit (abbreviated as T in the figure above) shall specify that the data is special.

The *More* bit (abbreviated as M in the figure above) shall specify that the Target request an ORB for data in the Target to Initiator direction. The type of ORB is further qualified by the C bit (defined below).

The Control bit (abbreviated as C in the figure above) is defined according to the following table.

Control bit (C)	Meaning
1	The Data associated with the ORB specifies a Control Function
0	The Action is a Reply

The *End of Message* bit (abbreviated as E in the figure above) shall specify that this is the end of a message.

The contents of the *residual* field contains the residue of the requested data transfer length minus the length of actual data to be transferred, in bytes. Negative values are indicated in two's complement notation. A non zero residual value is not necessarily an error.

6.1.3. Parameter Encoding

All parameters passed in the buffer associated with the ORB (whether to set or return values) shall be encoded using the format that follows. Parameter encodings may be packed together to form a list, which may be passed by reference, when supported by the command.

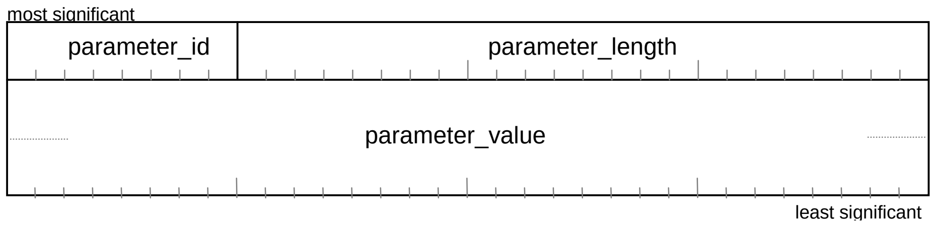


Figure 4 – Parameter ID and value format

The *parameter_length* field shall indicate the size (in bytes) of valid data in the *parameter_value* field.

The *parameter_value* field contains the actual data for the identified parameter. The data may be of an arbitrary byte length greater than zero. The data shall be padded with reserved bits to the next quadlet boundary. The *parameter_length* field shall not be adjusted to include this padding.

Data shall be stored in the least significant bits of the *parameter_value* field. For padding see section 3.2.2 SBP-2 revision 4 figure 3.

Example: the five byte number FFEEDDCCBB is encoded and padded as follows:

most significant

parameter_id	parameter_length		
00	00	00	FF
EE	DD	CC	BB

least significant

Example: the six character string “String” is encoded and padded as follows :

most significant

parameter_id	parameter_length		
“S”	t	“r”	“I”
“n”	“g”	00	00

least significant

6.2. Standard Parameters

6.2.1. Actions

Actions are used to manage connections and are accompanied by parameters. In general the Initiator and Target are required to return all of the specified parameters for a particular Action.

The exception is the queue_no with a Connect Action. When a connection is established the queue_no parameters determine the type of connection.The absence of a queue_no parameter indicates that it’s value is NULL. The Service_id_string determines whether the connection is uni-directional, bi-directional or mixed.

Queue_no, Parameter Returned	**Description**
I2T_queue_no	this specifies a single queue for communication in the Initiator to Target direction
T2I_queue_no	this specifies a single queue for communication in the Target to Initiator direction
I2T_queue_no, T2I_queue_no	this specifies two queues, one in each direction or a single mixed queue if the queue numbers are the same

ID	Parameter	Access	Size	Description
1	Action	RO	8 bits	Specifies the control function.

The parameter_value is encoded according to the following table.

Action	parameter_id	parameter_value	
		R	Action Number
RESERVED	1	*	0
CONNECT	1	*	1
DISCONNECT	1	*	2
ABORT	1	*	3

The parameter_value is further encoded using a Request/Reply bit (abbreviated as R in the diagram above) according to the following table.

Request bit (R)	Meaning
1	The Action is a Request
0	The Action is a Reply

6.2.2. Task_slots

This *Task_slots* parameter returns the maximum size of the active task set supported by this connection. The initiator must avoid queuing a greater number ORBS than the value of Task_slots on the Task List. All targets shall support at least one active task for each queue. This value will be established with a connect action.

ID	Parameter	Access	Size	Description
128	Task_slots	RO	16 bits	Returns the maximum number of pending ORBs for a connection.

6.2.3. Service_id_string

The *Service_id_string* is the name of the service requested. The Initiator or Target implicitly knows what services a device offers based on its function and other information provided through discovery.

ID	Parameter	Access	Size	Description
130	Service_id_string	RO	variable	Type service requested

6.2.4. I2T_queue_no

The *I2T_queue_no* is used in the *queue_field* of an ORB.

ID	Parameter	Access	Size	Description
131	I2T_queue_no	RO	8 bits	Queue number to be used for communication on this connection

6.2.5. T2I_queue_no

The *T2I_queue_no* is used in the *queue_field* of an ORB.

ID	Parameter	Access	Size	Description
132	T2I_queue_no	RO	8 bits	Queue number to be used for communication on this connection

6.2.6. Result

ID	Parameter	Access	Size	Description
133	Result	RO	8bits	Outcome of the Action

Result Value	**Description**
0	The Action was accomplished and the return parameters are valid
1	Connection could not be established at this time.
2	The requested service does not exist
3	Connection not permitted
4	Unknown Action
5	Parameter does not match the Action
6	Unknown Service_id_string
FF_{16}	An error occurred which is not specified in this table

6.3. Error Reporting Precedence

The precedence of error reporting, and the reported values, shall be as follows:

1. SBP-2 errors
2. ORB command dependent errors
3. Unknown or unspecified errors

7. Control Function Operations

7.1. Parameters for Initiator generated actions:

Action	d bit	Value	Parameters
CONNECT	0	0x01	Service_id_string
CONNECT	1	0x81	Task_slots, 2T_queue_no, T2I_queue_no, Result
DISCONNECT	0	0x02	I2T_queue_no, T2I_queue_no
DISCONNECT_REPLY	1	0x82	Result
ABORT_CONNECTION	0	0x03	
ABORT_CONNECTION	1	0x83	Result

7.2. Parameters for Target generated actions:

Action	**d bit**	**Value**	**Parameters**
CONNECT	1	0x01	Task_slots, I2T_queue_no, T2I_queue_no, Service_id_string
CONNECT	0	0x81	Result
DISCONNECT	1	0x02	I2T_queue_no, T2I_queue_no
DISCONNECT	0	0x82	Result
ABORT_CONNECTION	1	0x03	
ABORT_CONNECTION	0	0x83	Result

In order to setup the connection, the Initiator and Target must complete the following ORB commands and actions in sequence:

1. Action = CONNECT(Service_id_string) – Request a connection with the Target’s transport client.
2. Action = CONNECT(Task_slots, I2T_queue_no, T2I_queue_no, Result) – Receive the connection reply and the channel queues.

Once the connection has been established, the initiator may issue data ORBs for data to be transferred, provided there is room on the active task list. All transport client data shall be referenced by the ORB’s data_descriptor.

7.2.1. Initiator Connect

This action is used to establish a connection. The Initiator sends the Target a a Connect Action and a Service ID string. The Service_id_string is known to the application requesting the connection. Exactly how Service_id_strings are determined is beyond the scope of this specification.

In response to the Connect the Target returns the Task_slots, queue_no(s), and Result parameters.

7.2.2. Initiator Disconnect

The intent of the disconnect action is to perform a graceful disconnect. This Action shall be sent "synchronously" by the Initiator after all the data has been transferred and acknowledged with status blocks. The Action should be sent on the Data Queue The Initiator shall continue to provide ORBs for data until the Target is finished sending the data for any message that was in progress at the time of the disconnect.

After the Target has finished sending it's data it sends the disconnect Action. The Initiator and the Target shall wait for the reply before performing house keeping associated with the channel. After the Target sends the reply the Target is free to re-use the queue_no(s) associated with the disconnected channel.

7.2.3. Target Connect

This action is used to establish a connection from the Target to the Initiator. The Target provides the Task_slots, queue_no(s), and Service_id_string parameters

The Service_id_string is discovered or remembered? [Needs More explanation]

In response to the Connect the Initiator returns the outcome of the request in the Result

7.2.4. Target Disconnect

The intent of the Disconnect action is to perform a graceful disconnect. This Action shall be sent "synchronously" by the Target after all the data has been transferred and acknowledged with status blocks. The Action should be sent on the Data Queue. The Target shall continue to accept ORBs for data until the Initiator is finished sending the data for any message that was in progress at the time of the disconnect.

After the Initiator has finished sending it's data, it sends the Disconnect reply Action. The Initiator and the Target shall wait for the reply before performing house keeping associated with the channel. After the Initiator sends the reply the Target is free to re-use the queue_no(s) associated with the disconnected channel.