**LV33-151001**

**Interface Control Document (ICD) for Multi-Purpose Server (MPS) Qualification Software**

The information contained herein is proprietary and confidential to **Thales Avionics, Inc.** and shall not be used or disclosed, in whole or in part, without first obtaining the written permission of **Thales Avionics, Inc.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| REVISION | | | | |
| ECO | REV | DESCRIPTION | DATE | APPROVAL |
|  | 0.1 | Comments from review incorporated | 20160506 |  |
|  | 0.2 | Suggested changes/clarifications from tkLABS | 20160607 |  |
|  | 0.3 | Added HDDS Pass-Through (tkLABS) | 20160617 |  |
|  | 0.4 | Added ZMQ connection information (tkLABS) | 20160623 |  |
|  | 0.5 | Added REPORT function to SSD module (tkLABS) | 20160718 |  |

Signatures

(Not all documents require a signature page. Delete this page if it is not required.)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| Name |  | Date |  | Name |  | Date |
| Title |  |  |  | Title |  |  |
|  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| Name |  | Date |  | Name |  | Date |
| Title |  |  |  | Title |  |  |
|  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| Name |  | Date |  | Name |  | Date |
| Title |  |  |  | Title |  |  |
|  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| Name |  | Date |  | Name |  | Date |
| Title |  |  |  | Title |  |  |
|  |  |  |  |  |  |  |

[1 Scope 6](#_Toc456612220)

[1.1 Identification 6](#_Toc456612221)

[1.2 Document Overview 6](#_Toc456612222)

[2 Applicable Documents 7](#_Toc456612223)

[3 Interface Definitions 8](#_Toc456612224)

[3.1 Control and Reporting Interface 8](#_Toc456612225)

[3.1.1 CPU Loading 9](#_Toc456612226)

[3.1.2 Memory Bandwidth 11](#_Toc456612227)

[3.1.3 Analog Audio 13](#_Toc456612228)

[3.1.4 Carrier Card HD Audio 16](#_Toc456612229)

[3.1.5 Audio Video Encoder 18](#_Toc456612230)

[3.1.6 Ethernet 20](#_Toc456612231)

[3.1.7 ARINC 429 23](#_Toc456612232)

[3.1.8 ARINC 485 26](#_Toc456612233)

[3.1.9 ARINC 717 28](#_Toc456612234)

[3.1.10 GPIO 30](#_Toc456612235)

[3.1.11 RS-232 34](#_Toc456612236)

[3.1.12 RS-485 36](#_Toc456612237)

[3.1.13 SSD Application 38](#_Toc456612238)

[3.1.14 HDDS Pass-Through 40](#_Toc456612239)

[3.1.15 System Monitoring 42](#_Toc456612240)

# Scope

The purpose of this document is to define the external interfaces for the Multi-Purpose Server (MPS) Qualification Software.

## Identification

The MPS Qualification Software is the MPS resident component of an automated test suite designed to exercise the external hardware interfaces and simulate anticipated thermal loading of the LRU. This is to support system evaluation during environmental and EMI testing scenarios including HALT and HASS.

Although outside the scope of this document, consideration has been given to additional software being developed in support of the MPS program. Flexibility of design and leveraging existing software may outweigh simplicity to meet these long term objectives.

## Document Overview

This document consists of sections describing each type of interface, the external connections required, and the output of the MPS Qualification Software during test. The sections are listed below:

* CPU Loading
* Memory Bandwidth
* Analog Audio
* HD Audio
* Audio Video Encoder
* Ethernet
* ARINC 429
* ARINC 485
* ARINC 717
* GPIO
* RS-232
* RS-485
* SSD Application
* HDDS Pass-Through
* System Monitoring

# Applicable Documents

<http://wiki.libvirt.org/page/VirtualNetworking>

|  |  |  |
| --- | --- | --- |
| **LV33-160414** | Rev. A | **MPS Smart Embedded Management Agent (SEMA) Driver Interface Control Document** |

|  |  |  |
| --- | --- | --- |
| **LV33-160504** | Rev. | **MAP Host Domain Device Service Interface Control Document** |

|  |  |  |
| --- | --- | --- |
| **LV33-160408** | Rev. A | **MPS Power Supply Monitor Interface Control Document** |

|  |  |  |
| --- | --- | --- |
| **LV33-160419** | Rev. A | **MPS Power Supply Monitor Plugin Interface Control Document** |

# Interface Definitions

## Control and Reporting Interface

The MPS Qualification (MPS) software communicates with the test equipment (TE) via an Ethernet interface. Transactions between the TE and MPS are based on the client-server model implemented using the ZeroMQ (ZMQ) embeddable networking library. The ZMQ Request-Reply socket pair is implemented with the TE providing the Client (REQ) socket and the MPS providing the Server (REP) socket.

The messages are defined in this document using the Google Protocol Buffer (GPB) notation; however the messages are implemented in both GPB and JSON schemas. The following sections describe the contents of the messages relevant to each application function.

The MPS Qualification Software test application exposes two TCP ports for ZMQ messaging:

* Port 50001 accepts messages and returns replies encoded in GPB format, as defined in the “Thales Common Network Messaging ICD” document. These are ZMQ multipart messages with 3 frames:
  + Frame 0 contains the message name, as a string
  + Frame 1 contains a message header in GPB binary format (currently ignored by the MPS Qualification software)
  + Frame 2 contains the message body in GPB binary format
* Port 50002 accepts messages and returns replies encoded in JSON format. These are ZMQ multipart messages with 2 frames:
  + Frame 0 contains the message name, as a string
  + Frame 1 contains the JSON-encoded message, as a string

Note: The ZMQ service addresses for both the GPB and JSON listeners can be configured in the MPS Qualification Software configuration file, located at *qual/config/platform.ini* in the MPS Qualification Software installation directory. The parameter names and their default values are shown below.

[QualTestApp]

gpbServiceAddress = tcp://\*:50001

jsonServiceAddress = tcp://\*:50002

### CPU Loading

The CPU Loading function is not a test in itself but rather a mechanism to achieve a configurable percentage of CPU utilization. The goal is to establish a nominal load on the LRU while testing other aspects of the product.

#### CPU Loading Request

The CPU Loading Request Message is sent by the TE to run or stop the CPU loading function and request a report of the current state of the application and latest processor utilization statistics. If a report request is sent the reply is a report without changing the application state. The optional *level* message element sets the desired loading level in percentage of total CPU utilization (default 80%).

message CPULoadingRequest {

enum RequestTypeT {

STOP = 0;  
 RUN = 1;  
 REPORT = 2; }

required RequestTypeT requestType = 1 [default = STOP];

optional float level = 2;  
}

|  |  |
| --- | --- |
| Field | Description |
| STOP | Requests the application to halt and respond with a report |
| RUN | Requests the application to start and respond with a report |
| REPORT | Requests a report without any action |
| level | Desired loading level in percentage of total CPU utilization (default 80%); level is specified as percentage, e.g. 80% is 80.0 |

Table 1 CPULoadingRequest Message

#### CPU Loading Response

The CPU Loading Response Message is sent by the MPS to provide the current processor utilization statistics and the state of the application (RUN or STOP).

message CPULoadingResponse{

enum AppStateT {

STOPPED = 0;  
 RUNNING = 1;  
 }

required AppStateT state = 1;

required float totalUtilization = 2;

repeated float coreUtilization = 3;  
  
}

|  |  |
| --- | --- |
| Field | Description |
| STOPPED | The application is not running |
| RUNNING | The application is running |
| totalUtilization | Percentage of total CPU utilization; specified as percentage, e.g. 80% is 80.0 |
| coreUtilization | Percentage of CPU Core utilization, one value per core; specified as percentage, e.g. 80% is 80.0 |

Table 2 CPULoadingResponse Message

### Memory Bandwidth

The Memory Bandwidth function is not a test in itself but rather a mechanism for measuring the memory bandwidth under the test conditions.

#### Memory Bandwidth Request

The Memory Bandwidth Request Message is sent by the TE to run or stop the application and request the memory bandwidth statistics. If a report request is sent the reply is a report without changing the application state.

message MemoryBandwidthRequest {

enum RequestTypeT {

STOP = 0;  
 RUN = 1;  
 REPORT = 2;  
 }

required RequestTypeT requestType = 1 [default = STOP];  
}

|  |  |
| --- | --- |
| Field | Description |
| STOP | Requests the application to halt and reply with a report |
| RUN | Requests the application to run and reply with a report |
| REPORT | Requests a report without any action |

Table 3 MemoryBandwidthRequest Message

#### Memory Bandwidth Response

The Memory Bandwidth Response Message is sent by the MPS to report the current memory bandwidth statistics.

message MemoryBandwidthResponse {

enum AppStateT {

STOPPED = 0;  
 RUNNING = 1;  
 }

required AppStateT state = 1;

required float memoryBandWidth = 2;

}

|  |  |
| --- | --- |
| Field | Description |
| STOPPED | The application is not running |
| RUNNING | The application is running |
| memoryBandWidth | Memory Band Width measured in Gigabytes per Second |

Table 4 MemoryBandwidthResponse Message

### Analog Audio

The Analog Audio Application consists of loopback connections internal to the MPS such that the analog audio signal received on the designated input (signal source) is transmitted on the designated output (signal sink). The loopback path consists of digitizing the analog signal through the IFE card ADC and reconverting the digital stream through the IFE card DAC. If more than one output is connected to a given input the input stream will be looped back to each output.

#### Analog Audio Request

The Analog Audio Request Message is sent by the TE to establish or dissolve a loopback connection according to the test procedure. If the connection request matches the current state, the response is sent with no change of application state. If the source string is valid and the sink is the “ALL” string, all connections to the source are dissolved for a *DISCONNECT* request or all outputs are connected to the source for a *CONNECT* request.

message AnalogAudioRequest {

enum RequestTypeT {  
 DISCONNECT = 0;  
 CONNECT = 1;  
 REPORT = 2;  
 }

required RequestTypeT requestType = 1 [default = DISCONNECT];

required string sink = 2 [default = “ALL”];

optional string source = 3 [default = “”];

}

|  |  |  |
| --- | --- | --- |
| Name | | Description |
| CONNECT | The CONNECT request tells the MPS to loop back the source signal to the sink output | |
| DISCONNECT | The DISCONNECT request tells the MPS to cease output to the sink | |
| REPORT | The REPORT request tells the MPS to respond with a configuration report | |
| sink | Output sink of the signal connection referenced by the device names below | |
| source | Input source of the signal connection (ignored if present for DISCONNECT or REPORT requests) referenced by the device names below | |

Table 5 AnalogAudioRequest Message

#### Analog Audio Response

The Analog Audio Response Message is sent by the MPS to report the state of the connection after a *CONNECT* or *DISCONNECT* request is processed. In addition if the request was a *REPORT* type and the sink parameter is a single output the connection status for that sink is sent. If the request was a *REPORT* type and the sink parameter is “ALL”, the connection status for each sink is sent.

message AnalogAudioResponse {

enum ConnectionStateT {  
 DISCONNECTED = 0;  
 CONNECTED = 1;  
 }

Message OutputStatus {

required string sink = 1;

required string source = 2 [default = “”];

required ConnectionStateT state = 3 [default = DISCONNECTED];

}

repeated OutputStatus loopback = 1;

}

|  |  |
| --- | --- |
| Field | Description |
| loopback | The status on a per sink basis identifying the sink and source, if any, along with the connection state for each Analog Input devices named below. |
| sink | Output sink of the signal connection referenced by the device names below |
| source | Input source of the signal connection referenced by the device names below |
| state | This is the connection state of the sink (*CONNECTED* or *DISCONNECTED*) |
| DISCONNECTED | When the reply is generated the connection does not exist |
| CONNECTED | When the reply is generated the connection exists |

Table 6 AnalogAudioResponse Message

#### Analog Audio Device Names

The following logical names for analog audio outputs are used to specify the signal sink for a loop-back connection.

|  |  |  |
| --- | --- | --- |
| Logical Name | Positive Pin | Negative Pin |
| VA\_AUDOUT\_1 | Insert B : E-5 | Insert B : F-6 |
| VA\_ AUDOUT\_2 | Insert B : D-5 | Insert B : D-6 |
| VA\_ AUDOUT\_3 | Insert B : C-6 | Insert B : C-7 |
| VA\_ AUDOUT\_4 | Insert B : B-6 | Insert B : B-7 |
| VA\_ AUDOUT\_5 | Insert B : A-2 | Insert B : A-1 |
| VA\_ AUDOUT\_6 | Insert B : E-7 | Insert B : D-8 |

Table 7 Analog Ouput Names

The following logical names for analog audio inputs are used to specify the signal source for a loop-back connection.

|  |  |  |
| --- | --- | --- |
| Logical Name | Positive Pin | Negative Pin |
| PA\_70V\_AUDIN\_1 | Insert B : E-1/E-2 | Insert B : F-1 |
| PA\_AUDIN\_2 | Insert B : D-2 | Insert B : D-1 |
| PA\_AUDIN\_3 | Insert B : C-2 | Insert B : C-1 |
| PA\_AUDIN\_4 | Insert B : B-2 | Insert B : B-1 |
| PA\_AUDIN\_5 | Insert B : A-2 | Insert B : A-1 |
| PA\_AUDIN\_6 | Insert B : E-4 | Insert B : E-3 |
| PA\_AUDIN\_7 | Insert B : D-4 | Insert B : D-3 |
| PA\_AUDIN\_8 | Insert B : C-4 | Insert B : C-3 |

Table 8 Analog Input Names

### Carrier Card HD Audio

The HD Audio Application consists of the MPS outputting analog audio signals by converting the specified digital source through the Carrier Card HD Audio Codec.

#### HD Audio Request

The HD Audio Request Message is sent by the TE to start or stop the HD Audio Application. The optional source string consists of a path/filename to which the HD Audio Application has access. If the *CONNECT* request does not include the optional source string, a default file will be used. If the *DISCONNECT* request includes the optional source string, the string is ignored. If the optional volume parameter is included the output volume will be set accordingly.

message HDAudioRequest {

enum RequestTypeT {  
 DISCONNECT = 0;  
 CONNECT = 1;  
 REPORT = 2;  
 }

required RequestTypeT requestType = 1 [default = DISCONNECT];

optional string source = 2 [default = “”];

optional float volume = 3;

}

|  |  |  |
| --- | --- | --- |
| Name | | Description |
| CONNECT | The *CONNECT* request tells the application to stream the digital audio source to the HD Audio output | |
| DISCONNECT | The *DISCONNECT* request tells the application not to stream to the HD Audio output | |
| REPORT | The *REPORT* request tells the MPS to respond with a configuration report | |
| source | Input source path/filename of the signal connection (e.g. *HDAudio/test\_pattern.mp3)* | |
| volume | Volume setting to apply the output in percentage of full volume (0% to 100%) | |

Table 9 HDAudioRequest Message

#### HD Audio Response

The HD Audio Response Message is sent by the MPS to report the state of the connection after the request is processed.

message HDAudioResponse {

enum AppStateT {  
 DISCONNECTED = 0;  
 CONNECTED = 1;  
 }

required AppStateT appState = 1 [default = DISCONNECTED];

required string source = 2 [default = “”];

required float volume = 3;

}

|  |  |
| --- | --- |
| Field | Description |
| DISCONNECTED | When the reply is generated the connection does not exist |
| CONNECTED | When the reply is generated the connection exists |
| source | Input source of the signal connection (“*default”* or as specified in a request) |
| volume | Volume setting applied to the output in percentage of full volume (0% to 100%) |

Table 10 HDAudioResponse Message

### Audio Video Encoder

#### Encoder Request

The Encoder Request Message is sent by the TE to initiate or halt audio/video encoding and set the destination address (multi- or unicast) according to the test procedure. If the sink option has not been sent or is invalid, the encoded data is discarded rather than transmitted or stored.

message EncoderRequest {

enum RequestTypeT {  
 STOP = 0;  
 RUN = 1;  
 REPORT = 2; }

required RequestTypeT requestType = 1 [default = STOP];

optional string sink = 2;

}

|  |  |
| --- | --- |
| Name | Description |
| STOP | Requests the application to halt and reply with a report |
| RUN | Requests the application to run and reply with a report |
| REPORT | Requests a report without any action |
| sink | Output sink for the encoded A/V; an IP address and port (ignored if present for *DISCONNECT* or *REPORT* requests) |

Table 12 EncoderRequest Message

#### Encoder Response

The Encoder Response Message is sent by the MPS to acknowledge an encoder request or report the encoder status.

message EncoderResponse {  
 enum AppStateT {

STOPPED = 0;  
 RUNNING = 1;  
 }

required AppStateT state = 1;

required bool inputActive = 2 [default = FALSE];

required bool streamActive = 3 [default = FALSE];  
}

|  |  |
| --- | --- |
| Field | Description |
| STOPPED | The application is not running |
| RUNNING | The application is running |
| inputActive | *TRUE* if the test is running and an input signal is detected, otherwise *FALSE* |
| streamActive | *TRUE* if the test is running and the output stream is active, otherwise *FALSE* |

Table 13 EncoderResponse Message

### Ethernet

#### Ethernet Request

The Ethernet Request Message is sent by the TE to set enable or disable performance statistic gathering on an Ethernet channel and obtain a statistics report. When the statistic gathering is active the MPS connects the local Ethernet channel to the remote iPerf server and collects performance statistics. If the optional remote server string is present, the remote server address is updated. If the request type is RUN and the remote address has not been set, the response is sent without starting the measurement.

message EthernetRequest {

enum RequestTypeT {  
 STOP = 0;  
 RUN = 1;  
 REPORT = 2; }

required RequestTypeT requestType = 1 [default = OFF];

required string local = 2 [default = “”];

optional string remote = 3 [default = “”];

}

|  |  |
| --- | --- |
| Field | Description |
| STOP | Requests the application to halt and reply with a report; the report will include the bandwidth and retries values just prior to stopping |
| RUN | Requests the application to run and reply with a report |
| REPORT | Requests a report without any action |
| local | Channel to which the request applies |
| remote | Remote address to which the request applies (ignored if present for *STOP* or *REPORT* requests) |

Table 14 EthernetRequest Message

#### Ethernet Response

The Ethernet Response Message is sent by the MPS acknowledge an Ethernet Request message and provide statistics related to the iPerf measurement.

message EthernetResponse {

enum AppStateT {

STOPPED = 0;  
 RUNNING = 1;  
 }

required AppStateT state = 1;

required string local = 2;

required float bandwidth = 3;

required uint32 retries = 4;

}

|  |  |
| --- | --- |
| Field | Description |
| STOPPED | The application is not running |
| RUNNING | The application is running |
| local | Channel under test |
| bandwidth | Current instantaneous bandwidth as reported by iPerf, in megabits/second |
| retries | Number of TCP retries since the last transition from *STOPPED* to *RUNNING* |

Table 15 EthernetResponse Message

#### Ethernet Device Names

The following logical names are used to specify the MPS Ethernet channels in request or response messages.

| Logical Name | Positive A Pin | Positive B Pin | Positive C Pin | Positive D Pin | Shield Pin | Drain Pin | Neg. A Pin | Neg.  B Pin | Neg. C Pin | Neg.  D Pin |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ENET\_1 | A: F-1 | A: G-1 | A: J-1 | A : K-1 | A : H-1 | A : H-2 | A: G-2 | A: F-2 | A: K-2 | A: J-2 |
| ENET\_2 | A: F-13 | A: G-13 | A: J-13 | A : K-13 | A : H-13 | A : H-14 | A: G-14 | A: F-14 | A: K-14 | A: J-14 |
| ENET\_3 | A: F-4 | A: G-4 | A: J-4 | A : K-4 | A : H-4 | A : H-5 | A: G-5 | A: F-5 | A: K-5 | A: J-5 |
| ENET\_2 | A: F-10 | A: G-10 | A: J-10 | A : K-10 | A : H-10 | A : H-11 | A: G-11 | A: F-11 | A: K-11 | A: J-11 |
| ENET\_5 | A: A-1 | A: B-1 | A: D-1 | A : E-1 | A : C-1 | A : C-2 | A: B-2 | A: A-2 | A: E-2 | A: D-2 |
| ENET\_6 | A: A-4 | A: B-4 | A: D-4 | A : E-4 | A : C-4 | A : C-5 | A: B-5 | A: A-5 | A: E-5 | A: D-5 |
| ENET\_7 | A: F-7 | A: G-7 | A: J-7 | A : K-7 | A : H-7 | A : H-8 | A: G-8 | A: F-8 | A: K-8 | A: J-8 |
| ENET\_8 | A: A-7 | A: B-7 | A: D-7 | A : E-7 | A : C-7 | A : C-8 | A: B-8 | A: A-8 | A: E-8 | A: D-8 |
| ENET\_9 | A: A-10 | A: B-7 | XXXX | XXXX | A : C-10 | XXXX | A: B-11 | A: A-11 | XXXX | XXXX |
| ENET\_10 | A: D-10 | A: E-10 | XXXX | XXXX | A : C-11 | XXXX | A: E-11 | A: D-11 | XXXX | XXXX |
| ENET\_11 | B: G-2 | B: H-2 | B: J-1 | B : K-4 | B : G-1 | B: J-3 | B: H-3 | B: G-3 | B: K-2 | B: J-2 |
| ENET\_12 | B: G-5 | B: H-5 | B: J-4 | B : K-7 | B : G-4 | B: J-6 | B: H-6 | B: G-6 | B: K-5 | B: J-5 |
| ENET\_13 | B: G-8 | B: H-8 | B: J-7 | B : K-10 | B : G-7 | B: J-9 | B: H-9 | B: G-9 | B: K-8 | B: J-8 |
| ENET\_14 | B: G-11 | B: H-11 | B: J-10 | B : K-13 | B : G-10 | B: J-12 | B: H-12 | B: G-12 | B: K-11 | B: J-11 |
| ENET\_15 | B: G-14 | B: H-14 | B: J-13 | B : K-16 | B : G-13 | B: J-15 | B: H-15 | B: G-15 | B: K-14 | B: J-14 |

Table 16 Ethernet Channel Names

### ARINC 429

The purpose of the ARINC 429 Application is to simulate loading of the ARINC 429 bus components and detect failures thereof. The test strategy is to generate output on the transmitter channels (connection source) which will be externally looped back to the input channels (connection sink) in order to load and verify both the transmitter and receiver circuits. The TE will provide the MPS with the loopback mapping according to the test procedure.

Note that the MPS will allow a source to have multiple connections so that a source can be looped back to more than one sink. A sink, however, can have only one connection and therefore only one source.

#### ARINC 429 Request

The ARINC 429 Request Message is sent by the TE to establish or remove a loopback connection, or request a report for the sink. For each connection, the MPS will transmit a test pattern on the source channel and compare the data received on the connected sink channel. The TE may also request a report, in which case the MPS will respond with the status of the specified sink.

message ARINC429Request {

enum RequestTypeT {  
 DISCONNECT = 0;  
 CONNECT = 1;  
 REPORT = 2;  
 }

required RequestTypeT requestType = 1 [default = DISCONNECT];

required string sink = 2 [default = “”];

optional string source = 3 [default = “”];

}

|  |  |  |
| --- | --- | --- |
| Name | Description | |
| CONNECT | | The *CONNECT* request tells the MPS to loop back the source signal to the sink. If sink is “ALL” then all sinks are connected to the specified source |
| DISCONNECT | | The *DISCONNECT* request tells the MPS to cease output to the sink; the response message will include the counter values just prior to the disconnect. If sink is “ALL” then all sinks are disconnected from their source (if any) |
| REPORT | | The *REPORT* request tells the MPS to respond with a configuration report for the specified sink, or for all sinks if sink is “ALL” |
| requestType | | Type of request (*CONNECT* , *DISCONNECT*, or *REPORT*) |
| sink | | ARINC 429 input receiving the external loopback referenced by the device names below, or the special value “ALL” as described above |
| source | | ARINC 429 Output providing the external loopback (ignored if present for *DISCONNECT* or *REPORT* requests) referenced by the device names below |

Table 17 ARINC429Request Message

#### ARINC 429 Response

The ARINC 429 Response Message is sent by the MPS in response to any type of request.

message ARINC429Response {  
 enum ConStateT {  
 DISCONNECTED = 0;  
 CONNECTED = 1;  
 }

message StatusMessage {

required string sink = 1;

required string source = 2 [default = “”];

required ConStateT conState = 3;

required uint32 xmtCount = 4;

required uint32 rcvCount = 5;

required uint32 errorCount = 6;

}

repeated StatusMessage status = 1;  
}

|  |  |
| --- | --- |
| Field | Description |
| DISCONNECTED | When the reply is generated the connection does not exist |
| CONNECTED | When the reply is generated the connection exists |
| sink | Output sink of the signal connection referenced by the device names below |
| source | Input source of the signal connection (“” for disconnected sinks) referenced by the device names below |
| xmtCount | The number of messages sent since the connection was established. This value is reset when a connection is removed and restored. |
| rcvCount | The number of messages received since the connection was established. This value is reset when a connection is removed and restored. |
| errorCount | The number of mismatched messages detected since the connection was established. This value is reset when a connection is removed and restored. |

Table 18 ARINC429Response Message

#### ARINC 429 Device Names

The following logical names for ARINC 429 input channels are used to specify the channels for a loop-back connection.

|  |  |  |  |
| --- | --- | --- | --- |
| Logical Name | Positive Pin | Negative Pin | Shield Pin |
| ARINC\_429\_RX1 | Insert C: Pin 7 | Insert C: Pin 8 | Insert C: Pin 13 |
| ARINC\_429\_RX2 | Insert C: Pin 9 | Insert C: Pin 15 | Insert C: Pin 14 |
| ARINC\_429\_RX3 | Insert C: Pin 17 | Insert C: Pin 12 | Insert C: Pin 10 |
| ARINC\_429\_RX4 | Insert C: Pin 22 | Insert C: Pin 23 | Insert C: Pin 24 |
| ARINC\_429\_RX5 | Insert A: H-9 | Insert A: J-9 | Insert A: K-9 |
| ARINC\_429\_RX6 | Insert A: H-3 | Insert A: J-3 | Insert A: K-3 |
| ARINC\_429\_RX7 | Insert A: E-3 | Insert A: F-3 | Insert A: G-3 |

Table 19 ARINC Input Names

The following logical names for ARINC 429 output channels are used to specify the channels for a loop-back connection.

|  |  |  |  |
| --- | --- | --- | --- |
| Logical Name | Positive Pin | Negative Pin | Shield Pin |
| ARINC\_429\_TX1 | Insert C: Pin 16 | Insert C: Pin 11 | Insert C: Pin 18 |
| ARINC\_429\_TX2 | Insert C: Pin 19 | Insert C: Pin 20 | Insert C: Pin 21 |
| ARINC\_429\_TX3 | Insert A: J-15 | Insert A: K-15 | Insert A: L-15 |
| ARINC\_429\_TX4 | Insert A: J-12 | Insert A: K-12 | Insert A: L-12 |

`  
Table 20 ARINC Output Names

### ARINC 485

The purpose of the ARINC 485 Application is to simulate loading of the ARINC 485 bus components and detect failures thereof. The test strategy is to periodically initiate a Status\_Request/ Idle transaction between the Master channel and each Slave channel in turn to load and verify both the transmitter and receiver circuits of each channel.

#### ARINC 485 Request

The ARINC 485 Request Message is sent by the TE to start or stop the loopback application and to request the current transaction send and receive counts. When the MPS will periodically transmit a Status\_Request message addressed to each Slave node and the Slave node will respond with an Idle message.

message ARINC485Request {

enum RequestTypeT {  
 STOP = 0;  
 RUN = 1;  
 REPORT = 2; }

required RequestTypeT requestType = 1;

}

|  |  |  |
| --- | --- | --- |
| Name | | Description |
| STOP | Requests the application to halt and reply with a report |
| RUN | Requests the application to run and reply with a report |
| REPORT | The REPORT request tells the MPS to respond with a configuration report |

Table 21 ARINC485Request Message

#### ARINC 485 Response

The ARINC 485 Response Message is sent by the MPS to acknowledge a connection request and report the connection statistics. The response message contains one Loopback Stats message for each slave port.

message ARINC485Response {

enum AppStateT {

STOPPED = 0;  
 RUNNING = 1;  
 }

message LoopbackStats {

required string channel = 1;

required uint32 missed = 2;

required uint32 received = 3;

}

required AppStateT state = 1;

repeated LoopbackStats statistics = 2;

}

|  |  |
| --- | --- |
| Field | Description |
| STOPPED | The application is not running |
| RUNNING | The application is running |
| channel | The logical name of the channel for which this is the statistics message referenced by the device names below |
| missed | The number of missed Idle response messages for the channel |
| received | The number of received Idle response messages for the channel |
| statistics | One statistics message per slave channel (5 total) |

Table 22 ARINC485Response Message

### ARINC 717

The purpose of the ARINC 717 Application is to simulate loading of the ARINC 717 bus components and detect failures thereof. The test strategy is to externally generate data which is transmitted to the MPS ARINC 717 RX channel in order to load and verify the receiver circuit. The received data is output on the reporting interface for error checking.

#### ARINC 717 Frame Request

The ARINC 717 Frame Request Message is sent by the TE to control the application state and to request the most recent ARINC 717 frame data and status.

message ARINC717FrameRequest {

enum RequestTypeT {  
 STOP = 0;  
 RUN = 1;  
 REPORT = 2;  
 }

required RequestTypeT requestType = 1 [default = STOP];

}

|  |  |
| --- | --- |
| Name | Description |
| requestType | Type of request (STOP, RUN. or REPORT) |
| STOP | Requests the application to halt and reply with a report |
| RUN | Requests the application to run and reply with a report |
| REPORT | The REPORT request tells the MPS to respond with an ARINC 717 Frame Response Message |

Table 23 ARINC717FrameRequest Message

#### ARINC 717 Frame Response

The ARINC 717 Frame Response Message is sent by the MPS to report the application state, the frame sync state and most the recently received complete frame.

message ARINC717FrameResponse {

enum AppStateT {

STOPPED = 0;  
 RUNNING = 1;  
 }

enum FrameSync {  
 NO\_SYNC = 0;  
 SYNCED = 1;  
 }

required AppStateT state = 1;

required FrameSync syncState = 2 [default = NO\_SYNC];

repeated int32 arinc717frame = 3;

}

|  |  |
| --- | --- |
| Field | Description |
| STOPPED | The application is not running |
| RUNNING | The application is running |
| NO\_SYNC | The current state of the ARINC 717 interface is unsynchronized |
| SYNCED | The current state of the ARINC 717 interface is synchronized |
| arinc717frame | The 12-bit ARINC 717 words padded into 32 bit words with 20 MSBs set to 0 |

### GPIO

#### GPIO Request

The GPIO Request Message is sent by the TE to establish or remove a loopback connection, or request statistics regarding the connection. When an output pin is connected the MPS will assert and de-assert the gpOut pin with the expectation of sensing the same state on the gpIn pin(s).

When the request type is *CONNECT*, the gpIn parameter must be an unconnected GP input and the gpOut parameter must be a GP output. Any other combination will result in a response without establishing a connection (i.e. one output to multiple inputs is allowed but more than one output to one input is not.)

When the request type is *DISCONNECT*, the connection between the gpOut and gpIn is removed.

When the request type is *REPORT*, the connection for gpIn is unchanged and the response message is sent.

For all request types, i,

message GPIORequest {

enum RequestTypeT {

DISCONNECT = 0;  
 CONNECT = 1;  
 REPORT = 2;  
 }

required RequestTypeT requestType = 1 [default = DISCONNECT];

required string gpIn = 2 [default = “”];

optional string gpOut = 3 [default = “”];

}

|  |  |
| --- | --- |
| Name | Description |
| DISCONNECT | Requests the application to disconnect and reply with a report; report will include the counter values just prior to stopping |
| CONNECT | Requests the application to connect and reply with a report |
| REPORT | The REPORT request tells the MPS to respond with a report |
| gpIn | MPS discrete input |
| gpOut | MPS discrete output (ignored for *DISCONNECT* or *REPORT*) |

Table 24 GPIORequest Message

#### GPIO Response

The GPIO Response Message is sent by the MPS to provide the connection state and loopback statistics for an individual GP input.

message GPIOResponse {  
 enum ConStateT {  
 DISCONNECTED = 0;  
 CONNECTED = 1;  
 }

Message GpioStatus {

required ConStateT conState = 1;

required integer matchCount = 2;

required integer mismatchCount = 3;

required string gpIn = 4;

optional string gpOut = 5;

}

repeated GpioStatus status = 1;

}

|  |  |
| --- | --- |
| } Field | Description |
| DISCONNECTED | When the reply is generated the connection does not exist |
| CONNECTED | When the reply is generated the connection exists |
| matchCount | The number of matches for the specified connection since the last connection request for the pin pair, or zero if the connection does not exist. |
| mismatchCount | The number of mismatches for the specified connection since the last connection request for the pin pair, or zero if the connection does not exist. |
| gpIn | MPS discrete input |
| gpOut | MPS discrete output, if a connection exists |

Table 25 GPIOResponse Message

#### GPIO Device Names

The following logical names for GPIO input channels are used to specify the channels for a loop-back connection.

|  |  |
| --- | --- |
| Logical Name | Pin |
| PA\_KYLN\_IN1 | Insert B: C-9 |
| PA\_KYLN\_IN2 | Insert B: B-9 |
| PA\_KYLN\_IN3 | Insert B: A-9 |
| PA\_KYLN\_IN4 | Insert B: C-10 |
| PA\_KYLN\_IN5 | Insert B: B-10 |
| PA\_KYLN\_IN6 | Insert B: A-10 |
| PA\_KYLN\_IN7 | Insert B: C-11 |
| PA\_KYLN\_IN8 | Insert B: B-11 |
| PA\_ALL\_KYLN\_IN | Insert B: A-8 |
| PA\_MUTE\_KYLN\_IN | Insert B: A-8 |

|  |  |
| --- | --- |
| Logical Name | Pin |
| GP\_KYLN\_IN1 | Insert A: A-15 |
| GP\_KYLN\_IN2 | Insert A: C-15 |
| GP\_KYLN\_IN3 | Insert A: B-13 |
| GP\_KYLN\_IN4 | Insert A: B-14 |
| GP\_KYLN\_IN5 | Insert A: D-13 |
| GP\_KYLN\_IN6 | Insert B: A-11 |
| GP\_KYLN\_IN7 | Insert B: B-12 |
| GP\_KYLN\_IN8 | Insert B: A-12 |
| GP\_KYLN\_IN9 | Insert B: A-13 |

Table 26 GPIO Input Names

The following logical names for GPIO output channels are used to specify the channels for a loop-back connection.

|  |  |
| --- | --- |
| Logical Name | Pin |
| VA\_KYLN\_OUT1 | Insert A: E-15 |
| VA\_KYLN\_OUT2 | Insert A: D-15 |
| VA\_KYLN\_OUT3 | Insert A: C-15 |
| VA\_KYLN\_OUT4 | Insert A: B-15 |
| VA\_KYLN\_OUT5 | Insert A: A-15 |
| VA\_KYLN\_OUT6 | Insert A: B-14 |

|  |  |
| --- | --- |
| Logical Name | Pin |
| GP\_KYLN\_OUT1 | Insert A: A-13 |
| GP\_KYLN\_OUT2 | Insert A: C-13 |
| GP\_KYLN\_OUT3 | Insert A: C-14 |
| GP\_KYLN\_OUT4 | Insert A: D-15 |
| GP\_KYLN\_OUT5 | Insert A: D-14 |
| GP\_KYLN\_OUT6 | Insert A: E-15 |
| GP\_KYLN\_OUT7 | Insert B: C-13 |
| GP\_KYLN\_OUT8 | Insert B: B-13 |
| GP\_KYLN\_OUT9 | Insert B: A-14 |

Table 27 GPIO Output Names

### RS-232

The purpose of the RS-232 Application is to simulate loading of the RS-232 bus components and detect any failures thereof. The test strategy is to generate output on the transmitter channel which will be externally looped back to the receiver channel in order to load and verify both the transmitter and receiver circuits.

#### RS-232 Request

The RS232Request Message is sent by the TE to initiate or halt transmission and verification of RS-232 data according to the test procedure and to request a report in response..

message RS232Request {

enum RequestTypeT {

STOP = 0;  
 RUN = 1;  
 REPORT = 2;  
 }

required RequestTypeT requestType = 1 [default = STOP];

}

|  |  |
| --- | --- |
| Name | Description |
| requestType | Type of request (REPORT, STOP, or RUN); for STOP requests, the response message will include the counter values just prior to stopping |

Table 28 RS232Request Message

#### RS-232 Response

The RS232Response Message is sent by the MPS to acknowledge an RS-232 request and provide the number of data matches and mismatches since the application last received a RUN request.

message RS232Response {

enum AppStateT {

STOPPED = 0;  
 RUNNING = 1;  
 }

required AppStateT state = 1;

required uint32 xmtCount = 2 [default = 0];

required uint32 matches = 3 [default = 0];

required uint32 mismatches = 4 [default = 0];

}

|  |  |
| --- | --- |
| Field | Description |
| STOPPED | The application is not running |
| RUNNING | The application is running |
| xmtCount | The number of bytes transmitted since the last *RUN* request |
| matches | The number of matched bytes detected since the last *RUN* request |
| mismatches | The number of mismatched bytes detected since the last *RUN* request; in case of gaps in received data, system will track number of sent/received bytes, calculate number of lost bytes, and add to mismatches |

Table 29 RS232Response Message

### RS-485

The purpose of the RS-485 Application is to simulate loading of the RS-485 bus components and detect any failures thereof. The test strategy is to generate output on the RS-485 channel which will be externally echoed by the TE back to the MPS RS-485 channel in order to load and verify both the transmitter and receiver circuits. The transmitted and received data are then compared and mismatches reported.

#### RS-485 Request

The RS232Request Message is sent by the TE to initiate or halt transmission and verification of RS-485 data according to the test procedure.

message RS485Request {

enum RequestTypeT {

STOP = 0;  
 RUN = 1;  
 REPORT = 2;  
 }

required RequestTypeT requestType = 1 [default = STOP];

}

|  |  |
| --- | --- |
| Name | Description |
| requestType | Type of request (REPORT, STOP or RUN); for STOP requests, the response message will include the counter values just prior to stopping |

Table 30 RS232Request Message

#### RS-485 Response

The RS485Response Message is sent by the MPS to acknowledge an RS-485 request and provide the data mismatch statistics.

message RS485Response {

enum AppStateT {

STOPPED = 0;  
 RUNNING = 1;  
 }

required AppStateT state = 1;

required uint32 xmtCount = 2;

required uint32 matches = 3;

required uint32 mismatches = 4;

}

|  |  |
| --- | --- |
| Field | Description |
| STOPPED | The application is not running |
| RUNNING | The application is running |
| xmtCount | The number of bytes transmitted since the last *RUN* request |
| matches | The number of matched bytes detected since the last *RUN* request |
| mismatches | The number of mismatched bytes detected since the last *RUN* request; in case of gaps in received data, system will track number of sent/received bytes, calculate number of lost bytes, and add to mismatches |

Table 31 RS485Response Message

### SSD Application

#### SSD Application Request

The SSDRequest Message is sent by the TE to initiate or halt the SSD application according to the test procedure, or request application status.

message SSDRequest {

enum RequestTypeT {

STOP = 0;  
 RUN = 1;

REPORT = 2;  
 }

required RequestTypeT requestType = 1 [default = STOP];

}

|  |  |
| --- | --- |
| Name | Description |
| requestType | Type of request (STOP, RUN, or REPORT) |

Table 32 SSDRequest Message

#### SSD Application Response

The SSDResponse Message is sent by the MPS to acknowledge an SSD Application request or report an error.

message SSDResponse {

enum AppStateT {

STOPPED = 0;  
 RUNNING = 1;  
 }

required AppStateT state = 1;

required float readBandwidth = 2;

required float writeBandwidth = 3;

}

|  |  |
| --- | --- |
| Field | Description |
| STOPPED | The application is not running |
| RUNNING | The application is running |
| readBandwidth | The current bandwidth of data being read from the SSD RAID, in Mbytes/s |
| writeBandwidth | The current bandwidth of data being written to the SSD RAID, in Mbytes/s |

Table 33 SSDResponse Message

### HDDS Pass-Through

In order to get system information such as Remote Inventory, a pass-through interface to connect to the Host Domain Device Service (HDDS) will be provided. This interface will accept the same requests and return the same responses as the HDDS.

#### HDDS Get Request

The HDDS Get Request is a client request to the HDDS for one or more elements by key name.

message GetReq {

repeated string key = 1;

}

Table 34 HDDS GetReq Message

This message is passed unmodified to the HDDS. Refer to the *MPS Host Domain Device Service ICD* for details regarding the use of this message.

#### HDDS Get Response

The HDDS Get Response is the response for the HDDS Get Request message.

message GetResp {

repeated ValueResp HDDSValue = 1;

}



Table 35 HDDS GetResp Message

This message is returned unmodified from the HDDS. Refer to the *MPS Host Domain Device Service ICD* for details regarding the contents of this message.

#### HDDS Set Request

The HDDS Get Request is a client request to the HDDS to set given key(s) to the given value(s).

message GetReq {

repeated Property HDDSValue = 1;

}

Table 34 HDDS SetReq Message

This message is passed unmodified to the HDDS. Refer to the *MPS Host Domain Device Service ICD* for details regarding the use of this message.

#### HDDS Set Response

The HDDS Set Response is the response for the HDDS Set Request message.

message SetResp {

repeated ValueResp HDDSValue = 1;

}

Table 35 HDDS SetResp Message

This message is returned unmodified from the HDDS. Refer to the *MPS Host Domain Device Service ICD* for details regarding the contents of this message.

### System Monitoring

The System Monitoring interface is deprecated in favor of the HDDS Pass-Through interface.