PMCI Test Tools Interface and Design Specification

Supersedes: None

Document Class: Normative

Document Status: Published

Document Language: en-US
DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. Members and non-members may reproduce DMTF specifications and documents, provided that correct attribution is given. As DMTF specifications may be revised from time to time, the particular version and release date should always be noted.

Implementation of certain elements of this standard or proposed standard may be subject to third party patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose, or identify any or all such third party patent right, owners or claimants, nor for any incomplete or inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, disclose, or identify any such third party patent rights, or for such party's reliance on the standard or incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any party implementing such standard, whether such implementation is foreseeable or not, nor to any patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a patent owner for such implementations.

For information about patents held by third-parties which have notified DMTF that, in their opinion, such patent may relate to or impact implementations of DMTF standards, visit http://www.dmtf.org/about/policies/disclosures.php.

This document's normative language is English. Translation into other languages is permitted.
## CONTENTS

1 Foreword .......................................................... 5
   1.1 Acknowledgments ............................................. 5
2 Introduction ....................................................... 6
   2.1 Conventions ................................................. 6
      2.1.1 Document conventions ................................. 6
      2.1.2 Reserved and unassigned values ................... 6
      2.1.3 Byte ordering ........................................ 6
      2.1.4 Test Interface data types ......................... 6
      2.1.5 Version encoding .................................... 7
      2.1.6 Notations ............................................ 8
3 Scope ............................................................ 9
4 Normative references ............................................ 10
5 Terms and definitions .......................................... 11
6 Symbols and abbreviated terms ............................... 13
7 PMCI Test Architecture ......................................... 14
8 PMCI Test Tools Interface Concepts .......................... 16
   8.1 Interface Scope ............................................ 16
   8.2 Security .................................................... 16
      8.2.1 Overview ............................................ 16
      8.2.2 Security Requirements .............................. 16
      8.2.3 Security Best Practices ............................ 17
   8.3 Test Client and Test Service Interface .................. 17
      8.3.1 Admin and Test Protocols Messages Flow .......... 18
      8.3.2 Admin Messaging Protocol .......................... 18
         8.3.2.1 Client Session Establishment ................ 18
      8.3.3 Test Messaging Protocol ............................ 20
      8.3.4 NC-SI Testing Considerations ..................... 20
9 Test service behavior ........................................... 21
   9.1 Device security arbiter ................................... 21
   9.2 Connection Watchdog ...................................... 21
   9.3 Proxying of messages .................................... 21
9.4 Collection of timing information ......................... 22
9.5 Relaying of device-initiated messages to registered test clients ....................... 22
10 Messages ....................................................... 23
   10.1 Message structure ....................................... 23
      10.1.1 Test Service Wrapper .............................. 23
         10.1.1.1 Protocol Type ................................ 23
         10.1.1.2 Test Service Wrapper Flags ................ 24
      10.1.2 Message Response Codes ........................... 24
   10.2 Admin Messages .......................................... 25
      10.2.1 Command Codes .................................... 25
10.2.2 Connect (0x00) ................................................................. 26
10.2.3 Disconnect (0x01) ......................................................... 26
10.2.4 Query Capabilities (0x10) ............................................. 27
10.2.5 Query Status (0x11) ....................................................... 29
10.2.6 Query System Inventory (0x12) ...................................... 30
10.2.7 Configure Test Service (0x20) ........................................ 32
10.2.8 Configure Device Under Test (0x21) .............................. 33
10.2.9 Configure Device Under Test Examples .......................... 34
  10.2.9.1 Example 1 ............................................................... 35
  10.2.9.2 Example 2 ............................................................... 35
  10.2.9.3 Example 3 ............................................................... 35
  10.2.9.4 Example 4 ............................................................... 35
  10.2.9.5 Example 5 ............................................................... 35
10.2.10 Register to Protocol (0x22) ......................................... 36
10.2.11 Register Async Message Recipient (0x23) ..................... 37
10.2.12 Log Event (0x30) ....................................................... 37
10.3 Vendor Defined Admin .......................................................... 38
10.4 Test Messages ................................................................. 39
  10.4.1 Device-originated (Async) Protocol Messages ............... 39
10.5 Test Request and Response Messages ................................. 40
10.6 Handling MCTP Packets .................................................... 41
10.7 Handling NC-SI over RBT Packets ....................................... 42
11 ANNEX A SystemInventory Example (informative) and Schema (normative) .................. 45
  11.1 SystemInventory Example ............................................... 45
  11.2 SystemInventory Schema ............................................... 54
12 ANNEX B (informative) Change log ........................................ 60
13 Bibliography ........................................................................ 61
1 Foreword

The Platform Management Communications Infrastructure (PMCI) working group of the DMTF prepared the PMCI Test Tools Interface and Design Specification (DSP0280). DMTF is a not-for-profit association of industry members that promotes enterprise and systems management and interoperability. For information about the DMTF, see DMTF.

1.1 Acknowledgments

The DMTF acknowledges the following individuals for their contributions to this document:

Contributors:

- Daniil Egranov — Arm Ltd.
- Ira Kalman — Intel Corporation
- Justin King — IBM
- Guerney Hunt — IBM
- Manojkiran Eda — IBM
- Peter Lieber — Broadcom Inc.
- Greg Roth — Lenovo
- Bill Scherer — Hewlett Packard Enterprise
- Pat Schoeller — Intel Corporation
- Bob Stevens — Dell Technologies
- Tom Joseph — NVIDIA Corporation
2 Introduction

The PMCI Test Tools Interface and Design Specification defines messages, data objects, and sequences for testing PMCI protocol implementations in devices over a variety of transport media. The description of message exchanges includes secure registration of test clients, administrative messages for configuring test sessions, and specific messages for testing individual messages in the various PMCI protocols. The message exchanges are demonstrated via an example test client.

2.1 Conventions

The following conventions apply to this specification.

2.1.1 Document conventions

- Document titles appear in italics.
- The first occurrence of each important term appears in italics with a link to its definition.
- ABNF rules appear in a monospaced font.

2.1.2 Reserved and unassigned values

Unless otherwise specified, any reserved, unspecified, or unassigned values in enumerations or other numeric ranges are reserved for future definition by the DMTF.

Unless otherwise specified, reserved numeric and bit fields shall be written as zero (0) and ignored when read.

2.1.3 Byte ordering

Unless otherwise specified, for this specification byte ordering of multi-byte numeric fields or multi-byte bit fields is "Little Endian" (that is, the lowest byte offset holds the least significant byte, and higher offsets hold the more significant bytes).

2.1.4 Test Interface data types

The Test interface data types table lists the abbreviations and descriptions for common data types used in this specification. These definitions follow DSP0240.

Test Tool Interface data types
<table>
<thead>
<tr>
<th>Data type</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ver8</td>
<td>Eight-bit encoding of the PTTI version number. Version encoding defines the encoding of the version number.</td>
</tr>
<tr>
<td>bitfield8</td>
<td>Byte with 8-bit field. Each of these bit fields can be defined separately.</td>
</tr>
<tr>
<td>bitfield16</td>
<td>Two-byte word with 16-bit field. Each of these bit fields can be defined separately.</td>
</tr>
<tr>
<td>sint8</td>
<td>8-bit signed integer.</td>
</tr>
<tr>
<td>sint16</td>
<td>16-bit signed integer.</td>
</tr>
<tr>
<td>sint32</td>
<td>32-bit signed integer.</td>
</tr>
<tr>
<td>sint64</td>
<td>64-bit signed integer.</td>
</tr>
<tr>
<td>uint8</td>
<td>8-bit unsigned integer.</td>
</tr>
<tr>
<td>uint16</td>
<td>16-bit unsigned integer.</td>
</tr>
<tr>
<td>uint32</td>
<td>32-bit unsigned integer.</td>
</tr>
<tr>
<td>uint64</td>
<td>64-bit unsigned integer.</td>
</tr>
<tr>
<td>bool8</td>
<td>8-bit boolean value. 0x00 is false; all other values are true</td>
</tr>
<tr>
<td>real32</td>
<td>32-bit real value, formatted per ANSI/IEEE Standard 754</td>
</tr>
<tr>
<td>real64</td>
<td>64-bit real value, formatted per ANSI/IEEE Standard 754</td>
</tr>
<tr>
<td>strASCII</td>
<td>Null-terminated ASCII-encoded string.</td>
</tr>
<tr>
<td>strUTF8</td>
<td>Null-terminated UTF-8-encoded string.</td>
</tr>
<tr>
<td>enum8</td>
<td>A sequential enumeration, starting from 0 as the default, with mandatory numeric declarator. The number 8 indicates that the enumeration is encoded using an 8-bit binary number. Example: enum8 { CPU = 0, Memory = 1, Network = 2, Storage = 3 }</td>
</tr>
</tbody>
</table>

### 2.1.5 Version encoding

A field with data type ver8 encodes the supported version of the PTTI specification through a combination of Major and Minor nibbles, encoded as follows:

<table>
<thead>
<tr>
<th>Version</th>
<th>Matches</th>
<th>Incremented when</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major</td>
<td>Major version field</td>
<td>Protocol modification breaks backward compatibility.</td>
</tr>
<tr>
<td>Minor</td>
<td>Minor version field</td>
<td>Protocol modification maintains backward compatibility.</td>
</tr>
</tbody>
</table>

**EXAMPLE:**
A Test Agent that supports Version 1.2 can interoperate with an older Test Agent that supports Version 1.0 only, but the available functionality is limited to what specification Version 1.0 defines.

A Test Agent that supports Version 1.2 only and a Test Agent that supports Version 3.7 only are not interoperable and shall not attempt to communicate beyond the initial Connect admin message. If the Test Service detects that it is incompatible, it must respond to the Connect message with an INCOMPATIBLE_VERSION message response code and the Test Client must not communicate any further. If the Test Client receives a SUCCESS message response code from the Connect message, but is unable to use the Test Service Version from the Connect response message, it shall issue a Disconnect admin message and then cease communication with the Test Service.

2.1.6 Notations

This specification uses the following notations:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M:N</td>
<td>In field descriptions, this notation typically represents a range of byte offsets starting from byte ( M ) and continuing to and including byte ( N ) (( M \leq N )). The lowest offset is on the left. The highest offset is on the right.</td>
</tr>
<tr>
<td>[4]</td>
<td>Square brackets around a number typically indicate a bit offset. Bit offsets are zero-based values. That is, the least significant bit.</td>
</tr>
<tr>
<td>[M:N]</td>
<td>A range of bit offsets where ( M ) is greater than or equal to ( N ). The most significant bit is on the left, and the least significant bit is on the right.</td>
</tr>
<tr>
<td>1b</td>
<td>A lowercase ( b ) after a number consisting of ( 0 )s and ( 1 )s indicates that the number is in binary format.</td>
</tr>
<tr>
<td>0x12A</td>
<td>Hexadecimal, indicated by the leading 0x.</td>
</tr>
<tr>
<td>N+</td>
<td>Variable-length byte range that starts at byte offset ( N ).</td>
</tr>
</tbody>
</table>
3 Scope

This specification describes messages and flows used to capture PMCI upper (data model) layer data from a device. The data may be used to assess conformance of a device vendor firmware that implements PMCI protocols. This document specifies a specially protected interface to a Control Plane (such as a BMC) that supports a test API for tool clients.

Techniques for verifying that the captured data demonstrates compliance or conformance to PMCI specifications are not in scope for this specification. DMTF and PMCI Working Group will not certify protocol implementations in device vendor firmware. Any test clients provided by DMTF and the PMCI Working Group are offered as-is.
4 Normative references

The following documents are indispensable for the application of this specification. For dated or versioned references, only the edition cited, including any corrigenda or DMTF update versions, applies. For references without a date or version, the latest published edition of the referenced document (including any corrigenda or DMTF update versions) applies.

- DMTF DSP0218, Platform Level Data Model (PLDM) for Redfish Device Enablement, https://www.dmtf.org/dsp/DSP0218
- DMTF DSP0239, MCTP IDs and Codes, https://www.dmtf.org/dsp/DSP0239
- DMTF DSP0245, Platform Level Data Model (PLDM) IDs and Codes Specification, https://www.dmtf.org/dsp/DSP0245
- DMTF DSP0248, Platform Level Data Model (PLDM) for Platform Monitoring and Control Specification, https://www.dmtf.org/dsp/DSP0248
- DMTF DSP0249, Platform Level Data Model (PLDM) State Set Specification, https://www.dmtf.org/dsp/DSP0249
- DMTF DSP0261, NC-SI over MCTP Binding Specification, https://www.dmtf.org/dsp/DSP0261
- DMTF DSP0267, Platform Level Data Model (PLDM) for Firmware Update Specification, https://www.dmtf.org/dsp/DSP0267
5 Terms and definitions

In this document, some terms have a specific meaning beyond the normal English meaning. This clause defines those terms.

The terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"), "may", "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described in ISO/IEC Directives, Part 2 (see Clause 7). The terms in parenthesis are alternatives for the preceding term, for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that ISO/IEC Directives, Part 2 (see Clause 7) specifies additional alternatives. Occurrences of such additional alternatives shall be interpreted in their normal English meaning.

The terms “clause”, “subclause”, “paragraph”, and “annex” in this document are to be interpreted as described in ISO/IEC Directives, Part 2 (see Clause 6).

The terms “normative” and “informative” in this document are to be interpreted as described in ISO/IEC Directives, Part 2 (see Clause 3). In this document, clauses, subclauses, or annexes labeled “(informative)” do not contain normative content. Notes and examples are always informative elements.

The terms that DSP0236, DSP0239, and DSP0275 define also apply to this document.

This specification uses these terms:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous Message</td>
<td>A Request Message that originates at the Test Service and flows to the Test Client, either due to a message from a DUT or an administrative event within the Test Service.</td>
</tr>
<tr>
<td>Client Session</td>
<td>A session established between the Test Client and the Test Service for the purpose of PMCI upper layer protocols validation.</td>
</tr>
<tr>
<td>Command Code</td>
<td>A numeric identifier, typically one byte, inserted into a message to indicate an operation to be performed.</td>
</tr>
<tr>
<td>Control Plane</td>
<td>An entity that provides hardware monitoring and control functions for a platform. Commonly, this is a BMC as defined in DSP0236, though it may also be a primary/secondary BMC pair, a network of peer BMCS, or another management entity altogether in a complex, multi-chassis environment.</td>
</tr>
<tr>
<td>DUT</td>
<td>Device Under Test; A managed device like Host Firmware, Network Controller, etc.</td>
</tr>
<tr>
<td>Device Identifier</td>
<td>A value identifying a device. This does not determine a physical interface or path. There is only one Device Identifier per device.</td>
</tr>
<tr>
<td>DMTF</td>
<td>Formerly known as the Distributed Management Task Force, the DMTF creates open manageability standards that span diverse emerging and traditional information technology (IT) infrastructures, including cloud, virtualization, network, servers, and storage. Member companies and alliance partners worldwide collaborate on standards to improve the interoperable management of IT.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DUT Connection ID</td>
<td>A value returned by the <em>Configure Device Under Test</em> message which identifies a specific path and interface to a DUT for use in protocol Test Messages.</td>
</tr>
<tr>
<td>Interface Identifier</td>
<td>A value that identifies a specific interface to a device. There can be multiple Interface Identifiers per device.</td>
</tr>
<tr>
<td>Message</td>
<td>A sequence of data communicated between a requester and a responder, such as a <em>Test Client</em> and a <em>Test Service</em>, to effect an operation specified by a <em>Command Code</em>. A message may be classified as either a <em>Request Message</em> or a <em>Response Message</em>.</td>
</tr>
<tr>
<td>Request Message</td>
<td>A <em>Message</em> that asks the recipient to initiate an operation.</td>
</tr>
<tr>
<td>Response Message</td>
<td>A <em>Message</em> that provides the results of an operation to the entity that requested it.</td>
</tr>
<tr>
<td>Test Agent</td>
<td>Either a <em>Test Client</em> or a <em>Test Service</em>, entities which can communicate over the Test Interface using the protocols defined in this specification.</td>
</tr>
<tr>
<td>Test Client</td>
<td>The part of a testing tool or suite which is responsible for the communication with a <em>Test Service</em> via a <em>Test Interface</em>.</td>
</tr>
<tr>
<td>Test Interface</td>
<td>An interface that is used for the communication between the <em>Test Service</em> and the <em>Test Client(s)</em>.</td>
</tr>
<tr>
<td>Test Message</td>
<td>A <em>Request Message</em> that originates at the <em>Test Client</em> and flows to the <em>Test Service</em> that instructs the Test Service to send an Upper-layer protocol message to a DUT</td>
</tr>
<tr>
<td>Test Protocol</td>
<td><em>Messages</em> and sequences for testing PMCI protocol implementations in devices over a variety of transport mediums using the <em>Test Interface</em> that is defined in this specification.</td>
</tr>
<tr>
<td>Test Service</td>
<td>An application that communicates with the Test Client(s) through a <em>Test Interface</em> for the purpose of PMCI upper layer protocols validation. The Test Service runs on the <em>Control Plane</em>, and analyzes the messages sent via a <em>Test Interface</em> and sends the corresponding messages (like PLDM/NC-SI/SPDM) to a <em>DUT</em>.</td>
</tr>
</tbody>
</table>
# 6 Symbols and abbreviated terms

The following abbreviations are used in this document.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMTF</td>
<td>Formerly the <em>Distributed Management Task Force</em></td>
</tr>
<tr>
<td>PMCI</td>
<td>Platform Management Communications Infrastructure</td>
</tr>
<tr>
<td>DUT</td>
<td><em>Device Under Test</em></td>
</tr>
<tr>
<td>PTTI</td>
<td>PMCI Test Tools Interface</td>
</tr>
<tr>
<td>TC</td>
<td>Test Client</td>
</tr>
<tr>
<td>TLS</td>
<td><em>Transport Layer Security</em></td>
</tr>
<tr>
<td>TS</td>
<td>Test Service</td>
</tr>
</tbody>
</table>
7 PMCI Test Architecture

This clause provides a high-level view of the components of the PMCI Test Architecture. It contains three main components:

- Test client software
- Test service software or firmware
- Devices that support PMCI protocols that can be tested

The test client is a piece of software that performs test and validation functions for one or more PMCI protocols. Test clients may be written in any language so long as they have the ability to communicate with the test service over the network via the TLS protocol.

The next component is the test service, which can be either firmware or software. Typically a part of the system's control plane (often a management controller), the test service offers an API by which test clients can register themselves and interrogate the system under test as to the number or type of target devices against which they can perform test functions.

The last component is the collection of devices in the system that support various PMCI protocols. The test service provides APIs by which test clients can target specific devices with test messages that validate their implementations of PMCI protocols. The connection from the test service to a particular device under test can be via any of the hardware buses that are supported in the PMCI protocol stack.
System

Test Client

Test Service

Control Plane (such as BMC)

Device Under Test

System
8 PMCI Test Tools Interface Concepts

8.1 Interface Scope

The PMCI Test Tools Interface is intended for testing upper layer PMCI protocols. However, NC-SI Pass-through and MCTP Control are not within the scope of PTTI, and Test Service implementations may elect to prohibit testing these protocols altogether.

8.2 Security

8.2.1 Overview

The PMCI Test Tools Interface can be utilized in a number of circumstances, such as:

- Development of devices or device firmware
- Acceptance test of a new device or new device firmware
- Manufacturing of systems containing devices
- Debugging a device in a production system

PTTI opens up a communication path between an external entity (Test Client) and a device under test inside a system. Therefore, it is strongly recommended to consider the security of this communication path. In some cases it may be necessary for the Test Client to be on a remote network, and this scenario requires even more thought.

PTTI is built upon TLS, which provides a number of benefits:

- The Test Client and Test Server are authenticated to one another.
- The Test Client, using TLS certificates, verifies that it is talking to the expected control plane.
- The link between Test Agents is encrypted and can prevent replay and man-in-the-middle attacks.

8.2.2 Security Requirements

1. The Test Client shall connect to the Test Service with a secure, authenticated protocol (TLS).
2. The Test Client shall verify that the target system is the intended system by validating the Test Service’s TLS certificate.
3. The Test Client shall verify that the protocol version returned from the Connect message is supported.
8.2.3 Security Best Practices

Test Agents should log all interactions with their partner Test Agent, including authorization credentials.

The Test Service should be disabled by default on the control plane, and it should require special authorization to enable the Test Service. The control plane should log all authorization credentials when an administrator attempts to enable the Test Service.

The Test Client should verify that the control plane and the Test Service are at an acceptable level of firmware via the *Query System Inventory* message.

When a remote Test Client (that is, a Test Client on a separate network from the control plane) is required, the control plane administrator should only enable the Test Service to run when required, and shall disable the Test Service as soon as the connection with the Test Client is ended. The Test Client should be required to go through some additional form of authentication, such as a Virtual Private Network.

It is not recommended to enable the Test Service when the system is in production, when devices in the system are in production, or when devices in the system store persistent data. If the target system or devices are in production and the Test Service must be run, then:

- The control plane should not be directly connected to the internet.
- The control plane administrator should ensure that the Security Parameter to the *Connect* message is very strong (such as a long password or certificate).
- A device under test is offline whenever possible (for example, a network card is disconnected from any network and not available to users of the system).
- A device under test with persistent data should be reset before and after the connection with the Test Client.

8.3 Test Client and Test Service Interface

There are two kinds of messages that can be sent over the test client and test service interface:

- Messages related to Admin Messaging Protocol
- Messages related to Test Messaging Protocol

The differences between two protocols and also the scope, are described later on in this chapter. The *Admin and Test Protocols Messages Flow* depicts the high-level flow diagram for the Test Client and Test Service communication.
8.3.1 Admin and Test Protocols Messages Flow

```
<table>
<thead>
<tr>
<th>Test Client</th>
<th>Test Service</th>
<th>DUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect { Security parameter }</td>
<td>Response { Test Service version, Test Client ID }</td>
<td></td>
</tr>
<tr>
<td>Query Capabilities { }</td>
<td>Response { Capabilities of Test Service }</td>
<td></td>
</tr>
<tr>
<td>Configure Test Service { Capabilities }</td>
<td>Response</td>
<td></td>
</tr>
<tr>
<td>Query System Inventory { }</td>
<td>Response { System Inventory Data }</td>
<td></td>
</tr>
<tr>
<td>Configure DUT #N { Target Identifier, Identifier List }</td>
<td>Response { DUT Connection ID }</td>
<td></td>
</tr>
<tr>
<td>Register to Protocol #N { Protocol, DUT Connection ID }</td>
<td>Response { DUT Connection ID }</td>
<td></td>
</tr>
<tr>
<td>Register Async Message Recipient #N { Protocol, DUT Connection ID }</td>
<td>Response { DUT Connection ID }</td>
<td></td>
</tr>
<tr>
<td>Test Protocol Message #N</td>
<td>Response #N</td>
<td></td>
</tr>
<tr>
<td>Async Test Protocol Message #N</td>
<td>Response #N</td>
<td></td>
</tr>
</tbody>
</table>
```

8.3.2 Admin Messaging Protocol

The admin messaging protocol is used to set up the test service and to register a test client that will perform testing. Admin messages provide support for the following functionality:

- Establishing connections to test clients (see Client Session Establishment)
- Providing a system inventory of available devices against which testing may be performed
- Registering a test client to specific protocols and to specific target devices
- Configuring a device for testing, including the hardware path to be used for testing it
- Registering a test client to receive asynchronous messages sent from devices

8.3.2.1 Client Session Establishment

There are multiple steps in establishing a session between the Test Client and the Test Service. First, the Test Service
must open a TLS port and begin listening on that port. This document does not specify how the TLS port is opened, which will vary based on the vendor that produces the Test Service.

Second, the TLS port number and the Test Service's certificate must be provided as inputs to the Test Client. Again, this document does not specify any procedure for providing these inputs, and in many cases this will be a manual setup step. The Test Client is responsible for verifying the correctness of the provided certificate.

The Test Client will initiate a TLS connection with the Test Service as detailed in the TLS specification, using the provided certificate and port. Once the TLS connection is established, the data flowing between the Test Client and Test Service is encrypted and protected from man-in-the-middle attacks.

Next, the Test Client will send a Connect admin message over the secure channel. The Security Parameter of the Connect message proves that the Test Client has authorization to send PTTI messages to the Test Service. This document does not specify the manner by which the Test Service validates the Security Parameter.

If the Security Parameter is invalid, the Test Service will respond to the Connect message with an AUTHENTICATION_ERROR message response code. However, if the Security Parameter is valid, the Test Service will respond to the Connect message with a SUCCESS message response code as well as a Test Client ID and a Test Service Version. The Test Client ID serves as a token that must be passed as a parameter with all subsequent messages. This Test Client ID is just a handle and has no meaning other than to identify the Test Client to the Test Service. The Test Service Version returned by the Test Service should be validated by the Test Client to ensure that the Test Client is able to communicate with the Test Service.

Once the Test Client has obtained a Test Client ID and has validated that it can use the Test Service Version, it should issue the following admin messages:

- Query Capabilities to ensure that the Test Service provides all the capabilities necessary for the Test Client's operation.
- Query System Inventory to collect the inventory of available devices that may be tested.

From the inventory provided by the Test Service, the Test Client may select one device to designate as the Device Under Test. The Test Client shall use the Configure Device Under Test admin message to inform the Test Service that it wishes to test a specified device, and it optionally may select the path by which the Control Plane will talk to the device by using information from the system inventory.

After a DUT is successfully configured, the Test Client shall issue one or more Register To Protocol admin messages to request that the Test Service allow the Test Client to test particular protocols or protocol types. Optionally, the Test Client may issue one or more Register Async Message Recipient admin messages in order to receive messages asynchronously originating from the DUT.

Having completed all the messages to establish a test session, the Test Client may now begin testing using the Test Messaging Protocol.
8.3.3 Test Messaging Protocol

The test messaging protocol is used by a test client to send request messages to the test service for relay to a device under test. The test service then collects the device's response to the request messages and returns them to the test client for evaluation and further processing. The test messaging protocol supports a variety of upper-level protocols and provides support for testing messages in each of these protocols.

In addition to message relay, the test messaging protocol collects timing information for the amount of time that a device took to respond to the Test Message. It also reports on various errors that may have occurred in the processing of the message.

The Test Client is permitted at most one outstanding Test Message per Protocol Type to each Device Under Test at a time. The Test Service is responsible for ensuring that the DUT responds in the appropriate amount of time, and if it does not, the Test Service must send a response to the Test Client with the \textit{TIMEOUT} Message Response Code.

If the Test Service does not respond to the Test Message in a time frame that is reasonable to the Test Client, the Test Client may terminate its connection to the Test Service. In this case, the Test Client may wish to send a \textit{Log Event} admin message with Reason Code \textit{ClientTimeout} before disconnecting.

8.3.4 NC-SI Testing Considerations

Special care should be taken when implementing tests for NC-SI. There are no provisions made in this specification for testing NC-SI Pass-through traffic, and it is recommended that the Test Service reject any NC-SI Pass-through tests (for example, those using MCTP Type 3).

NC-SI Control is supported by this specification. However, in certain architectures, the Test Service running on a Control Plane (such as a BMC) may be connected to the external network or the Test Client using NC-SI Pass-Through. The recommendation is that the implementer review the architecture of the system and avoid using the NC-SI \textit{Deselect} command for a package that may be providing network connectivity to the Control Plane. Details about NC-SI packages may be found in the “NC-SIInfo” section of the \textit{System Inventory}, and it is recommended that Test Services return the “NC-SIInfo” JSON object when NC-SI Control testing is supported to a DUT.
9 Test service behavior

9.1 Device security arbiter

The Test Service may optionally maintain a database of Test Clients and the devices for which they have permission to access. The Test Service may allow or deny access to devices for test purposes, and the Test Service may allow or deny access to certain protocols or protocol types. Further, the Test Service may allow or deny access to asynchronous messages originating from the device, on a protocol or protocol type basis.

The Test Client may check which protocols it has been granted permission for by issuing the Query Status admin message.

9.2 Connection Watchdog

The Test Service may optionally implement a Connection Watchdog, by which the Test Service may disconnect Test Clients that are no longer communicating with the Test Service. The Test Service must reset the Connection Watchdog timeout after receiving any message from the Test Client that has a valid Test Client ID. Upon expiration of the Watchdog timer, the Test Service will send the Log Event message with Reason Code WatchdogTimeout to indicate the connection will be disconnected and the Test Client ID will become invalid afterwards. The Test Service should reject any further messages from the Test Client with an AUTHENTICATION_ERROR message response code.

If implemented, the Connection Watchdog timeout must be reported in the Query Capabilities response message. The Test Client can configure the watchdog timeout using the Configure Test Service message and the maximum watchdog timeout that can be configured must be reported in the Query Capabilities response message. Should the Test Client wish to keep the connection open without sending Test Messages to the Test Service, the Test Client may use the Query Status message with the Query Type set to Ping.

9.3 Proxying of messages

The primary purpose of the test service is to act as a proxy, receiving messages from test clients, submitting them to a device under test, collecting response messages from the device, and relaying those responses back to the test client. It is imperative that the test service be as transparent as possible; to the maximum extent possible, it should appear to both the test client and the device under test that they are communicating directly. The main exception to this is that the test service is responsible for simplifying the communication pathways, enabling the test client to reach the device under test via an abstract DUT Connection ID rather than needing to know the physical hardware addresses for the device under test. Similarly, low-level details such as the packetization of MCTP messages is taken care of by the control plane so that the test client can focus on higher-level protocol testing.
9.4 Collection of timing information

Upon request, the test service is responsible for collecting timing information for how long it takes a device to respond to a request message. This measurement shall be the amount of time elapsed between when the last byte of the request message is sent and the first byte of the response message is received. In the event that multiple tries are required to obtain a response from the device under test, the test service shall only report the timing for the final request.

9.5 Relaying of device-initiated messages to registered test clients

Test clients may optionally register to receive asynchronous communications for specific protocols from a device under test. If the test client has so registered and the device emits a message that matches the registration criteria, the test service shall intercept that message and relay it to the test client.

For PLDM events, the Control Plane is responsible for collecting events whether those events come from device-initiated PlatformEventMessage messages or by Control Plane-initiated PollForPlatformEventMessage messages. In either case, the Control Plane must deliver these events to its Test Service so that they may be relayed to the Test Client.
10 Messages

This clause details the various messages used in the test interface.

10.1 Message structure

The basic structure of all PTTI request and response messages consists of two parts: a Test Service Wrapper and Message Data. For Admin requests and responses, the first byte of the Message Data indicates the specific Admin Command Code. For Test requests and responses, the full Upper-layer message (header and data) is contained in the Message Data.

<table>
<thead>
<tr>
<th>Byte Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Service Wrapper</td>
</tr>
<tr>
<td>Message header for the test service</td>
</tr>
<tr>
<td>Message Data</td>
</tr>
<tr>
<td>Protocol-specific message payload data</td>
</tr>
</tbody>
</table>

10.1.1 Test Service Wrapper

The test service wrapper is a header specific to the test interface and shall be present for all request and response messages.

<table>
<thead>
<tr>
<th>Type</th>
<th>Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>ver8</td>
<td>PTTI Protocol Version</td>
</tr>
<tr>
<td>Protocol Type</td>
<td>byte</td>
<td>Protocol Type</td>
</tr>
<tr>
<td>Flags</td>
<td>bitfield16</td>
<td>Test Service Wrapper Flags</td>
</tr>
<tr>
<td>Test Client ID</td>
<td>uint32</td>
<td>Client ID as assigned by the Connect response message</td>
</tr>
</tbody>
</table>

10.1.1.1 Protocol Type

The Protocol Type field identifies the protocol for the current message.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTTI Admin</td>
<td>0xFF</td>
</tr>
<tr>
<td>PTTI Vendor Defined Admin</td>
<td>0xF1</td>
</tr>
<tr>
<td>Test Messages (values from DSP0239)</td>
<td>0x00 - 0x7F</td>
</tr>
</tbody>
</table>
10.1.2 Test Service Wrapper Flags

The test service wrapper flags contain attributes of the current message.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0:1]</td>
<td>Direction</td>
<td>0: TC → TS Request Message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: TS → TC Response Message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: TS → TC Request Message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: TC → TS Response Message</td>
</tr>
<tr>
<td>[2:15]</td>
<td>R</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

10.1.2 Message Response Codes

The following table enumerates the possible response codes for messages. Response codes with value 0x80 and higher are reserved for command-specific message responses.

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SUCCESS</td>
<td>The responding test agent received a request and performed all required actions successfully.</td>
</tr>
<tr>
<td>1</td>
<td>TIMEOUT</td>
<td>The responding test agent received a request but a timeout occurred prior to completing the request.</td>
</tr>
<tr>
<td>2</td>
<td>INVALID_PROTOCOL</td>
<td>The Test Service received a Test request for an unsupported Protocol Type such as MCTP Control (0) or NC-SI Pass-through (3)</td>
</tr>
<tr>
<td>3</td>
<td>TRANSPORT_ERROR</td>
<td>The Test Service was unable to send the Test Message due to a transport-level error.</td>
</tr>
<tr>
<td>4</td>
<td>PHYSICAL_ERROR</td>
<td>The Test Service was unable to send the Test Message due to a physical-level error.</td>
</tr>
<tr>
<td>5</td>
<td>AUTHENTICATION_ERROR</td>
<td>The Test Service was unable to send the Test Message as the Test Client ID failed authentication checks.</td>
</tr>
<tr>
<td>6</td>
<td>PRIVILEGE_ERROR</td>
<td>The Test Service was unable to process the request because the Test Client has insufficient privilege to perform the requested action.</td>
</tr>
<tr>
<td>7</td>
<td>INTEGRITY_CHECK_ERROR</td>
<td>The responding test agent received a request but could not complete the request due to a data integrity check error.</td>
</tr>
<tr>
<td>8</td>
<td>INCOMPATIBLE_VERSION</td>
<td>The responding test agent received a message, but the Test Service Wrapper Version is incompatible with the test agent.</td>
</tr>
<tr>
<td>9</td>
<td>INVALID_DUT_CONNECTION_ID</td>
<td>The DUT Connection ID given by the requesting test agent is invalid.</td>
</tr>
<tr>
<td>10</td>
<td>OUTSTANDING_MESSAGE</td>
<td>The Test Agent already has an outstanding Test Message to the specified DUT, or already has an outstanding Admin message to its partner Test Agent, so this message is not permitted.</td>
</tr>
<tr>
<td>Code</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>0x80..0xEF</td>
<td>COMMAND_SPECIFIC_RESERVED</td>
<td>Reserved for command-specific message responses.</td>
</tr>
<tr>
<td>0xF0..0xFF</td>
<td>OEM_SPECIFIC_RESERVED</td>
<td>Reserved for OEM-specific message responses.</td>
</tr>
</tbody>
</table>

### 10.2 Admin Messages

Admin messages are used to setup the test service and register a test client. Admin messages are distinguished by the Protocol Type in the Test Wrapper, and have a Command Code as the first byte of their Message Data.

The Test Client is permitted one outstanding Admin message to the Test Service at a time, in addition to having one outstanding Test Message to each Device Under Test. If the Test Service detects a second Admin message has been received before it responds to the first Admin message, it may respond with the OUTSTANDING_MESSAGE Message Response Code.

If the Test Service does not respond to the Admin message in a time frame that is reasonable to the Test Client, the Test Client may terminate its connection to the Test Service. In this case, the Test Client may wish to send a Log Event admin message with Reason Code ClientTimeout before disconnecting.

### 10.2.1 Command Codes

The Command Code is the first byte of the Message Data in an Admin request or response message. It identifies which Admin command is contained within the message and enables interpretation of the message payload.

<table>
<thead>
<tr>
<th>Command</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>0x00</td>
</tr>
<tr>
<td>Disconnect</td>
<td>0x01</td>
</tr>
<tr>
<td>Query Capabilities</td>
<td>0x10</td>
</tr>
<tr>
<td>Query Status</td>
<td>0x11</td>
</tr>
<tr>
<td>Query System Inventory</td>
<td>0x12</td>
</tr>
<tr>
<td>Configure Test Service</td>
<td>0x20</td>
</tr>
<tr>
<td>Configure Device Under Test</td>
<td>0x21</td>
</tr>
<tr>
<td>Register to Protocol</td>
<td>0x22</td>
</tr>
<tr>
<td>Register Async Message Recipient</td>
<td>0x23</td>
</tr>
<tr>
<td>Log Event</td>
<td>0x30</td>
</tr>
</tbody>
</table>
10.2.2 Connect ( 0x00 )

The Connect message connects a Test Client to a Test Service. The Connect message does not require a valid Test Client ID parameter in the Test Service Wrapper.

A security parameter must be passed in the Connect request message. The process for generating this security parameter is dependent on the Test Service vendor, and will not be specified in this document. The security parameter may be a password, encryption key, certificate, or other construct as determined by the Test Service vendor.

When the Test Service receives a security parameter that it accepts, the Test Service will respond to the Connect message with a 4-byte Test Client ID that must be passed in the Test Service Wrapper, by the Test Client, on all subsequent messages. The Test Service also provides the version of the Test Service API that it supports. In the event that the Test Client is incompatible with this version, it shall use the Disconnect message to cease testing.

While connected, this Test Service is unavailable to other Test Clients and will refuse connection requests. A connection is terminated by the Disconnect message or a Connection Watchdog timeout.

Connect request message format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x00 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Security Parameter Length</td>
<td>uint32</td>
<td>Length of the security parameter</td>
</tr>
<tr>
<td>5</td>
<td>Security Parameter</td>
<td>Variable</td>
<td>Security parameter provided by the Test Service vendor</td>
</tr>
</tbody>
</table>

Connect response message format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x00 - See Command Codes</td>
</tr>
</tbody>
</table>
| 1      | Response Code       | enum8 | See Test Service Response Codes and the below specific codes.  
0x80: OTHER_CLIENT_CONNECTED - The test service is already connected to another client. |
| 2      | Test Service Version| ver8  | The version of the Test Service API that the Test Service implements |
| 3      | Test Client ID      | uint32 | Test Client ID for use in the Test Service Wrapper on subsequent messages |

10.2.3 Disconnect ( 0x01 )

The Disconnect message terminates the connection between a Test Client and a Test Service. Any outstanding messages shall be canceled (if possible) or quiesced (otherwise) by the Test Service prior to response to this message.
Registration for asynchronous notification of messages initiated by devices under test shall be implicitly canceled by the issuing of this message.

After the *Disconnect* message is issued, the Test Client ID parameter in the *Test Service Wrapper* is no longer valid, and a new *Connect* message must be issued with a valid security parameter in order to obtain a new Test Client ID.

### Disconnect request message format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x01 - See Command Codes</td>
</tr>
</tbody>
</table>

### Disconnect response message format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x01 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Response Code</td>
<td>enum8</td>
<td>See Test Service Response Codes</td>
</tr>
</tbody>
</table>

### 10.2.4 Query Capabilities (0x10)

The *Query Capabilities* message shall be used by the Test Client to determine the capabilities of the Test Service. A Test Client must issue the *Query Capabilities* message after every successful *Connect* message. The Test Service must guarantee that the capabilities do not change while the Test Client ID obtained from the *Connect* message remains valid.

A Test Service may optionally support each of the following capabilities. For all one-bit fields, a value of 1b indicates that the Test Service supports the feature, and a value of 0b indicates that the Test Service does not support the feature.

### Test Service Capabilities

<table>
<thead>
<tr>
<th>Name</th>
<th>Capability ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Connection Watchdog Timeout (Read Only)</td>
<td>1</td>
<td>Maximum <em>Watchdog timeout</em> in seconds</td>
</tr>
<tr>
<td>Current Connection Watchdog Timeout</td>
<td>2</td>
<td>Current <em>Watchdog timeout</em> in seconds</td>
</tr>
<tr>
<td>Reserved for future use</td>
<td>3-32763</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>Vendor Capabilities Set #1 IANA Enterprise ID</td>
<td>32764</td>
<td>IANA Enterprise ID of the vendor owning the capabilities in ID range 32768-40959</td>
</tr>
<tr>
<td>Name</td>
<td>Capability ID</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Vendor Capabilities Set #2 IANA Enterprise ID</td>
<td>32765</td>
<td>IANA Enterprise ID of the vendor owning the capabilities in ID range 40960-49151</td>
</tr>
<tr>
<td>Vendor Capabilities Set #3 IANA Enterprise ID</td>
<td>32766</td>
<td>IANA Enterprise ID of the vendor owning the capabilities in ID range 49152-57343</td>
</tr>
<tr>
<td>Vendor Capabilities Set #4 IANA Enterprise ID</td>
<td>32767</td>
<td>IANA Enterprise ID of the vendor owning the capabilities in ID range 57344-65535</td>
</tr>
<tr>
<td>Vendor Capabilities Set #1</td>
<td>32768-40959</td>
<td>Vendor Capabilities Set #1</td>
</tr>
<tr>
<td>Vendor Capabilities Set #2</td>
<td>40960-49151</td>
<td>Vendor Capabilities Set #2</td>
</tr>
<tr>
<td>Vendor Capabilities Set #3</td>
<td>49152-57343</td>
<td>Vendor Capabilities Set #3</td>
</tr>
<tr>
<td>Vendor Capabilities Set #4</td>
<td>57344-65535</td>
<td>Vendor Capabilities Set #4</td>
</tr>
</tbody>
</table>

### Query Capabilities Message request format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x10 - See Command Codes</td>
</tr>
</tbody>
</table>

### Query Capabilities Message response format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x10 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Response Code</td>
<td>enum8</td>
<td>See Test Service Response Codes</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
<td>uint8</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>3</td>
<td>Number of Capabilities Fields</td>
<td>uint16</td>
<td>Count of capabilities in this structure</td>
</tr>
<tr>
<td>5</td>
<td>First Capability ID</td>
<td>uint16</td>
<td>From the Table of Capability IDs</td>
</tr>
<tr>
<td>7</td>
<td>First Capability Value</td>
<td>uint32</td>
<td>Value for the first Capability</td>
</tr>
<tr>
<td>11</td>
<td>Second Capability ID</td>
<td>uint16</td>
<td>From the Table of Capability IDs</td>
</tr>
<tr>
<td>13</td>
<td>Second Capability Value</td>
<td>uint32</td>
<td>Value for the second Capability</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N*6+5</td>
<td>Final Capability ID</td>
<td>uint16</td>
<td>From the Table of Capability IDs</td>
</tr>
<tr>
<td>N*6+9</td>
<td>Final Capability Value</td>
<td>uint32</td>
<td>Value for the final Capability</td>
</tr>
</tbody>
</table>
10.2.5 Query Status (0x11)

The Query Status message may be used by the Test Client to query the status of its connection to the Test Service and, optionally, the protocol and async-handling registrations for a device under test.

Query Status Message request format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x11 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Query Type</td>
<td>enum8</td>
<td>The type of status query being performed. See values in Query Status request.</td>
</tr>
</tbody>
</table>

Query Status Message response format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x11 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Response Code</td>
<td>enum8</td>
<td>See Test Service Response Codes</td>
</tr>
<tr>
<td>2</td>
<td>Query Type</td>
<td>enum8</td>
<td>The type of status query that was performed. See values in Query Status request.</td>
</tr>
<tr>
<td>3</td>
<td>Query Response Data Length</td>
<td>uint32</td>
<td>The length of the Query Response Data.</td>
</tr>
<tr>
<td>7</td>
<td>Query Response Data</td>
<td>variable</td>
<td>The data in response to the query, which is interpreted based on the Query Type.</td>
</tr>
</tbody>
</table>

Query Status Ping Response Data

If the Query Type is Ping, the Query Response Data Length will be 0, and there will be no Query Response Data.

Query Status Device List Response Data

If the Query Type is Device List, the Query Response Data will be in the following format:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Device Count</td>
<td>uint8</td>
<td>The number of devices returned in this query.</td>
</tr>
<tr>
<td>1</td>
<td>Device Data Structures</td>
<td>Repeating List</td>
<td>A repeating list of Query Status Device Data structures based on the Device Count</td>
</tr>
</tbody>
</table>

Query Status Device Data

The Test Client may use this structure to understand the test status of a particular DUT. In addition to a DUT...
Connection ID, the structure contains a list of protocol/type pairs. The `protocol` value of the pair indicates the specific protocol that is registered, such as `0x01` for PLDM. The type value of the pair indicates a type of message within the protocol, such as `0x02` for PLDM for Platform Monitoring and Control. For a protocol that does not have multiple types, the value `0x00` shall be used for the type. The Test Service will also indicate whether it will forward asynchronous messages that originate from the DUT to the Test Client.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DUT Connection ID</td>
<td>uint32</td>
<td>The DUT Connection ID for the device.</td>
</tr>
<tr>
<td>4</td>
<td>Device Registered Protocol Type Count</td>
<td>uint8</td>
<td>The count of protocol/type pairs for the device registered by the Test Client using <code>Register to Protocol</code></td>
</tr>
<tr>
<td>5</td>
<td>Device Registered Protocol #1</td>
<td>enum8</td>
<td>A protocol value for the first registered Protocol/Type pair</td>
</tr>
<tr>
<td>6</td>
<td>Device Registered Protocol Type #1</td>
<td>enum8</td>
<td>A protocol type value for the first registered Protocol/Type pair</td>
</tr>
<tr>
<td>7</td>
<td>Async Registered for Protocol Type #1</td>
<td>bool8</td>
<td>The Test Client has registered to receive asynchronous messages for the first Protocol/Type pair</td>
</tr>
<tr>
<td>8</td>
<td>Device Registered Protocol #2</td>
<td>enum8</td>
<td>A protocol value for the second registered Protocol/Type pair</td>
</tr>
<tr>
<td>9</td>
<td>Device Registered Protocol Type #2</td>
<td>enum8</td>
<td>A protocol type value for the second registered Protocol/Type pair</td>
</tr>
<tr>
<td>10</td>
<td>Async Registered for Protocol Type #2</td>
<td>bool8</td>
<td>The Test Client has registered to receive asynchronous messages for the second Protocol/Type pair</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N*3+5</td>
<td>Device Registered Protocol #N</td>
<td>enum8</td>
<td>A protocol value for the final registered Protocol/Type pair</td>
</tr>
<tr>
<td>N*3+6</td>
<td>Device Registered Protocol Type #N</td>
<td>enum8</td>
<td>A protocol type value for the final registered Protocol/Type pair</td>
</tr>
<tr>
<td>N*3+7</td>
<td>Async Registered for Protocol Type #N</td>
<td>bool8</td>
<td>The Test Client has registered to receive asynchronous messages for the final Protocol/Type pair</td>
</tr>
</tbody>
</table>

**10.2.6 Query System Inventory (0x12)**

The `Query System Inventory` message retrieves an inventory of the hardware and firmware present in the system exposed by the test service. The resulting inventory is supplied in JSON format per the `SystemInventory` JSON schema. A mockup and the schema may be found in Annex A.

The inventory returned by the Test Service may be constructed via both standard and vendor proprietary methods. For example, the `Protocol` information may be supplied by the cached results of an MCTP `Get Message Type Support` command, but the `PCI-ID` information would typically be obtained outside of PMCI protocol messages. As each platform and Control Plane are unique, the methods for constructing a System Inventory must also be unique to that platform and Control Plane.
The fields in the system inventory are as follows:

- **Schema definition**: A link to the SystemInventory schema
- **ControlPlane**: An object containing information about the control plane (such as a management controller) with which the test service interacts to provide test functionality
  - **Manufacturer**: The manufacturer of the control plane
  - **Model**: The model of the control plane (if applicable)
  - **Firmware versions**: An array of named firmware associated with the control plane and/or system environment. It is recommended that the "Name" of the firmware component be consistent with the Package Classification Type from the ComponentClassification values table in PLDM For Firmware Update whenever possible.
  - **Device Identifier**: The code identifying the control plane for the system under test. This is usually 0.
  - **Interfaces**: An array of the hardware and software interfaces the control plane supports for communicating with devices, as well as an indication of whether devices can initiate messages on them
- **Devices**: A list of the devices in the system that can be reached for test purposes. For each of these devices:
  - **Manufacturer**: The manufacturer of the device
  - **Location**: The physical location of the device, such as a particular slot in the chassis
  - **Device Identifier**: A code that can be used to test against the specific device when the test client doesn't care which interface messages are routed over
  - **UniqueID**: A unique value that can be used to identify a device that does not change over the lifetime of the device. The UniqueID can be MCTP Endpoint UUID, NC-SI Package UUID or any other unique identifier such as a serial number. The format of the UniqueID is outside the scope of this specification.
  - **PCI-ID**: The four-part PCI device identifier for the device:
    - **DID**: The device identifier
    - **SDID**: The sub-device identifier
    - **VID**: The vendor identifier
    - **SVID**: The sub-vendor identifier
  - **Firmware versions**: An array of named firmware associated with the device
  - **Interfaces**: An array of interfaces with which the MC can reach the device. For each interface:
    - **Name**: The name of the interface
    - **Interface Identifier**: An identifier that can be used as part of a compound sequence to build up a specific path to a device to force testing on specific paths and interfaces via the Configure Device Under Test message
    - **Parent Device Identifier**: The Device Identifier for the bridge or control plane to which the interface is attached. Zero for the main management controller/control plane.
  - **Protocol support**: A list of the protocols for which the device advertises support:
    - **Protocol**: The name of the protocol
    - **Types**: An array of sub-types of the protocol supported by the device:
      - **Type**: The numeric sub-type value
      - **Name**: The name of the protocol sub-type, such as PLDM for Platform Monitoring and Control
• **Versions**: A list of specific specification versions the device supports for the sub-type
• **NC-SIInfo**: Additional information if the Protocol is NC-SI
• **ChannelID**: Network Controller Channel ID to address the Network Controller Channel
• **Pass-through**: The status of the channel to allow Pass-through packets

**Admin Query System Inventory Message request format**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x12 - See Command Codes</td>
</tr>
</tbody>
</table>

**Admin Query System Inventory Message response format**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x12 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Response Code</td>
<td>enum8</td>
<td>See Test Service Response Codes</td>
</tr>
<tr>
<td>2</td>
<td>System Inventory</td>
<td>uint8</td>
<td>UTF-8 formatted JSON SystemInventory data</td>
</tr>
</tbody>
</table>

**10.2.7 Configure Test Service (0x20)**

The **Configure Test Service** shall be used by the test client to configure the capabilities of the test service. A Test Client must issue the **Query Capabilities** message after every successful **Connect** message. The Test Client can configure the Test Service after determining the capabilities of the Test Service from the response of **Query Capabilities**. Test client can issue **Query Capabilities after Configure Test Service** to ensure the capability is configured as expected. The test client might configure the test service before issuing the **Configure Device Under Test**, after which the test service can reject the **Configure Test Service** request from the test client.

**Admin Configure Test Service Message request format**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x20 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Number of Capabilities</td>
<td>uint16</td>
<td>Count of capabilities in this structure</td>
</tr>
<tr>
<td>3</td>
<td>First Capability ID</td>
<td>uint16</td>
<td>From the Table of Capability IDs</td>
</tr>
<tr>
<td>5</td>
<td>First Capability Value</td>
<td>uint32</td>
<td>Value for the first Capability</td>
</tr>
<tr>
<td>9</td>
<td>Second Capability ID</td>
<td>uint16</td>
<td>From the Table of Capability IDs</td>
</tr>
<tr>
<td>11</td>
<td>Second Capability Value</td>
<td>uint32</td>
<td>Value for the second Capability</td>
</tr>
</tbody>
</table>
170 **Admin Configure Test Service Message response format**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x20 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Response Code</td>
<td>enum8</td>
<td>See Test Service Response Codes</td>
</tr>
</tbody>
</table>

171 **10.2.8 Configure Device Under Test (0x21)**

The *Configure Device Under Test* message enables the test client to configure the Test Service to communicate with a device it wishes to perform testing against. This may either be via the Device Identifier from the System Inventory, if the test does not require a certain path or interface to reach the device; or it may be via a sequence of Device Identifiers and Interface Identifiers that specify the exact sequence of interfaces to use to reach it.

If the Target Identifier is a Device Identifier and the Identifier Count is 0, the Test Service determines the path to reach the device and the interface to use. If the Target Identifier is an Interface Identifier and the Identifier Count is 0, the Test Service will determine a path that uses the given interface to connect to the associated device.

In the case where an Identifier List is provided and the Identifier Count is greater than 0, the test service will verify that the path is valid. If the given path is invalid, the Test Service shall send a response with the Response Code set to **INVALID_PATH**. The DUT Connection IDs, Device Identifiers, and Interface Identifiers all exist in the same name-space, removing any ambiguities between them.

In all cases other than for invalid paths, the Test Service returns a Configure Device Under Test Response with the newly created DUT connection ID and the path chosen to reach the DUT. Using the DUT Connection ID in future messages guarantees the returned path will be used. To facilitate debugging in the invalid path case, the response Identifier List can contain the Interface Identifiers up to, but not including the first invalid connection.

176 **Admin Configure Device Under Test Message request format**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x21 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Target Identifier</td>
<td>uint32</td>
<td>The device or interface identifier for the device being configured</td>
</tr>
<tr>
<td>5</td>
<td>Identifier Count</td>
<td>uint8</td>
<td>The number of identifiers supplied in the remainder of this message</td>
</tr>
<tr>
<td>6</td>
<td>Identifier List</td>
<td>uint32 * N</td>
<td>The sequence of interface identifier(s) to the device under test if the Identifier Count is 0</td>
</tr>
</tbody>
</table>
### Admin Configure Device Under Test Message response format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x21 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Response Code</td>
<td>enum8</td>
<td>See Test Service Response Codes and the below specific codes. 0x90: INVALID_PATH - Path given in Interface List of Configure DUT request is invalid</td>
</tr>
<tr>
<td>2</td>
<td>DUT Connection ID</td>
<td>uint32</td>
<td>The DUT Connection ID to use with subsequent messages</td>
</tr>
<tr>
<td>6</td>
<td>Identifier Count</td>
<td>uint8</td>
<td>The number of interface identifiers in the path to the DUT</td>
</tr>
<tr>
<td>7</td>
<td>Identifier List</td>
<td>uint32 * N</td>
<td>The sequence of interface identifiers in the path to the DUT</td>
</tr>
</tbody>
</table>

### 10.2.9 Configure Device Under Test Examples

The figure below is an examples of a topology containing a control plane, two bridges, and three devices. Explanations of some example calls to Configure Device Under Test follow the figure below.

```
+----------+          +-----------+
|          |          | Device    |
|          |          | Dev ID 02 |
| Control  |          | IfcID 0C  |
| Plane    |          |           |
| Dev ID 00|          | IfcID 0F  |
|          | IfcID 0A+----------------+----------+
|          | +---------------------+          |
|          | | Device              | |
|          | | Dev ID 04           | |
|          | | IfcID 0D             | |
|          | +---------------------+          |
| IfcID 0B+----------------+IfcID 10 |
|                     | | IfcID 0E             |
|                     | +---------------------+          |
| IfcID 13+----------+IfcID 11 |
| | Device             | |
| | Dev ID 05           | |
| | IfcID 12------------|
| +---------------------+
```
10.2.9.1 Example 1

- Client sends **ConfigureDUTRequest** (TargetIdentifier: 02, IdentifierCount: 0, IdentifierList: [])
- Path: Test Service picks only available path via Interface 0A to Interface 0C
- DUT Connection ID: Test Service picks available Identifier 20
- Test Service sends **ConfigureDUTResponse** (ResponseCode: 0<SUCCESS>, DUTConnectionID: 20, IdentifierCount: 2, IdentifierList: [0A, 0C])

10.2.9.2 Example 2

- Client sends **ConfigureDUTRequest** (TargetIdentifier: 04, IdentifierCount: 0, IdentifierList: [])
- Path: Test Service has two choices of paths, and must choose any valid path. For example, Interface 0A to Interface 0F.
- DUT Connection ID: Test Service picks available Identifier 21
- Test Service sends **ConfigureDUTResponse** (ResponseCode: 0<SUCCESS>, DUTConnectionID: 21, IdentifierCount: 2, IdentifierList: [0A, 0F])

10.2.9.3 Example 3

- Client sends **ConfigureDUTRequest** (TargetIdentifier: 10, IdentifierCount: 3, IdentifierList: [0B, 0D, 0E])
- Path: Test Service determines the given path is valid.
- DUT Connection ID: Test Service picks available Identifier 22
- Test Service sends **ConfigureDUTResponse** (ResponseCode: 0<SUCCESS>, DUTConnectionID: 22, IdentifierCount: 4, IdentifierList: [0B, 0D, 0E, 10])

10.2.9.4 Example 4

- Client sends **ConfigureDUTRequest** (TargetIdentifier: 13, IdentifierCount: 5, IdentifierList: [0A, 0D, 0E, 12, 11])
- Path: Test Service determines the given path is invalid, noting the last good interface identifier along the path is 0A.
- DUT Connection ID: Test Service does not create a DUT Connection ID
- Test Service sends **ConfigureDUTResponse** (ResponseCode: 90<INVALID_PATH>, DUTConnectionID: 00, IdentifierCount: 1, IdentifierList: [0A])

10.2.9.5 Example 5

- Client sends **ConfigureDUTRequest** (TargetIdentifier: 13, IdentifierCount: 0, IdentifierList: [])
- Path: Test Service picks only available path to interface 13 of device 06.
• DUT Connection ID: Test Service picks available Identifier 23
• Test Service sends ConfigureDUTResponse(ResponseCode: 0<SUCCESS>, DUTConnectionID: 23, IdentifierCount: 6, IdentifierList: [0B, 0D, 0E, 12, 11, 13])

10.2.10 Register to Protocol (0x22)

The Register to Protocol message enables the test client to specify the particular device(s), protocol(s), and type(s) it plans to test against. This is a hook by which the test service may deny permission to a test client for protocol access. A response of success grants permission to access the protocol/type(s) on the requested device. If the device is not available on the requested protocol or the path given in the Configure Device DUT message does not support it, this message shall fail.

The Register to Protocol message includes a field to specify types of messages within a Protocol Type. For example, there are multiple types of PLDM messages (Platform Monitoring and Control, Redfish Device Enablement, etc.) defined in DSP0245, and a test client may register for any combination of these PLDM types. NC-SI and SPDM do not have types within their protocols, and so the Type Count would be set to 0 for these protocols.

This message only sets up communication from the Test Client to the DUT (via the Test Service). In order for the Test Client to receive asynchronous communication originating from the DUT, the Test Client must use the Register Async Message Recipient message.

Admin Register to Protocol Message request format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x22 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Protocol Type</td>
<td>enum8</td>
<td>The protocol the test client wishes to test against</td>
</tr>
<tr>
<td>2</td>
<td>DUT Connection ID</td>
<td>uint32</td>
<td>The DUT the test client is requesting to test against</td>
</tr>
<tr>
<td>6</td>
<td>Type Count</td>
<td>uint8</td>
<td>The number of types within the protocol the client wishes to test against (0: if no types in protocol)</td>
</tr>
<tr>
<td>7</td>
<td>Types</td>
<td>uint8 * N</td>
<td>The specific individual types within the Protocol Type (for example, Redfish Device Enablement)</td>
</tr>
</tbody>
</table>

Admin Register to Protocol Message response format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x22 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Response Code</td>
<td>enum8</td>
<td>See Test Service Response Codes</td>
</tr>
<tr>
<td>2</td>
<td>DUT Connection ID</td>
<td>uint32</td>
<td>The DUT that the test client requested to test against</td>
</tr>
</tbody>
</table>
10.2.11 Register Async Message Recipient (0x23)

The Register Async Message Recipient message informs the test service that the test client wishes to receive asynchronous messages caused by a particular device. The test service shall forward asynchronous messages for registered types to the test client for processing. Except in matters of physical safety of the hardware, the test service shall not process forwarded messages itself.

The Test Service shall not permit the Test Client to register for asynchronous messages from a device if the Test Client has not previously registered to send synchronous (Test Client-initiated) messages to that device using the Register To Protocol message.

Admin Register Async Message Recipient Message request format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x23 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Protocol Type</td>
<td>enum8</td>
<td>The protocol the test client wishes to test against</td>
</tr>
<tr>
<td>2</td>
<td>DUT Connection ID</td>
<td>uint32</td>
<td>The DUT for which the test client requests to receive asynchronous events</td>
</tr>
<tr>
<td>6</td>
<td>Type Count</td>
<td>uint8</td>
<td>The number of types within the protocol the client wishes to receive asynchronously (0 if no types in protocol)</td>
</tr>
<tr>
<td>7</td>
<td>Types</td>
<td>uint8 * N</td>
<td>The specific individual types within the Protocol Type (for example, Platform Monitoring and Control - see discussion in Register To Protocol)</td>
</tr>
</tbody>
</table>

Admin Register Async Message Recipient Message response format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x23 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Response Code</td>
<td>enum8</td>
<td>See Test Service Response Codes</td>
</tr>
<tr>
<td>2</td>
<td>DUT Connection ID</td>
<td>uint32</td>
<td>The DUT for which the test client requested to receive asynchronous events</td>
</tr>
</tbody>
</table>

10.2.12 Log Event (0x30)

The Log Event message is used by a Test Agent to log event messages to its peer Test Agent. The receiving Test Agent may, at its discretion, ignore the Log Data. This may be desirable in the case that the receiving Test Agent does not have the space to store the Log Data.

Log Event request message format
<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x30 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Reason Code</td>
<td>enum8</td>
<td>The reason for the Log Event</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{ CorruptMessage = 1, ClientTimeout = 2, WatchdogTimeout = 3, OemReserved = 0xF0..FF }</td>
</tr>
<tr>
<td>2</td>
<td>Log Data Format</td>
<td>enum8</td>
<td>The format for the Log Data (see DSP0267 String Type values)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{ Binary = 0, ASCII = 1, UTF8 = 2, UTF-16 = 3, UTF-16LE = 4, UTF16-BE = 5 }</td>
</tr>
<tr>
<td>3</td>
<td>Log Data Length</td>
<td>uint32</td>
<td>Length of the Log Data</td>
</tr>
<tr>
<td>7</td>
<td>Log Data</td>
<td>variable</td>
<td>Log Data</td>
</tr>
</tbody>
</table>

199 Log Event response message format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command Code</td>
<td>enum8</td>
<td>0x30 - See Command Codes</td>
</tr>
<tr>
<td>1</td>
<td>Response Code</td>
<td>enum8</td>
<td>See Test Service Response Codes</td>
</tr>
</tbody>
</table>

200 10.3 Vendor Defined Admin

202 PTTI Vendor Defined Admin messages are used when a tester wants to send custom messages between their Test Client and Test Service. In contrast, MCTP Vendor Defined Messages are used when a tester wants to send custom messages between their Test Client and DUT. MCTP Vendor Defined Messages are sent via Test Request and Response Messages with the appropriate Protocol Type from DSP0239, and not via the Vendor Defined Admin mechanism.

203 PTTI Vendor Defined Admin messages are identified using the Protocol Type of the Test Service Wrapper. The IANA Enterprise ID of the requester and responder must be provided as the first 4 bytes.

204 A PTTI Vendor Defined Admin message may not be sent when a standard PTTI Admin message is outstanding, and a standard PTTI Admin message may not be sent when a PTTI Vendor Defined Admin message is outstanding. In such cases, the Test Service shall respond with an OUTSTANDING MESSAGE Test Service Response Code.

205 Vendor Defined Admin request message format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IANA Enterprise ID</td>
<td>uint32</td>
<td>IANA Enterprise ID for Vendor. MSB first. This is formatted per the Vendor Data Field for IANA enterprise vendor ID in the MCTP Base Specification.</td>
</tr>
<tr>
<td>4+</td>
<td>Vendor Defined Data</td>
<td>Variable</td>
<td>Data for Vendor Defined Admin request message</td>
</tr>
</tbody>
</table>
**Vendor Defined Admin response message format**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IANA Enterprise ID</td>
<td>uint32</td>
<td>IANA Enterprise ID for Vendor MSB first. This is formatted per the Vendor Data Field for IANA enterprise vendor ID in the MCTP Base Specification.</td>
</tr>
<tr>
<td>4</td>
<td>Response Code</td>
<td>enum8</td>
<td>See Test Service Response Codes</td>
</tr>
<tr>
<td>5+</td>
<td>Vendor Defined Data</td>
<td>Variable</td>
<td>Data for Vendor Defined Admin request message</td>
</tr>
</tbody>
</table>

### 10.4 Test Messages

Test Messages instruct the Test Service to send a message, of the protocol specified in the Protocol Type field of the Test Service Wrapper, to the DUT using the connection specified in the DUT Connection ID. Test Clients must correctly set the Test Service Wrapper Flags to indicate that the Test Message Request originates with the Test Client (TC → TS Request). Likewise, the Test Service must set the flags indicating that the Response originates from the Test Service (TS → TC Response).

When constructing a PTTI Test Message, the Test Client should use the Protocol Type value defined in DSP0239, even when the underlying transport layer is not MCTP. MCTP Type 0 (MCTP Control) and Type 3 (NC-SI Pass-through) are not supported by this specification, and it is recommended that the Test Service reject such requests with the **INVALID_PROTOCOL** message response code.

If the Test Service detects that a second Test Message for the same Protocol Type has been received for a DUT before the Test Service has responded to the first Test Protocol message for the same Protocol Type and DUT, the Test Service may respond with the **OUTSTANDING_MESSAGE** Message Response Code and not forward the Test Message to the DUT.

### 10.4.1 Device-originated (Async) Protocol Messages

Certain upper-layer messages, such as AEN packets in NC-SI and the PlatformEventMessage message in PLDM for Platform Monitoring and Control, may asynchronously originate from the DUT and flow to the Control Plane. If the Test Client wishes to receive these messages, they may register this request using the **Register Async Message Recipient** message to the Test Service.

When the Test Service receives a message originating from the device that matches the forwarding criteria established by the Test Client, the Test Service will send a Test Message to the Test Client with the correct Protocol Type. The Test Service must take care that the Test Service Wrapper Flags are set such that Request originates from.
the Test Service (TS → TC Request). Likewise, the Test Client must set the flags indicating that the Response originates from the Test Client (TC → TS Response).

10.5 Test Request and Response Messages

Test Message request format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DUT Connection ID</td>
<td>uint32</td>
<td>DUT Connection ID returned by Configure Device Under Test message</td>
</tr>
<tr>
<td>4</td>
<td>Maximum Wait Time</td>
<td>uint32</td>
<td>The maximum time that the Test Service should wait, after transmitting the last byte of this request, before returning a TIMEOUT message response code. For protocols such as PLDM, this may be a fixed value for all messages. For certain SPDM messages, this value may vary based on negotiation between the Test Client and the DUT.</td>
</tr>
<tr>
<td>8</td>
<td>Upper-layer Protocol Request Message</td>
<td>Variable</td>
<td>A fully-formed Upper-layer Protocol message such as a complete PLDM, NC-SI Control, or SPDM message.</td>
</tr>
</tbody>
</table>

Test Message response format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Response Code</td>
<td>enum8</td>
<td>See Test Service Response Codes</td>
</tr>
<tr>
<td>1</td>
<td>DUT Connection ID</td>
<td>uint32</td>
<td>DUT Connection ID returned by Configure Device Under Test message</td>
</tr>
<tr>
<td>5</td>
<td>Elapsed Time (μs)</td>
<td>uint32</td>
<td>The elapsed time from when the Test Service sent the last byte of the test message request to when the Test Service received the first byte of the response.</td>
</tr>
<tr>
<td>9</td>
<td>Upper-layer Protocol Response Message</td>
<td>Variable</td>
<td>If the Response Code is SUCCESS, the response message from the DUT shall be in this field. If the Response Code is not SUCCESS, this field shall have zero length.</td>
</tr>
</tbody>
</table>
10.6 Handling MCTP Packets

To build one or more MCTP packets from a Test Request message, the Test Service (and the Control Plane hosting the Test Service) may take the following steps:

1. The Physical Medium-Specific Header and Trailer will typically be created by Control Plane logic that manages the appropriate physical interface. It may be necessary for the Test Service to look up the physical interface using the DUT Connection ID, and the Test Service may reject any unregistered requests with a PRIVILEGE_ERROR Message Response Code.

2. The MCTP Header version, Start Of Message, End Of Message, Packet Sequence #, Tag Owner, and Message tag fields shall be filled out as appropriate (for example, for a first/middle/last packet in a message). Packetization of Upper-layer Protocol messages is described in DSP0236, and it may be necessary for the Test Service or Control Plane to build multiple MCTP packets to handle a single Test request.

3. The Destination endpoint ID shall be obtained by using the DUT Connection ID to look up the DUT's Destination endpoint ID.

4. The Source endpoint ID shall be the Control Plane's MCTP EID.

5. The MCTP Message type may be obtained directly from the Test Service Wrapper's Protocol Type field.

6. The remainder of the MCTP packet payload is contained within the Upper-layer Protocol Request Message of the Test request.

7. Once the entire MCTP message is constructed and sent to the DUT, the Test Service must start a timer to capture the elapsed request-to-response time in a Test response. If the timer exceeds the Maximum Wait Time from the Test Client, the Test Service shall respond with a TIMEOUT Message Response Code.

Likewise, the Test Service may handle an MCTP message received from the DUT in this way:

1. If the Test Service receives a message that cannot be interpreted or correlated to a request message, the Test Service should send a Log Event message with Reason Code CorruptMessage to the Test Client with relevant information, and then continue waiting for a response if appropriate. It is possible
that the Test Service will eventually exceed the Maximum Wait Time in this scenario, in which case a `TIMEOUT` Message Response code should be sent.

2. The control plane should check the Physical Medium-Specific Header and Trailer for any errors. If one is found, but it is possible to correlate the message to a request message, it is appropriate to send a `PHYSICAL_ERROR` message response code to the Test Client.

3. The control plane or Test Service should check the MCTP Header version, Destination endpoint ID, Start Of Message, End Of Message, Packet Sequence #, Tag Owner, and Message tag fields for accuracy. If any errors are found, but it is possible to correlate the message to a request message, it is appropriate to send a `TRANSPORT_ERROR` message response code to the Test Client. It may be necessary to wait for multiple MCTP packets before the MCTP message is ready.

4. Use the Source endpoint ID, Message type, and Message type-specific Header fields to discover whether this is a response to an outstanding Test request. If so, a Test response should be constructed with the `TS → TC Response` bits set in the Test Service Wrapper. If the Test Service receives an asynchronous MCTP message, and the Test Client has registered to receive async messages from the DUT, then a new Test request should be constructed with the `TS → TC Request` bits set.

5. When constructing either a Test response or an async Test request to the Test Client, the Protocol Type (in the Test Service Wrapper) shall be set to the MCTP Message type from the MCTP packet payload, the DUT Connection ID shall be set to the appropriate value for the device and physical interface, and the Upper-layer Protocol Response Message shall be all bytes of the MCTP packet payload, excluding the MCTP Message type byte.

6. For Test response messages, the Elapsed Time shall be filled in using the timer that was started during the Test request.

### 10.7 Handling NC-SI over RBT Packets

<table>
<thead>
<tr>
<th>Bytes</th>
<th>31..24</th>
<th>23..16</th>
<th>15..08</th>
<th>07..00</th>
</tr>
</thead>
<tbody>
<tr>
<td>00..03</td>
<td>MC ID</td>
<td>Header Revision</td>
<td>Reserved</td>
<td>IID</td>
</tr>
<tr>
<td>04..07</td>
<td>Control Packet Type</td>
<td>Ch. ID</td>
<td>Flags</td>
<td>Payload Length</td>
</tr>
<tr>
<td>08..11</td>
<td></td>
<td></td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>12..15</td>
<td></td>
<td></td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

To build an NC-SI packet from a Test Request message, the Test Service (and the Control Plane hosting the Test Service) may take the following steps (refer to DSP0222). Implementers are reminded that NC-SI is a Big Endian protocol and that the bytes column of the preceding diagram represents the order of transmission.

1. The Test Service shall ensure that the Protocol type is set to NC-SI Control, even though this message will not be going over MCTP. If the Protocol type is not set to NC-SI control, the Test Service shall return an `INVALID_PROTOCOL` message response code to the Test Client.
2. The Ethernet frame header is populated by the Control Plane.

3. The NC-SI Control Packet Header must be included as part of the Upper-layer Protocol Request Message coming from the Test Client. However, the Control Plane and Test Service may validate or re-write certain fields as follows:
   i. The Management Controller ID shall be filled in by the Test Service using the DUT Connection ID. Typically, this value will be 0x00.
   ii. The Header Revision must be set to 0x01 by the Test Service.
   iii. The Instance ID must be re-written by the Control Plane or Test Service to ensure that it is monotonically increasing, as required in DSP0222. The Instance ID that was requested by the Test Client must be remembered so that it may be re-written into the Test response. It is possible that even Instance IDs sent from the Test Service will not be monotonically increasing, since the Control Plane may need to send its own NC-SI messages for critical operations such as thermal monitoring, and therefore the Test Service must maintain a bi-directional mapping of the requested Instance ID and the sent/received Instance ID.
   iv. The Control Packet Type, Flags, and Payload Length is provided by the Test Client as part of the Upper-layer Protocol Request.
   v. The Channel ID is provided by the Test Client as part of the Upper-layer Protocol Request, based on the NC-SIInfo section of the System Inventory. Implementers are encouraged to review the NC-SI Considerations section of this document related to the Channel ID.

4. The Control Packet payload will follow the Control Packet Header in the Upper-layer Protocol Request from the Test Client, and may be copied directly into the NC-SI over RBT packet. However, the Test Service or Control Plane may have to modify the Payload Pad as required by DSP0222.

5. Once the entire RBT message is constructed and sent to the DUT, the Test Service must start a timer to capture the elapsed request-to-response time in a Test response. If the timer exceeds the Maximum Wait Time from the Test Client, the Test Service shall respond with a TIMEOUT Message Response Code.

Likewise, the Test Service may handle an NC-SI Response packet or AEN packet received from the DUT in this way:

1. If the Test Service receives an NC-SI packet that cannot be interpreted as an AEN packet or a Response packet correlated to a Request message, the Test Service should send a Log Event message with Reason Code CorruptMessage to the Test Client with relevant information, and then continue waiting for a response if appropriate. It is possible that the Test Service will eventually exceed the Maximum Wait Time in this scenario, in which case a TIMEOUT Message Response code should be sent.

2. If the packet is valid, the Ethernet frame header shall be removed by the Test Service or Control Plane. If the packet is not valid, it is appropriate to send a TRANSPORT_ERROR message response code to the Test Client.

3. The Test Service shall construct a Test Response message with:
   i. The Test Service Wrapper with the Protocol type set to NC-SI Control (even though this was not an MCTP message) and the TS → TC Response bits set.
ii. The DUT Connection ID appropriate value for the device and physical interface.

iii. For NC-SI Response packets, the Elapsed Time shall be filled in using the timer that was started during the Test request.

iv. The appropriate Message response code (typically, **SUCCESS** if a valid NC-SI message was received).

v. For NC-SI Response packets, the requested Instance ID from the Test Client shall replace the received Instance ID from the DUT so that the Test Client may match up the request and response NC-SI messages. For AEN packets, the Instance ID field is set to 0.

vi. The Control Packet header (with the updated Instance ID), followed by the unmodified Response packet payload, as the Upper-layer Protocol Response Message.
11 ANNEX A SystemInventory Example (informative) and Schema (normative)

11.1 SystemInventory Example

The following is an example of System Inventory that might be returned from the GetSystemInventory message:

```json
{
    "SchemaDefinition": "SystemInventory.v1_0_0",
    "ControlPlane": {
        "Manufacturer": "Contoso",
        "Model": "ContoBMC",
        "FirmwareVersions": [
            {
                "Name": "name1",
                "Version": "version1"
            },
            {
                "Name": "name2",
                "Version": "version2"
            },
            {
                "Name": "name3",
                "Version": "version3"
            }
        ],
        "Interfaces": [
            {
                "Interface": "I2C",
                "MessageInitiationSupport": "ControlPlaneRequestorOnly"
            },
            {
                "Interface": "PCIeVDM",
                "MessageInitiationSupport": "AnyRequestor"
            },
            {
                "Interface": "RBT",
                "MessageInitiationSupport": "ControlPlaneRequestorOnly"
            }
        ],
        "Devices": [
            {
                "Manufacturer": "ContosoAdapters",
                "Location": "Slot 3"
            }
        ]
    }
}
```
"GeneralDeviceIdentifier": 3180,
"PCI-ID": {
    "DID": "0x1234",
    "SDID": "0x5678",
    "VID": "0x9ABC",
    "SVID": "0xDEF0"
},
"FirmwareVersions": [
    {
        "Name": "Management",
        "Version": "2.7.3"
    },
    {
        "Name": "Ethernet",
        "Version": "4.8.1"
    },
    {
        "Name": "Security",
        "Version": "1.3.7a"
    }
],
"Interfaces": [
    {
        "Interface": "I2C",
        "InterfaceIdentifier": 3187,
        "ParentDeviceIdentifier": 0,
        "ProtocolSupport": {
            "Protocol": "MCTP",
            "Types": {
                "Type": 0,
                "Name": "MCTP Base",
                "Versions": ["1.2.0"
            ]
        },
        {
            "Type": 1,
            "Name": "PLDM over MCTP",
            "Versions": ["1.0.0"
        ]
    },
    {
        "Type": 2,
        "Name": "NC-SI over MCTP",
        "Versions": ["1.0.0"
    ]
]


```
},
{
  "Type": 3,
  "Name": "Ethernet over MCTP",
  "Versions": [
    "1.0.0"
  ]
},
{
  "Type": 4,
  "Name": "NVM Express Management Messages over MCTP",
  "Versions": [
    "1.0.0"
  ]
},
{
  "Type": 5,
  "Name": "SPDM over MCTP",
  "Versions": [
    "1.0.0"
  ]
},
{
  "Type": 126,
  "Name": "Vendor Defined - PCI",
  "Versions": [
    "1.0.0"
  ]
},
{
  "Type": 127,
  "Name": "Vendor Defined - IANA",
  "Versions": [
    "1.0.0"
  ]
}
],
{
  "Protocol": "PLDM",
  "Types": [
    {
      "Type": 0,
      "Name": "PLDM Base",
      "Versions": [
        "1.0.0"
      ]
    },
    {
      "Type": 2,
      "Name": "...
```
"Name": "PLDM for Platform Monitoring and Control",
"Version": [
  "1.2.0"
]
],

"Interface": "PCIeVDM",
"InterfaceIdentifier": 3188,
"ParentDeviceIdentifier": 0,
"ProtocolSupport": [
  {
    "Protocol": "MCTP",
    "Types": [
      {
        "Type": 0,
        "Name": "MCTP Base",
        "Version": [
          "1.2.0"
        ]
      },
      {
        "Type": 1,
        "Name": "PLDM over MCTP",
        "Version": [
          "1.0.0"
        ]
      }
    ]
  },
  {
    "Protocol": "PLDM",
    "Types": [
      {
        "Type": 0,
        "Name": "PLDM Base",
        "Version": [
          "1.0.0"
        ]
      },
      {
        "Type": 2,
        "Name": "PLDM for Platform Monitoring and Control",
        "Version": [
          "1.2.0"
        ]
      }
    ]
  }
]
{
  "Type": 5,
  "Name": "PLDM for Firmware Update",
  "Versions": [
    "1.1.0"
  ]
},
{
  "Type": 6,
  "Name": "PLDM for Redfish Device Enablement",
  "Versions": [
    "1.0.1",
    "1.1.0"
  ]
}
{
  "Interface": "RBT",
  "InterfaceIdentifier": 3189,
  "ParentDeviceIdentifier": 0,
  "ProtocolSupport": [
    {
      "Protocol": "NC-SI",
      "Types": [
        {
          "Type": 0,
          "Name": "NC-SI",
          "Versions": [
            "1.1.1"
          ]
        }
      ],
      "NC-SIInfo": [
        {
          "ChannelID": "0x01",
          "Pass-through": "Enabled"
        },
        {
          "ChannelID": "0x02",
          "Pass-through": "Disabled"
        }
      ]
    }
  ]
}
{  
    "Manufacturer": "ContosoBridge",
    "Location": "Slot 5",
    "GeneralDeviceIdentifier": 6450,
    "PCI-ID": {  
        "DID": "0x0867",
        "SDID": "0x5309",
        "VID": "0x9ABC",
        "SVID": "0xDEF0"
    },
    "FirmwareVersions": [
        {  
            "Management": "1.6a"
        }
    ],
    "Interfaces": [  
        {  
            "Interface": "I2C",
            "PathDeviceIdentifier": 6457,
            "ParentDeviceIdentifier": 0,
            "ProtocolSupport": [  
                {  
                    "Protocol": "MCTP",
                    "Types": [
                        {  
                            "Type": 0,
                            "Name": "MCTP Base",
                            "Versions": ["1.2.0"]
                        },  
                        {  
                            "Type": 1,
                            "Name": "PLDM over MCTP",
                            "Versions": ["1.0.0"]
                        }
                    ]
                },  
                {  
                    "Protocol": "PLDM",
                    "Types": [
                        {  
                            "Type": 0,
                            "Name": "PLDM Base",
                            "Versions": ["1.0.0"]
                        }
                    ]
                }
            ]
        }
    ]
}
"Interface": "PCIeVDM",
"PathDeviceIdentifier": 6458,
"ParentDeviceIdentifier": 0,
"ProtocolSupport": [
  "Protocol": "MCTP",
  "Types": [
    {"Type": 0,
     "Name": "MCTP Base",
     "Versions": ["1.2.0"]
  },
  {"Type": 1,
   "Name": "PLDM over MCTP",
   "Versions": ["1.0.0"]
  }
]
},
{"Protocol": "PLDM",
"Types": [
  {"Type": 0,
   "Name": "PLDM Base",
   "Versions": ["1.0.0"]
  }
]
},
{"Manufacturer": "ContosoBackendDrive",
"Location": "Bridge Slot 1",
"GeneralDeviceIdentifier": 1240,
"UniqueID": "0x45237789056AB781",
"PCI-ID": {"DSP0280 PMCI Test Tools Interface and Design Specification
Version 1.0.0
Published
51"}
"DID": "0x2468",
"SDID": "0x1357",
"VID": "0x9ABC",
"SVID": "0xDEF0"
},
"FirmwareVersions": [
  {
    "Management": "1.0.1",
    "NVMe": "12.3.45"
  }
],
"Interfaces": [
  {
    "Interface": "I2C",
    "PathDeviceIdentifier": 1247,
    "ParentDeviceIdentifier": 6450,
    "ProtocolSupport": [
      {
        "Protocol": "MCTP",
        "Types": [
          {
            "Type": 0,
            "Name": "MCTP Base",
            "Versions": [
              "1.2.0"
            ]
          },
          {
            "Type": 1,
            "Name": "PLDM over MCTP",
            "Versions": [
              "1.0.0"
            ]
          }
        ]
      }
    ],
    "Protocol": "PLDM",
    "Types": [
      {
        "Type": 0,
        "Name": "PLDM Base",
        "Versions": [
          "1.0.0"
        ]
      }
    ]
  }
]
{  
  "Interface": "PCIeVDM",
  "PathDeviceIdentifier": 1248,
  "ParentDeviceIdentifier": 6450,
  "ProtocolSupport": [
    
    "Protocol": "MCTP",
    "Types": [
      
      "Type": 0,
      "Name": "MCTP Base",
      "Versions": [
        "1.2.0"
      ]
    },
    
    "Type": 1,
    "Name": "PLDM over MCTP",
    "Versions": [
      "1.0.0"
    ]
  ]
  
  },
  
  "Protocol": "PLDM",
  "Types": [
    
    "Type": 0,
    "Name": "PLDM Base",
    "Versions": [
      "1.0.0"
    ]
  },
  
    "Type": 5,
    "Name": "PLDM for Firmware Update",
    "Versions": [
      "1.1.0"
    ]
  ]
}
11.2 SystemInventory Schema

The following Schema dictates the contents of the System Inventory document returned from the GetSystemInventory message:

```
{
    "$id": "http://pmci.dmtf.org/tools/SystemInventory.v1_0_0.json",
    "$schema": "http://json-schema.org/draft-07/schema#",
    "description": "A system-level representation of the hardware devices and their connections."
    "copyright": "Copyright 2021 DMTF. For the full DMTF copyright policy, see http://www.dmtf.org/about/policies/copyright",
    "type": "object",
    "additionalProperties": false,
    "properties": {
        "SchemaDefinition": {
            "type": "string",
            "description": "The JSON schema that defines this SystemInventory document and can be used to validate its contents."
        },
        "ControlPlane": {
            "type": "object",
            "description": "Information on the Control Plane with which the test service interacts."
        },
        "Manufacturer": {
            "type": "string",
            "description": "The name of the organization that manufactures this Control Plane."
        },
        "Model": {
            "type": "string",
            "description": "The model name of this Control Plane."
        },
        "FirmwareVersions": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/FWVersion"
            }
        },
        "Interfaces": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/ControlPlaneInterface"
            }
        }
    }
}
```
"type": "array",
"items": {
   "$ref": "#/definitions/Device"
}
},
"definitions": {
   "Interface": {
      "type": "string",
      "enum": ["I2C", "I3C", "PCIeVDM", "RBT", "Serial", "Proprietary"],
      "enumDescriptions": {
         "I2C": "I2C or SMBus",
         "I3C": "I3C",
         "PCIeVDM": "PCIe supported via vendor defined messages",
         "RBT": "RBT/RMII used for NC-SI communications",
         "Serial": "Serial",
         "Proprietary": "Vendor-specific proprietary interface not otherwise listed here"
      },
      "description": "The name of the interface"
   },
   "Device": {
      "type": "object",
      "description": "Information about hardware device",
      "properties": {
         "Manufacturer": {
            "type": "string",
            "description": "The name of the organization that manufactures this Device."
         },
         "Location": {
            "type": "string",
            "description": "The physical location (slot, etc.) for this device."
         },
         "GeneralDeviceIdentifier": {
            "type": "number",
            "description": "A numeric identifier assigned by the testing service for this device."
         },
         "UniqueID": {
            "type": "string",
            "description": "An idempotent identifier unique to the device."
         },
         "PCI-ID": {
            "type": "object",
            "description": "The PCI identifier for this device",
            "properties": {
               "DID": {
                  "type": "string",
                  "description": "The four hexadecimal digit device identifier for this device, prefixed with 0x"
               },
               "SDID": {

```
"type": "string",
"description": "The four hexadecimal digit subdevice identifier for this device, prefixed with 0x"
},
"VID": {
  "type": "string",
  "description": "The four hexadecimal digit vendor identifier for this device, prefixed with 0x"
},
"SVID": {
  "type": "string",
  "description": "The four hexadecimal digit subvendor identifier for this device, prefixed with 0x"
}
},
"FirmwareVersions": {
  "type": "array",
  "items": {
    "$ref": "#/definitions/FWVersion"
  }
},
"Interfaces": {
  "type": "array",
  "items": {
    "$ref": "#/definitions/DeviceInterfaceInfo"
  }
},
"DeviceInterfaceInfo": {
  "type": "object",
  "description": "Information about a specific interface for a hardware device",
  "properties": {
    "Interface": {
      "$ref": "#/definitions/Interface"
    },
    "PathDeviceIdentifier": {
      "type": "number",
      "description": "A numeric identifier assigned by the testing service for this device as connected to via this interface."
    },
    "ParentDeviceIdentifier": {
      "type": "number",
      "description": "The testing service-assigned numeric identifier for the bridge to which this device is attached on this interface, or zero if the device is directly connected to the host system's Control Plane."
    },
    "ProtocolSupport": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/ProtocolInfo"
      }
    }
  }
}
"FWVersion": {
  "type": "object",
  "description": "Information about a firmware version for the Control Plane",
  "properties": {
    "Name": {
      "type": "string",
      "description": "The name of the firmware component"
    },
    "Version": {
      "type": "string",
      "description": "The version string for the firmware component"
    }
  }
},
"ControlPlaneInterface": {
  "type": "object",
  "description": "Information about an interface supported by the Control Plane",
  "properties": {
    "Interface": {
      "$ref": "#/definitions/Interface"
    },
    "MessageInitiationSupport": {
      "type": "string",
      "enum": [
        "AnyRequestor",
        "ControlPlaneRequestorOnly",
        "DeviceRequestorOnly"
      ],
      "enumDescriptions": {
        "AnyRequestor": "Either the Control Plane or the device may initiate messages",
        "ControlPlaneRequestorOnly": "Only the Control Plane may initiate messages",
        "DeviceRequestorOnly": "Only the device may initiate messages"
      },
      "description": "An indication of the directions from which messages may be initiated on this interface"
    }
  }
},
"ProtocolInfo": {
  "type": "object",
  "description": "Information about a protocol family supported by a device.",
  "properties": {
    "Protocol": {
      "type": "string",
      "enum": ["DSP0280", "PMCI Test Tools Interface and Design Specification"]
    }
  }
}
"MCTP",
"PLDM",
"NC-SI",
"NVMe",
"OEM"
],
"enumDescriptions": {
  "MCTP": "Management Control Transfer Protocol",
  "PLDM": "Platform Layer Data Model",
  "NC-SI": "Network Controller Sideband Interface",
  "NVMe": "Non-Volatile Memory Express",
  "OEM": "Original Equipment Manufacturer (proprietary)"
},
"Types": {
  "type": "array",
  "items": {
    "$ref": "#/definitions/ProtocolTypeSupport"
  }
}
},
"ProtocolTypeSupport": {
  "type": "object",
  "description": "Information about the types supported within a protocol family for a device.",
  "properties": {
    "Type": {
      "type": "number",
      "description": "The numeric identifier for the protocol type."
    },
    "Name": {
      "type": "string",
      "description": "The name of the protocol"
    },
    "Versions": {
      "type": "array",
      "items": {
        "patternProperties": {
          "^.*$": {
            "description": "A version of the protocol type supported by the device"
          }
        }
      }
    }
  }
},
"NC-SIAuxInfo": {
  "type": "object",
"description": "Information about the NC-SI implementation on the network controller",
"properties": {
  "ChannelID": {
    "type": "string",
    "description": "Network Controller Channel ID to address the Network Controller Channel."
  },
  "Pass-through": {
    "type": "string",
    "description": "The status of the channel to allow Pass-through packets.",
    "enum": [
      "Enabled",
      "Disabled"
    ],
    "enumDescriptions": {
      "Enabled": "Pass-through enabled on channel",
      "Disabled": "Pass-through disabled on channel"
    }
  }
}
## 12 ANNEX B (informative) Change log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0.0</td>
<td>2022-07-07</td>
<td>Initial Version</td>
</tr>
</tbody>
</table>
13 Bibliography

DMTF DSP4014, *DMTF Process for Working Bodies 2.6.*