PMCI Test Tools Interface
Message Flow

Work In Progress – August 2021

Justin King – IBM

Copyright © 2021 DMTF
Disclaimer

- The information in this presentation represents a snapshot of work in progress within the DMTF.
- This information is subject to change without notice. The standard specifications remain the normative reference for all information.
- For additional information, see the DMTF website.
- This information is a summary of the information that will appear in the specifications. See the specifications for further details.
PMCI Test Tools Architecture - Components

- **Test Client**
  - External Tool
  - Runs compliance tests
  - Connects to Test Service via out-of-band connection

- **Test Service**
  - Runs on Control Plane (BMC)
  - Interface to Test Client
  - Can produce inventory of available devices to test
  - Has interfaces to testable devices

- **Device Under Test** - runs unmodified
PMCI Test Tools Message Flow

1. Open TCP Port w/ TLS
2. Connect
3. Query Capabilities
4. Query System Inventory
5. Configure Device Under Test
6. Register To Protocol
7. Register Async Message Recipient
8. Test Messages
9. Disconnect
Open TCP Port w/ TLS

- Control Plane (BMC) opens a TCP port protected by TLS 1.2 or higher.
  - Spec does not define “how/when” port is opened - Test Service vendor policy
- Port number and Test Service Certificate Provided to the Test Client
  - Again, spec does not specify “how”
- Test Client connects to the port using the provided certificate
  - If done correctly, now have a secure channel for TC<->TS communication
Connect Message

Connect request message format

<table>
<thead>
<tr>
<th>Type</th>
<th>Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Parameter Length</td>
<td>uint32</td>
<td>0:3 Length of the security parameter</td>
</tr>
<tr>
<td>Security Parameter</td>
<td>Variable</td>
<td>4+ Security parameter provided by the Test Service vendor</td>
</tr>
</tbody>
</table>

→ Test Client sends a “Security Parameter” (Password, OTAC, Key, Certificate)

Connect response message format

<table>
<thead>
<tr>
<th>Type</th>
<th>Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Code</td>
<td>enum8</td>
<td>0 See Test Service Response Codes</td>
</tr>
<tr>
<td>Test Service Version</td>
<td>ver8</td>
<td>1 The version of the Test Service API that the Test Service implements</td>
</tr>
<tr>
<td>Test Client ID</td>
<td>uint32</td>
<td>2:5 Test Client ID for use in the Test Service Wrapper on subsequent commands</td>
</tr>
</tbody>
</table>

← Test Service responds with a Test Client ID that must be provided on all subsequent messages
Query Capabilities Message

→ Test Client requests Test Service capabilities
← Test Service responds with a set of pairs

<table>
<thead>
<tr>
<th>Type</th>
<th>Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Capability ID</td>
<td>uint16</td>
<td>4:5</td>
</tr>
<tr>
<td>First Capability Value</td>
<td>uint32</td>
<td>6:9</td>
</tr>
<tr>
<td>Second Capability ID</td>
<td>uint16</td>
<td>10:11</td>
</tr>
<tr>
<td>Second Capability Value</td>
<td>uint32</td>
<td>12:15</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Final Capability ID</td>
<td>uint16</td>
<td>...</td>
</tr>
<tr>
<td>Final Capability Value</td>
<td>uint32</td>
<td>...</td>
</tr>
</tbody>
</table>

Capabilities for timing, retry counts, Connection Watchdog timeout, and ranges for OEM Capabilities

1. Open TCP Port w/ TLS
2. Connect
3. Query Capabilities
4. Query System Inventory
5. Configure Device Under Test
6. Register To Protocol
7. Register Async Message Recipient
8. Test Messages
9. Disconnect
**Query System Inventory Message**

→ Test Client requests system inventory

← Test Service responds with JSON

(examples on next slides)

Warning: Text in the following slides is subject to change.
System Inventory – Identifiers and DUT Capabilities

"Devices": [
  { "Manufacturer": "ContosoAdapters", "Location": "Slot 3", "DeviceIdentifier": 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" } ],
    "Mediums": [
      { "Medium": "I2C", "InterfaceIdentifier": 3187, "ParentDeviceIdentifier": 0,
        "ProtocolSupport": [ 
          { "Transport": "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": 2, "Name": "PLDM for Platform Monitoring and Control", "Versions": [ "1.2.0" ] } ] }
        ] },
      { "Bus": "PCIeVDM", "InterfaceIdentifier": 3188, "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" ] },
            { "Type": 2, "Name": "PLDM for Platform Monitoring and Control", "Versions": [ "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" ] } ] } ]
    ]
]
System Inventory – MCTP Bridge

"Devices": [
    {
        "Manufacturer": "ContosoBridge",
        "Location": "Slot 5", "DeviceIdentifier": 6450,
        "HardwareBuses": [
            {
                "Bus": "I2C", "InterfaceIdentifier": 6457, "ParentDeviceIdentifier": 0,
                "ProtocolSupport": [ ... ]
            },
            {
                "Bus": "PCIeVDM", "InterfaceIdentifier": 6458, "ParentDeviceIdentifier": 0,
                "ProtocolSupport": [ ... ]
            }
        ]
    },
    {
        "Manufacturer": "ContosoBackendDrive",
        "Location": "Bridge Slot 1", "DeviceIdentifier": 1240,
        "HardwareBuses": [
            {
                "Bus": "I2C", "InterfaceIdentifier": 1247, "ParentDeviceIdentifier": 6450,
                "ProtocolSupport": [ ... ]
            },
            {
                "Bus": "PCIeVDM", "InterfaceIdentifier": 1248, "ParentDeviceIdentifier": 6450,
                "ProtocolSupport": [ ... ]
            }
        ]
    }
]
Configure Device Under Test

Admin Configure Device Under Test Message request format

<table>
<thead>
<tr>
<th>Type</th>
<th>Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment Count</td>
<td>uint8</td>
<td>0</td>
</tr>
<tr>
<td>Path Segments</td>
<td>uint32 * N</td>
<td>1+  The path identifier(s) for the device to test</td>
</tr>
</tbody>
</table>

→ Test Client sends one of:

→ DeviceIdentifier – “I want to talk to this device”
→ InterfaceIdentifier – “I want to talk to this device on this interface”
→ Sequence of InterfaceIdentifiers – “I want to follow this path to this interface on the device”

← Test Service responds with a DUT
Connection ID that can be used to send test messages

1. Open TCP Port w/ TLS
2. Connect
3. Query Capabilities
4. Query System Inventory
5. Configure Device Under Test
6. Register To Protocol
7. Register Async Message Recipient
8. Test Messages
9. Disconnect
Register To Protocol Message

<table>
<thead>
<tr>
<th>Type</th>
<th>Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol</td>
<td>enum8</td>
<td>The protocol the test client wishes to test against – see the command-type enumeration for values</td>
</tr>
<tr>
<td>DUT Connection ID</td>
<td>uint32</td>
<td>The device the test client is requesting to test against</td>
</tr>
<tr>
<td>Type Count</td>
<td>uint8</td>
<td>The number of types within the protocol the client wishes to test against</td>
</tr>
<tr>
<td>Types</td>
<td>uint8 * N</td>
<td>The specific individual types</td>
</tr>
</tbody>
</table>

→ Test Client sends a “Protocol” Value (PLDM, NC-SI, SPDM, MCTP VDM), possibly with “Types” (for PLDM – M&C, FW Update, RDE,…)
for a specific DUT Connection ID

→ Non-typed protocols (NC-SI, SPDM) skip
← Test Service responds with permission to use that protocol

1. Open TCP Port w/ TLS
2. Connect
3. Query Capabilities
4. Query System Inventory
5. Configure Device Under Test
6. Register To Protocol
7. Register Async Message Recipient
8. Test Messages
9. Disconnect
Register Async Message Recipient

<table>
<thead>
<tr>
<th>Type</th>
<th>Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol</td>
<td>enum8</td>
<td>The protocol the test client wishes to test against – see the command-type enumeration for values</td>
</tr>
<tr>
<td>DUT Connection ID</td>
<td>uint32</td>
<td>The device the test client is requesting to test against</td>
</tr>
<tr>
<td>Type Count</td>
<td>uint8</td>
<td>The number of types within the protocol the client wishes to test against</td>
</tr>
<tr>
<td>Types</td>
<td>uint8 * N</td>
<td>The specific individual types</td>
</tr>
</tbody>
</table>

→ Test Client sends a “Protocol” Value (PLDM, NC-SI, SPDM, MCTP VDM), possibly with “Types” (for PLDM – M&C, FW Update, RDE,…) for a specific DUT Connection ID

→ Non-typed protocols (NC-SI, SPDM) skip

← Test Service responds acknowledging that it will forward asynchronous messages to the Test Client for this Protocol (and Types)
PLDM Test Message
(NC-SI, SPDM, MCTP VDM identical)

PLDM 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>Maximum Retries</td>
<td>uint8</td>
<td>See Test Service Retry Mechanism</td>
</tr>
<tr>
<td>1+</td>
<td>PLDM Message</td>
<td>Variable</td>
<td>A fully-formed PLDM message</td>
</tr>
</tbody>
</table>

PLDM 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>Retry Count</td>
<td>uint8</td>
<td>The number of times that the Test Service re-tried the test message</td>
</tr>
<tr>
<td>2:5</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. If the Retry Count is non-zero, this field references the final retry.</td>
</tr>
<tr>
<td>6+</td>
<td>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>
**Disconnect Message**

→ Test Client sends *Disconnect*

← Test Service responds, Test Client ID is no longer valid and a new *Connect* message must be used to obtain a new one

1. Open TCP Port w/ TLS
2. Connect
3. Query Capabilities
4. Query System Inventory
5. Configure Device Under Test
6. Register To Protocol
7. Register Async Message Recipient
8. Test Messages
9. *Disconnect*
Watchdog Timer

- The Test Service may optionally implement a Watchdog Timer
- Upon expiration of the timer, Test Client ID is no longer valid, msgs fail
- Current Tools TF thinking (not in document yet):
  - Timer Start – when TS responds to a message from TC
  - Timer Expires – no messages received from TC
    - (Timer is paused while TS is handling message from TC, which includes waiting for a DUT to respond to a Test Message)

- Test Client may use Query Status message with “Ping” option to reset the Timer
- If implemented, TS must report expiration time in Query Capabilities
PTTI Message Format

### Basic Message Format

<table>
<thead>
<tr>
<th>Type</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Service Wrapper</td>
<td>0:11</td>
</tr>
<tr>
<td>Command Header</td>
<td>12:X-1</td>
</tr>
<tr>
<td>Command Data</td>
<td>X+</td>
</tr>
</tbody>
</table>

### Command Type

<table>
<thead>
<tr>
<th>Protocol</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin</td>
<td>0xFF</td>
</tr>
<tr>
<td>MCTP</td>
<td>0x00</td>
</tr>
<tr>
<td>PLDM</td>
<td>0x01</td>
</tr>
<tr>
<td>NC-SI</td>
<td>0x02</td>
</tr>
<tr>
<td>SPDM</td>
<td>0x05</td>
</tr>
</tbody>
</table>

### Test Service Wrapper Flags

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>dir</td>
<td>Direction 0: TC -&gt; TS Request Message 1: TS -&gt; TC Response Message</td>
</tr>
<tr>
<td>[1]</td>
<td>compound-id</td>
<td>Compound Connection ID</td>
</tr>
<tr>
<td>[2:15]</td>
<td>R</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

- Assigned by TS when TC connects
- Opaque TS-generated token from System Inventory + Configure DUT
### 10.2.1.1 Command Codes

The command code in the admin command header identifies which 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 Device Under Test</td>
<td>0x20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register to Protocol</td>
<td>0x21</td>
</tr>
<tr>
<td>Register Async Message Recipient</td>
<td>0x22</td>
</tr>
</tbody>
</table>