PMCI Test Tools Interface Message Flow

Work In Progress – August 2021

Justin King – IBM

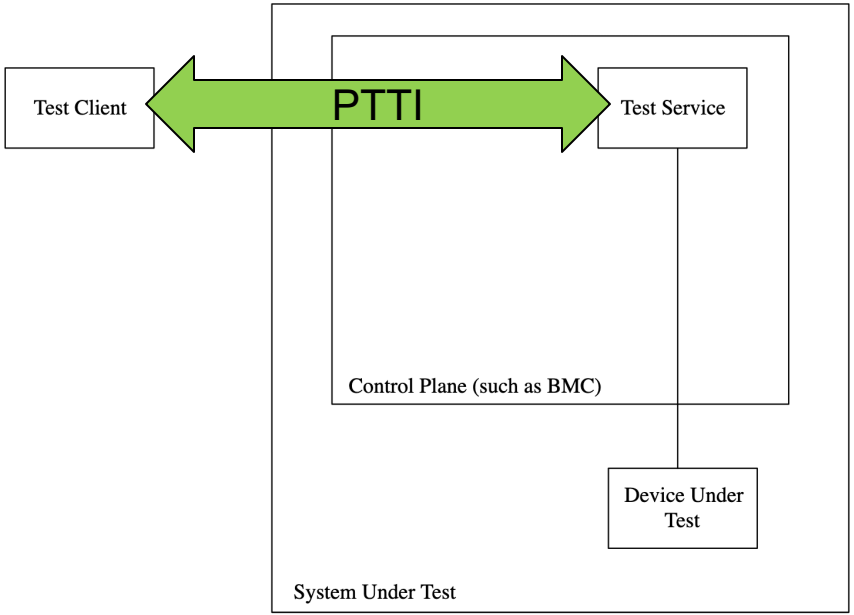
Copyright © 2021 DMTF

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

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*

Connect Message

***Connect* request message format**

	Type	Byte	Description
Security Parameter Length	uint32	0:3	Length of the security parameter
Security Parameter	Variable	4+	Security parameter provided by the Test Service vendor

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

***Connect* response message format**

	Type	Byte	Description
Response Code	enum8	0	See Test Service Response Codes
Test Service Version	`ver8`	1	The version of the Test Service API that the Test Service implements
Test Client ID	uint32	2:5	Test Client ID for use in the Test Service Wrapper on subsequent commands

← Test Service responds with a Test Client ID that must be provided on all subsequent 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*

Query Capabilities Message

→ Test Client requests Test Service capabilities

← Test Service responds with a set of pairs

	Type	Byte	Description
First Capability ID	`uint16`	4:5	From the Table of Capability IDs
First Capability Value	`uint32`	6:9	Value for the first Capability
Second Capability ID	`uint16`	10:11	From the Table of Capability IDs
Second Capability Value	`uint32`	12:15	Value for the second Capability
...	...	...	...
Final Capability ID	`uint16`	...	From the Table of Capability IDs
Final Capability Value	`uint32`	...	Value for the final Capability

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.

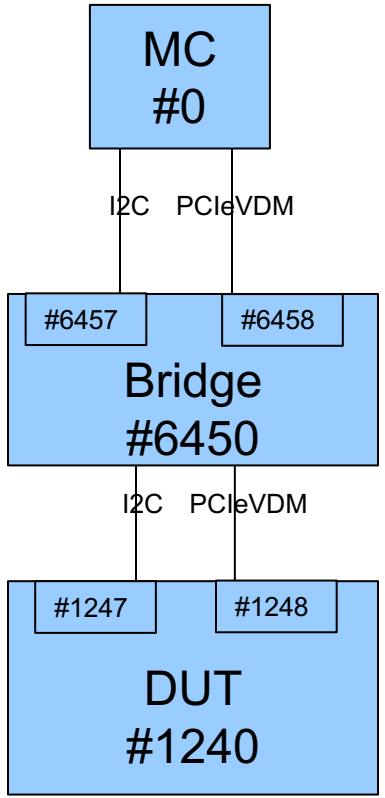
1. Open TCP Port w/ TLS
2. *Connect*
3. *Query Capabilities*
4. *Query System Inventory*
5. *Configure Device Under Test*
6. *Register To Protocol*
7. *Register Async Message Recipient*
8. Test Messages
9. *Disconnect*

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" ] },
         { "Type": 5, "Name": "SPDM 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

	Type	Byte	Description
Segment Count	uint8	0	The number of path identifiers supplied in the remainder of this message
Path Segments	uint32 * N	1+	The path identifier(s) for the device to test

→ 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

	Type	Byte	Description
Protocol	enum8	0	The protocol the test client wishes to test against -- see the command-type enumeration for values
DUT Connection ID	uint32	1:4	The device the test client is requesting to test against
Type Count	uint8	5	The number of types within the protocol the client wishes to test against
Types	uint8 * N	6+	The specific individual types

→ 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

	Type	Byte	Description
Protocol	enum8	0	The protocol the test client wishes to test against -- see the command-type enumeration for values
DUT Connection ID	uint32	1:4	The device the test client is requesting to test against
Type Count	uint8	5	The number of types within the protocol the client wishes to test against
Types	uint8 * N	6+	The specific individual types

→ 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)

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*

PLDM Test Message (NC-SI, SPDM, MCTP VDM identical)

PLDM Test Message request format

Offset	Field	Type	Value
0	Maximum Retries	uint8	See Test Service Retry Mechanism
1+	PLDM Message	Variable	A fully-formed PLDM message

PLDM Test Message response format

Offset	Field	Type	Value
0	Response Code	enum8	See Test Service Response Codes
1	Retry Count	uint8	The number of times that the Test Service re-tried the test message
2:5	Elapsed Time (µs)	uint32	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.
6+	Response Message	Variable	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.

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*

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*

Backup Slides

PTTI Message Format

Basic Message Format

	Byte
Test Service Wrapper	0:11
Command Header	12:X-1
Command Data	X+

Test Service Wrapper

	Type	Byte	Description
version	ver8	0	PTTI Protocol Version
command-type	byte	1	Command Type
flags	bitfield16	2:3	Test Service Wrapper Flags
test-client-id	uint32	4:7	Client ID or token as assigned during registration
dut-connection-id	uint32	8:11	DUT Connection ID

Command Type

Protocol	ID
Admin	0xFF
MCTP	0x00
PLDM	0x01
NC-SI	0x02
SPDM	0x05

Test Service Wrapper Flags

Bits	Field	Description
[0]	dir	Direction 0: TC -> TS Request Message 1: TS -> TC Response Message
[1]	compound-id	Compound Connection ID
[2:15]	R	Reserved

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.

Command	Code
Connect	0x00
Disconnect	0x01
Query Capabilities	0x10
Query Status	0x11
Query System Inventory	0x12
Configure Device Under Test	0x20

Command	Code
Register to Protocol	0x21
Register Async Message Recipient	0x22