Open Mailbox Communications

The Open Mailbox Communications specification is a point-to-point request-response protocol which supports shared physical busses, multiple API support, and minimal hardware and memory requirements.

Use Cases

An interposing device can use the Open Mailbox Communications specification with a minimal footprint on the address space of the interposed bus (for example, interposing on a SPI Nor/Nand flash interface).

A device wishing to support multiple APIs can use the Open Mailbox Communications specification to share the same underlying transport.

A device with minimal memory requirements can use the Open Mailbox Communications specification to implement an API which requires a larger transmission capacity. Because of the limited hardware requirements, many devices can support this specification.

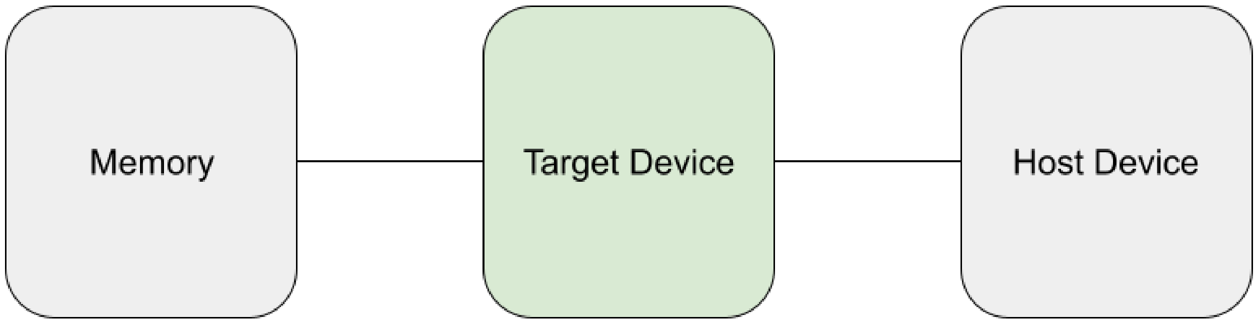
Terms

- Mailbox: a minimal memory-mapped interface used to communicate between two devices

Specification

Mailbox Protocol

The Open Mailbox Communications specification is a point-to-point request-response protocol driven by a host device. On startup, a target device indicates a mailbox interface at a particular memory address; the host device discovers the mailbox address and issues requests by writing to that location.



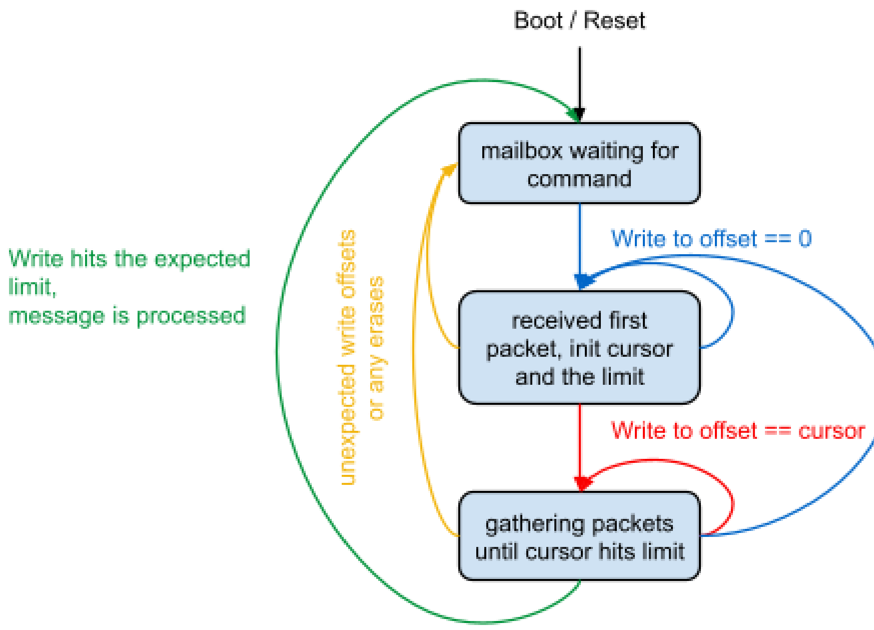
The actors in the Open Mailbox Communications specification. Communications via the mailbox interface occur between the host and target device. The target device may also facilitate the host device's access to some external memory, though this is not required by the specification.

Each request message written by the host to the mailbox corresponds to a response message written by the target to the mailbox. The host polls until the message is ready and then reads the response message from the mailbox. The mailbox protocol also supports a mechanism for sending and receiving messages larger than the mailbox size.

The minimum mailbox size is 1024 bytes.

The mailbox address is determined by the target device. Several options for how to determine the mailbox address are defined in the *Mailbox Location* section. The host device must know where to discover the mailbox address in a system; how the mailbox is discovered is dependent on the physical integration for the interface. See the *Physical Bindings* section for details for various interfaces.

To send a message to the target device, the host performs a series of writes to the mailbox region. Writes to the mailbox region shall always begin at offset 0, shall be written in chunks of the supported physical interface write size, and shall be written in order until the full message is received. The following state machine describes the write handling logic in the target device:


Boot / Reset
mailbox waiting for command
Write to offset == 0
received first packet, init cursor and the limit
Write to offset == cursor
gathering packets until cursor hits limit
Write hits the expected limit, message is processed
unexpected write offsets or any erases

The target device state machine for processing writes to the mailbox region.

After writing the full message, the host polls until a response is written by the target into the mailbox region; how the host determines that the message is ready to be read depends on the physical binding used. Once the response is written to the mailbox, the host receives the message by performing a series of reads of the mailbox region. Reads to the mailbox region shall begin at offset 0, shall be read in chunks of the supported physical interface read size, and shall be read in order until the full message is received.

Mailbox Location

The target device is responsible for determining the location of the mailbox in the address space. The mailbox will shadow a portion of the address space; therefore, wherever it is located, it must not interfere with the contents of the memory being shadowed.

Hard-Coded Mailbox Address

The target device firmware can hard-code the mailbox address and populate a known location in shared memory with the hard-coded mailbox address. This option is known as setting a static mailbox.

The known location in memory may be a metadata region (for SPI busses the SFDP table is a suitable choice), or a stable configuration location.

Scan Shared Memory for Mailbox Location

To avoid shadowing data at a particular address via a hard-coded value, the target device can scan the address space at each boot for a region indicating the mailbox placement. This option is known as setting a dynamic mailbox.

The following requirements are imposed for the dynamic mailbox:

- The marked region for a dynamic mailbox shall contain the pattern `_MAILBOX` repeated for the entirety of the dynamic mailbox address space.
- The marked region shall be a multiple of the minimum mailbox size.
- The marked region shall be aligned to a minimum of 1024 bytes. Specific implementations may increase the alignment requirements (The alignment of the mailbox affects the number of locations the target device has to check at each boot. It is recommended that this alignment be at least 64K).
- If multiple marked regions are found in the address space, then the last region shall be used for the dynamic mailbox.
- If the target device presents only a portion of the total address space to the host device, then only marked regions from the presented portion shall be considered for the dynamic mailbox.
- If no marked region is found, then the target device may map the mailbox at a default static fallback address.

Message Header Format

All numeric multi-byte fields shall use the little endian (LSB first order).

To ensure the entire message is received and to protect against data corruption, the following message header is used:

[31:24]		[23:16]	[15:8]	[7:0]
Protocol Revision	Status Code	Reserved	Checksum	
Message Type			Data Length	
Message Payload				

Protocol Revision (4 bits)

The protocol revision occurs at the beginning of the message and defines the message header format used for communication. The protocol revision is set to 1.

Status Code (4 bits)

The status code is used for all message types, though it is especially useful for message types that support breaking a logical request into several transmission units. See the *Message Sequences* section for details on how the status is set.

Status Code	Name	Definition	Set By
0	`REQUEST`	The message in the mailbox contains a request.	Host
1	`RESPONSE`	The message in the mailbox contains a response.	Target
2	`CONTINUE`	I've read the transmission unit. Please send another transmission unit.	Host / Target
3	`NO DATA`	There's no more data to send (used after a CONTINUE status code).	Target
4	`BAD DATA`	The message in the mailbox is ill-formed.	Target
5	`UNKNOWN`	The message type or protocol revision cannot be handled by the target device.	Target

Reserved (8 bits)

This byte is reserved for future use. The reserved bits shall be set to 0.

Checksum (16 bits)

The checksum is set to a value such that the sum of all the bytes in the message (discarding carry over bits) is equal to zero.

Message Type (16 bits)

A value indicating the type of message contained after the header. See the *API Codes* section for valid values for the message type field.

Data Length (16 bits)

The total length of the message in bytes (excluding this message header). It is equal to the number of bytes immediately following this field.

Note: This field necessarily imposes a maximum mailbox size. Future revisions of the protocol may reserve more space for the data length field to accommodate larger mailboxes.

Message Payload (Variable)

A message that conforms to the specification indicated by the Message Type.

Message Sequences

When a message (either a complete logical request or a transmission unit) is written to the mailbox by the host, the status code is set to `REQUEST`. The target reads the entire message, then checks to see if it constitutes a complete logical request. If it does not, then the target sets the status code to `CONTINUE`. The host then writes the next transmission unit and sets the status code back to `REQUEST`. This process repeats until all transmission units comprising the logical request have been transferred. The process of sending `REQUEST` and `CONTINUE` messages is known as a *request sequence*.

The same sequence also applies to logical responses that are broken into several transmission units, substituting the `REQUEST` code for the `RESPONSE` code. The process of sending `RESPONSE` and `CONTINUE` messages is known as a *response sequence*.

The protocol revision and message type shall remain constant throughout a request/response sequence. The data length field shall be set to `0` for a message with the `CONTINUE` status code.

If the target cannot interpret a message's protocol revision or message type, then the target shall set the status code to `UNKNOWN`. During a request sequence, if the target receives an ill-formed message, then the target shall set the status code to `BAD DATA`. During a response sequence, if the host sets the status code to `CONTINUE` and the target has no more data to send, then the target shall set the status code to `NO DATA`.

There are no timing constraints between setting any status and sending the next message.

Security

If the target device enforces a write-policy on the address space designated to it, then the mailbox shall be mapped to a write-enabled region.

The integrity of messages is enforced via the checksum field in the message header. The target device shall ensure that a full message received to the mailbox is well-formed. If the checksum validation fails, then the target device sets the `BAD DATA` status code. It is up to the host to determine how to handle this error condition.

Physical Bindings

Because the Open Mailbox Communications specification imposes minimal hardware requirements, it can be implemented on top of several physical bus protocols. Specifically, the protocol can be implemented on top of any physical protocol which supports reading and writing to addressable memory.

SPI Flash

The SPI Flash interface is a de-facto standard supported by most flash parts. Physical communication is defined using the SPI bus. All commands consist of a one-byte opcode, a three or four-byte address, and a variable-length data. Opcodes are used to indicate a read, write, or erase operation of the flash chip.

The Open Mailbox Communications standard is designed to provide a method of communication with devices that interpose on an SPI Flash interface. These interposers emulate SPI Flash to the host device, supporting the same read, write, and erase operations that the underlying flash device supports.

Typical SPI interposers leverage hardware to efficiently handle interposing on the SPI bus. The hardware can be configured to use an EEPROM mode of operation, which automatically handles instruction decoding and address translation of SPI transactions. A typical SPI interposer supports mapping a virtual address space to several physical regions, including the underlying external flash device, the interposer's internal flash, or the interposer's internal RAM.

For the Open Mailbox Communications specification, the SPI interposer acts as a target device.

The interposer can publish information about the mailbox region and supported APIs via the JEDEC Serial Flash Discoverable Parameters (SFDP) specification.

SFDP Parameter Header

The parameter header shall have the following format:

[31:24]	[23:16]	[15:8]	[7:0]
Parameter Length (in double words)	Parameter Major Revision	Parameter Minor Revision	Parameter ID LSB
Parameter ID MSB	Parameter Table Pointer (byte address)		

Parameter Table Version

This specification describes major version 01, minor version 00 of the parameter table.

Parameter ID

The vendor-defined Parameter ID 0x2709 is within the function-specific range for Parameter ID allocations as defined by the JEDEC SFDP specification. U+2709 is the envelope emoji, which is fitting for a mailbox protocol. Once standardized by the JEDEC office, a suggested parameter ID is 0xF4EC, corresponding to the unicode emoji for an open mailbox with a letter (U+1F4EC).

```
// The purpose of the Function Specific tables is to allow development of features and
associated parameter tables common to multiple manufacturers, prior to the parameter
tables being incorporated into the next revision of JESD216. Allocation of IDs for
Function Specific tables is requested through the JEDEC office, see Annex A.
```

SFDP Parameter Table

The parameter table shall have the following format:

[31:24]	[23:16]	[15:8]	[7:0]
Parameter Signature = "MAIL" (0x4D41494C)			
Mailbox Offset			
Mailbox Size		API 1	
API 2		API 3	
...		API N	

See the *API Codes* section for valid values for APIs.

USB

The Universal Serial Bus (USB) specification is a widely-used standard for connecting peripherals to a host device. Physical communication occurs over the USB bus, addressed to a particular endpoint on an interface of a physical device. Because a single interface supports multiple endpoints, the Open Mailbox Communications specification can use a dedicated endpoint for communications.

When communicating over USB, the data normally present in the message header is instead combined into control structures of the USB protocol to more efficiently use the USB bus.

For the Open Mailbox Communications specification, the target device can expose a USB interface that uses the following operations for mailbox communications.

Interface Details

An interface that supports the Open Mailbox Communications USB Physical Binding specification shall report that it implements the following class code triple:

Code	Value	Notes
Base Class	0xFF	Vendor-Specific Base Class
Sub Class	0xF4	Represents the unicode emoji U+1F4EC, an open mailbox with a letter
Protocol	0xEC	

The host device can use the standard USB APIs to discover an endpoint that supports Open Mailbox Communications.

Endpoint Details

An interface that supports the Open Mailbox Communications USB Physical Binding specification shall support one associated interrupt IN endpoint, known as the *interrupt* endpoint.

Target Device State

A target device that communicates via USB shall maintain an internal state, which must be one of the following values:

State	Code	Description
Initializing	0	Initial state, entered by `SetConfiguration` or `SetInterface`
Idle	1	Ready to process a command
Continue Request	2	Waiting for more data to be sent
Processing Command	3	Busy processing a command
Response Ready	4	A response is ready to be read

Transitions between states occur during a control request or interrupt as described below.

Control Requests and Interrupts

An interface that supports the Open Mailbox Communications USB Physical Binding specification shall support the following operations:

- The *InterfaceReady* interrupt
- The *GetAPIs* control request
- The *SendMessage* control request
- The *ResponseReady* interrupt
- The *GetResponse* control request
- The *GetState* control request

InterfaceReady Interrupt

This interrupt is issued by the interface after configuration is complete and the target device is ready to receive messages. It shall only be issued from the `Initializing` state. After issuing the interrupt, the target device shall transition to the `Idle` state.

During the data stage, the interface shall send an interrupt header which has the following format:

Offset	Field	Size	Value	Description
0	`bLength`	1	`2`	Size in bytes of this header
1	`bInterruptType`	1	`0`	`InterfaceReady` constant
2	`wMailboxSize`	2	*Mailbox Size*	The size of the mailbox supported. `wLength` of any Control Request must not exceed this size.
4	`wNumAPIs`	2	*Num APIs*	The number of APIs supported by the target device

GetAPIs Control Request

The host device issues the `GetAPIs` control request to determine the APIs that the target device supports. The host device may send this request when the target device is in any state other than the `Initializing` state. The target device shall not transition to a new state upon receiving this request.

The setup stage shall have the following format:

Offset	Field	Size	Value	Description
0	`bmRequestType`	1	`0b11000001`	Device-to-host, vendor type, interface recipient
1	`bRequest`	1	`7`	`GetAPIs` constant
2	`wValue`	2	`0`	Unused
4	`wIndex`	2	*Interface Index*	The OMC interface index
6	`wLength`	2	*2 * Num APIs*	2 * The number of APIs returned from the `InterfaceReady` interrupt

The data stage shall have the following format:

Offset	Field	Size	Value	Description
`k`	`wAPI[k]`	2	*API Code*	One of the values in the *API Codes* section

Note that `0 ≤ k < wLength` and `k % 2 = 0`. The target device shall report the same API code at index `k` across subsequent control requests.

SendMessage Control Request

The host device issues the `SendMessage` control request to transmit a message to the target device. The host device may only send this request when the target device is in the `Idle` or `Continue Request` states. Upon receiving this request, the target device shall transition to the `Idle` state if `wLength` is 0, shall transition to the `Processing Command` state if the complete logical request is received, or shall transition to the `Continue Request` state otherwise.

The setup stage shall have the following format:

Offset	Field	Size	Value	Description
0	`bmRequestType`	1	`0b01000001`	Host-to-device, vendor type, interface recipient
1	`bRequest`	1	`8`	`SendMessage` constant
2	`wValue`	2	*Message type*	One of the values in the *API Codes* section
4	`wIndex`	2	*Interface Index*	The OMC interface index
6	`wLength`	2	*Data length*	Length of the payload

The data stage shall be a message payload, formatted in accordance with the specification defined by the `wValue` field.

ResponseReady Interrupt

This interrupt is issued by the interface after a command has completed processing. It shall only be issued from the `Processing Command` or `Response Ready` states. After issuing the interrupt, the target device shall transition to the `Response Ready` state if not already in that state.

During the data stage, the interface shall send an interrupt header which has the following format:

Offset	Field	Size	Value	Description
0	`bLength`	1	`6`	Size in bytes of this header
1	`bInterruptType`	1	`1`	`ResponseReady` constant
2	`wLength`	2	*Response length*	The length in bytes of the response data

GetResponse Control Request

The host device issues the `GetResponse` control request to receive a message from the target device. The host device may only send this request when the target device is in the `Response Ready` state. Upon receiving this request, the target device shall transition to the `Idle` state if all responses have been sent to the host, or shall remain in the `Response Ready` state and send the `ResponseReady` interrupt if another response is ready.

The setup stage shall have the following format:

Offset	Field	Size	Value	Description
0	`bmRequestType`	1	`0b11000001`	Device-to-host, vendor type, interface recipient
1	`bRequest`	1	`9`	`GetResponse` constant
2	`wValue`	2	`0`	Unused
4	`wIndex`	2	*Interface Index*	The OMC interface index
6	`wLength`	2	*Response length*	Length returned from the `ResponseReady` interrupt

The data stage shall be a message payload, formatted in accordance with the specification corresponding to the previous `SendMessage` control request.

GetState Control Request

The host device issues the `GetState` control request to determine the current state of the target device. The host device may send this request when the target device is in any state. The target device shall not transition to a new state upon receiving this request.

The setup stage shall have the following format:

Offset	Field	Size	Value	Description
0	`bmRequestType`	1	`0b11000001`	Device-to-host, vendor type, interface recipient
1	`bRequest`	1	`10`	`GetState` constant
2	`wValue`	2	`0`	Unused
4	`wIndex`	2	*Interface Index*	The OMC interface index
6	`wLength`	2	`1`	Fixed length of the state response

The data stage shall have the following format:

Offset	Field	Size	Value	Description
0	`bState`	1	*State*	One of the state codes listed above

APIs

The Open Mailbox Communications specification can be supported as-is by open standards to allow communication using open message formats.

API Codes

A Supported API is identified by a unique 16-bit code. The following APIs are defined:

Code	API	Notes
0x0000	None	Reserved for empty entries for mailbox discovery
0x0001	MCTP	Defined in DSP0236_1.3.1
0x0002	TPM	Defined in TPM 2.0 Library Part 3: Commands
0x0003	SPDM	Defined in DSP0274_1.1.1
0xABC0 - 0xABCF	Vendor-defined	Used for non-standard APIs interpreted in a vendor-specific way

Several API codes list the standards which describe the format of messages. While a particular version of the API standard is listed here, the latest available version of the standard is considered supported. If an unsupported version of the API standard is sent to a target device, the target device should respond with an error message dictated by the API standard.

API Bindings

Most of the existing standards already provide a binding for communicating over the physical interfaces described in the *Physical Bindings* section. However, the bindings assume that those physical interfaces are exclusively dedicated to supporting that protocol. This section discusses the existing standards and provides recommendations for incorporating the Open Mailbox Communications specification.

MCTP

Existing Specifications

The MCTP over Serial Transport specification describes a method to encapsulate an MCTP message in an SPI frame. The SPI framing format is incompatible with devices which emulate SPI flash (SPI flash devices all support a 1-byte opcode, 3 or 4-byte address, and variable length data message format for SPI transactions, whereas the MCTP over Serial Transport specification describes a message that begins and ends with a framing flag). Therefore, the MCTP over Serial Transport specification is distinct from the Open Mailbox Communications specification using the SPI EEPROM flash physical binding.

The MCTP over SMBus/I2C specification describes a method to encapsulate an MCTP message in an SMBus block write command. While many of the fields are useful (and have equivalents in the message header format above), the SMBus header contains detailed addressing fields that are not necessary for an interposer architecture. Therefore, the MCTP over SMBus/I2C specification is not suitable for communicating via the Open Mailbox Communications standard.

Open Mailbox Communications Binding

The MCTP base specification supports a baseline transmission unit size of 64 bytes. However, the specification allows for larger transmission units between endpoints for specific message types. The MCTP base specification allows other standards to define the mechanism used for negotiating larger transmission units. The SPDM over MCTP binding specification does not mention transmission unit sizes.

For MCTP over Open Mailbox Communications, the recommended MCTP transmission unit size is equal to the mailbox size. This minimizes the number of MCTP messages needed to form a logical request and allows the entire mailbox region to be used for messages.

TPM

Existing Specifications

PC Client Specific

The PC Client Specific TPM Interface Specification (TIS) defines the requirements for TPM on an SPI bus integrated into a PC architecture. The TPM exposes a set of registers at well known addresses, and the PC architecture sends all requests within the known address range to the TPM.

The south bridge will route the entire address range from 0xFED4_0000 through 0xFED4_4FFF to the TPM over SPI.

The PC Client Specific TIS also defines the SPI wire format of messages, which is similar to but incompatible with the SPI EEPROM flash de-facto standard. The TPM SPI wire format is 1-bit indicating read/write, 1 reserved bit, 6 bits of data length, 3 address bytes, and 4 data bytes. The specification also reserves 60 additional bytes for future use (to write 64B registers)

The combination of an unconfigurable address space and incompatible wire format makes the PC Client Specific TIS not suitable for communicating via the Open Mailbox Communications specification using the SPI EEPROM flash physical binding.

Mobile Command Response Buffer

The Mobile Command Response Buffer (CRB) Interface defines a flexible interface for communicating with a TPM. The CRB interface is designed to work with a large number of hardware devices. It specifies a 48-byte control structure and a command/response buffer.

The command/response buffers may overlap, which fits well with the mailbox protocol described in this specification. The control structure does not map into the open mailbox communications standard;

however, part of the control structure is used for mobile-specific TPM functionality and another part is used for equivalent functionality described by the open mailbox communications standard.

Several control structures do not map to the open mailbox communications standard. The low-power (idle) state is a mobile-specific TPM feature, and is not mapped to this standard. Interrupt control is not mapped to this standard; instead, the host polls the SPI status register's BUSY bit until the response is in the mailbox.

Other control structures do map to the open mailbox communications standard. The unrecoverable error status can be communicated via the status code in the message header format. A host can cancel a long-running request message by setting the status code in the message header format. The start field is automatically "set" when a complete request message is received. The buffer sizes and addresses are equivalent to the mailbox size and offset.

TPM over Open Mailbox Communications can support the TPM API in a mechanism similar to the mobile CRB interface; however, it doesn't require a control structure, and the command/response buffers both map to the entire mailbox region.

SPDM

SPDM messages may be transported over the MCTP binding and can be supported by the RoT using Open Mailbox Communications with the MCTP API Code. However, a RoT may wish to support SPDM directly to reduce the complexity of firmware (with trade-offs in complexity for other firmware on the system).

Since the SPDM specification describes the byte order of SPDM messages, it can be used directly with the Open Mailbox Communications specification without requiring the MCTP transport.

References

- DMTF Standards
  - Security Protocol and Data Model (SPDM) Specification - DSP0274_1.1.1
  - Management Component Transport Protocol (MCTP) Base Specification - DSP0236_1.3.1
  - Security Protocol and Data Model (SPDM) over MCTP Binding Specification - DSP0275_1.0.0
  - MCTP Serial Transport Binding Specification - DSP0253_1.0.0
  - MCTP SMBus/I2C Transport Binding Specification - DSP0237_1.2.0
- JEDEC Standards
  - Serial Flash Discoverable Parameters (SFDP) - JESD216D.01
  - Standard Manufacturer's Identification Code - JEP106BC
- TCG Standards
  - PC Client Specific TPM Interface Specification (TIS) - 1.3 Rev 27
  - TPM 2.0 Mobile Command Response Buffer Interface - Level 00 Rev 12
  - TPM 2.0 Library - Part 3: Commands - Level 00 Rev 01.59
- USB Standards
  - Universal Serial Bus Specification - 2.0