Manufacturing Tool V2 (MFGTool2) Update Transport Protocol (UTP) Introduction

Contents

[**Universal Updater Goals** 2](#_Toc337478311)

[**Use Cases** 2](#_Toc337478312)

[**Architecture** 2](#_Toc337478313)

[**Design Details** 7](#_Toc337478314)

**Universal Updater Goals**

1. Decouple the host software from detailed knowledge of the device media layout and device command specifics (i.e. keep device commands opaque to host).
2. Avoid hardcoded installation logic in the host software.
3. Allow the firmware writer to control the installation sequence and design of installation commands. In other words, allow the system providers to specify and control installation without requiring changes to the host tools.
4. Ensure new update protocol is portable to transports beyond USB.

**Use Cases**

1. Manufacturing.
   * Supports multiple devices from one host.
   * Does unconditional full installation.
   * Writes OTP bits.
   * Preloads content.
   * Installs DRM keys and other one-time initialization.
   * May include production tests.
   * Could install from devices other than USB (e.g. SD card).
2. End-user de-bricking a dead device.
   * Conditionally writes firmware only or does a full installation.
   * May want to save/restore objects if doing a full installation.
3. Development support.
   * Write firmware only or force a full installation.
   * May want read back capabilities.

**Architecture**

The following diagram depicts the high-level architecture for the universal updater. The following is a brief description of the major elements of the design.

1. (Yellow) A set of device commands defined and interpreted by the Device Command Engine (DCE) firmware. Since DCE is system specific, each system is free to specify the supported commands as well as the format of those commands. This allows each system to implement only the needed functionality, while allowing each to implement system specialization.
2. (Blue) A set of host commands defined and interpreted by the Host Command Engine (HCE). These commands will perform control functions such as connecting to a target device and UI functions such as requesting operator input.
3. (Grey) A list of device and host commands referred to as the Update Command List (UCL) that is provided by the system developers. The UCL is the mechanism which developers use to alter the installation sequence and device commands. The form of the UCL is not specified by the architecture, but possibilities include an XML document, delimited ASCII file or some form of a script. The UCL interpreter could be implemented as a simple list parser with limited conditional execution capabilities or as a complete scripting engine.
4. (Dotted Line) An optional package container that would bundle together the UCL and all associated objects (e.g. firmware image, file system images, preloaded content, etc.). The form of the container could be an off-the-shelf installer package, self-extracting archive, or a custom format of our own design.
5. The windows host program that parses commands from the UCL and delivers device commands to the target or directly executes host commands.
6. (Green) A transport protocol for delivering device update commands over USB. The Update Transport Protocol (UTP) is designed to be carried over the Bulk Only Transport (BOT) of the Mass Storage Class (MSC). This design takes advantage of the bandwidth provided by BOT, while eliminating the need for a custom windows driver or inf file.
7. (Yellow) System specific device firmware that implements the UTP transport and DCE engine. The firmware is system specific because media partitioning, flash translation layers, and file systems vary greatly between the targeted systems.



**Device Commands**

Device commands are update operations performed on the target by the Device Command Engine (DCE). They encompass the target functionality needed to successfully install a complete firmware system on an uninitialized or corrupted device. Generally this means preparing the primary storage media and writing the firmware and loading content. It can also include other system and device dependent operations. Because each system manages media with their own unique driver stacks, and because end products have varied features and capabilities, the design of the device commands will remain system dependent and needs to be easily extensible.

**Update Command List (UCL)**

The Update Command List is a set of operations that are parsed and executed by the UCL interpreter. The UCL is written by the system developers and is delivered along with the firmware image and any other objects needed to complete the installation. Optionally, we may define a container to hold the UCL and related objects.

The UCL includes both host and device commands. Host commands include control and user interface operations that are defined and executed by the Host Command Engine (HCE). Device commands are transported over USB and executed by the Device Command Engine (DCE) which is system dependent. The UCL interpreter views device commands as an array of bytes and is not aware of their structure or meaning. This allows the system developers to modify device command sets and the UCL without impacting the host program.

The UCL should support conditional execution (i.e. the ability to skip commands based on the result of a previous command) and should be easy to create. This could take the form of a simple text file, an XML document, or possibly a script if the UCL interpreter is designed as a script engine. At a minimum, the UCL interpreter will need the following information for each instruction.

1. Command Type - Host Control, User Interface, File Copy, Device Read, Device Write, Device Call, and Device Reset.
2. Command Body - Byte array representing command and parameters. Meaning of this array is determined by the HCE or DCE.
3. Payload Data - Data object associated with the command (e.g. firmware image). Not all commands will include a payload.
4. Commands Executing Phase – indicate which phase the command should be executed.

**Update Transport Protocol (UTP)**

The Update Transport Protocol provides a means for transporting device commands and their payloads to a target device over USB. It is implemented as a thin layer on top of Mass Storage Class - Bulk Only Transport. MSC-BOT was selected because windows provides a default driver and inf file, and because it provides adequate bandwidth. On the firmware side, it should be straight forward to implement UTP on top of an existing MSC stack that supports vendor specific SCSI commands.

The device firmware should enumerate one LUN and report the media as removable and not present. To the host it will appear as a memory card reader without a card inserted. This allows us to use MSC on devices that do not have a FAT file system. It also reduces the number of SCSI commands that need to be implemented. Unsupported SCSI commands can simply report that the media is not present.

A complete UTP transaction is made up of one or more UTP messages. Each UTP message is a complete BOT transfer (CBW-Data-CSW) that is followed by an optional extended reply. The UTP messages are implemented using a vendor specific 16-byte CDB. The extended UTP reply is returned through a standard SCSI REQUEST\_SENSE command. This is easy to implement on a windows host as the usbstor.sys driver automatically sends the REQUEST\_SENSE command if the previous CSW returned FAIL.

The following table lists the UTP messages and their parameters. Per SCSI protocol, each field is sent in big-endian order.

|  |  |  |  |
| --- | --- | --- | --- |
| **CDB[1]** | **CDB[6-13]** | **BOT Data** |  |
| **UTP Message** | **Parameter(bits)** | **Type - Direction** | **Description** |
| Poll = 0 | type(32) | None - N/A | Used to determine when an asynchronous device command is finished. Also used to query the UTP version. |
| Exec = 1 | payload\_size(64) | Device Command - OUT | Transfers a device command and its parameters. |
| Get = 2 | seq\_number(32) | Payload Data - IN | Transfers the command payload to the host. Payloads will be split into multiple 64 KB messages. |
| Put = 3 | seq\_number(32) | Payload Data - OUT | Transfers the command payload to the device. Payloads will be split into multiple 64 KB messages. |

The operation code (CDB[0]) for UTP messages is 0xF0.

Each message carries a 32-bit tag (CDB[2-5]) used to group messages belonging to the same UTP transaction. The host increments the tag when starting a new transaction. It is included for sanity checking only, as UTP messages from multiple transactions are not allowed to be interleaved. In other words, UTP transactions are considered atomic.

If the type parameter for a Poll() message is 1, the device will return an EXIT reply with the result containing the UTP version.

Payload size is included with the Exec() message so the device can reject a command before the payload messages begin. For example, this can be used to test whether a firmware image will fit before transferring the image. The payload size is 64-bits to allow payloads greater than 4 GB.

The sequence number is used to confirm the order of payload messages. The host resets the sequence number to 0 when starting a new transaction. The number is incremented for each subsequent payload message.

Each UTP message either returns success or is followed by an extended reply. If a message completes without an error, the protocol simply returns PASS in the CSW. For extended replies, the CSW returns FAIL and the complete reply is returned through fixed format sense data. The following table lists the defined replies and their meaning. Per SCSI protocol, each field is sent in big-endian order.

|  |  |  |
| --- | --- | --- |
| **Sense[12-13].ASC+ASCQ** | **Sense[3-6].Information** |  |
| **Reply Code** | **Reply Info (bits)** | **Description** |
| PASS | N/A | Message completed successfully. Returned in CSW, without a sense stage. |
| EXIT = 0x8001 | Result (32) | Command terminated early with the indicated results. 0=Pass, NEG=Error, POS=Conditional result. |
| BUSY = 0x8002 | Progress Countdown (32) | Device is busy processing the command. Returned by asynchronous device commands. Host will Poll() to determine completion status. |
| SIZE = 0x8003 | Size in bytes (64) | Returned after the Exec() message of a DeviceIn command to provide payload size to the host. |

The sense key (Sense[2]) for UTP replies is 0x9.

The EXIT reply is used by the device to terminate a UTP transaction. It can signal command terminating errors and is also a means for reporting results that can be used to control the flow of the UCL. For example, a device command could be sent to test the room available for a firmware image. The EXIT reply could then be used to report the result.

One of the possible replies is BUSY. This is included to allow for asynchronous device commands. Certain update functionality such as erasing a media can take a very long time. Instead of waiting for the action to complete, the device should reply BUSY for operations taking longer than 5 secs. The host will send periodic Poll() messages until the device command is completed. This allows the host to report progress and prevents the USB driver from timing out.

The SIZE reply is used for Device Read transactions to inform the host of the payload size. After receiving a device command that returns payload data, the device would send a SIZE reply to the Exec() message so the host knows how much payload data to transfer. Unlike the other replies, the SIZE info field is 64-bits to support payloads larger than 4 GB. The upper 32-bits are sent in the command-specific information field (Sense[8-11]).

**Design Details**

**UTP Transaction State Machine**

The following diagram shows the UTP transaction state machine implemented by the host software.



**Typical UTP Transactions**

The following examples show the UTP message and reply sequences that make up typical UTP transactions.

**Synchronous command transaction without a payload.**

|  |  |
| --- | --- |
| **Exec(), cmd body** | PASS or EXIT |

**Asynchronous command transaction without a payload.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Exec(), cmd body** | BUSY | **Poll()** | BUSY | ... | **Poll()** | PASS or EXIT |

**Synchronous command transaction with a Get payload.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Exec(), cmd body** | SIZE | **Get(seq=0), data0** | PASS | ... | **Get(seq=N), dataN** | PASS or EXIT |

**Synchronous command transaction with a Put payload.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Exec(payload\_size), cmd body** | PASS | **Put(seq=0), data0** | PASS | ... | **Put(seq=N), dataN** | PASS or EXIT |

**Asynchronous command transaction with a Put payload.** (Note that it is also possible to return BUSY on Put() messages.)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Exec(payload\_size), cmd body** | BUSY | **Poll()** | BUSY | ... | **Poll()** | PASS | **Put(seq=0), data0** | PASS | ... | **Put(seq=N), dataN** | PASS or EXIT |

**UTP to SCSI Mapping**

The following tables show how UTP message fields are mapped on to a 16-byte SCSI CDB, and how UTP reply fields are mapped on the SCSI fixed format sense data.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Byte** | **16-Byte SCSI CDB** | | 0 | Operation Code = 0xF0 | | 1 | **UTP Message Type**  (0=Poll, 1=Exec, 2=Get, 3=Put) | | 2  3  4  5 | (MSB)  **UTP Message Tag**  (LSB) | | 6  7  8  9 | (MSB)  **UTP Message Parameter** [upper 32-bits]  (0 for all parameters except payload\_size) | | 10  11  12  13 | **UTP Message Parameter** [lower 32-bits]  (LSB) | | 14  15 | Reserved | | |  |  | | --- | --- | | **Byte** | **Fixed format sense data.** | | 0 | Response Code = 0x70 | | 1 | Per Spec | | 2 | Sense Key = 0x9 | | 3  4  5  6 | (MSB)  **UTP Reply Info** [lower 32-bits]  (LSB) | | 7 | Additional Sense Length (n-7) | | 8  9  10  11 | (MSB)  **UTP Reply Info** [upper 32-bits]  (0 for all replies except SIZE)  (LSB) | | 12 | Additional Sense Code = 0x80 | | 13 | Additional Sense Code Qualifier  **UTP Reply Code**  (0=PASS, 1=EXIT, 2=BUSY, 3=SIZE) | | n | Per Spec | |



