Seagate USM Update Tool User Guide

May 2023
Version 2.7

Revision history

Revision	Date	Change Description
2.1	2019-09-04	Initial release
2.2	2021-08-27	Section 4: Added New Exit Code
2.3	2021-10-27	Section 4: Added New Exit Code
2.4	2021-11-10	Added Username and Password Command
2.5	2022-11-14	Added -ivc command and added new exit code
2.6	2023-05-24	Updated document
2.7	2023-06-29	Added Note for TMPDIR

Table of contents

1 Tool overview

1.1 Introduction

The USM upgrade utility (UUT) is a utility used for updating the firmware on Seagate JBOD enclosures from an attached SAS host. It identifies compatible Seagate JBOD enclosures, determines whether an update is required, and sequences the entire update process to bring the firmware in-line with the levels described in this document.

1.2 Terms and abbreviations

ANSI	American National Standards Institute
AP	Application platform
CLI	Command Line Interface
GEM	Generic Enclosure Management
PSU	Power Supply Unit
SAS	Serial Attached SCSI
SCSI	Small Computer System Interface
SES	SCSI Enclosure Services
USB	Universal Serial Bus
USM	Unified System Management
X86	CPU Complex of an Applications Platform

1.3 Structure

This tool is distributed as a standalone executable package, incorporating the associated firmware components for the enclosures supported by the USM package. When the tool is executed, it extracts the encompassed firmware into a temporary location, updates the enclosure, and cleans up any temporary files created as part of the update.

1.4 Supported interfaces

This tool has been designed to function over the primary enclosure management interfaces for Seagate storage enclosures. This will vary from enclosure to enclosure.

For JBODs, the supported interfaces are:

SCSI - Provides in-band access to update firmware components of storage platform.

1.5 Supported platforms

The tool is expected to run on standard distributions of all platforms. Each platform has its own executable package, more information can be found in the release notes.

2 Usage

2.1 Update scenarios

Below are some of the scenarios that the user may encounter while using the tool.

2.2 In-band update

In-band updates utilize the primary interfaces used to access the storage for updating the firmware, for example, SAS.

The update sequence starts by discovering Seagate enclosure SES targets available within the SCSI topology visible to the host. From the SES target, an enclosure map is constructed along with its target information and WWN. Update is processed on a per enclosure basis. It will update each of the detected enclosure FRUs in the following order:

Case 1 : If a single path to the enclosure is attached to the host running UUT, the following sequence will be applied, assuming the target ID is not overridden using the -t option

a) Transfer, activate and verify Primary expander's components (Bootloader, Firmware, Flash Config, VPDs and CPLD)
b) Transfer, activate and verify Secondary expander's components (Bootlader, Firmware, Flash Config, and VPDs)
c) Transfer, activate and verify Sideplane expander's components (Bootloader, Firmware, Flash Config and VPDs)
d) Transfer, activate and verify Baseplane components (Baseplance CPLD and VPDs)
e) Update and verify Midplane VPD
f) Update and Verify PSU Firmware

Case 2: If multiple paths to the enclosure are attached to the host running UUT, the following sequence will be applied, assuming the target ID is not overridden using the -t option.

a) Transfer Primary expander's components (Bootloader, Firmware, Flash Config, VPDs and CPLD) on IOM 0
b) Transfer Primary expander's components (Bootloader, Firmware, Flash Config, VPDs and CPLD) on IOM 1
c) Activate Primary expander's components on IOM 0
d) Activate Primary expander's components on IOM 1
e) Verify activated components on IOM 0 and IOM 1
f) Transfer Secondary expander's components (Bootloader, Firmware, Flash Config, and VPDs) on IOM0

g) Transfer Secondary expander's components (Bootloader, Firmware, Flash Config, and VPDs) on IOM1
h) Activate Secondary expander's components on IOM 0
i) Activate Secondary expander's components on IOM 1
j) Verify activated components on IOM 0 and IOM
k) Transfer Sideplane expander's components (Bootloader, Firmware, Flash Config and VPDs) on IOM 0
l) Transfer Sideplane expander's components (Bootloader, Firmware, Flash Config and VPDs) on IOM 1
m) Activate Sideplane expander's components on IOM 0
n) Activate Sideplane expander's components on IOM 1
o) Verify activated components on IOM 0 and IOM 1
p) Transfer Baseplane components (Baseplane CPLD and VPDs) on IOM0
q) Transfer Baseplane components (Baseplane CPLD and VPDs) on IOM1
r) Activate Baseplane components on IOM0
s) Activate Baseplane components on IOM1
t) Verify activated components on IOM 0 and IOM 1
u) Update Midplane on IOM0
v) Update Midplane on IOM1
w) Verify activated components on IOM 0 and IOM 1
x) Update PSUs

The above sequence may be subject to change on a per release basis if the specific USM requires an alternative procedure. To minimise the update duration, the update script compares the component versions running on the enclosure against the versions of the firmware in the package. If the versions match, the update is skipped.

3 Tool command and parameters

This is a command line tool with interactive use prompts. For scripting purposes, the interactive prompts may be bypassed using command line options documented below.

Base command:

UUT is packaged as a standalone executable archive containing the enclosure firmware and upgrade logic. The executable is named after the firmware package and is launched from the command line as follows.

Syntax:

```
./<package_name>
```

Example:

```
./UUTXX_XXX_XX
```

Note: Ensure that the package file's execute permission bit is set while executing it on Linux based platforms and the user executing the command has superuser privileges.

TMPDIR:

At runtime, UUT extracts its package content to the operating system's default temporary file location. Along with the enclosure firmware, the package files include libraries and executables that UUT depends upon to interact with the enclosure.

Some Linux operating systems may have restrictions that prevent UUT from executing utilities and dynamic libraries from the default /tmp location, which can cause UUT to fail with a message similar to the following:

```
$ ./UUTX.X –list
./UUTX.X: error while loading shared libraries: libz.so.1: failed to map segment from
shared object: Operation not permitted
```

If the above message is seen when running UUT on Linux, the mount permissions for /tmp can be verified by running the mount command and checking for the noexec flag against the /tmp mount point.

```
$ mount | grep /tmp
tmpfs on /tmp type tmpfs (rw,noexec,nosuid,nodev,nr_inodes=409600)
```

If /tmp is mounted with the noexec flag, the TMPDIR environment variable can be set to point to an alternative location where execution is permitted prior to running UUT. For example:

```
$ mkdir -p /home/curruser/alt-tmp/
$ TMPDIR=/home/curruser/alt-tmp/ ./UUTX.X
```

In the example above, UUT will extract its package files to /home/curruser/alt-tmp instead of the default /tmp location.

3.1 Option parameters

This section details the additional command line options supported by UUT. These options are used to modify the way UUT behaves and allows a user to take control of certain aspects of the update process.

3.1.1 -t/--target

Using this option, a user can supply a comma separated list of the target device(s) to update. The target device can be a SCSI handle (/dev/sg*). In the case of a local update, a user can specify the handle (/dev/sg*) of a specific SCSI target device associated with the enclosure that UUT should use to perform the update. When this option is not specified, UUT attempts to auto-detect the attached enclosures and will select the most appropriate SCSI targets to use to update them.

Syntax:

```
./<package_name> -t /dev/sg*
```

Example:

```
./UUTXX_XXX_XX --target /dev/sg24
```

3.1.2 –stages

The update process comprises of multiple stages. Each stage consists of steps or actions that are required to be executed to update or verify the firmware of a component, or a group of related components. Each stage can be uniquely identified by its name. The name of the stage describes the actions that will be performed as part of the stage. This option displays information for all stages that are part of the update process, and includes the name of the stage, whether the stage is disruptive to I/O, and it's estimated execution time.

Syntax:

```
./<package_name> --stages
```

Example:

```
./UUTXX_XXX_XX --stages
```

3.1.3 –ls/--list

Before proceeding with the update, a tool provides facility for a user to see firmware version of components on the target and its comparison with respective version in the USM to get an idea of components that are eligible for updates. This option enables a user to invoke this facility. It also lists backup version of the components if it's available. The version comparison is displayed in tabular form with the help of Red, Green, White and Yellow colours if coloured output is enabled. Below table gives significance of each colour in the comparison table.

Red	Green	White	Yellow
Mismatch between target and USM version of component	Target and USM version of component is same	Component can't be updated due to certain limitations. Or if component's target version is not available.	Backup version if it's available

Syntax:

```
./<package_name> --list
```

3.1.4 -d/--dryrun

This option mocks the update process. When this option is used a tool will simulate all the steps of the update process without executing them. It is implemented to give a user an idea of all the steps that will be performed during the update process before performing the update.

Syntax:

```
./<package_name> --dryrun
```

Example:

```
./UUTXX_XXX_XX -d
```

3.1.5 -ff/--flipflop

This option directs a tool to select flip-flop image of the USM for update if it's available.

Syntax:

```
./<package_name> -ff
```

Example:

```
./UUTXX_XXX_XX -ff
```

3.1.6 –frus

This option can be used to update specific config on the system. If this option is provided during the update, then default config will be skipped, and the user provided config will be loaded on the system. A user can pass comma separated list of such FRU keywords.

Syntax:

```
./<package_name> --frus <fru_tag>
```

Example:

```
./UUTXX_XXX_XX --frus scsn
```

Note: Please refer to UUT Release notes for supported fru options. Above "--frus scsn" is just an example.

3.1.7 --skip

This option enables user to skip updating components even if there's a mismatch between their target and USM version. A user needs to pass the name of the stage to which that component belongs (which can be found out by using -cm/--component map option) as parameter to this option. If user wants to skip multiple stages in the update sequence, then user can do so by passing a comma separated list of stages to skip. It is mandatory to supply the stage name in quotes. The tool will ask for confirmation at the time of execution if user really wants to skip the stage(s). Note that in case a stage to be skipped has a dependent stage then the dependent stage will be skipped as well.

Syntax:

```
./<package_name> --skip "Primary Firmware Update (Deferred)"
```

Examples:

```
./UUTXX_XXX_XX --skip "Primary Firmware Update (Deferred)"
```

```
./UUTXX_XXX_XX --skip "Primary Firmware Update (Deferred)","Primary Bootloader Update (Deferred)"
```

3.1.8 --force

The tool skips update stage(s) of all those components on target whose firmware version match with the respective version in the USM. However, if a user explicitly wants one or more stages to be executed regardless of whether they can be skipped or not then a user can provide a comma separated list of stages as argument to this parameter. It is mandatory to supply the stage name in quotes. A user can also force a tool to execute all the stages by passing a string "all" as value to this parameter. The tool will then execute all the stages in the update sequence one by one. A user needs to be very careful while exercising "force" option because it overrides the default behaviour or checks that tool performs while executing forced stage.

Syntax:

```
./<package_name> --force "stage_name"
```

```
./<package_name> --force all
```

Examples:

```
./UUTXX_XXX_XX --force "Primary Firmware Update (Deferred)"
```

```
./UUTXX_XXX_XX --force "Primary Firmware Update (Deferred)","Primary Bootloader Update (Deferred)"
```

```
./UUTXX_XXX_XX --force all
```

3.1.9 -l/--logfile

UUT, by default, stores generated logs in a file named *debug.log,* located in the *usm_upgrade_tool/logs* directory, which is created under the home directory of a user executing the update command. This option enables a user to specify an alternative location to store the logs.

Syntax:

```
./<package_name> --logfile <path_to_store_logs>
```

Examples:

```
./UUTXX_XXX_XX -l /tmp/usm_upgrade/upgrade.log
```

3.1.10 –syslog

This option enables logging to system log file which means that logs generated by tool will be stored in system log file (i.e. */var/log/messages* on RedHat system and */var/log/syslog* on Debian system*)* as well. Tool auto detects the system's log file to store the logs

Syntax:

```
./<package_name> --syslog
```

Examples:

```
./UUTXX_XXX_XX --syslog
```

3.1.11 -uv/--usmver

This option displays the version of the USM bundled with the tool package.

Syntax:

```
./<package_name> -uv
```

Examples:

```
./UUTXX_XXX_XX --usmver
```

3.1.12 -y/--yes

The UUT may request input from a user in the form of yes or no to review an action and proceed with the update or not. This option directs UUT to assume a "yes" response from a user and proceed with the action unprompted.

Syntax:

```
./<package_name> -y
```

Examples:

```
./UUTXX_XXX_XX -yes
```

3.1.13 -c/--continueonfail

A typical update process comprises of multiple stages that are executed sequentially. By default, the tool terminates the update process if any of the stage fails to execute successfully. But a user can direct the tool to continue with process even if any of the intermediate stage does not execute successfully by using this option.

Syntax:

```
./<package_name> -c
```

Examples:

```
./UUTXX_XXX_XX --continueonfail
```

3.1.14 -nc/--nocolour

A tool displays the update information on console in various colours so that it's easier to identify various aspects of the update process as it's going on. A user however can disable colour formatting in the output by using this option.

Syntax:

```
./<package_name> -nc
```

Example:

```
./UUTXX_XXX_XX --nocolour
```

3.1.15 -V/--version

This option displays the version of a tool being used.

Syntax:

```
./<package_name> -V
```

Example:

```
./UUTXX_XXX_XX --version
```

3.1.16 -cm/--componentmap

Each stage in the update sequence updates specific firmware component(s) and a user can find out which firmware component(s) is part of which stage with the help of this option.

Syntax:

```
./<package_name> -cm
```

Example:

```
./UUTXX_XXX_XX --componentmap
```

3.1.17 -nul/--noutilitylogs

A tool by default collates logs (from console and file) of underlying utilities that tool uses. These logs are stored in the same directory where tool's log file is stored. However, if a user doesn't want tool to collate logs, he can direct tool to do with the help of this option. When this option is specified, the tool will not collate logs.

Syntax:

```
./<package_name> -nul
```

Example:

```
./UUTXX_XXX_XX --noutilitylogs
```

3.1.18 -v/--verbosity

Default verbosity level of the tool is WARNING meaning only those log statements will be logged on a console whose log level either matches or is higher than WARNING. This command line option enables the user to get more detailed logs. Specifying this option once will set the log level to ERROR, specifying it twice will set it INFO and specifying it thrice will set it to DEBUG which is the highest level of verbosity.

Syntax:

```
./<package_name> -v
```

Example:

```
./UUTXX_XXX_XX --verbosity
```

3.1.19 -oc/--onlycanister

This option enables a user to update only canister/controller specific components on the target system. When this option is used the tool will not update components which are common or updatable from both controllers (i.e. enclosure components) in a dual controller system even if there's a mismatch between their target and USM version.

Syntax:

```
./<package_name> -oc
```

Example:

```
./UUTXX_XXX_XX --onlycanister
```

3.1.20 -oe/--onlyenclosure

This option enables a user to update only such components on the target system that are updatable from both controllers/canisters (i.e., enclosure components) in a dual controller system. When this option is used a tool will not update canister/controller specific components on the target system even if there's a mismatch between their target and USM version.

Syntax:

```
./<package_name> -oe
```

Example:

```
./UUTXX_XXX_XX --onlyenclosure
```

3.1.21 -nt/--notimestamp

Log statements generated by tool are by default prefixed with the timestamp. A user can disable this timestamp using this option.

Syntax:

```
./<package_name> -nt
```

Example:

```
./UUTXX_XXX_XX --notimestamp
```

3.1.22 -eo/--executeonly

Using this option, a user can update selective component(s) on the target. A user needs to pass the stage name for which that component belongs (which can be found out by using -cm/--componentmap option) as a parameter to this option. If a user wants to execute multiple stages, then user can do so by passing a comma separated list of stages to be executed. It is mandatory to supply stage name in quotes. A tool will ask for confirmation at the time of execution if user really wants to execute the stage(s).

When this option is exercised tool will skip all other stages and will execute stage(s) passed to this option only if it's eligible of execution i.e., if there's a mismatch between target and USM version of corresponding components and there's no update constraint.

Syntax:

```
./<package_name> -eo "stage name"
```

Example:

```
./UUTXX_XXX_XX --executeonly  "PSU Update"
```

3.1.23 --username and --password

Using this option username and password is provided to UUT.

Syntax:

```
./<package_name> --username <name> --password <password>
```

Example:

```
./UUTXX_XXX_XX --username manage --password P@ssw0rd
```

3.1.24 -ivc/--ignore_version_check

Using this option "N/N+1" firmware check is ignored, and firmware is updated.

Syntax:

```
./< package_name> -ivc
```

Example:

```
./UUTXX_XXX_XX -ignore_version_check
```

4 Tool exit codes

The USM Upgrade Tool version 2.5 supports the following exit codes.

Exit Code	Description
0	Operation performed successfully
1	Failed to perform operation successfully
2	Operation Skipped
3	Operation performed by underlying utility failed
4	Upgrade to be performed is disruptive in nature
5	Target version not matching with USM version
6	User entered negative input
7	Utility method not implemented
8	Unable to detect target specified for USM upgrade
9	Unable to detect enclosure on target
10	Unable to detect slot connected to
11	Not enough data to fetch expected information
12	Operation failed due to unknown reason
13	Schema validation failed
14	Specified path does not exist
15	Keyboard interrupt occurred
16	Could not discover role of target
17	Failed to import hook module
18	Operation performed by hook failed
19	Hook method is not implemented
20	Device is not ready for upgrade
21	User entered invalid input
22	Partner update is in progress
23	Partner is not updating
24	Failed to establish SSH connection with target
25	Invalid user
26	Password Expired
27	All firmware versions on the enclosure match the package contents
28	Firmware versions on the enclosure are different than the package contents
29	Invalid username or password
30	Firmware is missing signature
31	N/N+1 firmware condition not match