USB Device Mass Storage Class Driver User's Guide

Version 3.00

For use with USBD Mass Storage Class Driver Version 6.06 and above

Date: 27-Aug-2014 13:54

All rights reserved. This document and the associated software are the sole property of HCC Embedded. Reproduction or duplication by any means of any portion of this document without the prior written consent of HCC Embedded is expressly forbidden.

HCC Embedded reserves the right to make changes to this document and to the related software at any time and without notice. The information in this document has been carefully checked for its accuracy; however, HCC Embedded makes no warranty relating to the correctness of this document.
# Table of Contents

System Overview .......................................................... 3  
Introduction ...................................................................... 3  
  Operation ........................................................................ 4  
  Getting Started .............................................................. 4  
Feature Check ..................................................................... 4  
Packages and Documents .................................................. 5  
  Packages ......................................................................... 5  
  Documents ....................................................................... 5  
Source File List ................................................................... 6  
  API Header Files ............................................................. 6  
  Configuration Files .......................................................... 6  
  Version File ...................................................................... 6  
Mass Storage System ......................................................... 7  
Configuration Options ....................................................... 8  
  System Configuration Options ........................................... 8  
  SCSI Configuration Options ............................................. 9  
    SCSI Identification Information ..................................... 9  
    config_scsi_tgt.c .......................................................... 10  
API .................................................................................... 11  
  Module Management ....................................................... 11  
    mstd_init ...................................................................... 11  
    mstd_start .................................................................... 12  
    mstd_stop ..................................................................... 13  
    mstd_delete ................................................................... 14  
    mstd_is_idle .................................................................. 15  
SCSI Device Management .................................................. 16  
  scsis_disable_lun ............................................................ 16  
  scsis_enable_lun .............................................................. 17  
  scsis_lun_disabled .......................................................... 18  
  scsis_media_remove_allowed .......................................... 19  
MST Device Management .................................................... 20  
  mstd_reg_unknown_rx_cb ................................................ 20  
Error Codes ........................................................................ 21  
Types and Definitions ....................................................... 22  
  scsim_table_entry_t ........................................................ 22  
  t_vmst_com_unknown_rx_cb ............................................. 22  
Integration .......................................................................... 23  
OS Abstraction Layer (OAL) ............................................... 23  
PSP Porting .......................................................................... 24
1 System Overview

1.1 Introduction

This guide is for those who want to implement an Embedded USB Device Mass Storage (MST) class driver. The USB Mass Storage class is designed for connecting block storage devices over a USB link to a host. The MST package is a function device implementation of this class. It allows a device to connect to a host system and be seen by the host system as one or more locally connected storage drives. This allows attachment of a media driver (that controls, for example, an SD card) to a PC over the Mass Storage interface, making it look much like a standard pen drive.

The system structure is shown in the diagram below:

On the embedded device side, these drives can be any block-addressable storage media that conform to the HCC Media Driver Interface Specification. If provided as described in the SCSI Device Management section of this guide, it enables a host system to use that drive as locally connected storage.

Each drive is called a Logical Unit in USB terminology and is referenced by its Logical Unit Number (LUN).
Note:

- This product conforms to the *Universal Serial Bus Class Definition for Mass Storage Devices Version 1.1*. This recommended reference may be downloaded from [www.usb.org](http://www.usb.org).
- This module is part of HCC’s Embedded USB Device (EUSBD) System, as described in the *HCC Embedded USB Device Base System User’s Guide*. This module communicates with the EUSBD base system through the EUSBD Device Interface, as described in the above manual.

**Operation**

This class driver consists of:

- A Mass Storage section – this processes Mass Storage requests from the host (in a task) and transfers data between the host and the SCSI slave driver.
- SCSI driver master and slave sections – the SCSI slave driver is the interface between the MST and the SCSI master, which contains a table of interface functions used to access the media driver.

However, developers do not need to understand the details of this interaction.

**Getting Started**

To use this class driver, do the following:

1. Generate a device configuration that includes an MST interface.
2. Set the configurable parameters as described in Configuration Options.
3. Call the `mstd_init()` function to initialize the class driver.
4. Create the Mass Storage task (mst_task()) that handles Mass Storage control protocol messages.
5. Call the `mstd_start()` function to start the drive(s).

After these steps are complete, and if the device is connected to the USB host, the configured drives will be available to the host system.

### 1.2 Feature Check

The main features of the class driver are the following:

- It conforms to the HCC Advanced Embedded Framework.
- It supports all devices that conform to the USB MST specification.
- It supports all devices that conform to the *HCC Media Driver Interface* specification.
- It is compatible with sample device files produced by using the *HCC USB Device Descriptor Generator*.
- It allows you to specify a callback for Unknown Data Received events.
1.3 Packages and Documents

Packages

The table below lists the packages that you need in order to use this module:

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcc_base_doc</td>
<td>This contains the two guides that will help you get started.</td>
</tr>
<tr>
<td>usbd_base</td>
<td>The USB device base package. This is the framework used by USB class drivers to communicate over USB using a specific USB device controller package.</td>
</tr>
<tr>
<td>usbd_cd_mst</td>
<td>The USB device MST class driver package described by this document.</td>
</tr>
</tbody>
</table>

Documents

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the Quick Start Guide when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC Embedded USB Device Base System User's Guide

This document defines the USB device base system upon which the complete USB stack is built.

HCC USB Device Mass Storage Class Driver User's Guide

This is this document.
2 Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

**Note:** Do not modify any files except the configuration files.

2.1 API Header Files

These files are the only files that should be included by an application using this module. **These files should only be modified by HCC.** For details of the API functions, see API.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src/api/api_scsi_tgt.h</td>
<td>SCSI header file.</td>
</tr>
<tr>
<td>src/api/api_usbd_mst.h</td>
<td>Mass Storage handler header file.</td>
</tr>
</tbody>
</table>

2.2 Configuration Files

These files contain all the configurable parameters. Configure these as required. For details of these options, see Configuration Options.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src/config/config_scsi_tgt.c</td>
<td>Source code for the media driver interface.</td>
</tr>
<tr>
<td>src/config/config_scsi_tgt.h</td>
<td>Defines the SCSI drive table.</td>
</tr>
<tr>
<td>src/config/config_usbd_mst.h</td>
<td>Header file for the Mass Storage handler.</td>
</tr>
</tbody>
</table>

2.3 Version File

The file `src/version/ver_usbd_mst.h` contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.
2.4 Mass Storage System

These files should only be modified by HCC.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>src/usb-device/class-drivers/mass-storage/scsi_tgt.c</code></td>
<td>SCSI source file.</td>
</tr>
<tr>
<td><code>src/usb-device/class-drivers/mass-storage/scsi_tgt.h</code></td>
<td>SCSI header file.</td>
</tr>
<tr>
<td><code>src/usb-device/class-drivers/mass-storage/usbd_mst.c</code></td>
<td>MST source file.</td>
</tr>
<tr>
<td><code>src/usb-device/class-drivers/mass-storage/usbd_mst.h</code></td>
<td>MST header file.</td>
</tr>
</tbody>
</table>
3 Configuration Options

All system configuration options are set in three configuration files. This section lists the available configuration options and their default values.

3.1 System Configuration Options

Set the system configuration options in the file src/config/config_usbd_mst.h.

MST_BUFFER_SIZE
The size of one RAM buffer. This must be longer than the sector size of the media. The default is 1 * 1024U.

MST_OVERLAPPED
Specifies whether Mass Storage transfers should be performed in overlapped mode. The default is 1.

If this is enabled, two buffers of MST_BUFFER_SIZE are allocated. This usually increases the transfer speed.

MST_EVENT_FLAG
The default is 1.

MST_USE_MST_IF
Enable this to use the normal MST interface. The default is 1.

MST_USE_VMST_IF
Enable this to use the Virtual MST (VMST) interface. The default is zero. It is possible to use both the MST and VMST interfaces together.

USBD_MST_TASK_PRIO
The task priority. The default is OAL_NORMAL_PRIORITY.

USBD_MST_STACK_SIZE
The stack size for MST. The default is 1024.

USBD_VMST_TASK_PRIO
The VMST task priority. The default is OAL_NORMAL_PRIORITY.

USBD_VMST_STACK_SIZE
The stack size for VMST. The default is 1024.
3.2 SCSI Configuration Options

Set the SCSI configuration options in the file `src/config/config_scsi_tgt.h`.

**N_LUNS_MST**

The number of supported Logical Units (LUNs) for MST. The default is 1U.

**N_LUNS_VMST**

The number of supported Logical Units for Virtual MST (VMST). The default is 1U.

**BOOT_MODE_ENABLE**

Set this to 1 (the default) if the device is used as boot device.

**SCSI Identification Information**

The following options control the information sent in the response to the inquiry command. The length of these strings must match that specified below; use space characters if necessary to pad to the required size.

**SCSI_VENDOR**

The 8 byte long vendor id. The default is:

"HCC     ".

**SCSI_PRODUCT**

The 16 byte long product name. The default is:

"test-drive     ".

**SCSI_VERSION**

The 4 byte long version number. The default is "1.00".
3.3 config_scsi_tgt.c

The file `src/config/config_scsi_tgt.c` defines the drive table which makes the link from the Mass Storage to the drive. This must conform to the HCC Media Driver Interface specification.

The way this is used depends on your settings for the MST_USE_MST_IF and MST_USE_VMST_IF options.

If you use the normal MST interface, the table is defined as follows, making the CD partition visible via MST:

```c
scsim_table_entry_t  scsi_media_table_mst [N_LUNS_MST] =
{ mmcsd_initfunc, 0, 0 }
};
```

If you use the Virtual MST (VMST) interface, the table is defined as follows:

```c
scsim_table_entry_t  scsi_media_table_vmst[N_LUNS_VMST] =
{ xxx_initfunc, 0, 0 }
};
```
4 API

This section documents the Application Programming Interface. It includes all the functions that are available to an application program.

4.1 Module Management

mstd_init

Use this function to initialize the class driver and allocate the required resources.

Format

```
int mstd_init ( void )
```

Arguments

Parameter

None.

Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSTD_SUCCESS</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>MSTD_ERROR</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>
**mstd_start**

Use this function to start the class driver.

**Note:** You must call `mstd_init()` before this function to initialize the module.

**Format**

```c
int mstd_start ( void )
```

**Arguments**

- **Parameter**
  - None.

**Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSTD_SUCCESS</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>MSTD_ERROR</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>
**mstd_stop**

Use this function to stop the class driver.

**Format**

```c
int mstd_stop ( void )
```

**Arguments**

| Parameter  | None. |

**Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSTD_SUCCESS</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>MSTD_ERROR</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>
**mstd_delete**

Use this function to delete the class driver and release the associated resources.

**Format**

```c
void mstd_delete ( void )
```

**Arguments**

- **Parameter**
  - None.

**Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSTD_SUCCESS</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>MSTD_ERROR</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>
**mstd_is_idle**

Use this function to check for an idle state.

**Format**

```
int mstd_is_idle ( uint8_t if_idx )
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>if_idx</td>
<td>The index of the MST or VMST interface.</td>
<td>uint8_t</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>Interface is not idle.</td>
</tr>
<tr>
<td>1</td>
<td>Interface is idle.</td>
</tr>
<tr>
<td>MSTD_ERROR</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>
4.2 SCSI Device Management

**scsis_disable_lun**

Use this function to disable the specified Logical Unit (LUN).

This function disables access to this media.

**Format**

```c
int scsis_disable_lun ( uint8_t   if_idx,
                       uint8_t   lun )
```

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>if_idx</td>
<td>The index of the MST or VMST interface.</td>
<td>uint8_t</td>
</tr>
<tr>
<td>lun</td>
<td>The number of the logical unit to disable.</td>
<td>uint8_t</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSI_TGT_SUCCESS</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>SCSI_TGT_ERROR</td>
<td>Failure.</td>
</tr>
</tbody>
</table>


**scsis_enable_lun**

Use this function to enable the specified Logical Unit (LUN).

**Format**

```c
int scsis_enable_lun (
    uint8_t   if_idx,
    uint8_t   lun,
    uint8_t   readonly )
```

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>if_idx</td>
<td>The index of the MST or VMST interface.</td>
<td>uint8_t</td>
</tr>
<tr>
<td>lun</td>
<td>The number of the logical unit to enable.</td>
<td>uint8_t</td>
</tr>
<tr>
<td>readonly</td>
<td>If this is set, initialize the driver as read-only.</td>
<td>uint8_t</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSI_TGT_SUCCESS</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>SCSI_TGT_ERROR</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>
**scsis_lun_disabled**

Use this function to test whether the specified Logical Unit (LUN) is disabled.

**Format**

```c
int scsis_lun_disabled (  
    uint8_t   if_idx,  
    uint8_t   lun )
```

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>if_idx</td>
<td>The index of the MST or VMST interface.</td>
<td>uint8_t</td>
</tr>
<tr>
<td>lun</td>
<td>One of the following:</td>
<td>uint8_t</td>
</tr>
<tr>
<td></td>
<td>· Zero = LUN is enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>· 1 = LUN is disabled.</td>
<td></td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSI_TGT_SUCCESS</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>SCSI_TGT_ERROR</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>
**scsis_media_remove_allowed**

Use this function to return a "media remove allowed" flag sent by the host.

**Format**

```c
int scsis_media_remove_allowed ( 
    uint8_t     if_idx, 
    uint8_t     lun, 
    uint8_t *   allowed )
```

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>if_idx</td>
<td>The index of the MST or VMST interface.</td>
<td>uint8_t</td>
</tr>
<tr>
<td>lun</td>
<td>The number of the logical unit.</td>
<td>uint8_t</td>
</tr>
<tr>
<td>allowed</td>
<td>A pointer to the allowed flag.</td>
<td>uint8_t *</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSI_TGT_SUCCESS</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>SCSI_TGT_ERROR</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>
4.3 MST Device Management

mstd_reg_unknown_rx_cb

Use this function to register a callback for Unknown Data Received events.

Note:
- This is only suitable for a VMST interface.
- It is the user's responsibility to provide any callback functions the application requires. Providing such functions is optional.

Format

```c
int mstd_reg_unknown_rx_cb ( t_vmst_com_unknown_rx_cb vmst_com_unknown_rx_cb )
```

Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmst_com_unknown_rx_cb</td>
<td>The type of callback.</td>
<td>t_vmst_com_unknown_rx_cb</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSTD_SUCCESS</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>MSTD_ERROR</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>
4.4 Error Codes

If a function executes successfully, it returns with a SUCCESS code, a value of zero. The following table shows the meaning of the error codes.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSI_TGT_SUCCESS</td>
<td>0</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>SCSI_TGT_ERROR</td>
<td>1</td>
<td>Failed SCSI operation.</td>
</tr>
<tr>
<td>MSTD_SUCCESS</td>
<td>0</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>MSTD_ERROR</td>
<td>1</td>
<td>Failed MST operation.</td>
</tr>
</tbody>
</table>
### 4.5 Types and Definitions

#### scsim_table_entry_t

This structure is used to build the LUN list:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver_init</td>
<td>Media driver init function.</td>
<td>F_DRIVERINIT</td>
</tr>
<tr>
<td>param</td>
<td>Media driver parameter.</td>
<td>uint32_t</td>
</tr>
<tr>
<td>iso</td>
<td></td>
<td>uint8_t</td>
</tr>
</tbody>
</table>

#### t_vmst_com_unknown_rx_cb

The `t_vmst_com_unknown_rx_cb` definition specifies the format of the callback function which will be called when a packet of unknown size arrives.

**Format**

```c
typedef void ( * t_vmst_com_unknown_rx_cb )(
    uint8_t *   p_buf,
    uint32_t    length )
```

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_buf</td>
<td>A pointer to the buffer containing the data.</td>
<td>uint8_t *</td>
</tr>
<tr>
<td>length</td>
<td>The length of the data pointed to by <code>p_buf</code>.</td>
<td>uint32_t</td>
</tr>
</tbody>
</table>
5 Integration

This section specifies the elements of this package that need porting, depending on the target environment.

5.1 OS Abstraction Layer (OAL)

All HCC modules use the OS Abstraction Layer that allows the module to run seamlessly with a wide variety of RTOSes, or without an RTOS.

This module uses the following OAL components:

<table>
<thead>
<tr>
<th>OAL Resource</th>
<th>Number Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>1</td>
</tr>
<tr>
<td>Mutexes</td>
<td>1</td>
</tr>
<tr>
<td>Events</td>
<td>2</td>
</tr>
</tbody>
</table>
5.2 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer.

The MST module makes use of the following standard PSP functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Package</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psp_memcpy()</td>
<td>psp_base</td>
<td>psp_string</td>
<td>Copies a block of memory. The result is a binary copy of the data.</td>
</tr>
<tr>
<td>psp_memset()</td>
<td>psp_base</td>
<td>psp_string</td>
<td>Sets the specified area of memory to the defined value.</td>
</tr>
</tbody>
</table>

The MST module makes use of the following standard PSP macros:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Package</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSP_RD_BE32</td>
<td>psp_base</td>
<td>psp_endianness</td>
<td>Reads a 32 bit value stored as big-endian from a memory location.</td>
</tr>
<tr>
<td>PSP_RD_LE32</td>
<td>psp_base</td>
<td>psp_endianness</td>
<td>Reads a 32 bit value stored as little-endian from a memory location.</td>
</tr>
<tr>
<td>PSP_WR_BE16</td>
<td>psp_base</td>
<td>psp_endianness</td>
<td>Writes a 16 bit value to be stored as big-endian to a memory location.</td>
</tr>
<tr>
<td>PSP_WR_BE32</td>
<td>psp_base</td>
<td>psp_endianness</td>
<td>Writes a 32 bit value to be stored as big-endian to a memory location.</td>
</tr>
<tr>
<td>PSP_WR_LE32</td>
<td>psp_base</td>
<td>psp_endianness</td>
<td>Writes a 32 bit value to be stored as little-endian to a memory location.</td>
</tr>
</tbody>
</table>