MonsterX CEC SDK

V 1.0.0

SKNET Corporation
2011/04/03

Change History

V 1.0.0	- Initial release

CHAPTERS

1. Introduction

Consumer Electronics Control (CEC) is a protocol that provides high-level control functions between all of the various HDMI audio/visual products in a user's environment. Through the CEC commands, MonsterX3 device can be used to control the connected HDMI devices such as:

- Power Up and Down HDMI devices
- Basic video operation such as start/stop, pause, FFWD, FRWD playback.
- Get basic playback information
- Add/Edit/Delete schedule.

It is not compulsory for HDMI device manufacturer to implement all CEC commands. Also, each manufacturer could optionally implement their own custom CEC commands. However, if a particular command (opcode) is implemented, it must always be conform to the HDMI specification. As such, to develop a CEC application, you must obtain a copy of the HDMI 1.3a specification. In this specification, the CEC specification is explained in "Supplement 1 – Consumer Electronics Control (CEC)" section.

This SDK allows developers to develop custom CEC applications for

- MonsterX3

OS Platform supported:

- Windows XP (32-bit)
- Windows Vista (32-bit/64-bit)
- Windows 7 (32-bit/64-bit)

Supported HDMI version:

- 1.3a

Development tools requirement:

- Visual Studio 2008 with Visual C++ compiler

SDK files:

.¥bin¥MX_CECUtil.exe	Pre-compiled sample application file
.¥bin¥MX_CECMgr.dll	Core CEC library

.¥bin¥CECData.ini	CEC commands created and saved by MX_CECUtil application.
.¥src¥	CEC sample application source codes
.¥doc¥UserManual	The user manual for the SDK.

2. Development Guide

Before you begin with the development, you should

1) Have the MonsterX3 application and driver installed.
2) Obtain a copy of the HDMI 1.3a specification

How to start developing a CEC application

1) Create a sample MFC application in Microsoft Visual Studio 2008
2) Add the .¥src¥MX_CECUtil¥CECMgr.cpp and .¥src¥MX_CECUtil¥CECMgr.h into the sample application's project.
3) Instantiate a CCECMgr object in the application.
4) Initialize the CCECMgr object by calling the CCECMgr::Init() function.
5) If initialization is successful, you can either create a Windows timer or a slave thread to monitor for any incoming CEC data. These data could contain information such as:
   - Executed command status (Reason for failure if command failed)
   - Miscellaneous information like device logical address, playback status.
6) Below is an example on how to listen for incoming CEC data.

```
IncomingDataLoop()
{
   BOOL bTemp = FALSE, bHasData = FALSE;
   if (g_CECMgr.HasReadData(&bTemp))
      bHasData = bTemp;
      if (bHasData)
      {
         BYTE CECReadData[20];
         Int iNumReadData = 20;
         ZeroMemory(CECReadData, sizeof(CECReadData));
         g_CECMgr.ReadData(CECReadData, &iNumReadData);
      }
   }
```

If call to `ReadData` function is successful, `CECReadData` will contain the incoming CEC data and `iNumReadData` will contain the number of data.

7) Below is an example to write a CEC command

```
BYTE CECData[5];
ZeroMemory(CECData,sizeof(CECData));

//Send Power on command a DVR1 device
int iTargetDevice;
iTargetDevice = 1; //Logical device address for DVR1 (Please refer to HDMI spec)
CECData[0] = 0x44; //Opcode for remote command (Please refer to HDMI spec)
CECData[1] = 0x40; //Power-On parameter (Please refer to HDMI spec)
g_CECMgr.WriteData(iTargetDevice,CECData,2);

//Broadcast power off command to all connected HDMI devices
iTargetDevice = 0xf //Broadcast address (Please refer to HDMI spec)
CECData[0] = 0x36 //Opcode for Power-Off (Please refer to HDMI spec)
g_CECMgr.WriteData(iTargetDevice,CECData,1);
```

3. SDK Function Reference

`int MX_CECMgr_Init()`

Return

0 – Initialization successful.

2 – Initialization failed

Description

This function loads the CEC library and checks for the existence of MonsterX3 hardware. This is the first function that must be called before calling any other CEC SDK functions. If value 2 is returned, it is most probably due to the driver or the MonsterX3 device not properly installed.

.

`void MX_CECMgr_Uninit()`

Description

This function unloads the CEC library. No CEC SDK function should be called after unloading the CEC library.

`int MX_CECMgr_SetDiscoverDevice(int iDevIndex)`

Parameter

`int iDevIndex`

Logical role of HDMI device (0x1 to 0xe)

According to the HDMI specification, the range of devices' logical role is as below:

0x0 – TV (Not used as MonsterX3 device is regarded as TV device)

0x1 – DVR1	0x8 – PLYR2
0x2 – DVR2	0x9 – DVR3
0x3 – TUNER1	0xa – TUNER4
0x4 – PLYR1	0xb – PLYR3
0x5 – AUDIO	0xc – Reserve1
0x6 – TUNER2	0xd – Reserve2
0x7 – TUNER3	0xe – Free

Return

0 – Successful.

1 – Failed (Initialization was not called)

Description

This function is called to discover device roles that are connected to the MonsterX3 device via the HDMI cable. It is possible that a single device that is connected to the MonnsterX3 device could have more than 1 role. For example, a DVR player could have a role of DVR1 and TUNER1 at the same time. Please refer to the HDMI specification for more detail. After this function is called, if a connected device has the appropriate role, it will respond and the incoming CEC data timer/thread loop should immediately detect this response to confirm the availability of this role.

`int MX_CECMgr_HasReadData(BOOL *pbStatus)`

Parameter

`BOOL *pbStatus`	Return status to determine the availability of incoming CEC data. Valid only if function call is successful TRUE – has incoming CEC data FALSE – has no incoming CEC data

Return

0 – Successful.

1 – Failed (Initialization was not called)

3 – Invalid parameter

Description

This function must be called in a timer or a thread loop to constantly monitor for any incoming CEC data. When the availability of incoming CEC data is detected, `MX_CECMgr_ReadData` function should be called to retrieve the data. Even if the data is not needed, `MX_CECMgr_ReadData` must be called to clear the incoming CEC data buffer or otherwise, `pbStatus` will always return TRUE.

Please refer to “2. Development Guide” chapter on how to monitor for incoming CEC data.

`int MX_CECMgr_ReadData(BYTE *pbData, int *piNumData)`

Parameter

`BYTE *pbData`	Byte array which will be used to store incoming CEC data.
`int *piNumData`	Size of pbData byte array. If the function call is successful, this parameter will contain the actual number of bytes returned.

Return

0 – Successful.

1 – Failed (Initialization was not called)

3 – Invalid parameter

4 – Byte array size is too small to store all incoming CEC data

Description

This function is called to retrieve all incoming CEC Data. This function should only be called when incoming CEC data availability is detected via `MX_CECMgr_HasReadData`. Also, this function must always be called whenever there is incoming CEC data, even if it is not needed otherwise `MX_CECMgr_HasReadData` will always return a positive result.

Please refer to “2. Development Guide” chapter on how to monitor for incoming CEC data.

`int MX_CECMgr_WriteData(int iTargetDev, BYTE *pbData, int *piNumData)`

Parameter

`int iTargetDev`	Target device role (0x1 to 0xe for specific target role or 0xf for broadcase)
`BYTE *pbData`	Byte array which will be used to store incoming CEC data.
`int iNumData`	Size of pbData byte array.

Return

0 – Successful.

1 – Failed (Initialization was not called)

3 – Invalid parameter

Description

This function is called to send a CEC command. pbData should typically contain an opcode and optionally one or more parameters, depending on what opcode is to be sent.

Please refer to “2. Development Guide” chapter on how to send a CEC command.