ENTTEC RDM STACK

Controller Library

Public Functions

Conforms to RDM Standard ANSI E1.20 -2006

www.enttec.com/rdm
Version 1.65
25/05/2007

ENTTEC RDM Controller Stack Documentation

Functions

- uint16_t RDMInit (int DeviceNum, int *VersionMSB, int *VersionLSB)
- void RDMTick (void)
- void RDMComms (void)
- uint16_t RDMDiscovery(uint8_t flag, volatile uint8_t* Ready, uint8_t* Message)
- uint16_t GetDeviceCount (void)
- void GetDeviceID(uint8_t position, uint8_t * DeviceArray)
- void RDMSet (uint8_t *DestinationUID, uint16_t ParamID, uint8_t DataLength, void* Data, void* ResponseData, uint32_t *max_size, uint32_t Timeout,uint16_t Sub_Device, uint16_t ResponseType)
- void RDMGet (uint8_t *DestinationUID, uint16_t ParamID, uint8_t DataLength, void* Data, void* ResponseData, uint32_t *max_size, uint32_t Timeout, uint16_t Sub_Device ,uint16_t ResponseType)
- void ClosePort (void)

Detailed Description

This documentation covers all the functions required to use ENTTEC RDM Stack for Controllers. The included library 'Controller_Lib.lib' is a Visual C library intended to be used with C/C++ code compiled under Visual Studio environment.

Note:

The way RDM Stack has been defined, it is imperative that the RDMTick() Function [Timer Control] and the RDMComms() Function [Send / Receive RDM] should be called in separate threads (run every millisecond).

Function Documentation

uint16_t RDMInit (int DeviceNum, int * *VersionMSB*, int * *VersionLSB*)

Initializes the data structures used in the RDM Controller Stack. Also opens the device specified and makes the USB Pro Ready to Send / Receive RDM Packets. The 'DeviceNum' is the no. of "USB PRO" connected to the PC (sequentially starting from 1).

A VersionMSB of '2' indicates that the USB Pro supports RDM. Make sure that the VersionMSB is checked for, before proceeding further. Any versions before 2.xx do not support RDM natively.

Parameters:

Device Num integer
VersionMSB Pro Firmware Version MSB (int*)
VersionLSB Pro Firmware Version LSB (int*)

Returns:

Returns TRUE (1) on Success, and FALSE (0) if it fails to open the port.

void RDMTick (void)

Calls the Stack Timer Routine that enables the RDM Stack to send packets based on all the possible scenarios. The Stack stores each packet (header) sent, and matches the response to the sent RDM Packet via this Timer functionality.

This function must be run via a timer thread that runs every millisecond for an accurate Send / Receive cycle to succeed.

All RDM Packets stored in the RDM Stack are sent via this routine. It is therefore necessary that this function is called separately in its own thread (that runs every millisecond).

Warning:

This function must be called in a separate thread (running every 1 millisecond for accurate response)

Parameters:

void

Returns:

void

void RDMComms (void)

Enables the Send and Receive functionality of the Controller Stack, thus, allowing the Controller to "wait for a valid response" for every successful RDM packet sent. Since this routine may keep the Controller waiting for a long time, it must be run in a separate thread running every 10 milliseconds.

Warning:

This function must be called in a separate thread (running every 10 milliseconds)

Parameters:

void

Returns:

void

uint16_t RDMDiscovery(uint8_t flag, volatile uint8_t* Ready, uint8_t* Message)

Calls the RDM discovery routine, which is a recursive process of finding all the Controller devices connected to the DMX network. On completion it populates the device List with the device Id's of the devices found.

Normally Discovery begins by sending an "unmute" RDM packet to all the devices connected, and finds the RDM compatible devices by muting those successfully found.

After a successful discovery please use the GetDeviceCount() function to check if the Discovery found any devices.

Parameters:

flag (integer) value = 1 Continuous Discovery
(doesn't "Unmute" All devices before beginning discovery branch)
flag (integer) value = 0 Normal Discovery
ready will be set to 1; when Discovery completes
Message will be updated for any debug messages during Discovery

Returns:

Number of Devices "found in the last iteration".
(Keep doing Discovery till it returns 0).

uint16_t GetDeviceCount (void)

Returns the number of devices found (in the Device list), after a successful discovery process.

Parameters:

void

Returns:

no. of devices found after discovery

void GetDeviceID(uint8_t position, uint8_t * DeviceUID)

retrieve the UID of the selected device in the "Found Device List". Please refer to the examples section for more detail on how to use this function to get the UID of all the found devices.

Parameters:

Position index of the Device in tha Found Devices List
DeviceUID an array of integer to hold 6 byte UID for this Device.

void RDMSet (uint8_t * DestinationUID, uint16_t ParamID, uint8_t DataLength, void* Data, void * Response_Data, uint32_t * Response_Size_Expected, uint32_t Timeout, uint16_t Sub_Device, uint16_t* ResponseType)

The RDMSet routine is used to send a RDM SET Request. The function forms the RDM SET request based on the parameters passed. Among the parameters are the pointers to Data Structure holding the Response (if any), and an expected Length for the Response.

Since a RDM SET Request would usually keep waiting for the response (which could vary according to the time defined via Responder), this function also takes as a parameter a Timeout (in millisecond) that is the max. time the SET function should wait for a response.

Parameters:

DestinationUID pointer to an array of integers holding the Device ID (6 bytes)
where the request is to be sent. (Destination Device ID)
ParamID 2 byte long Parameter ID (integer)
DataLength Length of the Data to be sent in the Set Request (integer)
Data pointer to a Data Structure holding the data to send
Response_Data pointer to a structure that will hold the response.
Response_Size_Expected Maximum Size Expected for the Response
Timeout Maximum time that the controller stack should wait for a response (in millisecond) When = 0 means NO_TIMEOUT
Sub_Device Use 0 for Root Device, Else use a valid sub_device_id (1 -512)
ResponseType One of either:-

```
   // Response Types
#define RESPONSE_TYPE_ACK           0x00
#define RESPONSE_TYPE_ACK_TIMER     0x01
#define RESPONSE_TYPE_NACK_REASON   0x02
#define RESPONSE_TYPE_ACK_OVERFLOW  0x03
```

Returns:

void

void RDMGet (uint8_t * DestinationUID, uint16_t ParamID, uint8_t DataLength, void * Data, void * Response_Data, uint32_t * Response_Size_Expected, uint32_t Timeout, uint16_t Sub_Device, uint16_t* ResponseType)

The RDMGet routine is used to send a RDM GET Request.. The function forms the RDM GET request based on the parameters passed. Among the parameters are the pointers to Data Structure holding the Response (if any), and an expected Length for the Response.

Since a RDM GET Request would usually keep waiting for the response (which could vary according to the time defined via Responder), this function also takes as a parameter a Timeout (in millisecond) that is the max. time the GET function should wait for a response.

Parameters:

DestinationUID pointer to an array of integers holding the Device ID (6 bytes) where the request is to be sent. (Destination Device ID)

ParamID 2 byte long Parameter ID (integer)

DataLength Length of the Data to be sent in the Get Request (integer)

Data pointer to a Data Structure holding the data to send

Response_Data pointer to a structure that will hold the response.

Response_Size_Expected Maximum Size Expected for the Response

Timeout Maximum time that the controller stack should wait for a response (in millisecond) When = 0 means NO_TIMEOUT

Sub_Device Use 0 for Root Device, Else use a valid sub_device_id (1 -512)

ResponseType One of either:-

```
    // Response Types
#define RESPONSE_TYPE_ACK            0x00
#define RESPONSE_TYPE_ACK_TIMER      0x01
#define RESPONSE_TYPE_NACK_REASON    0x02
#define RESPONSE_TYPE_ACK_OVERFLOW   0x03
```

Returns:

void

void ClosePort (void)

Closes the port that was opened for RDM Communication. And Frees all the data structures used in the RDM Stack.

Parameters:

void

Returns:

void

Function Examples [C++]

Get Devices/ Device Count / Device ID

```
   // Populate the Device List
    int num_devices = GetDeviceCount();

    for( int i =0; i < num_devices; i++)
    {
            uint8_t *Dev = new uint8_t[6];
            GetDeviceID(i,Dev);

      // Send a GET SUPPORTED_PARAMETERS Request to this Device

     }
```

void RDMSet (uint8_t * DestinationUID, uint16_t ParamID, uint8_t DataLength, uint8_t * Data, uint8_t * Response_Data, uint32_t * Response_Size_Expected, uint32_t Timeout, uint16_t* Response_Type)

```
// Form the Request parameters
      uint8_t Response_Data[1220];
      uint32_t Response_Len = 1220;
      uint16_t Response_Type;

      uint8_t* Data = " Please reset your clock ";
      uint32_t Length = (uint32_t) strlen(Data);
      uint8_t DeviceSelected[] = {0x12,0x34,0x56,0x78,0x9A,0xBC};


// Once called we would have to wait till a response is received for
the function to return. 0 means NO_TIMEOUT . ROOT_DEVICE 0x0000

RDMSet(DeviceSelected,SET_TIMER,Length,(uint8_t*)Data,Response_Data,
            &Response_Len,NO_TIMEOUT,ROOT_DEVICE,& Response_Type);
```

void RDMGet (uint8_t * DestinationUID, uint16_t ParamID, uint8_t DataLength, uint8_t * Data, uint8_t * Response_Data, uint32_t * Response_Size_Expected, uint32_t Timeout, uint16_t* Response_Type)

```
// Form the Request parameters
      uint8_t Response_Data[150];
      uint32_t Res_Len = 150;

      uint8_t DeviceSelected[] = {0x12,0x34,0x56,0x78,0x9A,0xBC};


// Sending no data in the request
RDMGet (DeviceSelected,DEVICE_INFO,0,NULL,Response_Data,&Res_Len,200,
ROOT_DEVICE, &Response_Type);
```