DTS-HD® PBR API Library Description

Document Number: 9302J19200
Revision: E
Version: 1.2
Effective Date: December 2011

DTS, Inc.
5220 Las Virgenes Road
Agoura Hills, CA 91302
USA

www.dts.com

Confidential

No Warranty

Copyright

DTS-HD® PBR API Library Description

Table of Contents

1 INTRODUCTION

1.1 Overview

This document describes the DTS PBR (Peak Bit-rate) and Bit-stream Parser API which provides a set of "C++" Application Programming Interface functions to the DTS PBR and Parser Library so that it can be linked in and called by a host application. The primary objective of this library is to provide disc authoring applications with the ability to perform PBR smoothing on non-PBR smoothed DTS-HD Master Audio™ encoded files and frame parsing. This functionality is achieved by providing API functions for traversing the entire encoded file and extracting each frame of the bit-stream. The frame bit-stream returned from the frame retrieval is PBR smoothed for DTS-HD Master Audio bit-streams only. For all other bit-stream types, the actual encoded raw frame from the bit-stream is returned. For applications requesting just a section of the bit-stream, there exists API functions to define the region of extraction (and therefore, the region of PBR smoothing). To extend the functionality of this library, the API frame extraction functions will also allow extraction of CBR encoded bit-stream frames.

This API provides the following functionality:

- Functions to extract out individual encoded frames from DTS-HD (.dtshd) files or legacy DTS CD/DVD (.cpt) files.
- Functions to extract the frame payload/schedule (payload and PTS) from DTS-HD (.dtshd) files or legacy DTS CD/DVD (.cpt) files.
- Simultaneous PBR smoothing and frame parsing of non-PBR smoothed DTS-HD Master Audio encoded (VBR encoded) .dtshd files
- Frame parsing of previously PBR scheduled DTS-HD master audio encoded files.
- Ability to designate region (start frame/end frame) for frame extraction (and PBR scheduling).
- Ability to navigation to specific frame location to commence bit-stream extraction
- Functions to extract Mp2TS and Mp4 DTS audio descriptors for each frame.

Library operations consists of the following steps

- Library initialization which consists of handle allocation, library initialization, and bit-stream loading
- Selection of retrieval region to the entire bit-stream (default) or designated section
- Initializing library for retrieval of first frame schedule/payload and/or data along with Mp2TS/Mp4 audio descriptor
- Tranversing entire retrieval region for frame schedule/payload and/or data along with Mp2TS/Mp4 audio descriptor

2 Initialization

The library initialization consists of the following steps:

- Instantiation
- Initialization
- Loading DTS encoded bit-stream

2.1 Library Instantiation

The DTS PBR smoothing and parser library instantiation encompasses allocating an instance of the library via DTSPBR_Allocate_Hdl which returns an instance to the library handle CDTSPBRHdl.

```
CDTSPBRHdl pDtsHDPBRHdl;
pDtsHDPBRHdl = DTSPBR_Allocate_Hdl(void);
if ( pDtsHDPBRHdl == NULL )
        Throw “Cannot create handle”;
```

2.2 Library Initialization

Once the library has been instantiated, the next step is initialization of the library via the DTSPBR_Init API function. During this step, the name of the DTS encoded file (either DTS-HD with .dtshd file extension or legacy CD/DVD with .cpt file extension) and any initialization flags are passed to the library. Initialization will fail if the DTS encoded file can not be opened or the DTS-HD file header can not be read. For normal operation of the library, the initialization flag is set to 0.

```
unsigned int nPBRInitFlags = 0;
if ( ! DTSPBR_Init(pDtsHDPBRHdl, pcInFile, nPBRInitFlags) )
{
        DTSPBR_Deallocate_Hdl(pDtsHDPBRHdl);
        return false;
}
```

2.3 Loading of bit stream

After library initialization, the next step is to load the DTS encoded file. For CBR streams, the library load process will involve loading a single frame to determine meta-data information encoded in the bit-stream. For a non-PBR smoothed VBR bit-stream (DTS coherent acoustics core with lossless extension), the entire DTS-HD Master Audio bit-stream must be loaded by the library and parsed to facilitate PBR analysis and smoothing. This processing time for the load process is dependent upon the size of the DTS-HD file and the I/O speed of the hard drive. The bit stream loading is accomplished by calling the DTSPBR_Load_EncFile API function.

```
if ( ! DTSPBR_Load_EncFile(pDtsHDPBRHdl) )
{
        DTSPBR_Deallocate_Hdl(pDtsHDPBRHdl);
        return false;
}
```

For multi-threaded GUI applications where load progress is being reported, one thread can be assigned to performing the call DTSPBR_Load_EncFile function while another thread calls the DTSPBR_Get_Load_Progress() API function to check on the file loading percentage complete.

2.3.1 Loading of bit-stream using PBR database data

As mentioned previously, the library is required to load the entire bit-stream for non-PBR smoothed DTS-HD Master Audio bit-streams (coherent acoustics core with lossless extension). Once an application shuts down the library, the PBR data is no longer in memory. Any new subsequent application sessions with the same non-PBR smoothed master audio bit-stream will require a complete traversal of the entire master audio bit-stream upon session initialization. Although loading and traversing the bit-stream load process is not a computational exhaustive step, it is limited by the I/O speed of the disc. To eliminate this start-up load time required for future sessions using the same non-PBR smoothed master audio bit-stream, an image of the PBR database may be generated by the library and used for future application sessions. In this scenario, the calling application loads the entire master audio encoded file during the inaugural session using this particular DTS encoded bit-stream. Then prior to exiting the inaugural session, the applications queries the library for the PBR database image and stores it for initializing future sessions with the same master audio bit-stream. It is the responsibility of the calling application to manage and store the PBR database image. In subsequent sessions, the calling application will load the bit-stream using the PBR database image and the entire master audio bit-stream will NOT have to be loaded and parsed. This is accomplished by calling the DTSPBR_Load_EncFile API function and passing in the PBR database image.

```
if ( ! DTSPBR_Load_EncFile(pDtsHDPBRHdl, pucPBRDBaseBuffer, nPBRDBaseSize) )
{
        DTSPBR_Deallocate_Hdl(pDtsHDPBRHdl);
        return false;
}
```

If the library detects an error with the PBR database image, it will automatically discard using the PBR database image and continue with the load process by defaulting back to parsing and traversing the entire master audio bit-stream to initialize the PBR database. The API function DTSPBR_Get_PBR_DBase_Loaded_Flag can be used to determine if the loading of the PBR database image was successful or if the library defaulted back to loading and parsing the bit-stream.

3 Retrieval of frame schedule/payload and frame data

Once the encoded DTS file has been loaded, an application can either retrieve frame schedule/payload data and/or actual encoded frame bit-steam data. The library provides API functions to retrieve the frame data as either a single sub-stream (combined core + extension) or as two separate sub-streams (core and extension). By default, the library assumes, the calling application will retrieve data for the entire bit-stream. When the library operates upon a non-PBR smoothed master audio file, the schedule and frame data returned from the library is PBR smoothed.

For applications retrieving only frame by frame schedule/payload information, the library provides API (DTSPBR_Get_Frame_Schedule_*) functions for retrieving the frame data as either as a single sub-stream (combines core and extension) or two separate sub-streams (separate core and extension sub-streams). Both API functions (single and seperate sub-stream) will return the PTS and DTS of the frame. API function DTSPBR_Get_Frame_Schedule_TwoSubStream models separate core and extension sub-streams and returns separate frame payload values for the core and extension sub-stream. For bit-streams containing only a core sub-stream, the function will return 0 bytes for the extension sub-stream payload. Similarly the function will return 0 bytes for the core sub-stream payload for bit-streams containing only an extension sub-stream. For applications modeling the DTS bit-stream as a single sub-stream, function DTSPBR_Get_Frame_Schedule_SingleSubStream is provided. It models a single sub-stream and returns a single value for the total number of bytes in the frame (combines core sub-stream frame payload with extension sub-stream frame payload).

For applications retrieving frame by frame encoded data (and schedule/payload) information, the API functions DTSPBR_Get_Frame_Data_TwoSubStream() and DTSPBR_Get_Frame_Data_SingleSubStream() are provided. In addition to the frame bit-stream data, both functions will return the PTS and DTS of the frame. DTSPBR_Get_Frame_Data_TwoSubStream separates the frame data into core and extension sub-stream data buffers. For DTS bit-streams containing only a core sub-stream, the function will return 0 bytes for the extension sub-stream payload. Similarly the function will return 0 bytes for the core sub-stream payload for bit-streams containing only an extension sub-stream. API function DTSPBR_Get_Frame_Data_SingleSubStream returns the frame data in a single sub-stream data buffer (core and extension sub-streams are modeled as a single sub-stream).

By default, the library assumes the calling application will retrieve data for all frames of the DTS encoded bit-stream. This includes the codec delay frames. Therefore any application not requiring the codec delay frames should extract them and discard them. For DTS bit-streams contained in a ".dtshd" file, the codec delay can be determined from meta-data in the .dtshd file header. See .dtshd file header (DTS-HD_File_Header.doc) documentation to determine number of codec delay frames at the beginning of the bitstream. For legacy DVD DTS bit-streams (files ending with ".cpt" file extensions), the codec delay is one frame.

To extract out all frames, the calling application continues to call the DTSPBR_Get_Frame_* API functions until the function returns false. This indicates that the library frame pointer has traversed to the end of the encoded bit-stream. (If there is an error in the bit-stream such that a DTS Sync word is NOT at the next frame location, the DTSPBR_Get_Frame_* API functions will also return false.)

Upon retrieving the frame payload/schedule data, API functions are provided to retrieve Mp2TS and Mp4 audio descriptors. The API functions DTSPBR_Get_Frame_Mpeg2_DTS_audio_stream_descriptor and DTSPBR_Get_Frame_Mpeg2_DTSHD_audio_stream_descriptor are provided for retrieving DTS audio frame Mp2TS audio descriptors for both DTS and DTS-HD. Similarily, API function DTSPBR_Get_Frame_Mpeg4_DTSSampleEntryData returns the frame Mp4 audio descriptor.

3.1 Initializing library start state

Prior to the retrieval of any frame data, the library frame pointer must be initialized to start pulling data from the start of bit-stream (default operation) or the first frame of the selected region (if a bit-stream region has been selected via DTSPBR_Set_Schedule_Region) by calling the DTSPBR_Init_FrameNavigation API function. As previously discussed, the DTSPBR_Get_Frame_* API functions will return false when the either the library's frame pointer tranverses past the end of the encoded bit-stream or past the end of the selected region. To re-start tranversal using DTSPBR_Get_Frame_* API functions, a call to DTSPBR_Init_FrameNavigation must be made.

3.2 Frame retrieval for separate core and extension sub-streams.

3.2.1 Example code for retrieving only schedule/payload data for each frame of the encoded bit-stream (separate core and extension sub-stream method).

```
fprintf(pFile,"PTS");
fprintf(pFile,"\tDTS");
fprintf(pFile,"\tCoreSubStreamBytes");
fprintf(pFile,"\tExtSubStreamBytes\n");
// Set library state to start from beginning of bit-stream
if ( ! DTSPBR_Init_FrameNavigation(pDtsHDPBRHdl) )
        return false;
while (DTSPBR_Get_Frame_Schedule_TwoSubStream(pDtsHDPBRHdl, nuPTS,
nuDTS, nuCoreBytes, nuExtBytes) )
{
#ifdef _WIN32
        if ( nuPTS == 0 )
                fprintf(pFile, "0\t0\t%d\t%d\n", nuCoreBytes, nuExtBytes);
        else
                fprintf(pFile, "%.I64u\t0\t%d\t%d\n", nuPTS, nuCoreBytes,
nuExtBytes);
#else
        fprintf(pFile, "%llu\t%llu\t%d\t%d\n",nuPTS, nuDTS, nuCoreBytes,
nuExtBytes);
#endif
}
```

NOTE: Example extracts all frames from file including any codec delay frames.

3.2.2 Example code for retrieving both the encoded data and schedule/payload data for each frame of the encoded bit-stream (separate core and extension sub-stream method).

```
// Set library state to start from beginning of bit-stream
if ( ! DTSPBR_Init_FrameNavigation(pDtsHDPBRHdl) )
        return false;
n = 0;
while (DTSPBR_Get_Frame_Data_TwoSubStream(pDtsHDPBRHdl, &coreData,
nuCoreBytes, &extData, nuExtBytes, nuPTS, nuDTS) )
{
        if ( nuCoreBytes > 0 )
        {
                if ( fwrite(coreData.m_pucEncodedRawFrame,
sizeof(char),nuCoreBytes,pFile) != nuCoreBytes )
                {
                        fprintf(stderr, “Error writing %s.\n”, sFileName);
                        fclose(pFile);
                        return false;
                }
        }
        if ( nuExtBytes > 0 )
        {
                if ( fwrite(extData.m_pucEncodedRawFrame,
sizeof(char),nuExtBytes,pFile) != nuExtBytes )
                {
                        fprintf(stderr, “Error writing %s.\n”, sFileName);
                        fclose(pFile);
                        return false;
                }
        }
#ifdef ENABLE_MPEG_DESCRIPTOR
        // retrieve Mp2TS audio descriptor for current frame
        if ( bEnableMp2TS && pFileMP2 != NULL)
        {
                CDTSMpeg2DTSHD_audio_stream_descriptor DTSHDdescriptor;
                CDTSMpeg2DTS_audio_stream_descriptor DTSdescriptor;

                if
( DTSPBR_Get_Frame_Mpeg2_DTSHD_audio_stream_descriptor(pDtsHDPBRHdl,
&DTSHDdescriptor) )
                {
                        fprintf(pFileMP2, "Frame #%d\n", n);
                        Dump_Mpeg2_DTSHD_audio_stream_descriptor(pFileMP2,
&DTSHDdescriptor);
                }
                if
( DTSPBR_Get_Frame_Mpeg2_DTS_audio_stream_descriptor(pDtsHDPBRHdl, &DTSdescriptor) )
                {
                        fprintf(pFileMP2, "Frame #%d\n", n);
```

```
                                        Dump_Mpeg2_DTS_audio_stream_descriptor(pFileMP2,
&DTSdescriptor);
                                }
                        }
                        // retrieve Mp2TS audio descriptor for current frame
                        if (bEnableMp && pFileMP4 != NULL )
                        {
                                CDTSSampleEntry       DTSSampleEntry;
                                if
( DTSPBR_Get_Frame_Mpeg4_DTSSampleEntryData(pDtsHDPBRHdl, &DTSSampleEntry) )
                                {
                                        fprintf(pFileMP4, "Frame #%d\n", n);
                                        Dump_Mpeg4_DTSSampleEntry(pFileMP4,
&DTSSampleEntry);
                                }
                        }
#endif   // #ifdef ENABLE_MPEG _DESCRIPTOR
                        n++;
                        fprintf(stderr, “Updating frame #:%d\r”, n);
                }
```

NOTE: Example extracts all frames from file including any codec delay frames.

3.3 Frame retrieval for single sub-stream (combined core and extension)

3.3.1 Example code for retrieving only schedule/payload data for each frame of the encoded bit-stream (combined sub-stream method)

```
                fprintf(pFile,”PTS”);
                fprintf(pFile,”\tDTS”);
                fprintf(pFile,”\tFrmBytes\n”);
                // Set library state to start from beginning of bit-stream
                if ( ! DTSPBR_Init_FrameNavigation(pDtsHDPBRHdl) )
                        return false;
                while (DTSPBR_Get_Frame_Schedule_SingleSubStream(pDtsHDPBRHdl, nuPTS,
                nuDTS, nuFrmBytes) )
                {
#ifdef _WIN32
                        if ( nuPTS == 0 )
                                fprintf(pFile, “0\t0\t%d\n”, nuFrmBytes);
                        else
                                fprintf(pFile, “%.I64u\t0\t%d\n”, nuPTS, nuFrmBytes);
#else
                        fprintf(pFile, “%llu\t%llu\t%d\n”,nuPTS, nuDTS, nuFrmBytes);
#endif
                }
```

NOTE: Example extracts all frames from file including any codec delay frames

3.3.2 Example code for retrieving both the encoded data and schedule/payload data for each frame of the encoded bit-stream (combined sub-stream method).

```
// Set library state to start from beginning of bit-stream
if ( ! DTSPBR_Init_FrameNavigation(pDtsHDPBRHdl) )
        return false;
n = 0;
while (DTSPBR_Get_Frame_Data_SingleSubStream(pDtsHDPBRHdl, & nuFrmDatas,
nuFrmBytes, nuPTS, nuDTS) )
{
        if ( fwrite(nuFrmData.m_pucEncodedRawFrame, sizeof(char),
nuFrmBytes,pFile) != nuFrmBytes)
        {
                fprintf(stderr, “Error writing %s.\n”, sFileName);
                fclose(pFile);
                return false;
        }

        n++;
        fprintf(stderr, “Updating frame #:%d\r”, n);
}
```

NOTE: Example extracts all frames from file including any codec delay frames

3.4 Navigation or designating section of bit-stream to extract

The library provides the ability to select a section of the bit-stream to schedule and extract. This provides the application with the ability to “trim” out sections of the bit-stream. For non-PBR smoothed master audio bit-streams, only the selected section will be scheduled. Since the library supports seamless switching, the first two frames and last two frames will not be PBR smoothed. Designation of a section of the bit-stream for smoothing and extraction is done using the DTSPBR_Set_Schedule_Region() function by designating the start and end frame of the selected region. When this function is not used to set a region, the library will operate upon the entire bit-stream. The library will not allow navigation to a designated frame or selection of a frame region when operating upon any DTS-HD Master Audio encoded bit-streams which has already been PBR smoothed prior to being loaded into the library. The library will only allow navigation and selection of bit-stream regions for CBR bit-streams and non-PBR smoothed master audio bit-streams.

NOTE: The library uses frame number “one” as the first frame. **For bit-streams with codec delays, frame number one is a codec delay frame**. **The selected region may include codec delay frames. Codec delay frames will NOT be PBR smoothed but are exactable from the library. Therefore any application which does not require the codec delay frames should discard them if they are included in the selection region. (Or alternatively designate a selection region with out the codec delay frames and they will be “trimmed” out.) Also when converting timecode to frame number, the application should be aware of the codec delay frames included in the file.** Setting the start and end frame to -1 or 0 will select the entire bit-stream (including codec delay frames).

3.4.1 Designating bit-stream section for retrieving frame data into separate core and extension sub-streams.

3.4.1.1 Example code for retrieving only schedule/payload data for designated region of the encoded bit-stream (separate core and extension sub-stream method).

```
// Sets bit-stream region using start and end frame index
if ( ! DTSPBR_Set_Schedule_Region(pDtsHDPBRHdl, nFrameStart, nFrameEnd) )
        return false;

fprintf(pFile,”PTS”);
fprintf(pFile,”\tDTS”);
fprintf(pFile,”\tCoreSubStreamBytes”);
fprintf(pFile,”\tExtSubStreamBytes\n”);
// Set library state to start from beginning of selected region
if ( ! DTSPBR_Init_FrameNavigation(pPBRHdl) )
        return false;
while (DTSPBR_Get_Frame_Schedule_TwoSubStream(pDtsHDPBRHdl, nuPTS,
nuDTS, nuCoreBytes, nuExtBytes) )
{
#ifdef _WIN32
        if ( nuPTS == 0 )
                fprintf(pFile, “0\t0\t%d\t%d\n”, nuCoreBytes, nuExtBytes);
        else
                fprintf(pFile, “%.I64u\t0\t%d\t%d\n”, nuPTS, nuCoreBytes,
nuExtBytes);
#else
        fprintf(pFile, “%llu\t%llu\t%d\t%d\n”,nuPTS, nuDTS, nuCoreBytes,
nuExtBytes);
#endif

#ifdef ENABLE_MPEG_DESCRIPTOR
        // retrieve Mp2TS audio descriptor for current frame
        if ( bEnableMp2TS && pFileMP2 != NULL)
        {
                CDTSMpeg2DTSHD_audio_stream_descriptor DTSHDdescriptor;
                CDTSMpeg2DTS_audio_stream_descriptor DTSdescriptor;

                if
( DTSPBR_Get_Frame_Mpeg2_DTSHD_audio_stream_descriptor(pDtsHDPBRHdl,
&DTSHDdescriptor) )
                {
                        fprintf(pFileMP2, "Frame #%d\n", n);
                        Dump_Mpeg2_DTSHD_audio_stream_descriptor(pFileMP2,
&DTSHDdescriptor);
                }
                if
( DTSPBR_Get_Frame_Mpeg2_DTS_audio_stream_descriptor(pDtsHDPBRHdl, &DTSdescriptor) )
                {
```

```
                                        fprintf(pFileMP2, "Frame #%d\n", n);
                                        Dump_Mpeg2_DTS_audio_stream_descriptor(pFileMP2,
&DTSdescriptor);
                                }
                        }
                        // retrieve Mp2TS audio descriptor for current frame
                        if (bEnableMp && pFileMP4 != NULL )
                        {
                                CDTSSampleEntry       DTSSampleEntry;
                                if
( DTSPBR_Get_Frame_Mpeg4_DTSSampleEntryData(pDtsHDPBRHdl, &DTSSampleEntry) )
                                {
                                        fprintf(pFileMP4, "Frame #%d\n", n);
                                        Dump_Mpeg4_DTSSampleEntry(pFileMP4,
&DTSSampleEntry);
                                }
                        }
#endif   // #ifdef ENABLE_MPEG _DESCRIPTOR
                }
```

NOTE: The designation of nFrameStart and nFrameEnd in DTSPBR_Set_Schedule_Region should account for any codec delay frame(s) in the bit-stream.

3.4.1.2 Example code for retrieving both the encoded data and schedule/payload data for designated region of the encoded bit-stream (separate core and extension sub-stream method).

```
// Sets bit-stream region using start and end frame index
if ( ! DTSPBR_Set_Schedule_Region(pDtsHDPBRHdl, nFrameStart, nFrameEnd) )
        return false;


// Set library state to start from beginning of selected region
if ( ! DTSPBR_Init_FrameNavigation(pDtsHDPBRHdl) )
        return false;
n = 0;
while (DTSPBR_Get_Frame_Data_TwoSubStream(pDtsHDPBRHdl, &coreData,
nuCoreBytes, &extData, nuExtBytes, nuPTS, nuDTS) )
{
        if ( nuCoreBytes > 0 )
        {
                if ( fwrite(coreData.m_pucEncodedRawFrame,
sizeof(char),nuCoreBytes,pFile) != nuCoreBytes )
                {
                        fprintf(stderr, “Error writing %s.\n”, sFileName);
                        fclose(pFile);
```

```
                        return false;
                    }
                }
                if ( nuExtBytes > 0 )
                {
                    if ( fwrite(extData.m_pucEncodedRawFrame,
sizeof(char),nuExtBytes,pFile) != nuExtBytes )
                    {
                        fprintf(stderr, “Error writing %s.\n”, sFileName);
                        fclose(pFile);
                        return false;
                    }
                }
#ifdef ENABLE_MPEG_DESCRIPTOR
                // retrieve Mp2TS audio descriptor for current frame
                if ( bEnableMp2TS && pFileMP2 != NULL)
                {
                    CDTSMpeg2DTSHD_audio_stream_descriptor DTSHDdescriptor;
                    CDTSMpeg2DTS_audio_stream_descriptor DTSdescriptor;

                    if
( DTSPBR_Get_Frame_Mpeg2_DTSHD_audio_stream_descriptor(pPBRHdl, &DTSHDdescriptor) )
                    {
                        fprintf(pFileMP2, "Frame #%d\n", n);
                        Dump_Mpeg2_DTSHD_audio_stream_descriptor(pFileMP2,
&DTSHDdescriptor);
                    }
                    if
( DTSPBR_Get_Frame_Mpeg2_DTS_audio_stream_descriptor(pPBRHdl, &DTSdescriptor) )
                    {
                        fprintf(pFileMP2, "Frame #%d\n", n);
                        Dump_Mpeg2_DTS_audio_stream_descriptor(pFileMP2,
&DTSdescriptor);
                    }
                }
                // retrieve Mp2TS audio descriptor for current frame
                if (bEnableMp && pFileMP4 != NULL )
                {
```

```
                        CDTSSampleEntry       DTSSampleEntry;
                        if ( DTSPBR_Get_Frame_Mpeg4_DTSSampleEntryData(pPBRHdl,
&DTSSampleEntry) )
                        {
                                fprintf(pFileMP4, "Frame #%d\n", n);
                                Dump_Mpeg4_DTSSampleEntry(pFileMP4,
&DTSSampleEntry);
                        }
                }
#endif   // #ifdef ENABLE_MPEG _DESCRIPTOR
                n++;
                fprintf(stderr, “Updating frame #:%d\r”, n);
        }
```

NOTE: The designation of nFrameStart and nFrameEnd in DTSPBR_Set_Schedule_Region should account for any codec delay frame(s) in the bit-stream.

3.4.2 Designating bit-stream section for retrieving frame data into single (combined core and extension) sub-streams.

3.4.2.1 Example code for retrieving only schedule/payload data for designated bit-stream region (combined sub-stream method).

```
// Sets bit-stream region using start and end frame index
if ( ! DTSPBR_Set_Schedule_Region(pDtsHDPBRHdl, nFrameStart, nFrameEnd) )
        return false;

fprintf(pFile,”PTS”);
fprintf(pFile,”\tDTS”);
fprintf(pFile,”\tFrmBytes\n”);
// Set library state to start from beginning of selected region
if ( ! DTSPBR_Init_FrameNavigation(pDtsHDPBRHdl) )
        return false;
while (DTSPBR_Get_Frame_Schedule_SingleSubStream(pDtsHDPBRHdl, nuPTS,
nuDTS, nuFrmBytes) )
{
#ifdef _WIN32
        if ( nuPTS == 0 )
                fprintf(pFile, “0\t0\t%d\n”, nuFrmBytes);
        else
                fprintf(pFile, “%.I64u\t0\t%d\n”, nuPTS, nuFrmBytes);
#else
        fprintf(pFile, “%llu\t%llu\t%d\n”,nuPTS, nuDTS, nuFrmBytes);
#endif
}
```

NOTE: The designation of nFrameStart and nFrameEnd in DTSPBR_Set_Schedule_Region should account for any codec delay frame(s) in the bit-stream.

3.4.2.2 Example code for retrieving encoded frame data and schedule/payload data for designated bit-stream region (combined sub-stream method).

```
// Sets bit-stream region using start and end frame index
if ( ! DTSPBR_Set_Schedule_Region(pDtsHDPBRHdl, nFrameStart, nFrameEnd) )
        return false;
// Set library state to start from beginning of selected region
if ( ! DTSPBR_Init_FrameNavigation(pDtsHDPBRHdl) )
        return false;
n = 0;
while (DTSPBR_Get_Frame_Data_SingleSubStream(pDtsHDPBRHdl, & nuFrmDatas,
nuFrmBytes, nuPTS, nuDTS) )
{
        if ( fwrite(nuFrmData.m_pucEncodedRawFrame, sizeof(char),
nuFrmBytes,pFile) != nuFrmBytes)
        {
                fprintf(stderr, “Error writing %s.\n”, sFileName);
                fclose(pFile);
                return false;
        }
#ifdef ENABLE_MPEG_DESCRIPTOR
        // retrieve Mp2TS audio descriptor for current frame
        if ( bEnableMp2TS && pFileMP2 != NULL)
        {
                CDTSMpeg2DTSHD_audio_stream_descriptor DTSHDdescriptor;
                CDTSMpeg2DTS_audio_stream_descriptor DTSdescriptor;

                if
( DTSPBR_Get_Frame_Mpeg2_DTSHD_audio_stream_descriptor(pPBRHdl, &DTSHDdescriptor) )
                {
                        fprintf(pFileMP2, "Frame #%d\n", n);
                        Dump_Mpeg2_DTSHD_audio_stream_descriptor(pFileMP2,
&DTSHDdescriptor);
                }
```

```
                        if
( DTSPBR_Get_Frame_Mpeg2_DTS_audio_stream_descriptor(pPBRHdl, &DTSdescriptor) )
                        {
                                fprintf(pFileMP2, "Frame #%d\n", n);
                                Dump_Mpeg2_DTS_audio_stream_descriptor(pFileMP2,
&DTSdescriptor);
                        }
                }
                // retrieve Mp2TS audio descriptor for current frame
                if (bEnableMp && pFileMP4 != NULL )
                {
                        CDTSSampleEntry       DTSSampleEntry;
                        if ( DTSPBR_Get_Frame_Mpeg4_DTSSampleEntryData(pPBRHdl,
&DTSSampleEntry) )
                        {
                                fprintf(pFileMP4, "Frame #%d\n", n);
                                Dump_Mpeg4_DTSSampleEntry(pFileMP4,
&DTSSampleEntry);
                        }
                }
#endif   // #ifdef ENABLE_MPEG _DESCRIPTOR

                n++;
                fprintf(stderr, “Updating frame #:%d\r”, n);
        }
```

NOTE: The designation of nFrameStart and nFrameEnd in DTSPBR_Set_Schedule_Region should account for any codec delay frame(s) in the bit-stream.

3.5 Generation of PBR database image

For applications processing a single non-PBR smoothed DTS-HD Master Audio bit-stream over multiple sessions, it may be beneficial to create and store an image of the PBR database to reduce start up time in subsequent sessions. Recall in order to perform PBR scheduling on a non-PBR smoothed DTS-HD Master Audio bit-stream, the library must first initialize the PBR database by parsing frame payload data from the entire bit-stream during the load step (via a call to the DTSPBR_Load_EncFile API function). The PBR database is persistent in memory until the library handle is de-allocated. Any new subsequent application sessions with the same master audio bit-stream will require a complete traversal of the entire bit-stream upon session initialization. Although the bit-stream load process is not a computational exhaustive step, it is limited by the I/O speed of the disc. As an estimate, a 2.2GB bit-stream may take 2 minutes to load and parse. An alternative to re-traversing the entire master audio bit-stream for each new session, an application may elect to create, store, and manage a PBR database image. After initially loading a non-PBR smoothed DTS-HD Master Audio encoded file for an inaugural session, an application can command the library to create an image of the PBR database and use this database image doing the load process of subsequent sessions instead of traversing the entire bit-stream during session start-up. (As an estimate, a 2.2 Gbytes bit-stream yields a PBR database image of 11 Mbytes.)

Prior to requesting an image of the PBR database, the application should first query the library for the database image size and allocate a buffer for this size.

```
if ( ! DTSPBR_Get_PBR_DBase_Size(pDtsHDPBRHdl, nPBRDataSize) ||
nPBRDataSize == 0 )
        {
                DTSPBR_Deallocate_Hdl(pDtsHDPBRHdl,);
                return false;
        }
        unsigned char *pucPBRBuffer = new unsigned char[nPBRDataSize];
```

After creating a buffer of the appropriate size, the application can call the DTSPBR_Get_PBR_DBase_Data API function to fill into the allocated buffer with the PBR database image.

```
        if ( ! DTSPBR_Get_PBR_DBase_Data(pDtsHDPBRHdl, pucPBRBuffer, nPBRDataSize) )
        {
                DTSPBR_Deallocate_Hdl(pDtsHDPBRHdl);
                return false;
        }
```

The PBR database image is managed by the application and is reloaded using the DTSPBR_Load_EncFile API function.

3.6 Deallocate Library

Use DTSPBR_Deallocate_Hdl to deallocate the library instance.

4 Appendix

4.1 API Headers

The following list details the headers which need to be included in applications integrating the the DTS PBR and Parser library.

- DTSPbr.h contains the DTS PBR Parser API function prototypes and must be included in the DTS PBR Parser application.
- DTSVersion.h is optional and is only required for accessing the version of the DTS PBR scheduling and frame parser library.

The following list details additional headers included in the DTS PBR and Parser library SDK package:

- DTSDecFrameBuffer.h (referenced in DTSPbr.h) details the unsigned char buffer class holding the encoded frame bit-stream.
- DTSEncDecDefs.h contains some enumerated types.
- DTSMpeg2Descriptor.h (referenced in DTSPbr.h) details the classes for retrieving the Mp2TS audio descriptors for DTS and DTS-HD.
- DTSMpeg4Descriptor.h (referenced in DTSPbr.h) details the classes for retrieving the Mp4 audio descriptors.

4.2 API Functions

4.2.1 DTSPBR_Allocate_Hdl

Allocates and returns a handle (CDTSPBRHdl) to an instance of the DTS PBR Parser library used to access methods in the API. When the return value is NULL an instance of encoder was unable to be created.

CDTSPBRHdl DTSPBR_Allocate_Hdl (void);

Return Values

NULL Failed to create.

4.2.2 DTSPBR_Deallocate_Hdl

De-allocates instance of DTS PBR Parser library.

void DTSPBR_Deallocate_Hdl(CDTSPBRHdl pDTSPbrHdl);

Parameters

pDTSPbrHdl Library Instance handle created from DTSPBR_Allocate_Hdl

Return Values

None

4.2.3 DTSPBR_Init

Initialize the DTS PBR Parser library with name of DTS encoded bit-stream and optional initialization flags.

```
bool DTSPBR_Init(CDTSPBRHdl pDTSPbrHdl,
                 const char *pcEncFile,
                 unsigned long nInitFlags);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
pcEncFile	Full path file name of DTS encoded file
nInitFlags	Library initialization flags for special cases. (Should be defaulted to 0).

Return Values

true	Successful Initialization.
false	Unable to open file or invalid file designated or invalid handle

```
enum DTSPBRINIT
{
        DTSPBRINIT_NONE                         // == 0,
        DTSPBRINIT_APAK_2FRM_CODEC_DELAY        // Add 2 frame codec delay for APAK
        DTSPBRINIT_NO_SEAMLESS_SWITCHING ,      // Disables seamless switching
        DTSPBRINIT_NO_CBR                       // Disables parsing of CBR files
};
```

4.2.4 DTSPBR_Load_EncFile

Load the DTS encoded file designated in DTSPBR_Init.

```
bool DTSPBR_Load_EncFile(CDTSPBRHdl pDTSPbrHdl);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl

Return Values

true	Successful loading of DTS encoded file.
false	Invalid file designated in DTSPBR_Init

4.2.5 DTSPBR_Load_EncFile using PBR Database image

Load the DTS encoded file designated in DTSPBR_Init using PBR database image.

```
bool DTSPBR_Load_EncFile(CDTSPBRHdl pDTSPbrHdl,
                         unsigned char *pcPBRDBaseBuffer,
                         unsigned int nuPBRDBaseBufferSize);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
pcPBRDBaseBuffer	PBR Database image for designated DTS encoded file.
nuPBRDBaseBufferSize	Total buffer size of PBR Database image

Return Values

true	Successful loading of DTS encoded file.
false	Invalid file designated in DTSPBR_Init

4.2.6 DTSPBR_Get_Load_Progress

Retrieve the percentage complete for loading DTS encoded file in DTSPBR_Load_EncFile().

```
bool DTSPBR_Get_Load_Progress(CDTSPBRHdl pDTSPbrHdl,
                              unsigned int &nPercentDone);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
nPercentDone	Percentage of DTS encoded file loaded

Return Values

true	Successful retrieval of DTS encoded file load percentage.
false	Invalid handle

4.2.7 DTSPBR_Init_FrameNavigation

Sets the state of the library for retrieval of frame data by setting to first frame (either first frame of bit-stream or first frame in designated bit-stream region).

```
bool DTSPBR_Init_FrameNavigation(CDTSPBRHdl pDTSPbrHdl);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl

Return Values

true	Successful initialization of retrieval state
false	Invalid handle

4.2.8 DTSPBR_Set_Schedule_Region

Selects bit-stream region or section for library to extract frames.

```
bool DTSPBR_Set_Schedule_Region(CDTSPBRHdl pDTSPbrHdl, int nFrmStartIndex, int
nFrmEndIndex);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
nFrmStartIndex	Bit-stream frame index to start extracting frames. (Set to -1 or 0 for first frame)
nFrmEndIndex	Bit-stream frame index to end extracting frames. (Set to -1 for last frame in file)

Return Values

true	Successful setting of selected bit-stream region.
false	Invalid handle or invalid region designated.

NOTE: The designation of nFrameStart and nFrameEnd in DTSPBR_Set_Schedule_Region should account for any codec delay frame(s) in the bit-stream.

4.2.9 DTSPBR_Get_Frame_Schedule_SingleSubStream

Retrieves the frame schedule/payload data and returns value as a single sub-stream (combines core and extension sub-stream).

```
bool DTSPBR_Get_Frame_Schedule_SingleSubStream(CDTSPBRHdl pDTSPbrHdl,
                                               unsigned DTS__int64 &nuPTS,
                                               unsigned DTS__int64 &nuDTS,
                                               unsigned int &nuBytes);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
nuPTS	Presentation time
nuDTS	Decode time (always set to 0.)
nuBytes	Frame payload in bytes

Return Values

true	Successful retrieval of DTS frame schedule.
false	Invalid handle or end of bit-stream

4.2.10 DTSPBR_Get_Frame_Data_SingleSubStream

Retrieves the encoded frame data and schedule/payload and returns value as a single sub-stream (combines core and extension sub-stream).

```
bool DTSPBR_Get_Frame_Data_SingleSubStream(CDTSPBRHdl pDTSPbrHdl,
                                           CDTSDecRawFrameBuffer *pFrmData,
                                           unsigned int &nuFrmDataBytes,
                                           unsigned DTS__int64 &nuPTS,
                                           unsigned DTS__int64 &nuDTS);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
pFrmData	Buffer holding encoded frame data
nuFrmDataBytes	Frame payload in bytes.
nuPTS	Presentation time
nuDTS	Decode time (always set to 0.)

Return Values

true	Successful retrieval of DTS frame data.
false	Invalid handle or end of bit-stream

4.2.11 DTSPBR_Get_AccessUnit_Schedule_SingleSubStream

Retrieves the frame audio pack schedule/payload data and returns value as a single sub-stream (combines core and extension sub-stream).

```
bool DTSPBR_Get_AccessUnit_Schedule_SingleSubStream(CDTSPBRHdl pDTSPbrHdl,
                                                    int &nENOBUFlag,
                                                    unsigned DTS__int64 &nuPTS,
                                                    unsigned DTS__int64 &nuDTS,
                                                    unsigned int &nuApakBytes,
                                                    unsigned int &nuBufOccup);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
nENOBUFlag	ENOBU Flag value
nuPTS	Presentation time
nuDTS	Decode time (always set to 0.)
nuApakBytes	Payload size in bytes for audio pack frame data.
nuBufOccup	Buffer occupancy in bytes

Return Values

true	Successful retrieval of DTS frame schedule.
false	Invalid handle or end of bit-stream

4.2.12 DTSPBR_Get_AccessUnit_Data_SingleSubStream

Retrieves the frame audio pack encoded data and schedule/payload data and returns value as a single sub-stream (combines core and extension sub-stream).

```
bool DTSPBR_Get_AccessUnit_Data_SingleSubStream(CDTSPBRHdl pDTSPbrHdl,
                                                int &nENOBUFlag,
                                                unsigned DTS__int64 &nuPTS,
                                                unsigned DTS__int64 &nuDTS,
                                                CDTSDecRawFrameBuffer *pFrmData,
                                                unsigned int &nuApakBytes,
                                                unsigned int &nuBufOccup);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
nENOBUFlag	ENOBU Flag value
nuPTS	Presentation time
nuDTS	Decode time (always set to 0.)
pFrmData	Buffer holding audio pack data
nuApakBytes	Payload size in bytes for audio pack frame data.
nuBufOccup	Buffer occupancy in bytes

Return Values

true	Successful retrieval of DTS frame data.
false	Invalid handle or end of bit-stream

4.2.13 DTSPBR_Get_Frame_Schedule_TwoSubStream

Retrieves the frame schedule/payload data and returns value as a separate core and extension sub-streams.

```
bool DTSPBR_Get_Frame_Schedule_TwoSubStream(CDTSPBRHdl pDTSPbrHdl,
                                            unsigned DTS__int64 &nuPTS,
                                            unsigned DTS__int64 &nuDTS,
                                            unsigned int &nuCoreBytes,
                                            unsigned int &nuExtBytes);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
nuPTS	Presentation time
nuDTS	Decode time (always set to 0.)
nuCoreBytes	Frame payload of core sub-stream in bytes
nuExtBytes	Frame payload of extension sub-stream in bytes

Return Values

true	Successful retrieval of DTS frame data.
false	Invalid handle or end of bit-stream

4.2.14 DTSPBR_Get_Frame_Data_TwoSubStream

Retrieves the encoded frame data and schedule/payload and returns value as a separate core and extension sub-streams.

```
bool DTSPBR_Get_Frame_Data_TwoSubStream(CDTSPBRHdl pDTSPbrHdl,
                                        CDTSDecRawFrameBuffer *pCoreFrmData,
                                        unsigned int &nuCoreFrmBytes,
                                        CDTSDecRawFrameBuffer *pExtFrmData,
                                        unsigned int &nuExtFrmBytes,
                                        unsigned DTS__int64 &nuPTS,
                                        unsigned DTS__int64 &nuDTS);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
pCoreFrmData	Buffer holding encoded frame data for core sub-stream
nuCoreFrmBytes	Frame payload of core sub-stream in bytes
pExtFrmData	Buffer holding encoded frame data for extension sub-stream
nuExtFrmBytes	Frame payload of extension sub-stream in bytes
nuPTS	Presentation time
nuDTS	Decode time (always set to 0.)

Return Values

true	Successful retrieval of DTS frame data.
false	Invalid handle or end of bit-stream

4.2.15 DTSPBR_Get_Stream_Duration

Retrieve frame duration and total number of frames in the entire bit-stream file.

```
bool DTSPBR_Get_Stream_Duration(CDTSPBRHdl pDTSPbrHdl,
                                double &dFrameDuration,
                                unsigned int &nNumTotalFrames);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
dFrameDuration	Frame duration in seconds.
nNumTotalFrames	Total number of frames in encoded file.

Return Values

true	Successful loading of DTS encoded file.
false	Invalid file designated in DTSPBR_Init

4.2.16 DTSPBR_Get_NumCodecDelayFrame

Retrieve number of codec delay frames at start of encoded file.

```
bool DTSPBR_Get_NumCodecDelayFrame (CDTSPBRHdl pDTSPbrHdl,
                                    unsigned int & nNumCodecDelayFrames);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
nNumCodecDelayFrames	Number of codec delay frames at start of encoded file.

Return Values

true	Successful retrieval.
false	Invalid file designated in DTSPBR_Init

4.2.17 DTSPBR_Get_Schedule_BitRate

Returns the max PBR and avg PBR for entire selected region of the bit-stream.

```
bool DTSPBR_Get_Schedule_BitRate(CDTSPBRHdl pDTSPbrHdl,
                                 int &nPBRMaxKBPS,
                                 int &nPBRAvgKBPS);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
nPBRMaxKBPS	Max PBR in KPBS.
nPBRAvgKBPS	Avg PBR in KPBS.

Return Values

true	Successful loading of DTS encoded file.
false	Invalid file designated in DTSPBR_Init

4.2.18 DTSPBR_Get_PBR_DBase_Size

Queries library for buffer size required to hold PBR database image. A size of 0 indicates no PBR database is available.

```
bool DTSPBR_Get_PBR_DBase_Size(CDTSPBRHdl pDTSPbrHdl,
                               unsigned int &nuPBRDBaseSize);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
nuPBRDBaseSize	Number of bytes required for PBR database image.

Return Values

true	Successful creation of PBR Database image
false	No PBR database for current file or unable to create

4.2.19 DTSPBR_Get_PBR_DBase_Data

Create PBR database image.

```
bool DTSPBR_Get_PBR_DBase_Data(CDTSPBRHdl pDTSPbrHdl,
                               unsigned char *pcBuffer,
                               unsigned int nuBufferAllocSize);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
pcBuffer	Buffer holding PBR database image
nuBufferAllocSize	Size of buffer for holding PBR database image

Return Values

true	Successful creation of PBR Database image
false	No PBR database for current file or unable to create

4.2.20 DTSPBR_Get_PBR_DBase_Loaded_Flag

Check if DTSPBR_Load_EncFile using PBR Database image was successful. If the PBR database image was corrupted, the DTSPBR_Load_EncFile will default back to loading the entire bit-stream.

```
bool DTSPBR_Get_PBR_DBase_Loaded_Flag (CDTSPBRHdl pDTSPbrHdl);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl

Return Values

true	Successful DTSPBR_Load_EncFile using PBR Database image during
false	DTSPBR_Load_EncFile defaulted back to traversing bit-stream.

4.2.21 DTSPBR_Get_Frame_Mpeg2_DTSHD_audio_stream_descriptor

Retrieves the Mp2TS DTS-HD audio descriptor associated with the DTSH format identifier. This function may be called after any DTSPBR_Get_Frame_Data_[SingleSubStream,TwoSubStream] or DTSPBR_Get_Frame_Schedule_[SingleSubStream,TwoSubStream] function call.

```
bool DTSPBR_Get_Frame_Mpeg2_DTSHD_audio_stream_descriptor (
                               CDTSPBRHdl pDTSPbrHdl
                               CDTSMpeg2DTSHD_audio_stream_descriptor *pDescriptor);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
pDescriptor	Mp2TS DTS-HD audio descriptor

Return Values

true	Successful retrieval
false	Unsuccessful retrieval.

NOTE: All bit-streams are supported with a DTS-HD audio descriptor.

4.2.22 DTSPBR_Get_Frame_Mpeg2_DTS_audio_stream_descriptor

Retrieves the Mp2TS DTS audio descriptor associated with format_identifiers ‘DTS1’, ‘DTS2’, or ‘DTS3’. Function will return false if bit-stream can only be described using Mp2TS DTS-HD. This function may be called after any DTSPBR_Get_Frame_Data_[SingleSubStream,TwoSubStream] or DTSPBR_Get_Frame_Schedule_[SingleSubStream,TwoSubStream] function call.

```
bool DTSPBR_Get_Frame_Mpeg2_DTS_audio_stream_descriptor (
                                CDTSPBRHdl pDTSPbrHdl
                                CDTSMpeg2DTS_audio_stream_descriptor *pDescriptor);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
pDescriptor	Mp2TS DTS audio descriptor

Return Values

true	Successful retrieval
false	Unsuccessful retrieval.

4.2.23 DTSPBR_Get_Frame_Mpeg4_DTSSampleEntryData

Retrieves the Mp4 DTS sample entry data audio descriptor. This function may be called after any DTSPBR_Get_Frame_Data_[SingleSubStream,TwoSubStream] or DTSPBR_Get_Frame_Schedule_[SingleSubStream,TwoSubStream] function call.

```
bool DTSPBR_Get_Frame_Mpeg4_DTSSampleEntryData (
                                CDTSPBRHdl pDTSPbrHdl
                                CDTSSampleEntry *pDescriptor);
```

Parameters

pDTSPbrHdl	Library instance handle created from DTSPBR_Allocate_Hdl
pDescriptor	Mp4 DTS sample entry audio descriptor

Return Values

true	Successful retrieval
false	Unsuccessful retrieval.

4.3 CDTSDecRawFrameBuffer

The CDTSDecRawFrameBuffer class is used for returning the encoded bit-stream frame and is declared in DTSDecFrameBuffer.h.

```
class DLLEXPORT CDTSDecFrameBufferBase
{
public:
        int             m_nBufferType;
        int             m_nFrmSize;    // size of current buffer
        int             m_nStrmType;  // Width of the encoded stream word
        bool            m_bAllocFlag;  // false if buffer is pointer
                                       // true if buffer is allocated here
        int             m_nAllocSize;  // size of allocated buffer
}

class DLLEXPORT CDTSDecRawFrameBuffer : public CDTSDecFrameBufferBase
{
public:
        unsigned char   *m_pucEncodedRawFrame;      // buffer for encoded using char array
}
```

4.4 CDTSMpeg2DTS_audio_stream_descriptor

The CDTSMpeg2DTS_audio_stream_descriptor class is used for returning the Mp2TS DTS audio descriptor for encoded bit-stream frame and is declared in DTSMpeg2Descriptor.h.

```
class DLLEXPORT CDTSMpeg2DTS_audio_stream_descriptor
{
public:
        unsigned char       m_ucsample_rate_code;
        unsigned char       m_ucbit_rate_code;
        unsigned char       m_ucnblks;
        unsigned short      m_usfsize;
        unsigned char       m_ucsurround_mode;
        bool                m_blfe_flag;
        unsigned char       m_ucextended_surround_flag;
        unsigned char       m_uccomponent_type;
};
```

4.5 CDTSMpeg2DTSHD_audio_stream_descriptor

The CDTSMpeg2DTSHD_audio_stream_descriptor class is used for returning the Mp2TS DTS-HD audio descriptor for encoded bit-stream frame and is declared in DTSMpeg2Descriptor.h.

```
class CDTSMpeg2Asset
{
public:
        unsigned char   m_ucasset_construction;
        bool            m_bvbr_flag;
        bool            m_bpost_encode_br_scaling_flag;
        bool            m_bcomponent_type_flag;
        bool            m_blanguage_code_flag;
        unsigned short  m_usbit_rate_scaled;
        unsigned short  m_usbit_rate;
        unsigned char   m_uccomponent_type;
        unsigned int    m_unISO_639_language_code;
};

class CDTSMpeg2SubStream
{
public:
        unsigned char   m_ucNum_Assets;
        unsigned char   m_ucChannel_Count;
        bool            m_bLFE_Flag;
        unsigned char   m_ucSampling_frequency;
        unsigned char   m_ucSample_Resolution;
        CDTSMpeg2Asset      m_sAsset[8];
};

class DLLEXPORT CDTSMpeg2DTSHD_audio_stream_descriptor
{
public:
        bool                        m_bsubstream_core_flag;
        bool                        m_bsubstream_0_flag;
        bool                        m_bsubstream_1_flag;
        bool                        m_bsubstream_2_flag;
        bool                        m_bsubstream_3_flag;
        CDTSMpeg2SubStream          m_ssubstream_core;
        CDTSMpeg2SubStream          m_ssubstream_0;
        CDTSMpeg2SubStream          m_ssubstream_1;
        CDTSMpeg2SubStream          m_ssubstream_2;
        CDTSMpeg2SubStream          m_ssubstream_3;
};
```

4.6 CDTSSampleEntry

The CDTSSampleEntry class is used for returning the Mp4 DTS audio descriptor for encoded bit-stream frame and is declared in DTSMpeg4Descriptor.h.

```
class CDTSSpecificBox
{
public:
        unsigned int     m_nSize;
        unsigned char    m_ucType[4];
        unsigned int     m_nDTSSamplingFrequency;
        unsigned int     m_nMaxBitrate;
        unsigned int     m_nAvgBitRate;
        unsigned char    m_nPCMSampleDepth;
        unsigned char    m_nFrameDuration;
        unsigned char    m_nStreamConstruction;
        bool             m_bCoreLFEPresent;
        unsigned char    m_nCoreLayout;
        unsigned short   m_nCoreSize;
        bool             m_bStereoDownmix;
        unsigned char    m_nRepresentationType;
        unsigned short   m_nChannelLayout;
        bool             m_bMultiAssetFlag;
        bool             m_bLBRDurationMod;
};

class CDTSSampleEntry
{
public:
        unsigned int     m_nSize;
        unsigned char    m_ucType[4];
        unsigned int     m_nChannelCount;
        unsigned int     m_nSampleSize;
        unsigned int     m_nPredefined;
        unsigned int     m_nSampleRate;
        CDTSSpecificBox      m_dtssbox;
};
```