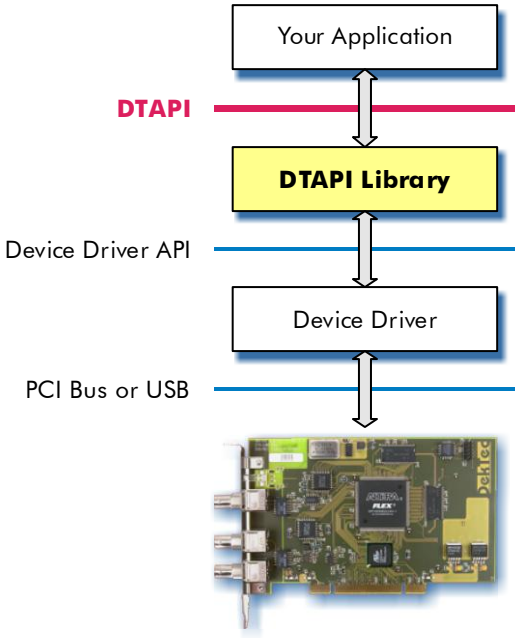
DTAPI – DVB-C2/T2 Multi-PLP Extensions

- Create custom multi-PLP applications
- Big-TS splitting into multiple PLPs
- DVB-T2 Receiver Buffer Model validation

FEATURES

- Extends DTAPI with classes to perform multi-PLP modulation for DVB-C2 or DVB-T2 from a custom application
- All parameters and modes in DVB-C2 and DVB-T2 can be set by the user program
- Single-PLP modulation is supported as a special case
- API can be used for direct RF output, T2-MI output over ASI or IP and offline generation of T2-MI and I/Q samples
- Implements DVB-T2 Receiver Buffer Model validation using callbacks
- Includes "Big-TS splitting" with SI processing for easy generation of multi-PLP streams with a common PLP
- Available for C++ and .NET languages
- Same API classes and methods can be used on Windows and Linux



APPLICATION

- DVB-C2 / DVB-T2 signal generator that supports all of C2 and T2
- Automated testing or validation of DVB-C2 or DVB-T2 receivers

Table of Contents

DTAPI Revision History

Version	Date	Change Description
V4.10.0.145	2011.02.07	• First DTAPI with support for multi-PLP modulation • General changes see *"C++ API for DekTec Devices"*

1. General Description

The DTAPI is the API that enables application programs to access the functions of DekTec devices in a uniform way. The basic concepts and object model of DTAPI are specified in "DTAPI – *C++ API for DekTec Devices*".

Multi-PLP modulation is a specific DTAPI function that enables application programs to create single-PLP and multi-PLP modulators for DVB-C2 and DVB-T2. The DTAPI classes and structures that are related multi-PLP modulation are specified in this document.

The DTAPI is composed of a header file, to be included by the application source code, and a library file, to be linked to the application's executable. The main DTAPI header file and library include the class definitions required for multi-PLP modulation.

1.1. Multi-PLP Object Model

The multi-PLP modulator is represented by a "Multi-PLP Modulator" object that is encapsulated by the `DtOutpChannel` class. When multi-PLP modulation parameters are set through a `SetModControl` method, multi-PLP modulation is enabled and input FIFOs are created for each PLP source. Methods are provided to write into the individual MPLP FIFOs and to control them. The `DtOutpChannel` object transfers the modulation results through the device driver to the device.

In case single-PLP modulation parameters are set, only one FIFO is created and the multi-PLP modulator acts as a single-PLP modulator.

1.2. Licensing

The multi-PLP classes require a DVB-C2 (DTC-379), a DVB-T2 (DTC-378) or GOLD license on the modulator card. If access to I/Q samples is required, an additional I/Q license (DTC-371) must be present.

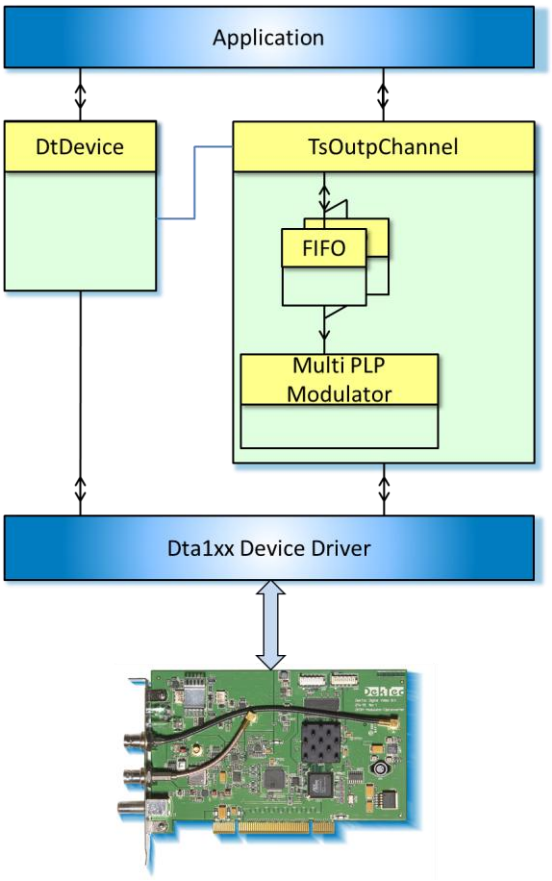


Figure 1. Example of a DtDevice object and a TsOutpChannel object encapsulation a multi-PLP modulator object.

1.3. References

- DTAPI – *C++ API for DekTec Devices, DekTec Digital Video B.V., 2010.*
- *ETSI EN 302 769, Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation transmission system for cable systems (DVB-C2).*
- *ETSI EN 302 755, Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2).*
- *ETSI EN 102 773, Digital Video Broadcasting (DVB); Modulator Interface (T2-MI) for a second generation digital terrestrial television broadcasting system (DVB-T2).*

2. Using Multi-PLP Extensions

This section discusses the usage of the multi-PLP functions in the DTAPI library. Code snippets are provided to illustrate key methods.

2.1. Attaching to a Channel

Using the DTAPI multi-PLP extensions is no different from using a standard modulator channel. First, a **DtDevice** object has to be instantiated and attached to the hardware, and then a **DtOutpChannel** object has to be attached to the device. For further details, please refer to the DTAPI documentation.

```
// Error-handling code has been omitted
DtDevice Dvc;
Dvc.AttachToSerial(4115123456);
DtOutpChannel TsOut;
TsOut.AttachToPort(&Dvc, 1);
```

Figure 2. Attaching to the hardware.

2.2. Virtual Channels

A standard output channel writes modulated I/Q samples directly to the hardware. The DTAPI multi-PLP extensions support a new type of channel, a *virtual* channel, enabling custom processing of the multi-PLP modulator output. For example, the modulated I/Q samples can be written to a file.

A virtual channel can be created using the channel's **AttachVirtual** member function. The first parameter of this function, a pointer to a **DtDevice** object, identifies the hardware device carrying the licenses to enable the MPLP extensions. The second parameter specifies the callback function and the third parameter an opaque pointer. When DTAPI has generated new output, the callback function is invoked with the opaque pointer and I/Q samples as arguments

Example code to create a virtual channel is shown in Figure 3.

```
{
  DtDevice Dvc;
  // Code to attach to device goes here

  DtOutpChannel TsOut;
  if (TsOut.AttachVirtual(&Dvc,
        ::WriteMySsmps, NULL) != DTAPI_OK)
  {
    // Error-handling code
  }
  etc.
}

bool WriteMySamps(void *pOpaque,
                  DtVirtualOutData* pVirtOut)
{
  // Code processing the generated data,
  // e.g. writing to file
}
```

Figure 3. Attaching a **DtOutpChannel** object to a virtual output.

Just like a **DtDevice** object, a virtual **DtOutpChannel** object should be detached from the hardware after all operations on the channel have been completed.

2.3. Streaming MPLP Data

The core of a multi-PLP modulator program is shown in Figure 4. The code assumes:

- **DtDevice** object **Dvc** and **DtOutpChannel** object **TsOut** have been attached to the hardware;
- Multi-PLP modulation parameters have been set;
- **GetTsData(i, Buf, Max)** is the user-supplied function that writes maximally **Max** new Transport-Stream data bytes in **Buf** for MPLP-FIFO/PLP index **i**, and returns the number of bytes written.

The transmission control is set to **Hold**, which enables multi-PLP modulation and DMA but keeps actual transmission disabled.

```
// PRE: Dvc and TsOut attached
//      MPLP modulation parameters set
char Buf[BUFSIZE];
int NumBytes = 1;

int TxControl = DTAPI_TXCTRL_HOLD;
TsOut.SetTxControl(TxControl);

// Main loop
while (NumBytes != 0)
{
  // Transmission in hold?
  if (TxControl == DTAPI_TXCTRL_HOLD)
  {
    // Check whether initial load reached
    int Load;
    TsOut.GetFifoLoad(Load);
    if (Load >= INILOAD)
    {
      TxControl = DTAPI_TXCTRL_SEND;
      TsOut.SetTxControl(TxControl);
    }
  }

  // Try to fill all input FIFOs
  bool AllFifosFilled = true;
  for (int i=0; i<NumInputs
                    && NumBytes!=0; i++)
  {
    // MPLP FIFO (still) filled?
    int NumFree;
    TsOut.GetMplpFifoFree(i, NumFree);
    if (NumFree < BUFSIZE)
      continue; // Yes; Next FIFO

    AllFifosFilled = false;
    NumBytes = GetTsData(i, Buf, BUFSIZE);
    TsOut.WriteMplp(i, Buf, NumBytes);
  }

  // All FIFOs filled?
  if (AllFifosFilled)
     Sleep(10);    // Sleep for a while

}
```

Figure 4. Streaming data to an output.

When the Transmit FIFO contains its initial load, actual transmission can be started by setting transmission control to **Send**. The main loop then supplies additional data to the MPLP FIFOs until the data sources are exhausted.

The following factors should be considered to achieve optimal results:

- Modulation of a frame is only possible when sufficient data is available for all PLPs. A lengthy transfer to one MPLP FIFO may cause underflow of another MPLP FIFO, stalling the modulation process. To prevent this, the transfer size should not be too large. For efficiency reasons, the transfer size should not be too small either. Therefore it is recommended to use a transfer size between 4K bytes and 32K bytes.
- The initial transmit-FIFO load (**INILOAD**) should not be too small, to prevent an early transmit-FIFO underflow in the main loop. A value close to the maximum hardware FIFO size is recommended.
  Warning: The initial load cannot be larger than the transmit-FIFO size: when the transmit FIFO is full, DMA will stall and the application "hangs."
- As far as DTAPI is concerned, the **GetTsData** function may return Transport-Stream data aligned at arbitrary 4-byte boundaries. However, for many data-generating algorithms, alignment on packet boundaries will be a natural choice. In such applications it is convenient and efficient to set the buffer size to a multiple of the packet size.

2.4. Complete Example

Figure 5 shows the code of a simple DVB-T2 stream generator containing 2 data PLPs and a common PLP.

Obviously, this example is just a first step towards a production-quality stream generator application.

```
// Command-line program T2Sample
// Outputs DVB-T2 signal according to V&V402 through DTA-115
#include "DTAPI.h"
#include <stdio.h>

int main(int argc, char* argv[])
{
    char TempRdBuf[8192];
    DTAPI_RESULT dr;
    DtDevice Dvc;
    DtOutpChannel Outp;

    // Attach to the DTA-115
    dr = Dvc.AttachToType(115);
    if (dr != DTAPI_OK)
        exit(dr);

    // Use the modulator port
    dr = Outp.AttachToPort(&Dvc, 2);
    if (dr != DTAPI_OK)
        exit(dr);

    // Set RF frequency to 666MHz
    dr = Outp.SetRfControl(666000000);
    if (dr != DTAPI_OK)
        exit(dr);

    // Set RF level -20.0 dBm
    dr = Outp.SetOutputLevel(-200);
    if (dr != DTAPI_OK)
        exit(dr);

    // Set default DVB-T2 values
    DtDvbT2Pars DvbT2Pars;

    // Below you'll find the parameter setting according to VV402
    // General parameters
    DvbT2Pars.m_T2Version       = DTAPI_DVBT2_VERSION_1_2_1;
    DvbT2Pars.m_Bandwidth       = DTAPI_DVBT2_8MHZ;
    DvbT2Pars.m_FftMode         = DTAPI_DVBT2_FFT_32K;
    DvbT2Pars.m_GuardInterval   = DTAPI_DVBT2_GI_1_128;
    DvbT2Pars.m_Miso            = DTAPI_DVBT2_MISO_OFF;
    DvbT2Pars.m_Papr            = DTAPI_DVBT2_PAPR_NONE;
    DvbT2Pars.m_BwtExt          = true;
    DvbT2Pars.m_PilotPattern    = 7;
    DvbT2Pars.m_L1Modulation    = DTAPI_DVBT2_BPSK;
    DvbT2Pars.m_CellId          = 0;
    DvbT2Pars.m_NetworkId       = 12421;
    DvbT2Pars.m_T2SystemId      = 32769;
    DvbT2Pars.m_L1Repetition    = false;

    // T2-Frame related parameters
    DvbT2Pars.m_NumT2Frames     = 2;
    DvbT2Pars.m_NumDataSyms     = 27;
    DvbT2Pars.m_NumSubslices    = 108;
```

```
// No FEF
DvbT2Pars.m_FefEnable        = false;

// 1 RF channel
DvbT2Pars.m_NumRfChans       = 1;
DvbT2Pars.m_StartRfIdx       = 0; // n.a. for non-TFS
DvbT2Pars.m_RfChanFreqs[0]   = 666000000;

// 3 PLPs
DvbT2Pars.m_NumPlps          = 3;

// PLP[0] First data PLP
DvbT2Pars.m_Plps[0].m_Id                = 0;
DvbT2Pars.m_Plps[0].m_GroupId           = 0;
DvbT2Pars.m_Plps[0].m_Type              = DTAPI_DVBT2_PLP_TYPE_2;
DvbT2Pars.m_Plps[0].m_Modulation        = DTAPI_DVBT2_QPSK;
DvbT2Pars.m_Plps[0].m_CodeRate          = DTAPI_DVBT2_COD_1_2;
DvbT2Pars.m_Plps[0].m_FecType           = DTAPI_DVBT2_LDPC_64K;
DvbT2Pars.m_Plps[0].m_Hem               = true;
DvbT2Pars.m_Plps[0].m_Npd               = true;
DvbT2Pars.m_Plps[0].m_Issy              = DTAPI_DVBT2_ISSY_LONG;
DvbT2Pars.m_Plps[0].m_IssyBufs          = 1048576;
DvbT2Pars.m_Plps[0].m_IssyTDesign       = 949777;
DvbT2Pars.m_Plps[0].m_CompensatingDelay = -1;   // Auto
DvbT2Pars.m_Plps[0].m_TimeIlType        = DTAPI_DVBT2_IL_ONETOONE;
DvbT2Pars.m_Plps[0].m_TimeIlLength      = 1;
DvbT2Pars.m_Plps[0].m_FrameInterval     = 1;
DvbT2Pars.m_Plps[0].m_FirstFrameIdx     = 0;
DvbT2Pars.m_Plps[0].m_Rotation          = true;
DvbT2Pars.m_Plps[0].m_InBandAFlag       = true;
DvbT2Pars.m_Plps[0].m_NumOtherPlpInBand = 0;
DvbT2Pars.m_Plps[0].m_InBandBFlag       = false;
DvbT2Pars.m_Plps[0].m_FfFlag            = false;
DvbT2Pars.m_Plps[0].m_FirstRfIdx        = 0;
DvbT2Pars.m_Plps[0].m_NumBlocks         = 14;
DvbT2Pars.m_Plps[0].m_TsRate            = 6000000;

// PLP[1] Second data PLP
DvbT2Pars.m_Plps[1].m_Id                = 1;
DvbT2Pars.m_Plps[1].m_GroupId           = 0;
DvbT2Pars.m_Plps[1].m_Type              = DTAPI_DVBT2_PLP_TYPE_2;
DvbT2Pars.m_Plps[1].m_Modulation        = DTAPI_DVBT2_QPSK;
DvbT2Pars.m_Plps[1].m_CodeRate          = DTAPI_DVBT2_COD_1_2;
DvbT2Pars.m_Plps[1].m_FecType           = DTAPI_DVBT2_LDPC_64K;
DvbT2Pars.m_Plps[1].m_Hem               = true;
DvbT2Pars.m_Plps[1].m_Npd               = true;
DvbT2Pars.m_Plps[1].m_Issy              = DTAPI_DVBT2_ISSY_LONG;
DvbT2Pars.m_Plps[1].m_IssyBufs          = 1048576;
DvbT2Pars.m_Plps[1].m_IssyTDesign       = 949777;
DvbT2Pars.m_Plps[1].m_CompensatingDelay = -1;   // Auto
DvbT2Pars.m_Plps[1].m_TimeIlType        = DTAPI_DVBT2_IL_ONETOONE;
DvbT2Pars.m_Plps[1].m_TimeIlLength      = 1;
DvbT2Pars.m_Plps[1].m_FrameInterval     = 1;
DvbT2Pars.m_Plps[1].m_FirstFrameIdx     = 0;
DvbT2Pars.m_Plps[1].m_Rotation          = true;
DvbT2Pars.m_Plps[1].m_InBandAFlag       = true;
```

```
DvbT2Pars.m_Plps[1].m_NumOtherPlpInBand = 0;
DvbT2Pars.m_Plps[1].m_InBandBFlag       = false;
DvbT2Pars.m_Plps[1].m_FfFlag            = false;
DvbT2Pars.m_Plps[1].m_FirstRfIdx        = 0;
DvbT2Pars.m_Plps[1].m_NumBlocks         = 14;
DvbT2Pars.m_Plps[1].m_TsRate            = 6000000;

// PLP[2] Common PLP
DvbT2Pars.m_Plps[2].m_Id                = 2;
DvbT2Pars.m_Plps[2].m_GroupId           = 0;
DvbT2Pars.m_Plps[2].m_Type              = DTAPI_DVBT2_PLP_TYPE_COMM;
DvbT2Pars.m_Plps[2].m_Modulation        = DTAPI_DVBT2_QPSK;
DvbT2Pars.m_Plps[2].m_CodeRate          = DTAPI_DVBT2_COD_1_2;
DvbT2Pars.m_Plps[2].m_FecType           = DTAPI_DVBT2_LDPC_16K;
DvbT2Pars.m_Plps[2].m_Hem               = true;
DvbT2Pars.m_Plps[2].m_Npd               = true;
DvbT2Pars.m_Plps[2].m_Issy              = DTAPI_DVBT2_ISSY_LONG;
DvbT2Pars.m_Plps[2].m_IssyBufs          = 1048576;
DvbT2Pars.m_Plps[2].m_IssyTDesign       = 949777;
DvbT2Pars.m_Plps[2].m_CompensatingDelay = -1;   // Auto
DvbT2Pars.m_Plps[2].m_TimeIlType        = DTAPI_DVBT2_IL_ONETOONE;
DvbT2Pars.m_Plps[2].m_TimeIlLength      = 1;
DvbT2Pars.m_Plps[2].m_FrameInterval     = 1;
DvbT2Pars.m_Plps[2].m_FirstFrameIdx     = 0;
DvbT2Pars.m_Plps[2].m_Rotation          = true;
DvbT2Pars.m_Plps[2].m_InBandAFlag       = true;
DvbT2Pars.m_Plps[2].m_NumOtherPlpInBand = 0;
DvbT2Pars.m_Plps[2].m_InBandBFlag       = false;
DvbT2Pars.m_Plps[2].m_FfFlag            = false;
DvbT2Pars.m_Plps[2].m_FirstRfIdx        = 0;
DvbT2Pars.m_Plps[2].m_NumBlocks         = 9;
DvbT2Pars.m_Plps[2].m_TsRate            = 6000000;


// PLP Inputs
// PLP[0] input uses MPLP FIFO index 0
DvbT2Pars.m_PlpInputs[0].m_DataType     = DtPlpInpPars::TS188;
DvbT2Pars.m_PlpInputs[0].m_FifoIdx      = 0;
DvbT2Pars.m_PlpInputs[0].m_BigTsSplit.m_Enabled = false;

// PLP[1] input uses MPLP FIFO index 1
DvbT2Pars.m_PlpInputs[1].m_DataType     = DtPlpInpPars::TS188;
DvbT2Pars.m_PlpInputs[1].m_FifoIdx      = 1;
DvbT2Pars.m_PlpInputs[1].m_BigTsSplit.m_Enabled = false;

// PLP[2] input uses MPLP FIFO index 2
DvbT2Pars.m_PlpInputs[2].m_DataType     = DtPlpInpPars::TS188;
DvbT2Pars.m_PlpInputs[2].m_FifoIdx      = 2;
DvbT2Pars.m_PlpInputs[2].m_BigTsSplit.m_Enabled = false;

// No virtual output is used through callback functions
DvbT2Pars.m_VirtOutput.m_Enabled = false;

// No test point data output
DvbT2Pars.m_TpOutput.m_Enabled = false;
```

```
// No PAPR ACE
DvbT2Pars.m_PaprPars.m_AceEnabled   = false;

// Only P2 P2 PAPR TR
DvbT2Pars.m_PaprPars.m_TrEnabled    = true;
DvbT2Pars.m_PaprPars.m_TrP2Only     = true;
DvbT2Pars.m_PaprPars.m_TrMaxIter    = 1;
DvbT2Pars.m_PaprPars.m_TrVclip      = 4.32;

// Enable L1 PAPR
DvbT2Pars.m_PaprPars.m_L1AceEnabled = true;
DvbT2Pars.m_PaprPars.m_L1AceCMax    = 0.0;

// PAPR Bias ballancing and bias ballancing cells
DvbT2Pars.m_PaprPars.m_BiasBalancing   = 1;
DvbT2Pars.m_PaprPars.m_NumBiasBalCells = 0;

// No TX signalling
DvbT2Pars.m_TxSignature.m_TxSigAuxEnabled   = false;
DvbT2Pars.m_TxSignature.m_TxSigFefEnabled   = false;

// We have RF output so no T2MI output
DvbT2Pars.m_T2Mi.m_Enabled  = false;

// No RBM validation
DvbT2Pars.m_RbmValidation.m_Enabled = false;

// Check whether parameters are valid
dr = DvbT2Pars.CheckValidity();
if (dr != DTAPI_OK)
    exit(dr);

// Get the TSRates of the PLPs
for (int i=0; i<3; i++)
{
    int TsRate;
    ::DtapiModPars2TsRate(TsRate, DvbT2Pars, i);
    printf("TS-rate PLP[%d]: %d bps\n", i, TsRate);
}

// Set transmitter to IDLE
dr = Outp.SetTxControl(DTAPI_TXCTRL_IDLE);
if (dr != DTAPI_OK)
    exit(dr);

// Initialize the modulator. Multi PLP
dr = Outp.SetModControl(DvbT2Pars, true);

// Set transmitter to HOLD
dr = Outp.SetTxControl(DTAPI_TXCTRL_HOLD);
if (dr != DTAPI_OK)
    exit(dr);
bool InSendMode = false;    // Not in SEND mode (yet)

// Determine the FIFO load threshold
int RfFifoSize;
```

```
dr = Outp.GetFifoSize(RfFifoSize);
if (dr != DTAPI_OK)
    exit(dr);

// Threshold is set to 75% of the FIFO size
int IniLoad = 3*RfFifoSize / 4;

// Open the input files and check the opened files
const int NumInputs = 3;
FILE* Files[NumInputs];
Files[0] = fopen("C:\\Data\\VV402_Plp0.ts", "rb");
Files[1] = fopen("C:\\Data\\VV402_Plp1.ts", "rb");
Files[2] = fopen("C:\\Data\\VV402_Plp2.ts", "rb");
if (Files[0]==NULL || Files[1]==NULL || Files[2]==NULL)
    dr = DTAPI_E;

printf("Press any key to stop...");

// Do while no keyboard key is hit and all is OK
while (!_kbhit() && dr == DTAPI_OK)
{
    // If not in SEND mode yet, check whether we can go to SEND mode
    if (!InSendMode)
    {
        // Get the FIFO load of the RF output
        int RfFifoLoad;
        dr = Outp.GetFifoLoad(RfFifoLoad);
        if (dr != DTAPI_OK)
            break;

        if (RfFifoLoad >= IniLoad)
        {
            // Goto SEND mode
            dr = Outp.SetTxControl(DTAPI_TXCTRL_SEND);
            if (dr != DTAPI_OK)
                break;

            // Now we can enter SEND mode
            InSendMode = true;
        }
    }

    // Lets assume all MPLP FIFOs are filled until found otherwise
    bool AllFifosFilled = true;
    for (int FifoIdx=0; FifoIdx<NumInputs && dr == DTAPI_OK; FifoIdx++)
    {
        // Check the amount free in the MPLP FIFO
        int NumFree;
        dr = Outp.GetMplpFifoFree(FifoIdx, NumFree);
        if (dr != DTAPI_OK)
            break;
        // Skip this MPLP FIFO if too less room is available
        if (NumFree < sizeof(TempRdBuf))
            continue;   // next FIFO

        // This MPLP FIFO is not filled enough
```

```
            AllFifosFilled = false;

            // Read a chunck of data
            int NumRead = (int)::fread(TempRdBuf, 1, sizeof(TempRdBuf),
                                                        Files[FifoIdx]);
            // EOF? then goto begin of file
            if (feof(Files[FifoIdx]))
                ::fseek(Files[FifoIdx], 0, SEEK_SET);

            // Write the data to the MPLP FIFO
            dr = Outp.WriteMplp(FifoIdx, TempRdBuf,  NumRead);
            if (dr != DTAPI_OK)
                break;
        }

        // All FIFOs filled? then sleep for a while, to prevent an endless loop.
        if (AllFifosFilled)
            Sleep(10);
    }

    // Get and print the status of the DVB-T2 modulation
    DtDvbT2ModStatus ModStatus;
    Outp.GetMplpModStatus(&ModStatus );
    printf("\nDVB-T2 Modulator Status:"
            "\n\t#BitrateOVF: %I64d"
            "\n\t#BlockOVF  : %I64d"
            "\n\t#TTO-Error : %I64d\n",
            ModStatus.m_BitrateOverflows,
            ModStatus.m_PlpNumBlocksOverflows,
            ModStatus.m_TtoErrorCount);

    // Set transmitter to IDLE again
    Outp.SetTxControl(DTAPI_TXCTRL_IDLE);

    // Detach hardware
    Outp.Detach(DTAPI_INSTANT_DETACH);
    Dvc.Detach();

    // Close the input files
    for (int i=0; i<NumInputs; i++)
        if (Files[i] != NULL)
            fclose(Files[i]);

    return dr;
}
```

Figure 5. DVB-T2 stream generator with the DTA-115.

3. DTAPI Methods for multi-PLP Modulation

3.1. Overview

Table 1. DTAPI – Global Functions

API Function	Description
::DtapiModPars2TsRate	Compute TS rate from modulation parameters

Table 2. DTAPI – `DtOutpChannel` Functions

API Function	Description
AttachVirtual	Attach channel to virtual output
GetMplpFifoFree	Get amount free in a multi-PLP modulator FIFO
GetMplpFifoSize	Get size of a multi-PLP modulator FIFO
GetMplpModStatus	Get status of the multi-PLP modulator
SetMplpChannelModelling	Set channel modelling parameters for the multi-PLP modulator
SetModControl	Set modulation parameters
WriteMplp	Write data to a multi-PLP modulator FIFO

Struct DtBigTsSplitPars

Structure for specifying the parameters for the "Big-TS splitting" operation. This operation splits one "big" Transport Stream into multiple SPTSes (Single Program Transport Streams), one for each data PLP in the group. Each SPTS will contain one service and adapted PSI/SI. The Transport Stream for the common PLP gets the common SI.

The parameters in this structure are used for the creation and modification of PAT, SDT and EIT tables for a single PLP. Furthermore it specifies the PIDs to be included in the Transport Stream. This structure is used in class **DtPlpInPars**.

```
struct DtBigTsSplitPars
{
  bool  m_Enabled;            // Enable "Big-TS splitting"
  bool  m_IsCommonPlp;        // Common PLP (yes/no)
  bool  m_SplitSdtIn;         // SDT is already split (yes/no)
  std::vector<int>  m_Pids;   // Series of PIDs to include

  // Parameters below are not used in case m_IsCommonPlp == true
  int  m_OnwId;               // Original Network ID of the Big TS
  int  m_TsId;                // Transport Stream ID of the Big TS
  int  m_ServiceId;           // ID of the service to include in PLP
  int  m_PmtPid;              // PID of the PMT table of selected service
  int  m_NewTsId;             // Transport Stream ID of the TS in the PLP

  // Parameters below are not used in case m_SplitSdtIn == true
  int  m_SdtLoopDataLength;           // SDT loop data length
  unsigned char  m_SdtLoopData[168];  // The SDT-actual loop data
};
```

Members

m_Enabled
If true, "Big-TS splitting" is enabled, otherwise it is disabled and the remaining parameters are not used.

m_IsCommonPlp
If true, the type of the associated PLP is a common PLP, otherwise the type is a data PLP.

m_SplitSdtIn
If true, the "Big TS" is "MPLP-prepared" and already contains separated SDT subtables for each PLP.

m_Pids
Series of PID values that specify the elementary streams to be included in Transport Stream for the associated PLP (e.g. for the data PLP: service components, ECM and PCR PIDs and for the common PLP: CAT, NIT, TOT, TDT-table PIDs).

The following parameters are not used if parameters are related to a common PLP (*m_IsCommonPlp* equals *true*).

m_OnwId, m_TsId, m_ServiceId
Identifies a service from the "Big TS" to include in the Transport Stream for the PLP.

m_PmtPid
The PID of the PMT-table of the selected service, needed for the creation of a new PAT-table.

m_NewTsId
Specifies the Transport Stream ID of the newly created TS in the PLP.

The following parameters are not used if the "Big TS" already contains separated SDT subtables for each PLP (*m_SplitSdtIn* equals *true*); otherwise, a new SDT-actual table is created for the selected service with the aid of the parameters below.

m_SdtLoopDataLength
Length of the new SDT-loop data for the selected service. The valid range is 0, 5 … 168.

m_SdtLoopData
Specifies the new SDT-actual loop data for the selected service. The SDT-loop data starts with the service_id field and includes the SDT-loop descriptors. The maximum length of the SDT-loop data is 168 bytes.

Struct DtComplexFloat

Structure describing a complex floating-point number.

```
Struct DtComplexFloat
{
   int   m_Re;                  // Real part
   int   m_Im;                  // Imaginary part
};
```

Members

m_Re
The real part of the complex floating-point number.

m_Im
The imaginary part of the complex floating-point number.

Struct DtPlpInpPars

Structure for specifying the input stream for a PLP. This structure is used in class **DtDvbC2Pars** and in class **DtDvbT2Pars**, in an array of structs. The index in the array corresponds to the index of the related PLP.

```
struct DtPlpInpPars
{
   int   m_FifoIdx;                    // Index of input FIFO
   InDataType   m_DataType;            // Input data type
   DtBigTsSplitPars   m_BigTsSplit;    // Big-TS splitting parameters
};
```

Members

m_FifoIdx

The index of the FIFO used by the associated PLP. PLPs in the same group that have "Big-TS" splitting enabled can share the same input FIFO.

The index will be used in several methods that operate on a specific FIFO (e.g. **DtOutpChannel::WriteMplp()**).

The default value of *m_FifoIdx* is equal to the index in the array of **DtPlpInpPars** structs. For writing data to the n^{th} PLP (which is specified at index n in the array of **DtPlpInpPars**) you have to use FIFO index n.

The valid range of *m_FifoIdx* is 0 … 255.

m_DataType

Specifies the type of the input data.

Value	Meaning
TS188	188-byte TS packets
TS204	204-byte TS packets

m_BigTsSplit

Specifies (for this PLP) the parameters for the "Big-TS" splitting operation.

Struct DtTestPointOutPars

Test-point data generation is specified by the Verification and Validation (V&V) group for DVB-C2 and DVB-T2 as a means for the verification and validation of the DVB specifications. Structure **DtTestPointOutPars** enables or disables test-point data generation, and – if enabled – specifies the associated handler.

This structure is used in class **DtDvbC2Pars** and in class **DtDvbT2Pars**.

```
struct DtTestPointOutPars
{
  bool  m_Enabled;                    // Enable test points (yes/no)
  void*  m_pTpWriteDataOpaque;        // Opaque pointer
  DtTpWriteDataFunc*  m_pTpWriteDataFunc; // Test-point data handler
};
```

Members

m_Enabled

If true, the generation of test point data is enabled. Whenever test point data is available, the callback function is called and the test point data is passed to the callback function. Note that test point data generation cannot be performed in real time.

m_pTpWriteDataOpaque

Opaque pointer that is passed to the callback function.

m_pTpWriteDataFunc

Pointer to the callback function of type **DtTpWriteDataFunc** that handles the generated test point data.

Struct DtVirtualOutData

Structure describing the type of output data generated by a virtual output.

```
struct DtVirtualOutData
{
 OutDataType  m_DataType;                // Output data type

 union {

   struct {                              // 16bit int I/Q samples
     const unsigned char**  m_pBuffer;   // Array of buffers
     int  m_NumBuffers;                  // #Buffers
     int  m_NumBytes;                    // #Bytes in each buffer
   } IqSamplesInt16;

   struct {                              // 32bit float I/Q samples
     const unsigned char**  m_pBuffer;   // Array of buffers
     int  m_NumBuffers;                  // #Buffers
     int  m_NumBytes;                    // #Bytes in each buffer
   } IqSamplesFloat32;

   struct {                              // 188byte T2MI TS packets
     const unsigned char*  m_pBuffer;    // Pointer to TS packet(s)
     int  m_NumBytes;                    // #Bytes
     __int64  m_T2MiFrameNr;             // T2MI frame counter
   } T2MiTs188;
 } u;
};
```

Members

m_DataType

Type of output data.

Value	Meaning
IQ_INT16	Pairs of signed 16-bit integers in I, Q order, little Endian
IQ_FLOAT32	Pairs of 32-bit floats in I, Q order
T2MI_TS188	T2-MI packets encapsulated into DVB/MPEG Transport Stream packets

u.IqSamplesInt16

Structure used in case *m_DataType* equals **IQ_INT16**.

u.IqSamplesInt16.m_pBuffer

Pointer to an array of *m_NumBuffers* pointers to buffers of length *m_NumBytes*.
The buffers contain pairs of signed 16-bit integers in I, Q order, little Endian.

u.IqSamplesInt16.m_NumBuffers

The number of buffers. There is one output buffer for each output channel (e.g. 2 buffers in case of MISO).

u.IqSamplesInt16.m_NumBytes

The number of bytes in each buffer.

u.IqSamplesFloat32
Structure used in case *m_DataType* equals **IQ_Float32**.

u.IqSamplesFloat32.m_pBuffer
Pointer to an array of *m_NumBuffers* pointers to buffers of *m_NumBytes* length.
The buffers contain pairs of 32-bit floats in I, Q order.

u.IqSamplesFloat32.m_NumBuffers
The number of buffers. There is one output buffer for each output channel (e.g. 2 buffers in case of MISO).

u.IqSamplesFloat32.m_NumBytes
The number of bytes in each buffer.

u.T2MiTs188
Structure used in case *m_DataType* equals **T2MI_TS188**.

u.T2MiTs188.m_pBuffer
Pointer to a buffer with 188-byte Transport Packets encapsulating T2-MI packets.

u.T2MiTs188.m_NumBytes
The number of bytes in the buffer.

u.T2MiTs188.m_T2MiFrameNr
DVB-T2 superframe counter. The counter is incremented each time the buffer contains a packet that contributes to a new DVB-T2 superframe. This parameter enables cutting of the output data stream at DVB-T2 superframe boundaries.

Struct DtVirtualOutPars

Structure for specifying the output data type in case the output data is generated for a virtual output.

```
struct DtVirtualOutPars
{
  bool  m_Enabled;                              // Parameters enabled
  DtVirtualOutData::OutDataType  m_DataType;    // Output data type
  double  m_Gain;                               // RMS of the I/Q samples
};
```

Members

m_Enabled

If true, the parameters in **DtVirutalOutPars** overrule the default values; otherwise, default output data type and gain will be used.

m_DataType

Specifies the type of output data for the virtual output.

Value	Meaning
IQ_INT16	Pairs of signed 16-bit integers in I, Q order, little Endian
IQ_FLOAT32	Pairs of 32-bit floats in I, Q order
T2MI_TS188	T2-MI packets encapsulated into DVB/MPEG Transport Stream packets

m_Gain

If the output data type is either **IQ_INT16** or **IQ_FLOAT32**, this field specifies the Root Mean Square (RMS) of the complex samples. This value should be set as large as possible to have the largest SNR, but small enough to avoid saturation. When a DekTec card is used for play-out of the I/Q samples, the value 5000 is an appropriate value.

Struct DtDvbC2DSlicePars

Structure describing DVB-C2 parameters for one data slice. This structure is used in class **DtDvbC2Pars**, in an array **of DTAPI_DVBC2_NUM_DSLICE_MAX** structs for the data slices.

```
struct DtDvbC2DSlicePars
{
   int   m_Id;                  // Data slice ID
   int   m_TunePosition;        // Tune position
   int   m_OffsetLeft;          // Data slice left offset (start position)
   int   m_OffsetRight;         // Data slice right offset (end position)
   int   m_TiDepth;             // Time interleaving depth
   int   m_Type;                // Data slice type
   int   m_FecHdrType;          // FEC header type
   bool   m_ConstConfig;        // Constant data slice configuration(yes/no)
   bool   m_LeftNotch;          // Left notch present (yes/no)
   std::<vector<DtDvbC2PlpPar> m_Plps; // PLPs
};
```

Members

m_Id

Unique identification of the data slice within a C2-System. The valid range is 0 … 255.

m_TunePosition

Tune position of the associated data slice relative to the start frequency of the C2-System, in multiples of pilot carrier spacing.
The valid range is 0 … 8191 if the guard interval is 1/128.
The valid range is 0 … 16383 if the guard interval is 1/64.

m_OffsetLeft

Start position of the associated data slice by means of the distance to the left from the tuning position, in multiples of the pilot carrier spacing.
The valid range is -128 … 127 if the guard interval is 1/128.
The valid range is -256 … 255 if the guard interval is 1/64.

m_OffsetRight

End position of the associated data slice by means of the distance to the right from the tuning position, in multiples of the pilot carrier spacing.
The valid range is -128 … 127 if the guard interval is 1/128.
The valid range is -256 … 255 if the guard interval is 1/64.
If *m_OffsetLeft* equals *m_OffsetRight*, the data slice is empty and no input streams are created for the PLPs of the data slice.

m_TiDepth

Time interleaving depth within the associated data slice.

Value	Meaning
DTAPI_DVBC2_TIDEPTH_NONE	No time interleaving
DTAPI_DVBC2_TIDEPTH_4	4 OFDM symbols
DTAPI_DVBC2_TIDEPTH_8	8 OFDM symbols
DTAPI_DVBC2_TIDEPTH_16	16 OFDM symbols

m_Type

Data slice type.

Value	Meaning
DTAPI_DVBC2_DSLICE_TYPE_1	Data slice type 1
DTAPI_DVBC2_DSLICE_TYPE_2	Data slice type 2

m_FecHdrType

FEC frame header type.

Value	Meaning
DTAPI_DVBC2_FECHDR_TYPE_ROBUST	Robust mode
DTAPI_DVBC2_FECHDR_TYPE_HEM	High efficiency mode

m_ConstConfig

If true, indicates that the configuration of the associated data slice shall not change; otherwise, the configuration is assumed to be variable.

m_LeftNotch

If true, indicates the presence of a left neighboured notch band.

m_Plps

A vector specifying the DVB-C2 modulation parameters for the physical layer pipes.

Struct DtDvbC2ModStatus

Structure containing the status of the DVB-C2 modulator. This structure is an output parameter of **`DtOutpChannel::GetMplpModStatus`**.

```
struct DtDvbC2ModStatus
{
   __int64  m_DjbOverflows;   // Number of DJB overflows
   __int64  m_DjbUnderflows;  // Number of DJB underflows
};
```

Members

`m_DjbOverflows`
Total number De-Jitter Buffer overflows.
If such overflow occurs, the *`DtDvbC2PlpPars::m_IssyOutputDelay`* parameter must be decreased or *`DtDvbC2PlpPars::m_IssyBufs`* must be increased.

`m_DjbUnderflows`
Total number De-Jitter Buffer underflows.
If such underflow occurs, the *`DtDvbC2PlpPars::m_IssyOutputDelay`* parameter must be increased.

Struct DtDvbC2NotchPars

Structure specifying a DVB-C2 notch band. This structure is used in class **DtDvbC2Pars**, in an array **of DTAPI_DVBC2_NUM_NOTCH_MAX** structs.

```
struct DtDvbC2NotchPars
{
   int  m_Start;              // Notch start
   int  m_Width;              // Notch width
};
```

Members

m_Start

Start position of the notch band relative to the start frequency of the C2-System. The start position is indicated in multiples of pilot carrier spacing.
The valid range is 0 … 8191 if the guard interval is 1/128.
The valid range is 0 … 16383 if the guard interval is 1/64.

m_Width

Width of the notch band indicated in multiples of pilot carrier spacing.
The valid range is 0 … 255 if the guard interval is 1/128.
The valid range is 0 … 511 if the guard interval is 1/64.

Struct DtDvbC2PaprPars

Structure for specifying PAPR reduction parameters. This structure is used in class **DtDvbC2Pars**.

```
struct DtDvbC2PaprPars
{
   bool   m_TrEnabled;          // PAPR TR enabled
   double  m_TrVclip;           // Clipping threshold
   int   m_TrMaxIter;           // Maximum number of iterations
};
```

Members

m_TrEnabled
If true, PAPR TR is active, otherwise PAPR TR is not active.

m_TrVclip
PAPR TR clipping threshold. The valid range is 1 … 4.32 (Volt).

m_TrMaxIter
Maximum number of iterations. Must be greater than or equal to 1.
Note: PAPR TR processing time is proportional to this parameter.

Struct DtDvbC2ParamInfo

Structure containing the DVB-C2 "derived" parameters: the value of the members follows from the basic DVB-C2 modulation parameters.

This structure is an output parameter of **`DtDvbC2Pars::GetParamInfo`**.

```
struct DtDvbC2ParamInfo
{
   int  m_L1Part2Length;      // Number of bits of the L1 part2 data
   int  m_NumL1Symbols;       // Total number of symbols per frame
   int  m_NumSymbols;         // Number of L1 symbols
   int  m_PilotSpacing;       // Distance between pilots
   int  m_FftSize;            // FFT size
   int  m_MinCarrierOffset;   // Lowest used carrier offset
   int  m_CenterFrequency;    // Center frequency
};
```

Members

m_L1Part2Length
Number of bits of the L1 part 2 data (including CRC).

m_NumL1Symbols
Number of L1 symbols (L_P).

m_NumSymbols
Total number of symbols per frame ($L_P + L_{data}$).

m_PilotSpacing
The number of carriers between pilots (D_X).

m_FftSize
FFT size.

m_MinCarrierOffset
The lowest used carrier offset.

m_CenterFrequency
Center frequency, expressed as the distance from 0 Hz in multiples of the carrier spacing.

Struct DtDvbC2PlpPars

Structure specifying the DVB-C2 modulation parameters for one physical layer pipe. This structure is used in class **DtDvbC2DSlicePars**.

```
struct DtDvbC2PlpPars
{
   bool  m_Hem;                 // High Efficiency Mode (yes/no)
   bool  m_Npd;                 // Null Packet Deletetion (yes/no)
   int   m_Issy;                // ISSY mode
   int   m_IssyBufs;            // ISSY BUFS
   int   m_IssyOutputDelay;     // ISSY output delay in T units
   int   m_TsRate;              // Transport stream rate
   int   m_Ccm;                 // ACM/CCM bit in the BBFrame header 0 or 1
   int   m_Id;                  // PLP ID
   int   m_Type;                // PLP type
   bool  m_Bundled;             // PLP bundled (yes/no)
   int   m_GroupId;             // PLP group ID
   int   m_FecType;             // FEC type
   int   m_CodeRate;            // Code rate
   int   m_Modulation;          // Modulation type
   int   m_HdrCntr;             // Header counter
   std::vector<DtDvbC2XFecFrameHeader>  m_AcmHeadres;    // ACM headers
   bool  m_PsiSiReproc;         // PSI/SI reprocessing is performed (yes/no)
   int   m_TsId;                // Transport stream ID
   int   m_OnwId;               // Original network ID
};
```

Members

m_Hem
If true, the PLP uses High Efficiency Mode (HEM), otherwise Normal Mode (NM) is used.

m_Npd
If true, null-packet deletion is active, otherwise it is not active.

m_Issy
ISSY mode, according to the table below.

Value	Meaning
DTAPI_DVBC2_ISSY_NONE	No ISSY field is used
DTAPI_DVBC2_ISSY_SHORT	2 byte ISSY field is used
DTAPI_DVBC2_ISSY_LONG	3 byte ISSY field is used

m_IssyBufs
ISSY 'BUFS' value. The valid range is 0 … 2097151

m_IssyOutputDelay
Delay (in T units) between the incoming data and the output data in the receiver model. This value determines the minimum and maximum dejitter buffer usage and is used to compute the ISSY 'BUFSTAT' field.

m_TsRate
Transport-Stream rate in bps. If *m_TsRate* is set to '0', no ISSY is used and null-packet deletion is not active then the transport stream rate is computed from the PLP parameters.

m_Ccm
ACM/CCM-field (Adaptive Coding and Modulation or Constant Coding and Modulation) in the BBFrame header 0 or 1.

m_Id
Unique identification of the PLP within a C2-System. The valid range is 0 … 255.

m_Bundled
If true, the associated PLP is bundled with other PLP(s) within the current C2 System. All the bundled PLPs have the same PLP ID. An input stream is created only for the first PLP of the bundle.

m_Type
PLP type.

Value	Meaning
DTAPI_DVBC2_PLP_TYPE_COMMON	Common PLP
DTAPI_DVBC2_PLP_TYPE_GROUPED	Grouped data PLP
DTAPI_DVBC2_PLP_TYPE_NORMAL	Normal data PLP

m_GroupId
Identifies the PLP group with which the PLP is associated. The valid range is 0 … 255.

m_FecType
FEC type used by the PLP.

Value	Meaning
DTAPI_DVBC2_LDPC_16K	16K LDPC
DTAPI_DVBC2_LDPC_64K	64K LDPC

m_CodeRate
Convolutional coding rate used by the PLP.

Value	Meaning
DTAPI_DVBC2_COD_2_3	2/3
DTAPI_DVBC2_COD_3_4	3/4
DTAPI_DVBC2_COD_4_5	4/5
DTAPI_DVBC2_COD_5_6	5/6
DTAPI_DVBC2_COD_8_9	8/9 (for 16K FEC)
DTAPI_DVBC2_COD_9_10	9/10 (for 64K FEC)

m_Modulation

Modulation used by the PLP.

Value	Meaning
DTAPI_DVBC2_QAM16	16-QAM
DTAPI_DVBC2_QAM64	64-QAM
DTAPI_DVBC2_QAM256	256-QAM
DTAPI_DVBC2_QAM1024	1024-QAM
DTAPI_DVBC2_QAM4096	4096-QAM
DTAPI_DVBC2_QAM16384	16384-QAM
DTAPI_DVBC2_QAM65536	65536-QAM

m_HdrCtr

Header counter field, number of FECFrames following the FECFrame header: 0=1 FECFrame; 1=2 FECFrames.

m_AcmHeaders

A vector that holds the XFEC Frame modulation parameters for Adaptive Coding and Modulation (ACM) testing. If the number of ACM headers is greater than zero, then the successive XFEC frames of this PLP use the modulation and coding parameters from the m_AcmHeaders vector. After the last value is used, it loops again to the start of the vector. In this case the *m_FecType*, *m_Modulation*, *m_CodeRate* and *m_HdrCntr* parameters from the **DtDvbC2PlpPars** structure are ignored.

m_PsiSiReproc

If true, indicates that PSI/SI has been reprocessed.

m_TsId, *m_OnwId*

If *m_PsiSiReproc* is set to 'false', these members specify the Transport Stream ID and Original Network ID of the TS in the PLP. A receiver will use these fields if it can't rely on the PSI/SI.

Struct DtDvbC2XFecFrameHeader

Structure describing the coding and modulation parameters for a series of XFEC frames. This structure is used in class **DtDvbC2PlpPars**.

```
struct DtDvbC2XFecFrameHeader
{
   int   m_FecType;            // PLP FEC type
   int   m_Modulation;         // PLP modulation
   int   m_CodeRate;           // PLP code rate
   int   m_HdrCntr;            // Header counter
   int   m_XFecFrameCount;     // Number XFEC frames using these parameters
};
```

Members

m_FecType
PLP FEC type. See **DtDvbC2PlpPars** for a list of applicable values.

m_Modulation
PLP modulation. See **DtDvbC2PlpPars** for a list of applicable values.

m_CodeRate
PLP code rate. See **DtDvbC2PlpPars** for a list of applicable values.

m_HdrCntr
PLP header counter. See **DtDvbC2PlpPars** for a list of applicable values.

m_XFecFrameCount
Number of XFEC frames using the parameters.

Struct DtDvbT2AuxPars

Structure for specifying AUX stream parameters, which can be inserted for test purposes. This structure is used in class **DtDvbT2Pars**.

```
struct DtDvbT2AuxPars
{
   int    m_NumDummyStreams;   // Number of dummy AUX streams
};
```

Members

m_NumDummyStreams

Number of dummy AUX streams added for test purposes.
If TX signature through AUX streams is enabled, the valid range is 0 ...14; otherwise, the valid range is 0 ...15.

Struct DtDvbT2MiPars

Structure for enabling T2-MI generation, and for specifying its parameters. This structure is used in class **DtDvbT2Pars**.

```
Struct DtDvbT2MiPars
{
  bool  m_Enabled;            // Enable T2-MI output
  int   m_Pid;                // T2-MI data PID
  int   m_PcrPid;             // T2-MI PCR PID
  int   m_PmtPid;             // T2-MI PMT PID
  int   m_TsRate;             // T2-MI Transport-Stream rate
  int   m_TimeStamping;       // T2-MI timestamping
  __int64  m_SecSince2000;    // First T2-MI output timestamp value
  int   m_Subseconds;         // Number of subseconds
  int   m_T2miUtco;           // Offset in seconds between UTC and Y2000
  bool  m_EncodeFef;          // Encode FEF (yes/no)
};
```

Members

m_Enabled
If true, T2-MI generation is enabled. An MPEG-2 Transport Stream is generated containing Transport Packets that encapsulate the T2-MI packets.

m_Pid
PID carrying the T2-MI packet data. The valid range is 0 … 8190.

m_PcrPid
PID carrying PCR values. If *m_PcrPid* equals -1, no PCRs are inserted in the Transport Stream; otherwise a PCR is inserted on the indicated PID once per 40ms. The valid range is -1 … 8190.

m_PmtPid
PID carrying the PMT-table. If *m_PmtPid* equals -1, no PAT and no PMT-table are inserted in the Transport Stream; otherwise, PAT and PMT are inserted on PID 0 once per 100ms. The valid range is -1 … 8190.

m_TsRate
T2-MI Transport-Stream rate in bits per second.

m_TimeStamping
Type of DVB-T2 timestamps to insert.

Value	Meaning
DTAPI_DVBT2MI_TIMESTAMP_NULL	Null timestamp
DTAPI_DVBT2MI_TIMESTAMP_REL	Relative timestamps. Use *m_Subseconds*.
DTAPI_DVBT2MI_TIMESTAMP_ABS	Absolute timestamps. Use *m_SecSince2000*, *m_Subseconds* and *m_T2MiUtco*.

m_SecSince2000

Number of seconds since 2000-01-01 00:00:00 UTC. This value is inserted in the first DVB-T2 timestamp that is generated. Subsequent timestamps are computed.
This field is used if *m_TimeStamping* equals **DTAPI_DVBT2MI_TIMESTAMP_ABS.**

m_Subseconds

Number of subsecond units (T_{sub}) elapsed since the time expressed in the seconds field. This value is inserted in the first generated DVB-T2 timestamp. Subsequent timestamps are computed.
This field is used if *m_TimeStamping* is either **DTAPI_DVBT2MI_TIMESTAMP_REL** or **TAPI_DVBT2MI_TIMESTAMP_ABS.**
The T2 system bandwidth defines the units of the subseconds as shown in the table below.

Bandwidth	Subseconds units, T_{sub}
1.7 MHz	1/131 μs
5 MHz	1/40 μs
6 MHz	1/48 μs
7 MHz	1/56 μs
8 MHz	1/64 μs
10 MHz	1/80 μs

m_T2MiUtco

Offset in seconds between UTC and *m_SecSince2000*. As of February 2009 the value shall be 2 and shall change as a result of each new leap second. This field is used if *m_TimeStamping* equals **DTAPI_DVBT2MI_TIMESTAMP_ABS.**

m_EncodeFef

If true, generates a FEF part composite packet with the required subpart. Otherwise, only generates a FEF part NULL packet when FEF is enabled.

Struct DtDvbT2ModStatus

Structure containing the status of the DVB-T2 modulator. This structure is an output parameter of **DtOutpChannel::GetMplpModStatus**.

```
struct DtDvbT2ModStatus
{
   __int64  m_PlpNumBlocksOverflows;   // Number of PLP block overflows
   __int64  m_BitrateOverflows;        // Number of bitrate overflows
   __int64  m_TtoErrorCount;           // Number of invalid TTOs
   // T2MI specific
   __int64  m_T2MiOutputRateOverflows; // Number of T2MI rate overflows
   int  m_T2MiOutputRate;              // Current effective T2MI rate
};
```

Members

m_PlpNumBlocksOverflows
Total number of FEC frames for which the requested number of PLP blocks is greater than *DtDvbT2PlpPars::m_NumBlocks*. An overflow results in an invalid stream.

m_BitrateOverflows
Total number FEC frames for which too many bits were allocated. An overflow results in an invalid stream.

m_TtoErrorCount
Number of times the generated TTO value was invalid. Typically this occurs if *DtDvbT2PlpPars::m_IssyTDesign* is too small.

m_T2MiOutputRateOverflows
Number of T2-MI bitrate overflows. The *DtDvbT2MiPars::m_TsRate* must be increased for reliable operation.

m_T2MiOutputRate
Current T2-MI rate excluding null packets in bps.

Struct DtDvbT2PaprPars

Structure for specifying the PAPR reduction parameters. This structure is used in class **DtDvbT2Pars**.

```
struct DtDvbT2PaprPars
{
  bool   m_AceEnabled;        // PAPR ACE enabled
  double  m_AceVclip;         // ACE clipping threshold
  double  m_AceGain;          // ACE gain
  double  m_AceLimit;         // ACE limit
  int   m_AceInterpFactor;    // ACE interplation factor
  int   m_AcePlpIndex;        // PLP used for PAPR ACE

  bool   m_TrEnabled;         // PAPR TR enabled
  bool   m_TrP2Only           // PAPR TR is only applied on the P2 symbol
  double  m_TrVclip;          // TR clipping threshold
  int   m_TrMaxIter;          // TR maximum number of iterations

  int   m_L1ExtLength;        // L1 extension field length

  bool   m_L1AceEnabled;      // L1 ACE enabled
  double  m_L1AceCMax;        // L1 ACE max constellation extension value

  // Parameters only applicable for DVB-T2 V1.2.1
  int   m_NumBiasBalCells;    // Number cells added to reduce P2 PAPR
  int   m_BiasBalancing;      // L1 bias compensation
};
```

Members

m_AceEnabled
If true, PAPR ACE is active, otherwise PAPR ACE is not active.

m_AceVclip
PAPR ACE clipping threshold. The valid range is 1 … 4.32 (Volt).

m_AceGain
PAPR ACE gain. The valid range is 0 … 31 (steps of 1).

m_AceLimit
PAPR ACE limit. The valid range is 0.7 … 1.4 (steps of 0.1).

m_AceInterpFactor
PAPR ACE interpolation factor. The valid range is 1 … 4.
Note: PAPR ACE processing time is proportional to this parameter.

m_AcePlpIndex
PLP used for the PAPR ACE.

m_TrEnabled
If true, PAPR TR is active, otherwise PAPR TR is not active.

m_TrP2Only
If true, PAPR TR is only applied on the P2 symbol, otherwise PAPR TR is applied on all symbols.

m_TrVclip
PAPR TR clipping threshold. The valid range is 1 … 4.32 (Volt).

m_TrMaxIter
Maximum number of iterations. Must be greater than or equal to 1.
Note: PAPR TR processing time is proportional to this parameter.

m_L1ExtLength
L1 extension field length. The valid rang is 0 … 65535.

m_L1AceEnabled
If true, L1 ACE is active, otherwise L1 ACE is not active.

m_L1AceCMax
Maximum value added to extend the QAM constellation values of L1.

m_NumBiasBalCells
Number of dummy cells added to reduce the P2 PAPR.
The valid range is 0 … *DtDvbT2ParamInfo::m_BiasBalCellsMax.*

m_BiasBalancing
L1 bias balancing.

Value	Meaning
DTAPI_DVBT2_BIAS_BAL_OFF	No L1 bias compensation
DTAPI_DVBT2_BIAS_BAL_ON	Modify the L1 reserved fields and L1 extension field padding to compensate the L1 bias

Struct DtDvbT2ParamInfo

Structure containing the DVB-T2 "derived" parameters: the value of the members follows from the basic DVB-T2 modulation parameters.

This structure is an output parameter of **`DtDvbT2Pars::GetParamInfo`** and **`DtDvbT2Pars::OptimisePlpNumBlocks`**.

```
struct DtDvbT2ParamInfo{
   int  m_TotalCellsPerFrame;     // Total number of cells per frame
   int  m_L1CellsPerFrame;        // #L1 cells per frame
   int  m_AuxCellsPerFrame;       // #Aux stream cells per frame
   int  m_BiasBalCellsPerFrame;   // #Bias balancing cells per frame
   int  m_BiasBalCellsMax;        // Max #bias balancing cells
   int  m_DummyCellsPerFrame;     // #Dummy cells per frame
};
```

Members

m_TotalCellsPerFrame
Total number of cells per frame.

m_L1CellsPerFrame
Total number of cells per frame used for L1 signaling.

m_AuxCellsPerFrame
Total number of auxiliary stream cells per frame.

m_BiasBalCellsPerFrame
Total number of L1 bias balancing cells per frame.

m_BiasBalCellsMax
Maximum number of L1 bias balancing cells per P2.

m_DummyCellsPerFrame
Total number of cells lost per frame; dummy cells overhead = *m_DummyCellsPerFrame* / *m_TotalCellsPerFrame*. It is only computed for the first frame.

Struct DtDvbT2PlpPars

Structure specifying the DVB-T2 modulation parameters for one PLP (Physical Layer Pipe). This structure is used in class **DtDvbT2Pars**, in an array of **DTAPI_DVBT2_NUM_PLP_MAX** structs for the physical layer pipes.

```
struct DtDvbT2PlpPars
{
  bool  m_Hem;                // High Efficiency Mode (yes/no)
  bool  m_Npd;                // Null Packet Deletetion (yes/no)
  int   m_Issy;               // ISSY mode
  int   m_IssyBufs;           // ISSY BUFS
  int   m_IssyTDesign;        // ISSY T_design value
  int   m_CompensationDesign;// Additional delay in samples
  int   m_TsRate;             // Transport Stream rate
  int   m_Id;                 // PLP ID
  int   m_GroupId;            // PLP group ID
  int   m_Type;               // PLP type
  int   m_CodeRate;           // Code rate
  int   m_Modulation;         // Modulation type
  bool  m_Rotation;           // Constellation rotation (yes/no)
  int   m_FecType;            // FEC type
  int   m_FrameInterval;      // T2-frame interval
  int   m_FirstFrameIdx;      // First frame index
  int   m_TimeIlLength;       // Time interleaving length
  int   m_TimeIlType;         // Timer interleaving type
  bool  m_InBandAFlag;        // In-band A signalling information (yes/no)
  bool  m_InBandBFlag;        // In-band B signalling information (yes/no)
  bool  m_NumBlocks;          // Maximum number of FEC blocks per IL frame

  int   m_NumOtherPlpInBand; // Number of other PLPs in the in-band sign
  int   m_OtherPlpInBand[DTAPI_DVBT2_NUM_PLP_MAX-1];
                              // Array of IDs of the other in band PLPs

  // Parmeters below are only meaningful for type 1 PLPs in TFS system.
  bool  m_FfFlag;             // FF flag
  int   m_FirstRfIdx;         // First TFS RF channel where PLP occurs
};
```

Members

m_Hem
If true, the PLP uses High Efficiency Mode (HEM); otherwise, Normal Mode (NM) is used.

m_Npd
If true, null-packet deletion is active.

m_Issy
ISSY mode.

Value	Meaning
DTAPI_DVBT2_ISSY_NONE	No ISSY field is used
DTAPI_DVBT2_ISSY_SHORT	2-byte ISSY field is used
DTAPI_DVBT2_ISSY_LONG	3-byte ISSY field is used

m_IssyBufs
ISSY 'BUFS' value. The valid range is 0 … 2097151

m_IssyTDesign
T_design value for TTO generation. Set to '0' to have the modulator choose the value. T_design is defined as the delay (in samples) between the start of the first T2 frame in which the PLP is mapped and the first output bit of the Transport Stream.

m_CompensatingDelay
Additional delay (in samples) before the TS data is sent. Set to '-1' to have the modulator choose the value.

m_TsRate
Tranport stream rate in bps. If *m_TsRate* is set to '0' and no null-packet deletion is active then the transport stream rate is computed from the PLP paramters.

m_Id
Unique identification of the PLP within a T2 system. The valid range is 0 … 255.

m_GroupId
Identifies the PLP group with which the PLP is associated. The valid range is 0 … 255.

m_Type
PLP type.

Value	Meaning
DTAPI_DVBT2_PLP_TYPE_COMM	Common PLP
DTAPI_DVBT2_PLP_TYPE_1	Data PLP type1
DTAPI_DVBT2_PLP_TYPE_2	Data PLP type2

m_CodeRate
Convolutional coding rate used by the PLP.

Value	Meaning
DTAPI_DVBT2_COD_1_2	1/2
DTAPI_DVBT2_COD_3_5	3/5
DTAPI_DVBT2_COD_2_3	2/3
DTAPI_DVBT2_COD_3_4	3/4
DTAPI_DVBT2_COD_4_5	4/5
DTAPI_DVBT2_COD_5_6	5/6

m_Modulation

Modulation used by the PLP.

Value	Meaning
DTAPI_DVBT2_BPSK	BPSK
DTAPI_DVBT2_QPSK	QPSK
DTAPI_DVBT2_QAM16	16-QAM
DTAPI_DVBT2_QAM64	64-QAM
DTAPI_DVBT2_QAM256	256-QAM

m_Rotation

If true, constellation rotation is used.

m_FecType

FEC type used by the PLP.

Value	Meaning
DTAPI_DVBT2_LDPC_16K	16K LDPC
DTAPI_DVBT2_LDPC_64K	64K LDPC

m_FrameInterval

The T2-frame interval within the super-frame for this PLP. The valid range is 1 ... 255.

m_FirstFrameIdx

The index of the first frame of the super-frame in which this PLP occurs. The valid range is 0 ... *m_FrameInterval*-1.

m_TimeIlLength

Time interleaving length. The valid range is 0 ... 255.
If *m_TimeIlType* is set to '0' (**DTAPI_DVBT2_IL_ONETOONE**), this parameter specifies the number of TI-blocks per interleaving frame.
If *m_TimeIlType* is set to '1' (**DTAPI_DVBT2_IL_MULTI**), this parameter specifies the number of T2 frames to which each interleaving frame is mapped.

m_TimeIlType

Type of interleaving used by the PLP.

Value	Meaning
DTAPI_DVBT2_IL_ONETOONE	One interleaving frame corresponds to one T2 frame
DTAPI_DVBT2_IL_MULTI	One interleaving frame is carried in multiple T2 frames

m_InBandAFlag

If true, the in-band A flag is set and in-band A signalling information is inserted in this PLP.

m_InBandBFlag

If true, the in-band B flag is set and in-band B signalling information is inserted in this PLP.

m_NumBlocks
The maximum number of FEC blocks contained in one interleaving frame for this PLP. The valid range is 0 ... 2047.

m_NumOtherPlpInBand
Specifies the number of other PLPs in the in-band signalling. The valid range is 0 ... **DTAPI_DVBT2_NUM_PLP_MAX-1**.

m_OtherPlpInBand
Array specifying the IDs of the other PLPs in the in-band signalling.

m_FfFlag
If true, the PLP occurs on the same RF channel in each T2-frame; otherwise, inter-frame TFS is applied. This parameter is only meaningful for a type 1 PLP in a TFS system.

m_FirstRfIdx
The RF channel where this PLP occurs on in the first frame of a super-frame in a TFS system. If, *m_FfFlag* is set to 'true' the field indicates the RF channel the PLP occurs on in every T2-frame. This parameter is only meaningful for a type 1 PLP in TFS system.

Struct DtDvbT2RbmEvent

Structure containing the Receiver Buffer Model (RBM) event data. If RBM-validation is enabled then on an RBM-event the **DtDvbT2RbmEvent** parameters are sampled and passed to the RBM-event handler.

```
struct DtDvbT2RbmEvent
{
  int  m_DataPlpId;                    // Data PLP ID
  int  m_DataPlpIndex;                 // Data PLP index
  double  m_Time;                      // Time in T units
  int  m_IsCommonPlp;                  // Common PLP
  DtDvbT2RbmEvent  m_EventType;        // RBM event type

  union {
    struct {
       // DTAPI_DVBT2_RBM_EVENT_PLOT parameters
       int  m_TdiWriteIndex;           // TDI write index
       int  m_TdiReadIndex;            // TDI read index
       int  m_TdiReadAvailable;        // Available cells in TDI buffer
       int  m_DjbSize;                 // Dejitter buffer size in bits
    } Plot;

    struct {
       // DTAPI_DVBT2_RBM_EVENT_BUFS_TOO_SMALL parameters
       int  m_Bufs;                    // BUFS value
    } BufsTooSmall;

    struct {
       // DTAPI_DVBT2_RBM_EVENT_TTO_IN_THE_PAST parameters
       int  m_Tto;                     // TTO value
    } TtoInThePast;

    struct {
       // DTAPI_DVBT2_RBM_EVENT_DJB_OVERFLOW parameters
       int  m_DjbSize;                 // Dejitter buffer size in bits
       int  m_DjbMaxSize;
    } DjbOverflow;

    struct {
       // DTAPI_DVBT2_RBM_EVENT_CRC8_ERROR_HEADER parameters
       int  m_Val;                     // CRC8 value
    } Crc8ErrorHeader;

    struct {
       // DTAPI_DVBT2_RBM_EVENT_DFL_TOO_LARGE parameters
       int  m_SyncD;                   // SYNCD
       int  m_Dfl;                     // DFL
    } SyncDTooLarge;

    struct {
       // DTAPI DVBT2 RBM EVENT INVALID SYNCD parameters
       int  m_SyncD;                   // SYNCD
       int  m_Left;                    // #bytes left
    } InvalidSyncD;

    struct {
       // DTAPI_DVBT2_RBM_EVENT_TDI_OVERFLOW parameters
       int  m_TdiWriteIndex;           // TDI write index
       int  m_TdiReadIndex;            // TDI read index
    } TdiOverflow;
```

```
    struct {
        // DTAPI_DVBT2_RBM_EVENT_INVALID_PLP_START parameters
        int  m_PlpId1;                  // IDs of overlapping PLPs
        int  m_PlpId2;
    } InvalidPlpStart;

    struct {
        // DTAPI_DVBT2_RBM_EVENT_ISCR_ERROR parameters
        int  m_Delta;                   // Delta time in T units
    } IscrError;

    struct {
        // DTAPI_DVBT2_RBM_EVENT_BUFS_NOT_CONSTANT parameters
        int  m_CufBufs;                 // Current and new BUFS values
        int  m_NewBufs;
    } BufsNotConstant;

    struct {
        // DTAPI_DVBT2_RBM_EVENT_PLP_NUM_BLOCKS_TOO_SMALL parameters
        int  m_PlpNumBlocks;            // Number of blocks
    } PlpNumBlocksTooSmall;

  } u;
};
```

Members

m_DataPlpId
Data PLP ID identifying the stream.

m_DataPlpIndex
Data PLP index.

m_Time
Time in T units.

m_IsCommonPlp
Indicates whether the event refers to a common PLP.
Possible values:
-1 : Event doesn't refer to a specific PLP
0 : Data PLP
1 : Common PLP

m_EventType
Type of Receiver Buffer Model event

Value	Meaning
DTAPI_DVBT2_RBM_EVENT_PLOT	Plot event
DTAPI_DVBT2_RBM_EVENT_DJB_UNDERFLOW	De-jitter buffer underflow
DTAPI_DVBT2_RBM_EVENT_BUFS_TOO_SMALL	BUFS gives too small dejitter buffer
DTAPI_DVBT2_RBM_EVENT_TTO_IN_THE_PAST	TTO gives time in the past
DTAPI_DVBT2_RBM_EVENT_DJB_OVERFLOW	De-jitter buffer overflow
DTAPI_DVBT2_RBM_EVENT_CRC8_ERROR_HEADER	CRC8 error in BBFrame

DTAPI_DVBT2_RBM_EVENT_DFL_TOO_LARGE	DFL too large in BBFrame
DTAPI_DVBT2_RBM_EVENT_SYNCD_TOO_LARGE	SYNCD too large in BBFrame
DTAPI_DVBT2_RBM_EVENT_INVALID_UPL	Invalid UPL in BBFrame
DTAPI_DVBT2_RBM_EVENT_INVALID_SYNCD	Invalid SYNCD in BBFrame
DTAPI_DVBT2_RBM_EVENT_TDI_OVERFLOW	TDI overflow
DTAPI_DVBT2_RBM_EVENT_TOO_MANY_TI_BLOCKS	Too many TI blocks queued
DTAPI_DVBT2_RBM_EVENT_INVALID_PLP_START	PLP-start values gives overlap
DTAPI_DVBT2_RBM_EVENT_FDI_OVERFLOW	Frequency/L1 de-interleaver overflow
DTAPI_DVBT2_RBM_EVENT_NO_TS_RATE	Not enough ISCR data to estimate TS rate
DTAPI_DVBT2_RBM_EVENT_ISCR_ERROR	ISCR error
DTAPI_DVBT2_RBM_EVENT_BUFS_NOT_CONSTANT	BUFS not constant
DTAPI_DVBT2_RBM_EVENT_ISSYI_NOT_CONSTANT	ISSYI not constant
DTAPI_DVBT2_RBM_EVENT_HEM_NOT_CONSTANT	HEM not constant
DTAPI_DVBT2_RBM_EVENT_PLP_NUM_BLOCKS_TOO_SMALL	PLP numblocks for this interleaving frame is too small

u.Plot
Structure used for event type **DTAPI_DVBT2_RBM_EVENT_PLOT**.

u.Plot.m_TdiWriteIndex
Write index in time de-interleaver buffer.

u.Plot.m_TdiReadIndex
Read index in time de-interleaver buffer.

u.Plot.m_TdiReadAvailable
Number of available cells in the time de-interleaver read buffer.

u.Plot.m_DjbSize
De-jitter buffer size in number of bits.

u.BufsTooSmall
Structure used for event type **DTAPI_DVBT2_RBM_EVENT_BUFS_TOO_SMALL**.

u.BufsTooSmall.m_Bufs
BUFS value.

u.TtoInThePast
Structure used for event type **DTAPI_DVBT2_RBM_EVENT_TTO_IN_THE_PAST**.

u.TtoInThePast.m_Tto
TTO value from the ISSY-field

u.DjbOverflow
Structure used for event type **DTAPI_DVBT2_RBM_EVENT_DJB_OVERFLOW**.

u.DjbOverflow.m_DjbSize
De-jitter buffer size in bits.

u.DjbOverflow.m_DjbMaxSize
Maximum de-jitter buffer size in bits.

u.Crc8ErrorHeader
Structure used for event type **DTAPI_DVBT2_RBM_EVENT_CRC8_ERROR_HEADER.**

u.Crc8ErrorHeader.m_Val
CRC-8 value from the baseband header.

u.SyncDTooLarge
Structure used for event type **DTAPI_DVBT2_RBM_EVENT_DFL_TOO_LARGE.**

u.SyncDTooLarge.m_SyncD
SYNCD value from the baseband header.

u.SyncDTooLarge.m_Dfl
DFL value from the baseband header.

u.InvalidSyncD
Structure used for event type **DTAPI_DVBT2_RBM_EVENT_INVALID_SYNCD.**

u.InvalidSyncD.m_SyncD
SYNCD value from the baseband header.

u.InvalidSyncD.m_Left
Number of bits remaining from the last baseband frame.

u.TdiOverflow
Structure used for event type **DTAPI_DVBT2_RBM_EVENT_TDI_OVERFLOW.**

u.TdiOverflow.m_TdiWriteIndex
Write index in time de-interleaver buffer.

u.TdiOverflow.m_TdiReadIndex
Read index in time de-interleaver buffer.

u.InvalidPlpStart
Structure used for event type **DTAPI_DVBT2_RBM_EVENT_TDI_OVERFLOW.**

u.InvalidPlpStart.m_Plp1, u.InvalidPlpStart.m_Plp2
IDs of the overlapping PLPs.

u.IscrError
Structure used for event type **DTAPI_DVBT2_RBM_EVENT_ISCR_ERROR.**

u.IscrError.m_Delta
Delta time in T-units.

u.BufsNotConstant
Structure used for event type **DTAPI_DVBT2_RBM_EVENT_TDI_OVERFLOW.**

u.BufsNotConstant.m_CurBufs, u.BufsNotConstant.m_NewBufs
Current and new BUFS values

u.PlpNumBlocksTooSmall
Structure used for event type **DTAPI_DVBT2_RBM_EVENT_PLP_NUM_BLOCKS_TOO_SMALL.**

u.PlpNumBlocksTooSmall.m_PlpNumBlocks
NUM_BLOCKS value for this PLP.

Struct DtDvbT2RbmValidation

Structure for enabling Receiver Buffer Model (RBM) validation, and specifying its parameters. This structure is used in class **DtDvbT2Pars**.

```
Struct DtDvbT2RbmValidation
{
   bool   m_Enabled;            // Enable RBM validation
   bool   m_PlotEnabled;        // Enable RBM plotting events
   int    m_PlotPeriod;         // Plot period in T-units
   void*  m_pCallbackOpaque;    // Opaque pointer for the callback function
   void   (*m_pCallbackFunc)( void*, const DtDvbT2RbmEvent*);
                                // Pointer to the callback function
};
```

Members

m_Enabled

If true, Receiver Buffer Model (RBM) validation is enabled. When a RBM-violation occurs, the callback function *(*m_pCallbackFunc)* is called and an RBM-event is passed.
Note that RBM-validation consumes a substantial amount of CPU cycles and therefore cannot always be performed in real time.

m_PlotEnabled

If true, Receiver Buffer Model (RBM) plotting is enabled. Periodically, the callback function will be called passing a **DTAPI_DVBT2_RBM_EVENT_PLOT** event.

m_PlotPeriod

Plot period time in T-units.

m_pCallbackOpaque

Opaque pointer that is passed to the callback function.

m_pCallbackFunc

Pointer to the callback function that handles the RBM-events.

Struct DtDvbT2TxSigPars

Structure for enabling and specifying the DVB-T2 transmitter signature. This structure is used in class **DtDvbT2Pars**.

```
Struct DtDvbT2TxSigPars
{
   bool  m_TxSigAuxEnabled;    // Enable TX signature through AUX streams
   int   m_TxSigAuxId;         // Transmitter ID
   int   m_TxSigAuxP;          // P-value
   int   m_TxSigAuxQ;          // Q-value
   int   m_TxSigAuxR;          // R-value

   bool  m_TxSigFefEnabled;    // Enable TX signature through FEF
   int   m_TxSigFefId1;        // Transmitter ID for 1st period
   int   m_TxSigFefId2;        // Transmitter ID for 2nd period
};
```

Members

m_TxSigAuxEnabled
If true, transmitter signature transmission through AUX streams is enabled.

m_TxSigAuxId
Transmitter ID. The valid range is 0 … 3071.

m_TxSigAuxP
The total number of possible transmitter IDs (M) is derived from *m_TxSigAuxP* (P). M = 3 * (P+1). The valid range for *m_TxSigAuxP* is 0 … 1023.

m_TxSigAuxQ
The number of cells used per transmitter (N) is derived from *m_TxSigAuxQ* (Q). N = 2 ^ Q. The valid range for *m_TxSigAuxQ* is 0 … 15.

m_TxSigAuxR
The number of T2-frames used per transmitter signature (L) is derived from *m_TxSigAuxR* (R). L = R+1. The valid range for *m_TxSigAuxR* is 0 … 255.

m_TxSigFefEnabled
If true, transmitter signature transmission through FEF is enabled. To use this, FEF generation must be enabled and the FEF length must be greater than or equal to **DTAPI_TXSIG_FEF_LEN_MIN**.

m_TxSigFefId1
Transmitter ID for the first signature period. The valid range is 0 … 7.

m_TxSigFefId2
Transmitter ID for the second signature period. The valid range is 0 … 7.

Class DtDvbC2Pars

Class specifying parameters for DVB-C2 modulation.

```
class DtDvbC2Pars
{
   int   m_Bandwidth;          // Bandwidth (channel raster)
   int   m_NetworkId;          // Network ID
   int   m_C2SystemId;         // C2-System ID
   int   m_StartFrequency;     // Start frequeny
   int   m_C2Bandwidth;        // Bandwidth of the generated signal
   int   m_GuardInterval;      // Guard interval
   bool  m_ReservedTone;       // Reserved tones present (yes/no)
   int   m_L1TiMode;           // L1 time interleaving mode

   // Data-slice parameters
   int   m_NumDSlices;         // Number of data slices
   DtDvbC2DSlicePars  m_DSlices[DTAPI_DVBC2_NUM_DSLICE_MAX];

   // Notches
   int   m_NumNotches;         // Number of notches
   DtDvbC2NotchPars  m_Notches[DTAPI_DVBC2_NUM_NOTCH_MAX];

   // Parameters specifying the source for each PLP
   int   m_NumPlpInputs;       // Number of PLP input streams
   DtPlpInpPars  m_PlpInputs[DTAPI_DVBC2_NUM_PLP_MAX];

   // Miscellaneous: PAPR, Virtual output, Test-point output
   DtDvbC2PaprPars  m_PaprPars;
   DtVirtualOutPars  m_VirtOutput;
   DtTestPointOutPars  m_TpOutput;
};
```

Public members

m_Bandwidth

Channel raster of the network.

Value	Meaning
DTAPI_DVBC2_6MHZ	6 MHz
DTAPI_DVBC2_8MHZ	8 MHz

m_NetworkId

Network ID. Unique identification of the DVB-C2 network. The valid range is 0 … 0xFFFF.

m_C2SystemId

C2-System ID. Unique identification of a C2-System. The valid range is 0 … 0xFFFF.

m_StartFrequency

Start frequency of the C2-System by means of the distance from 0Hz in multiples of the carrier spacing. The valid range is 0 … 0xFFFFFF and multiples of D_X. (D_X=24 for guard interval 1/128 and D_X=12 for guard interval 1/64).

m_C2Bandwidth
Bandwidth of the generated signal in multiples of pilot carrier spacing. The valid range is 0 ... 65535.

m_GuardInterval
The guard interval between OFDM symbols.

Value	Meaning
DTAPI_DVBC2_GI_1_128	1/128
DTAPI_DVBC2_GI_1_64	1/64

m_ReservedTone
If true, indicates one or more reserved tones (carriers) are used. When carriers are reserved (e.g PAPR TR is enabled) it shall be set to true.

m_L1TiMode
L1 time interleaving mode.

Value	Meaning
DTAPI_DVBC2_L1TIMODE_NONE	No time interleaving
DTAPI_DVBC2_L1TIMODE_BEST	Best fit
DTAPI_DVBC2_L1TIMODE_4	4 OFDM symbols
DTAPI_DVBC2_L1TIMODE_8	8 OFDM symbols

m_NumDSlices
Specifies the number of data slices in the C2-System. The valid range is 1 ... **DTAPI_DVBC2_NUM_DSLICE_MAX**.

m_DSlices
Array specifying the DVB-C2 parameters for the data slices.

m_NumNotches
Specifies the number of notch bands in the C2-System. The valid range is 0 ... **DTAPI_DVBC2_NUM_NOTCH_MAX**.

m_Notches
Array specifying the notch bands in the C2-System.

m_NumPlpInputs
Specifies the number of PLP inputs in the C2-System. The valid range is 1 ... **DTAPI_DVBC2_NUM_PLP_MAX**.

m_PlpInputs
Array specifying the PLP input streams. The index in the array is related to the index of a PLP in the C2 System (i.e. the first **DtPlpInpPars** in the array is related to the first PLP in the C2 System, which is the first PLP in the first data slice).
Note that PLPs in empty data slices are not taken into account and in case of bundled PLPs only the first PLP occurrence is taken into account.

m_PaprPars
Specifies the PAPR reduction parameters.

m_VirtOutput
In case of a virtual output *m_VirtOutput* specifies the virtual output data parameters.

m_TpOutput
In case of a virtual output *m_VirtOutput* specifies the virtual output data parameters.

Remarks

This class is used both for the initialization of the multi-PLP modulator and the traditional single-PLP DVB-C2 modulator. Which modulator is addressed, is indicated through parameter *MplpMod* of the **DtOutpChannel::SetModControl()** method.

As its name implies, the single-PLP DVB-C2 modulator can only be used for single-PLP parameter sets. The **DtOutpChannel::Write** method is used to write the data to the output channel.

The multi-PLP modulator can be used for both single-PLP and multi-PLP parameter sets. The **DtOutpChannel::WriteMplp** method is used to write data to the output channel.

DtDvbC2Pars::CheckValidity

Check DVB-C2 parameters for validity.

```
DTAPI_RESULT DtDvbC2Pars::CheckValidity (
      void
);
```

Parameters

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Parameters are valid
DTAPI_E_INVALID_PARS	Invalid parameter value (generic error)
DTAPI_E_INVALID_START_FREQ	Invalid start frequency
DTAPI_E_DSLICE_TUNE_POS	Invalid data slice tune position
DTAPI_E_DSLICE_OFFSETS	Invalid data slice offset
DTAPI_E_DSLICE_OVERLAP	Data slices cannot overlap
DTAPI_E_NOTCH_OFFSETS	Invalid notch
DTAPI_E_PLP_ID	Duplicate PLP IDs
DTAPI_E_PLP_BUNDLED	Inconsistent PLP bundled parameters
DTAPI_E_BROADBAND_NOTCH	Broadband notch cannot be inside a data slice
DTAPI_E_L1_PART2_TOO_LONG	L1 part 2 data is too long
DTAPI_E_DSLICE_T1_NDP	Null-packet deletion not allowed for type1 data slices
DTAPI_E_DSLICE_T1_TSRATE	TS-rate/ISSY combination not possible for type1 data slice
DTAPI_E_NO_TSRATE	PLP TS-rate is not specified

DtDvbC2Pars::GetParamInfo

Get the DVB-C2 "derived" parameters.

```
DTAPI_RESULT DtDvbC2Pars::GetParamInfo (
 [out] DtDvbC2ParamInfo&  ParamInfo   // DVB-C2 derived information
);
```

Parameters

ParamInfo

Output parameter that receives the DVB-C2 "derived" parameters.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Parameters are valid
Other result values	Error in modulation parameters, please refer to **DtDvbC2Pars::CheckValidity**

DtDvbT2Pars

Class describing parameters for DVB-T2 modulation.

```
class DtDvbT2Pars
{
  int   m_T2Version;          // DVB-T2 specification version
  int   m_Bandwidth;          // DVB-T2 channel bandwidth
  int   m_FftMode;            // FFT mode (or size)
  int   m_Miso;               // MISO mode
  int   m_GuardInterval;      // Guard interval
  int   m_Papr;               // PAPR reduction mode
  int   m_BwtExt;             // Bandwidth extension
  int   m_PilotPatern;        // Pilot pattern
  int   m_L1Modulation;       // L1 modulation type
  int   m_CellId;             // Cell ID
  int   m_NetworkId;          // Network ID
  int   m_T2SystemId;         // T2 system ID
  bool  m_L1Repetition;       // L1 repetition (yes/no)
  int   m_NumT2Frames;        // Number of T2 frames in a super frame
  int   m_NumDataSyms;        // Number of data OFDM symbols per T2-frame
  int   m_NumSubslices;       // Number of subslices per T2-frame

  bool  m_FefEnable;          // Insert FEF (yes/no)
  int   m_FefType;            // FEF type
  int   m_FefS1;              // FEF S1 field value
  int   m_FefS2;              // FEF S2 field value
  int   m_FefLength;          // FEF length
  int   m_FefInterval;        // FEF interval
  int   m_FefSignal;          // Type of signal during FEF period
  int   m_NumRfChans;         // Number of RF channels
  int   m_RfChanFreqs[DTAPI_DVBT2_NUM_RF_MAX];
                              // Array of RF channel frequencies
  int   m_StartRfIdx;         // First used RF channel
  int   m_NumPlps;            // Number of PLPs
  DtDvbT2PlpPars m_Plps[DTAPI_DVBT2_NUM_PLP_MAX];
                              // Array of PLP parameters
  DtPlpInpPars m_PlpInputs[DTAPI_DVBT2_NUM_PLP_MAX];
                              // Array of PLP input streams
  DtVirtualOutPars m_VirtOutput;
                              // Virtual-output parameters
  DtDvbT2AuxPars m_Aux;       // AUX streams
  DtDvbT2PaprPars m_PaprPars;// PAPR reduction parameters
  DtDvbT2TxSigPars m_TxSignature;
                              // Transmitter signature parameters
  DtDvbT2MiPars m_T2Mi;       // T2-MI output parameters
  DtDvbT2RbmValidation m_RbmValidation;
                              // Receiver Buffer Model validation
  DtTestPointOutPars m_TpOutput;
                              // Test point data output parameters
};
```

Public members

m_T2Version

DVB-T2 specification version.

Value	Meaning
DTAPI_DVBT2_VERSION_1_1_1	Version 1.1.1
DTAPI_DVBT2_VERSION_1_2_1	Version 1.2.1

m_Bandwidth

The bandwidth of the channel.

Value	Meaning
DTAPI_DVBT2_1_7MHZ	1.7 MHz
DTAPI_DVBT2_5MHZ	5 MHz
DTAPI_DVBT2_6MHZ	6 MHz
DTAPI_DVBT2_7MHZ	7 MHz
DTAPI_DVBT2_8MHZ	8 MHz
DTAPI_DVBT2_10MHZ	10 MHz

m_FftMode

The FFT size used for computing OFDM symbols.

Value	Meaning
DTAPI_DVBT2_FFT_1K	1K FFT
DTAPI_DVBT2_FFT_2K	2K FFT
DTAPI_DVBT2_FFT_4K	4K FFT
DTAPI_DVBT2_FFT_8K	8K FFT
DTAPI_DVBT2_FFT_16K	16K FFT
DTAPI_DVBT2_FFT_32K	32K FFT

m_Miso

MISO mode. This mode can be used to simulate antenna 1 (TX1), antenna 2 (TX2) or the average of antenna 1 and antenna 2 (TX1+TX2) to simulate reception halfway between the antennas.

Value	Meaning
DTAPI_DVBT2_MISO_OFF	No MISO
DTAPI_DVBT2_MISO_TX1	TX1 only
DTAPI_DVBT2_MISO_TX2	TX2 only
DTAPI_DVBT2_MISO_SUM	TX1+ TX2 through one output channel
DTAPI_DVBT2_MISO_BOTH	Both TX1and TX2 through two output channels

m_GuardInterval

The guard interval between OFDM symbols.

Value	Meaning
DTAPI_DVBT2_GI_1_128	1/128
DTAPI_DVBT2_GI_1_32	1/32
DTAPI_DVBT2_GI_1_16	1/16
DTAPI_DVBT2_GI_19_256	19/256
DTAPI_DVBT2_GI_1_8	1/8
DTAPI_DVBT2_GI_19_128	19/128
DTAPI_DVBT2_GI_1_4	1/4

m_Papr

The peak to average power reduction method. This is used to fill PAPR field in the L1-post signalling block.

Value	Meaning
DTAPI_DVBT2_PAPR_NONE	None
DTAPI_DVBT2_PAPR_ACE	ACE - Active Constellation Extension
DTAPI_DVBT2_PAPR_TR	TR - Power reduction with reserved carriers
DTAPI_DVBT2_PAPR_ACE_TR	ACE and TR

m_BwtExt

If true, the extended carrier mode is used.

m_PilotPattern

The Pilot Pattern used.

Value	Meaning
DTAPI_DVBT2_PP_1	PP1
DTAPI_DVBT2_PP_2	PP2
DTAPI_DVBT2_PP_3	PP3
DTAPI_DVBT2_PP_4	PP4
DTAPI_DVBT2_PP_5	PP5
DTAPI_DVBT2_PP_6	PP6
DTAPI_DVBT2_PP_7	PP7
DTAPI_DVBT2_PP_8	PP8

m_L1Modulation
The modulation type used for the L1-post signalling block.

Value	Meaning
DTAPI_DVBT2_BPSK	BPSK
DTAPI_DVBT2_QPSK	QPSK
DTAPI_DVBT2_QAM16	16-QAM
DTAPI_DVBT2_QAM64	64-QAM

m_CellId
Cell ID. Unique identification of a geographic cell in a DVB-T2 network. The valid range is 0 … 0xFFFF.

m_NetworkId
Network ID. Unique identification of the DVB-T2 network. The valid range is 0 … 0xFFFF.

m_T2SystemId
T2 system ID. Unique identification of the T2 system. The valid range is 0 … 0xFFFF.

m_L1Repetition
If true, L1 signalling is provided for the next frame.

m_NumT2Frames
The number of T2 frames in a super frame. The valid range is 1 … 255.

m_NumDataSyms
The number of data OFDM symbols per T2 frame, excluding P1 and P2.

m_NumSubslices
The number of subslices per T2-frame for type-2 PLPs.

m_FefEnable
If true, FEFs (Future Extension Frames) are inserted.

m_FefType
Specifies the FEF type. The valid range is 0 ... 15.

m_FefS1
The S1-field value in the P1 signalling data. Valid values: 2, 3, 4, 5, 6 and 7.

m_FefS2
The S2-field value in the P1 signalling data. Valid values: 1, 3, 5, 7, 9, 11, 13 and 15.

m_FefLength
The length of a FEF-part in number of T-units (= samples). The valid range is 0 … 0x3FFFFF.

m_FefInterval
The number of T2 frames between two FEF parts. The valid range is 1 … 255 and *m_NumT2Frames* shall be divisible by *m_FefInterval*.

m_FefSignal

The type of signal generated during the FEF period.

Value	Meaning
DTAPI_DVBT2_FEF_ZERO	Zero I/Q samples
DTAPI_DVBT2_FEF_1K_OFDM	1K OFDM symbols with 852 active carriers containing BPSK symbols
DTAPI_DVBT2_FEF_1K_OFDM_384	1K OFDM symbols with 384 active carriers containing BPSK symbols

m_NumRfChans

The number of frequencies in the T2 system. The valid range is 1 … **DTAPI_DVBT2_NUM_RF_MAX**.

m_RfChanFreqs

Array specifying the center frequencies of the RF channels. This is only used to fill the L1-post FREQUENCY fields. The valid range is 1 … 0xFFFFFFFF.

m_NumPlps

Specifies the number of physical layer pipes in the T2 system. The valid range is 1 … **DTAPI_DVBT2_NUM_PLP_MAX**. Must be set to '1' in case not using the Multi-PLP modulator.

m_Plps

Array specifying the DVB-T2 modulation parameters for the PLPs.

m_PlpInputs

Array specifying the PLP input streams. This is only used in case of using the Multi-PLP modulator. Default the FIFO index and PLP index maps 1:1 and "Big-TS splitting" is disabled.

m_VirtOutput

When the output channel has been attached to a virtual output, *m_VirtOutput* specifies the virtual output data parameters. This can only be used with the Multi-PLP modulator.
By default, the virtual output parameters are disabled.

m_Aux

Specifies the AUX stream parameters.
By default, the generation of AUX streams is disabled.

m_PaprPars

Specifies the PAPR reduction parameters.
By default, PAPR reduction is disabled.

m_TxSignature

Specifies the transmission of the DVB-T2 transmitter signature.
By default, the transmission of a transmitter signature is disabled.

m_T2Mi

Specifies the parameters for generation of T2-MI. This can only be used with the Multi-PLP modulator.
By default, the output of T2-MI is disabled.

m_RbmValidation
: Specifies the Receiver Buffer Model validation. This can only be used with the Multi-PLP modulator.
By default, RBM-validation is disabled.

m_TpOutput
: Specifies the generation of test point data.

Remarks

This class is used for both the initialization of the multi-PLP modulator and the traditional single-PLP DVB-T2 modulator. Which modulator is addressed is indicated through parameter *MplpMod* of the **DtOutpChannel::SetModControl()** method.

Note that the traditional single-PLP DVB-T2 modulator can only be used for single-PLP parameter sets. The **DtOutpChannel::Write** method is used to write the data to the output channel.

The multi-PLP modulator can be used for both single-PLP and multi-PLP parameter sets. The **DtOutpChannel::WriteMplp** method is used to write data to the output channel.

DtDvbT2Pars::CheckValidity

Check DVB-T2 parameters for validity.

```
DTAPI_RESULT DtDvbT2Pars::CheckValidity (
      void
);
```

Parameters

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Parameters are valid
DTAPI_E_BIAS_BAL_CELLS	Invalid number of bias balancing cells
DTAPI_E_BUFS	Invalid BUFS values
DTAPI_E_COMMON_PLP_COUNT	More than one common PLP per group ID
DTAPI_E_FEF	Error in FEF parameters
DTAPI_E_FIXED_CELL_PARS	Invalid fixed cell parameters
DTAPI_E_FRAME_INTERVAL	Frame interval must divide number of T2 frames
DTAPI_E_INVALID_BWT_EXT	Invalid bandwidth extension
DTAPI_E_INVALID_FFTMODE	Invalid FFT mode
DTAPI_E_INVALID_GUARD	Invalid guard interval
DTAPI_E_INVALID_NUMDTSYM	Invalid number of data symbols
DTAPI_E_INVALID_NUMT2FRM	Invalid number of T2 frames
DTAPI_E_INVALID_PARS	Invalid parameter value (generic error)
DTAPI_E_INVALID_TIME_IL	Invalid time interleaver length
DTAPI_E_NO_TSRATE	PLP TS-rate is not specified
DTAPI_E_NUM_PLP	Too many PLPs (i.e. L1 data too large)
DTAPI_E_OTHER_PLP_IN_BAND	Invalid PLP ID in m_OtherPlpInBand array
DTAPI_E_PILOT_PATTERN	Pilot pattern not allowed in combination with other parameters
DTAPI_E_PLP_ID	Duplicate PLP IDs
DTAPI_E_PLP_NUM_BLOCKS	Invalid number of PLP blocks (not enough bandwidth)
DTAPI_E_SUBSLICES	Number of subslices and/or TIME_IL_LENGTH does not give an integer number of cells per subslice
DTAPI_E_TI_MEM_OVF	Too many cells in time interleaver

DtDvbT2Pars::GetParamInfo

Get the DVB-T2 “derived” parameters.

```
DTAPI_RESULT DtDvbT2Pars::GetParamInfo (
 [out] DtDvbT2ParamInfo&  ParamInfo   // DVB-T2 derived information
);
```

Parameters

ParamInfo

Output parameter that receives the DVB-T2 “derived” parameters.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Parameters are valid
Other result values	Error in modulation parameters, please refer to **DtDvbT2Pars::CheckValidity**

DtDvbT2Pars::OptimisePlpNumBlocks

Compute the optimum value of DVB-T2 parameters to maximise the DVB-T2 channel's bitrate and compute the achieved efficiency.

```
// Overload #1 - Get optimum value for PLP_NUM_BLOCKS
DTAPI_RESULT DtDvbT2Pars::GetParamInfo (
 [out] DtDvbT2ParamInfo&  ParamInfo    // DVB-T2 efficiency information
 [out] Int&   OptPlpNumBlocks          // Optimum number of blocks
);
// Overload #2 - Get optimum value for PLP_NUM_BLOCKS and NUM_DATA_SYMBOLS
DTAPI_RESULT DtDvbT2Pars::GetParamInfo (
 [out] DtDvbT2ParamInfo&  ParamInfo    // DVB-T2 efficiency information
 [out] Int&   OptPlpNumBlocks          // Optimum number of blocks
 [out] Int&   OptNumDataSyms           // Optimum number data symbols
);
```

Parameters

ParamInfo

Output parameter that receives the DVB-T2 "derived" parameters based on the optimum parameter values.

OptPlpNumBlocks

Output parameter that is set to the optimum value for the number of FEC blocks per IL frame for PLP0 to maximise the DVB-T2 channel's bitrate.

OptNumDataSyms

Output parameter that is set to the optimum value for the number of data OFDM symbols per T2 frame to maximise the DVB-T2 channel's bitrate.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Parameters are valid
Other result values	Error in modulation parameters, please refer to **DtDvbT2Pars::CheckValidity**

Remarks

These methods can only be used in case of a single PLP (member variable *m_NumPlps* equals 1).

Callback Functions

DtTpWriteDataFunc

User-supplied callback function used for the processing of test-point data. The data can be written to a file, or processed otherwise.

```
void  DtTpWriteDataFunc(
  [in] void*  pOpaque,        // Opaque pointer
  [in] int  TpIndex,          // Test point
  [in] int  StreamIndex       // Stream index
  [in] const void*  pBuffer, // Test-point data buffer
  [in] int  Length,           // Number of data items
  [in] int  Format,           // Test point data format
  [in] float  Mult,           // Multiplication factor
  [in] int  IsNewFrame        // New frame (yes/no)
);
```

Parameters

pOpaque

The opaque pointer that was specified in **DtTestPointOutPars**.

TpIndex

Specifies the test point.

For DVB-C2 the following test points are defined:

Value	Meaning
DTAPI_DVBC2_TPnn	DVB-C2 test point nn

Where nn is: 07, 08, 10, 13, 15, 18, 20, 22, 26, 27, 31, 32, 33, 37, 40, 41 and 42.

For DVB-T2 the following test points are defined:

Value	Meaning
DTAPI_DVBT2_TPnn	DVB-T2 test point nn

Where nn is: 03, 04, 06, 08, 09, 11, 12, 15, 16, 19, 20, 21, 22, 23, 24, 25, 26, 28, 29, 30, 32, 33, 34, 50, 51 and 53.

StreamIndex

Identifies the stream. For DVB-C2 bits 0..7 specify the PLP-ID and bits 8..15 specify the data slice I. For DVB-T2 bits 0..7 specify the PLP-index and bit 8 is set when the PLP-type is a common PLP.

pBuffer

Pointer to a buffer containing the test point data.

Length

Number of test points data items available in buffer.

Format

The data format of the test-point data items.

Value	Meaning
DTAPI_TP_FORMAT_HEX	Byte data
DTAPI_TP_FORMAT_BIT	Bit data. Eight bits are packaged per byte, most significant bit first
DTAPI_TP_FORMAT_CFLOAT32	Complex 32-bit floating-point data of type **DtComplexFloat**
DTAPI_TP_FORMAT_INT64	64-bit integer data

Mult

Multiplication factor for the complex floating point data.

IsNewFrame

If true, the test point data relates to a new frame.

Global Functions

::DtapiModPars2TsRate

Compute Transport-Stream rate from modulation parameters. There are two new overloads one for DVB-C2 and one for DVB-T2 modulation type.

```
// Overload to be used for DVB-C2
DTAPI_RESULT ::DtapiModPars2TsRate(
 [out] int&  TsRate          // Computed Transport-Stream rate
  [in] DtDvbC2Pars  C2Pars,  // DVB-C2 modulation parameters
  [in] int  PlpIdx           // PLP index
);
// Overload to be used for DVB-T2
DTAPI_RESULT ::DtapiModPars2TsRate(
 [out] int&  TsRate          // Computed Transport-Stream rate
  [in] DtDvbT2Pars  T2Pars,  // DVB-T2 modulation parameters
  [in] int  PlpIdx           // PLP index
);
```

Parameters

TsRate
The Transport-Stream rate in bps computed from the modulation parameters.

C2Pars
DVB-C2 modulation parameters; see description of **class DtDvbC2Pars**.

T2Pars
DVB-T2 modulation parameters; see description of **class DtDvbT2Pars**.

PlpIdx
The index of the PLP for which the Transport-Stream rate is computed.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The TS rate has been computed from the modulation parameters successfully
Other result values	Error in modulation parameters, please refer to **DtDvbC2Pars::CheckValidity** and **DtDvbT2Pars::CheckValidity**

DtOutpChannel::AttachVirtual

Attach the output-channel object to a virtual output using the licenses of a particular device. A virtual output lets the user pass the output data to the specified callback function, instead of DTAPI writing the data to a physical output.

```
DTAPI_RESULT DtOutpChannel::AttachVirtual (
  [in] DtDevice*  pDtDevice, // Object representing a DekTec device
  [in] bool (*pFunc)(void*, DtVirtualOutData*),
                             // Pointer to the callback function
  [in] void* pOpaque         // Opaque pointer for the callback function
);
```

Parameters

pDtDvc

Pointer to the object that represents a DekTec device. The **DtDevice** object must be attached to the device hardware. The device is used only for reading licenses.

pFunc

Pointer to the callback function that will handle the generated output data. When the virtual-output calls this function the opaque pointer and a pointer to a **DtVirtualOutData** struct describing the output data are passed. To prevent hanging of the application, the callback function is not allowed to block. In case the callback function has to wait for a certain condition, it can return the Boolean value false. After a few milliseconds the virtual-output will call this function again with the same parameters and will repeat this until the callback function returns the Boolean value true.

pOpaque

Opaque pointer that is passed to the callback function.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Channel object has been attached successfully
DTAPI_E_ATTACHED	The channel object is already attached a hardware function
DTAPI_E_DEVICE	The **DtDevice** pointer is not valid or the **DtDevice** object is not attached to the device hardware
DTAPI_E_INVALID_ARG	The value of one of the parameters is invalid

Remarks

The intended usage for this method is to allow the user to output the multi-PLP modulator result to file or to a specific device. The licenses are taken from the DekTec device.

DtOutpChannel::GetMplpFifoFree

Get the number of free bytes in the specified multi-PLP modulator FIFO.

```
DTAPI_RESULT DtOutpChannel::GetMplpFifoFree (
  [in] int  FifoIndex        // FIFO index
 [out] int&  FifoFree        // Number of free bytes in the FIFO
);
```

Parameters

FifoIndex

Specifies the FIFO index.

FifoFree

Free space in the specified multi-PLP modulator FIFO, in number of bytes.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	FIFO free has been retrieved successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached

Remarks

If a Data transfer is in progress and/or the transmit-control state is **DTAPI_TXCTRL_HOLD** or **DTAPI_TXCTRL_SEND**, then every call to **GetMplpFifoFree** may return a different value.

DtOutpChannel::GetMplpFifoSize

Get the current size of the multi-PLP modulator FIFO.

```
DTAPI_RESULT DtOutpChannel::GetMplpFifoSize (
  [in] int  FifoIndex        // FIFO index
 [out] int&  FifoSize        // Size of Transmit FIFO in bytes
);
```

Parameters

FifoIndex

Specifies the FIFO index.

FifoSize

Size of the multi-PLP modulator FIFO in number of bytes.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	FIFO size has been retrieved successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

The size of the multi-PLP modulator FIFOs is fixed, it cannot be changed.

DtOutpChannel::GetMplpModStatus

Get the status of the multi-PLP modulator. There are overloads for DVB-C2 and for DVB-T2.

```
DTAPI_RESULT DtOutpChannel::GetMplpModStatus (
 [out] DtDvbC2ModStatus*  pDvbC2ModStat        // Status of DVB-C2
modulator
);
DTAPI_RESULT DtOutpChannel::GetMplpModStatus (
 [out] DtDvbT2ModStatus*  pDvbT2ModStat        // Status of DVB-T2
modulator
);
```

Parameters

pDvbC2ModStat

DVB-C2 modulator status; see description of **struct DtDvbC2ModStatus**.

pDvbT2ModStat

DVB-T2 modulator status; see description of **struct DtDvbT2ModStatus**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The status of the MPLP modulator has been retrieved successfully
DTAPI_E_IDLE	Not allowed when in IDLE state
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The currently active modulator does not support the request

Remarks

DtOutpChannel::SetMplpChannelModelling

Set channel-modelling parameters. This function may only be called when using the multi-PLP modulator while the transmit-control state is **DTAPI_TXCTRL_IDLE**.

```
DTAPI_RESULT DtOutpChannel::SetMplpChannelModelling (
  [in] bool   CmEnable,          // Enable/disable channel modelling
  [in] DtCmPars&  CmPars         // Channel modelling parameters
  [in] int   ChannelIdx=0        // Output channel index
);
```

Parameters

CmEnable

Enable channel modelling. This parameter provides an easy way to turn off channel modelling entirely for the specified output channel.

CmPars

Channel-modelling parameters. See description of struct **DtCmPars** in *"C++ API for DekTec Devices"*.

ChanneIdx

Index of the output channel (e.g. to specify the channel modelling parameters for the individual transmitters in case of MISO).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Channel-modelling parameters have been applied successfully
DTAPI_E_CM_NUMPATHS	The number of paths specified in **CmPars** exceeds the maximum number of paths
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The channel has no license for channel-modelling, or channel modelling is not supported for this type of channel

Remarks

DtOutpChannel::SetModControl

Set modulation-control parameters for modulator channels. There are two new overloads defined: one for DVB-C2 and one for DVB-T2.

```
// Overload to be used for DVB-C2
DTAPI_RESULT DtOutpChannel::SetModControl (
  [in] DtDvbC2Pars&  DvbC2Pars     // DVB-C2 modulation parameters
  [in] bool  MplpMod = false       // Use the multi-PLP modulator (yes/no)
);
// Overload to be used for DVB-T2
DTAPI_RESULT DtOutpChannel::SetModControl (
  [in] DtDvbT2Pars&  DvbT2Pars     // DVB-T2 modulation parameters
  [in] bool  MplpMod = false       // Use the multi-PLP modulator (yes/no)
);
```

Parameters

DvbC2Pars

DVB-C2 modulation parameters; see description of **class DtDvbC2Pars**.

DvbT2Pars

DVB-T2 modulation parameters; see description of **class DtDvbT2Pars**.

MplpMod

Specifies whether to use the multi-PLP modulator or the traditional single-PLP modulator. If true, the multi-PLP modulator is used.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The modulation parameters have been set successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_IDLE	Transmit-control state is not **DTAPI_TXCTRL_IDLE**; The requested modulation parameters can only be set in idle state
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The output channel does not support the specified modulation type

Remarks

DtOutpChannel::WriteMplp

Write data to a multi-PLP modulator FIFO.

```
DTAPI_RESULT DtOutpChannel::WriteMplp (
  [in] int  FifoIdx,          // FIFO index
  [in] char*  pBuffer,        // Pointer to data to be written to the FIFO
  [in] int  NumBytesToWrite // Number of bytes to be written
);
```

Parameters

FifoIndex

Specifies the FIFO index.

pBuffer

Pointer to the buffer containing the data to be written to the multi-PLP modulator FIFO. The pointer must be aligned to a 32-bit word boundary.

NumBytesToWrite

Number of bytes to be to be written to the multi-PLP modulator FIFO. The buffer size must be positive and a multiple of four.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Write operation has been completed successfully
DTAPI_E_INVALID_BUF	The buffer is not aligned to a 32-bit word boundary
DTAPI_E_INVALID_FIFO_IDX	Invalid FIFO index. FIFO index has not been specified in **DtOutpChannel::SetModControl** parameters
DTAPI_E_INVALID_SIZE	The specified transfer size is negative or not a multiple of four
DTAPI_E_IDLE	Cannot write data because transmission-control state is **DTAPI_TXCTRL_IDLE**
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

The data buffer can be any buffer in user space that is aligned to a 4-byte boundary. The data is only written when the transmit-control state is **DTAPI_TXCTRL_HOLD** or **DTAPI_TXCTRL_SEND** (see SetTxControl), *and* sufficient space is available in the FIFO. **WriteMplp** returns when all data has been transferred to the multi-PLP modulator FIFO.

The data from a multi-PLP modulator FIFO is only transferred to the modulator when all other multi-PLP modulator FIFOs have data to contribute. For this reason the thread executing **WriteMplp** will sleep forever if *NumBytesToWrite* is greater than the number of free bytes in the MPLP FIFO and one of the other MPLP FIFOs is empty.