

# DTAPI

## | Multi-PLP Extensions

## REFERENCE

November 2013



## Table of Contents

<b>Structures</b> .....	<b>3</b>	Class DtDvbC2Pars .....	<b>41</b>
Struct DtBigTsSplitPars .....	3	DtDvbC2Pars::CheckValidity .....	44
Struct DtComplexFloat .....	5	DtDvbC2Pars::GetParamInfo .....	45
Struct DtPlpInpPars .....	6	<b>DtDvbT2ComponentPars</b> .....	<b>46</b>
Struct DtTestPointOutPars .....	7	DtDvbT2ComponentPars .....	46
Struct DtVirtualOutData .....	8	<b>DtDvbT2Pars</b> .....	<b>52</b>
Struct DtVirtualOutPars .....	10	DtDvbT2Pars .....	52
<b>DVB-C2 Data Structures</b> .....	<b>11</b>	DtDvbT2Pars::CheckValidity .....	53
Struct DtDvbC2DSlicePars .....	11	DtDvbT2Pars::GetParamInfo .....	54
Struct DtDvbC2L1UpdateDSlicePars .....	13	DtDvbT2Pars::OptimisePlpNumBlocks .....	55
Struct DtDvbC2L1UpdatePlpPars .....	14	<b>DtIsdbTmmPars</b> .....	<b>56</b>
Struct DtDvbC2L1UpdatePars .....	15	Class DtIsdbTmmPars .....	56
Struct DtDvbC2ModStatus .....	16	DtIsdbTmmPars::CheckValidity .....	57
Struct DtDvbC2NotchPars .....	17	<b>Callback Functions</b> .....	<b>58</b>
Struct DtDvbC2Paprpars .....	18	DtTpWriteDataFunc .....	58
Struct DtDvbC2ParamInfo .....	19	<b>Global Functions</b> .....	<b>60</b>
Struct DtDvbC2PlpPars .....	20	::DtapiModPars2TsRate .....	60
Struct DtDvbC2XfecFrameHeader .....	23	<b>DtMplpOutpChannel</b> .....	<b>62</b>
<b>DVB-T2 Data Structures</b> .....	<b>24</b>	DtMplpOutpChannel .....	62
Struct DtDvbT2AuxPars .....	24	DtMplpOutpChannel::AttachVirtual .....	63
Struct DtDvbT2MiPars .....	25	DtMplpOutpChannel::GetMplpFifoFree .....	64
Struct DtDvbT2ModStatus .....	27	DtMplpOutpChannel::GetMplpFifoSize .....	65
Struct DtDvbT2Paprpars .....	28	DtMplpOutpChannel::GetMplpModStatus .....	66
Struct DtDvbT2ParamInfo .....	30	DtMplpOutpChannel::SetMplpChannelModelling .....	67
Struct DtDvbT2PlpPars .....	31	DtMplpOutpChannel::SetModControl .....	68
Struct DtDvbT2RbmEvent .....	35	DtMplpOutpChannel::WriteMplp .....	69
Struct DtDvbT2RbmValidation .....	39	DtMplpOutpChannel::WriteMplpPacket .....	70
Struct DtDvbT2TxSigPars .....	40		
<b>DtDvbC2Pars</b> .....	<b>41</b>		

Copyright © 2013 by DekTec Digital Video B.V.

DekTec Digital Video B.V. reserves the right to change products or specifications without notice. Information furnished in this document is believed to be accurate and reliable, but DekTec assumes no responsibility for any errors that may appear in this material.

## Structures

### Struct DtBigTsSplitPars

Structure for specifying the parameters for the “Big-TS splitting” operation, which is defined for DVB-C2 and DVB-T2. 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. Big-TS splitting is supported for DVB-C2 and DVB-T2.

*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**, **DtDvbT2ComponentPars** and in class **DtIsdbTmmPars**, in an array of structs. The index in the array corresponds to the index of the related PLP (or TS in case of ISDB-Tmm).

```
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. **DtMplpOutChannel::WriteMplp()**).

The default value of *m\_FifoIdx* is equal to the index in the array of **DtPlpInpPars** structs. For writing data to the *n*<sup>th</sup> 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
<b>GSE</b>	Generic Stream Encapsulation (GSE) packets
<b>TS188</b>	188-byte TS packets
<b>TS204</b>	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 **DtDvbT2ComponentPars**.

```
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 {                     // 16-bit 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 {                     // 32-bit 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
<b>IQ_INT16</b>	Pairs of signed 16-bit integers in I, Q order, little Endian
<b>IQ_FLOAT32</b>	Pairs of 32-bit floats in I, Q order
<b>T2MI_TS188</b>	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 **DtVirtualOutPars** 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
<b>IQ_INT16</b>	Pairs of signed 16-bit integers in I, Q order, little Endian
<b>IQ_FLOAT32</b>	Pairs of 32-bit floats in I, Q order
<b>T2MI_TS188</b>	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.

## DVB-C2 Data Structures

### 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_TIDEPH_NONE	No time interleaving
DTAPI_DVBC2_TIDEPH_4	4 OFDM symbols
DTAPI_DVBC2_TIDEPH_8	8 OFDM symbols
DTAPI_DVBC2_TIDEPH_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 DtDvbC2L1UpdateDSlicePars

Structure describing DVB-C2 parameter updates for one data slice. This structure is used in class `DtDvbC2L1UpdatePars`.

```
struct DtDvbC2L1UpdateDSlicePars
{
    bool    m_Enable;           // Enable the data slice (yes/no)
    int     m_OffsetLeft;       // Updated data slice left offset
    int     m_OffsetRight;      // Updated data slice right offset
    std::vector<DtDvbC2L1UpdatePlpPar> m_Plps; // L1 PLP updates
};
```

### Members

#### *m\_Enable*

If true, the data slice is enabled, otherwise it is disabled and the remaining parameters are not used. Only enabled data slices will occur in the L1 signalling.

Note that only “empty” data slices can be disabled. An empty data slice is either a data slice where `m_OffsetLeft == m_OffsetRight` in the global configuration, or a data slice where all PLPs have `m_NoData == true`.

#### *m\_OffsetLeft*

Updated 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*

Updated 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 the data slice is not empty then for type 1 data slices no change is accepted and for type 2 must hold that `m_OffsetLeft < m_OffsetRight`. It is up to the user to ensure that there is sufficient bandwidth and no bitrate overflow.

#### *m\_Plps*

A vector specifying the DVB-C2 parameters updates for the physical layer pipes. Note that the number of physical layer pipes and the order of physical layer pipes must be the same as in the global configuration.

## Struct DtDvbC2L1UpdatePlpPars

Structure describing DVB-C2 parameter updates for one physical layer pipe. This structure is used in class `DtDvbC2L1UpdateDSlicePars`.

```
struct DtDvbC2L1UpdatePlpPars
{
    bool    m_Enable;           // Enable the PLP (yes/no)
};
```

### Members

*m\_Enable*

If true, the physical layer pipe is enabled, otherwise it is disabled. Only enabled physical layer pipes will occur in the L1 signalling.

Note that only physical layer pipes where *m\_NoData==true* can be disabled.

## Struct DtDvbC2L1UpdatePars

Structure describing the updated DVB-C2 L1 signalling part2 parameters. This structure is used in class **DtDvbC2Pars**.

```
struct DtDvbC2L1UpdatePars
{
    int    m_NumFrames;           // Number of C2 frames the update is used
    // L1 data slice updates
    std::vector<DtDvbC2L1UpdateDSlicePars> m_DSlices;
};
```

### Members

*m\_NumFrames*

Number of C2 frames the updated data slice parameters are used.

*m\_DSlices*

A vector specifying for each data slice the updated data slice parameters.

Note that the number of data slices and the order of data slices must be the same as in **DtDvbC2Pars**.

## Struct DtDvbC2ModStatus

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

```
struct DtDvbC2ModStatus
{
    int    m_MplpModFlags;           // Multi-PLP-modulator flags
    __int64 m_DjbOverflows;         // Number of DJB overflows
    __int64 m_DjbUnderflows;        // Number of DJB underflows
};
```

### Members

*m\_MplpModFlags*

Multi-PLP-modulator flags. If the modulator stalls *m\_MplpModFlags* is set to a nonzero value.

*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 DtDvbC2PaprrPars

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

```
struct DtDvbC2PaprrPars
{
    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 Deletion (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_AcmHeaders; // ACM headers
    bool    m_PsiSiReproc;        // PSI/SI reprocessing is performed (yes/no)
    int     m_TsId;               // Transport stream ID
    int     m_OnwId;              // Original network ID
    bool    m_NoData;             // No input data is provided for this PLP
};
```

### 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.

### *m\_NoData*

If true, no input data is provided for this PLP. It is implicitly true for all PLPs in a data slice where *m\_OffsetLeft* == *m\_OffsetRight*.

## Struct DtDvbC2XFecFrameHeader

Structure describing the coding and modulation parameters for a series of XFEC frames for Adaptive Coding and Modulation (ACM) tests. 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. The valid range is 1 ... 256.

## DVB-T2 Data Structures

### Struct DtDvbT2AuxPars

Structure for specifying AUX stream parameters, which can be inserted for test purposes. This structure is used in class **DtDvbT2ComponentPars**.

```
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;               // (First) T2-MI data PID
    int     m_StreamId;          // Stream-id for the (first) T2-MI stream
    int     m_Pid2;              // Second T2-MI data PID
    int     m_StreamId2;         // Stream-id for the second T2-MI stream
    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\_StreamId*

Stream-id for the generated T2-MI stream. The valid range is 0 ... 7.

*m\_Pid2*

A second PID carrying the T2-MI packet data, used in case of multi-profile stream generation. The valid range is 0 ... 8190.

*m\_StreamId2*

Stream-id for the second generated T2-MI stream, used in case of multi-profile stream generation. The valid range is 0 ... 7.

*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
<b>DTAPI_DVBT2MI_TIMESTAMP_NULL</b>	Null timestamp
<b>DTAPI_DVBT2MI_TIMESTAMP_REL</b>	Relative timestamps. Use <i>m_Subseconds</i> .
<b>DTAPI_DVBT2MI_TIMESTAMP_ABS</b>	Absolute timestamps. Use <i>m_SecSince2000</i> , <i>m_Subseconds</i> and <i>m_T2MiUtco</i> .

*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 **DTAPI\_DVBT2MI\_TIMESTAMP\_ABS**.

The T2 system bandwidth defines the units of the subseconds as shown in the table below.

Bandwidth	Subseconds units, $T_{sub}$
<b>1.7 MHz</b>	1/131 $\mu s$
<b>5 MHz</b>	1/40 $\mu s$
<b>6 MHz</b>	1/48 $\mu s$
<b>7 MHz</b>	1/56 $\mu s$
<b>8 MHz</b>	1/64 $\mu s$
<b>10 MHz</b>	1/80 $\mu s$

*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 `DtMplpOutpChannel::GetMplpModStatus`.

```
struct DtDvbT2ModStatus
{
    int    m_MplpModFlags;           // Multi-PLP-modulator flags
    __int64 m_PlpNetBlocksOverflows; // 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\_MplpModFlags*

Multi-PLP-modulator flags. If the modulator stalls *m\_MplpModFlags* is set to a nonzero value.

*m\_PlpNetBlocksOverflows*

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 **DtDvbT2ComponentPars**.

```
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 interpolation 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
    bool   m_L1Scrambling;        // L1 Post Scrambling (for V1.3.1 only)
    // 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. Only applicable when DVB-T2 V1.3.1 is selected.

*m\_L1AceCMax*

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

*m\_L1Scrambling*

If true, L1-Post scrambling is active.

*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
<b>DTAPI_DVBT2_BIAS_BAL_OFF</b>	No L1 bias compensation
<b>DTAPI_DVBT2_BIAS_BAL_ON</b>	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
    int  m_SamplesPerFrame;      // #Samples per frame
};
```

### Members

*m\_TotalCellsPerFrame*

Total number of cells per frame.

*m\_L1CellsPerFrame*

Total number of cells per frame used for L1 signalling.

*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.

*m\_SamplesPerFrame*

Total number of samples per frame.

## Struct DtDvbT2PlpPars

Structure specifying the DVB-T2 modulation parameters for one PLP (Physical Layer Pipe). This structure is used in class **DtDvbT2ComponentPars**, in an array of **DTAPI\_DVBT2\_NUM\_PLP\_MAX** structs for the physical layer pipes.

```
struct DtDvbT2PlpPars
{
    // Mode adaptation layer: TS input
    bool m_Hem;                // High Efficiency Mode (yes/no)
    bool m_Npd;                // Null Packet Deletion (yes/no)
    int m_Issy;                // ISSY mode
    int m_IssyBufs;            // ISSY BUFS
    int m_IssyTDesign;         // ISSY T_design value
    int m_CompensatingDelay;    // Additional delay in samples
    int m_TsRate;              // Transport Stream rate

    // Mode adaptation layer: GSE input
    int m_GseLabelType;        // GSE-label type

    // L1 parameters
    int m_Id;                  // PLP ID
    int m_GroupId;             // PLP group ID
    int m_Type;                // PLP type
    int m_PayloadType;         // PLP payload 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

    // Parameters 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*

Transport 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 parameters.

*m\_GseLabelType*

GSE-label type.

Value	Meaning
DTAPI_DVBT2_GSE_LABEL_3BYTE	3-byte GSE label
DTAPI_DVBT2_GSE_LABEL_6BYTE	6-byte GSE label
DTAPI_DVBT2_GSE_LABEL_NONE	No GSE label

*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\_PayloadType*

PLP payload type.

Value	Meaning
DTAPI_DVBT2_PAYLOAD_GSE	Generic Stream Encapsulation
DTAPI_DVBT2_PAYLOAD_TS	Transport Stream

*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 (not for T2-Lite)
DTAPI_DVBT2_COD_5_6	5/6 (not for T2-Lite)
DTAPI_DVBT2_COD_1_3	1/3 (only for T2-Lite)
DTAPI_DVBT2_COD_2_5	2/5 (only for T2-Lite)

*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
<b>DTAPI_DVBT2_IL_ONETOONE</b>	One interleaving frame corresponds to one T2 frame
<b>DTAPI_DVBT2_IL_MULTI</b>	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;            // SYNCED
            int    m_Dfl;              // DFL
        } SyncDTooLarge;
        struct {
            // DTAPI DVBT2 RBM EVENT_INVALID_SYNCED parameters
            int    m_SyncD;            // SYNCED
            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_SYNCED_TOO_LARGE	SYNCD too large in BBFrame
DTAPI_DVBT2_RBM_EVENT_INVALID_UPL	Invalid UPL in BBFrame

<code>DTAPI_DVBT2_RBM_EVENT_INVALID_SYNCED</code>	Invalid SYNCED in BBFrame
<code>DTAPI_DVBT2_RBM_EVENT_TDI_OVERFLOW</code>	TDI overflow
<code>DTAPI_DVBT2_RBM_EVENT_TOO_MANY_TI_BLOCKS</code>	Too many TI blocks queued
<code>DTAPI_DVBT2_RBM_EVENT_INVALID_PLP_START</code>	PLP-start values gives overlap
<code>DTAPI_DVBT2_RBM_EVENT_FDI_OVERFLOW</code>	Frequency/L1 de-interleaver overflow
<code>DTAPI_DVBT2_RBM_EVENT_NO_TS_RATE</code>	Not enough ISCR data to estimate TS rate
<code>DTAPI_DVBT2_RBM_EVENT_ISCR_ERROR</code>	ISCR error
<code>DTAPI_DVBT2_RBM_EVENT_BUFS_NOT_CONSTANT</code>	BUFS not constant
<code>DTAPI_DVBT2_RBM_EVENT_ISSYI_NOT_CONSTANT</code>	ISSYI not constant
<code>DTAPI_DVBT2_RBM_EVENT_HEM_NOT_CONSTANT</code>	HEM not constant
<code>DTAPI_DVBT2_RBM_EVENT_PLP_NUM_BLOCKS_TOO_SMALL</code>	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 **DtDvbT2ComponentPars**.

```
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 **DtDvbT2ComponentPars**.

```
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.



## DtDvbC2Pars

### 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 frequency
    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;

    // Parameters specifying the generated carriers of one C2-system
    int    m_OutpFreqOffset;     // Output frequency offset
    int    m_OutpBandwidth;      // Output bandwidth (0 means default)
    // L1 updates
    std::vector<DtDvbC2L1UpdatePars> m_L1Updates;
};
```

#### 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 ... 0xFFFFFFFF 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.

*m\_OutpFreqOffset*

Output frequency offset from *m\_StartFrequency* (in carriers) of the generated spectrum. Must be a multiple of the carrier spacing ( $D_x=24$  for guard interval 1/128 and  $D_x=12$  for guard interval 1/64). *m\_OutpFreqOffset* in combination with *m\_OutpBandwidth* can be used to output a part of carriers of one C2-system.

*m\_OutpBandwidth*

Output bandwidth (in carriers). 0 selects the default output bandwidth. Must be a multiple of the carrier spacing ( $D_x=24$  for guard interval 1/128 and  $D_x=12$  for guard interval 1/64).

*m\_L1Updates*

A series of L1 signalling part2 parameters updates. The first update is applied immediately. After the last update is applied, it loops to the first one .

## Remarks

This class is used both for the initialization of the multi-PLP modulator and the traditional single-PLP DVB-C2 modulator. The **DtOutpChannel::SetModControl()** method sets the parameters for the single-PLP DVB-C2 modulator. Thereafter **DtOutpChannel::Write** method is used to write the data to the output channel.

The **DtMplpOutpChannel::SetModControl()** method sets the parameters for the multi-PLP DVB-C2 modulator. The multi-PLP modulator can be used for both single-PLP and multi-PLP parameter sets. The **DtMplpOutpChannel::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_BROADBAND_NOTCH	Broadband notch cannot be inside a data slice
DTAPI_E_DSLICE_OFFSETS	Invalid data slice offset
DTAPI_E_DSLICE_OVERLAP	Data slices cannot overlap
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_DSLICE_TUNE_POS	Invalid data slice tune position
DTAPI_E_INVALID_PARS	Invalid parameter value (generic error)
DTAPI_E_INVALID_RATE	PLP TS-rate is too high
DTAPI_E_INVALID_START_FREQ	Invalid start frequency
DTAPI_E_NO_TSRATE	PLP TS-rate is not specified
DTAPI_E_NOTCH_OFFSETS	Invalid notch
DTAPI_E_L1_PART2_TOO_LONG	L1 part 2 data is too long
DTAPI_E_PLP_BUNDLED	Inconsistent PLP bundled parameters
DTAPI_E_PLP_ID	Duplicate PLP IDs

## 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 <b>DtDvbC2Pars::CheckValidity</b>

## DtDvbT2ComponentPars

### DtDvbT2ComponentPars

Class describing the modulation parameters for one DVB-T2 component (e.g. base or lite).

```
class DtDvbT2ComponentPars
{
    int    m_T2Version;           // DVB-T2 specification version
    int    m_T2Profile;           // DVB-T2 profile
    bool   m_T2BaseLite;          // T2-Lite is used in a base profile
    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
    int    m_ComponentStartTime;  // Offset (T) at which the component starts
    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 stream
    DtDvbT2AuxPars m_Aux;        // AUX streams
    DtDvbT2PaprPars m_PaprPars; // PAPR reduction parameters
    DtDvbT2TxSigPars m_TxSignature; // Transmitter signature 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
DTAPI_DVBT2_VERSION_1_3_1	Version 1.3.1

*m\_T2Profile*

DVB-T2 profile.

Value	Meaning
DTAPI_DVBT2_PROFILE_BASE	Base profile
DTAPI_DVBT2_PROFILE_LITE	Lite profile (Requires DVB-T2 version 1.3.1)

*m\_T2BaseLite*

If true, T2 lite is used in a base profile component.

*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 TX1 and 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\_ComponentStartTime*

Specifies the offset in number of T-units at which the T2 component starts. Note: it should be set to 0 for the first component.

#### *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). For the base profile the valid range is 0 ... 0x3FFFFFF, for the lite profile the valid range is 0 ... 0xFFFFF.

#### *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
<b>DTAPI_DVBT2_FEF_ZERO</b>	Zero I/Q samples
<b>DTAPI_DVBT2_FEF_1K_OFDM</b>	1K OFDM symbols with 852 active carriers containing BPSK symbols
<b>DTAPI_DVBT2_FEF_1K_OFDM_384</b>	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\_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\_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.

## DtDvbT2Pars

### DtDvbT2Pars

Class describing parameters for DVB-T2 modulation, it describes the modulation parameters of a DVB-T2 component and optionally the parameters of a second component (e.g. base and a lite profile). The class **DtDvbT2ComponentPars** describes the component parameters.

```
class DtDvbT2Pars : public DtDvbT2ComponentPars
{
    int    m_NumFefComponents;    // Number of DVB-T2 components in FEF part
    DtDvbT2ComponentPars m_FefComponent[1];
                                // Parameters for a second DVB-T2 component

    DtVirtualOutPars m_VirtOutput;
                                // Virtual-output parameters
    DtDvbT2MiPars m_T2Mi;        // T2-MI output parameters
};
```

#### Inherited Public members

The public members inherited from **DtDvbT2ComponentPars** describe the modulation parameters for the first DVB-T2 component; see description of class **DtDvbT2ComponentPars**.

#### Public members

*m\_NumFefComponents*

The number of DVB-T2 components transmitted in the FEF part of the first DVB-T2 component. The parameters for these DVB-T2 components are specified in *m\_FefComponent*. The valid range is 0 ... 1.

*m\_FefComponent*

Array specifying the DVB-T2 modulation parameters for the DVB-T2 components transmitted in the FEF part of the first DVB-T2 component.

*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\_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.

#### Remarks

This class is used both for the initialization of the multi-PLP modulator and the traditional single-PLP DVB-T2 modulator. The **DtOutputChannel::SetModControl()** method sets the parameters for the single-PLP DVB-T2 modulator. Thereafter **DtOutputChannel::Write** method is used to write the data to the output channel.

The **DtMplpOutputChannel::SetModControl()** method sets the parameters for the multi-PLP DVB-T2 modulator. The multi-PLP modulator can be used for both single-PLP and multi-PLP parameter sets. The **DtMplpOutputChannel::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_COMP_OVERLAP	The frames of two components (lite and base profile) overlap.
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_MULTI_COMPS	Invalid mix of parameters in multi component configuration
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_T2_LITE	Invalid T2 lite profile parameters
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    // (First) T2-component information  
);  
  
DTAPI_RESULT DtDvbT2Pars::GetParamInfo(  
    [out] DtDvbT2ParamInfo& ParamInfo1, // First T2-component information  
    [out] DtDvbT2ParamInfo& ParamInfo2  // Second T2-component information  
);
```

### Parameters

*ParamInfo, ParamInfo1*

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

*ParamInfo2*

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

### Result

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

## 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 <b>DtDvbT2Pars::CheckValidity</b>

### Remarks

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

## DtIsdbTmmPars

### Class DtIsdbTmmPars

Class specifying parameters for ISDB-Tmm modulation.

```
class DtIsdbTmmPars
{
    int m_Bandwidth;           // Bandwidth (channel raster)
    int m_SubChannel;         // Sub-channel of the center segment
    // TS parameters
    int m_NumTss;              // Number of transport streams
    DtIsdbtPars m_Tss[DTAPI_ISDBT_NUM_TS_MAX];
    DtPlpInpPars m_TsInputs[DTAPI_ISDBT_NUM_TS_MAX];
    // Virtual output
    DtVirtualOutPars m_VirtOutput;
};
```

#### Public members

*m\_Bandwidth*

Channel raster of the network.

Value	Meaning
DTAPI_ISDBT_BW_6MHZ	6 MHz
DTAPI_ISDBT_BW_7MHZ	7 MHz
DTAPI_ISDBT_BW_8MHZ	8 MHz

*m\_SubChannel*

Sub channel of the center segment of the spectrum, which implicitly specifies the sub-channels of the 1-segment streams in the signal. The valid range is 0 ... 41.

*m\_NumTss*

Specifies the number of transport streams in the ISDB-Tmm system. The valid range is 0 ... DTAPI\_ISDBT\_NUM\_TS\_MAX.

*m\_Tss*

An array of **DtIsdbtPars**, specifying the modulation parameters for each transport stream. The index in the array is related to the index of the transport stream in the ISDB-Tmm system. See the description of struct **DtIsdbtPars** in “DTAPI Reference – Core Classes”.

*m\_TsInputs*

Array specifying the input transport streams. The index in the array is related to the index of the transport stream in the ISDB-Tmm system.

*m\_VirtOutput*

In case of a virtual output *m\_VirtOutput* specifies the virtual output data parameters.

#### Remarks

This class is only used for the initialization of the multi-PLP modulator. The class **DtIsdbtPars** can be used to initialize the traditional single-PLP modulator.



## DtIsdbTmmPars::CheckValidity

Check ISDB-T parameters for validity.

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

### Parameters

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Parameters are valid
DTAPI_E_INVALID_BANDWIDTH	Invalid value for bandwidth or not equal for all streams
DTAPI_E_INVALID_BTYPE	Invalid value for broadcast type
DTAPI_E_INVALID_GUARD	Invalid value for guard-interval length or not equal for all streams
DTAPI_E_INVALID_MODE	Invalid value for transmission mode or not equal for all streams
DTAPI_E_INVALID_NUMSEGM	Number of segments is more than 33 or the number of segments in a streams is not equal to 1, 3 or 13, or number of segments is invalid for the current broadcast type does not match number of segments specified in m_LayerPars
DTAPI_E_INVALID_PARTIAL	'Partial Reception' is selected for a stream but the number of segments in layer A is not 1
DTAPI_E_INVALID_SIZE	'No hierarchical multiplexing' (use TMCC) is selected for a stream where the input type is 188-byte TS packets
DTAPI_E_INVALID_SUBCH	Invalid sub-channel number

## 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
<code>DTAPI_TP_FORMAT_HEX</code>	Byte data
<code>DTAPI_TP_FORMAT_BIT</code>	Bit data. Eight bits are packaged per byte, most significant bit first
<code>DTAPI_TP_FORMAT_CFLOAT32</code>	Complex 32-bit floating-point data of type <code>DtComplexFloat</code>
<code>DTAPI_TP_FORMAT_INT64</code>	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 three new overloads one for DVB-C2, one for DVB-T2 and one for ISDB-Tmm 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
);
// Overload to be used for ISDB-Tmm
DTAPI_RESULT ::DtapiModPars2TsRate(
    [out] int& TsRate           // Computed Transport-Stream rate
    [in] DtIsdbTmmPars TmmPars, // ISDB-Tmm modulation parameters
    [in] int  TsIdx            // TS 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`.

*TmmPars*

ISDB-Tmm modulation parameters; see description of `class DtIsdbTmmPars`.

*PlpIdx*

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

*TsIdx*

The index of the TS in the ISDB-Tmm system 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

	<code>DtDvbC2Pars::CheckValidity</code> , <code>DtDvbT2Pars::CheckValidity</code> and <code>DtIsdbTmmPars::CheckValidity</code>
--	---

## ***DtMplpOutpChannel***

### **DtMplpOutpChannel**

Class representing a multi-PLP modulator for modulation of DVB-C2, DVB-T2 and ISDB-Tmm signals. Class **DtMplpOutpChannel** is derived from **DtOutpChannel**. For the inherited methods, please refer to the [DTAPI](#) documentation.

```
class DtMplpOutpChannel : public DtOutpChannel;
```

## DtMplpOutpChannel::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 DtMplpOutpChannel::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 <b>DtDevice</b> pointer is not valid or the <b>DtDevice</b> 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.

## DtMplpOutpChannel::GetMplpFifoFree

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

```
DTAPI_RESULT DtMplpOutpChannel::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.



## DtMplpOutpChannel::GetMplpFifoSize

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

```
DTAPI_RESULT DtMplpOutpChannel::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.

## DtMplpOutpChannel::GetMplpModStatus

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

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

DTAPI_RESULT DtMplpOutpChannel::GetMplpModStatus (
    [out] DtDvbT2ModStatus* pDvbT2ModStat      // Status of DVB-T2
    modulator
);

DTAPI_RESULT DtMplpOutpChannel::GetMplpModStatus (
    [out] DtDvbT2ModStatus* pDvbT2ModStat1     // Status of DVB-T2
    modulator for the first component
    [out] DtDvbT2ModStatus* pDvbT2ModStat2     // Status of DVB-T2
    modulator for the second component
);
```

### Parameters

*pDvbC2ModStat*

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

*pDvbT2ModStat, pDvbT2ModStat1*

DVB-T2 modulator status for the first component; see description of **struct DtDvbT2ModStatus**.

*pDvbT2ModStat2*

DVB-T2 modulator status for the second component; 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

## DtMplpOutpChannel::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 DtMplpOutpChannel::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 “DTAPI Reference – Core Classes”.

#### *ChannelIdx*

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

### Remarks

## DtMplpOutpChannel::SetModControl

Set modulation-control parameters for modulator channels. There are three overloads defined for the multi-PLP modulator output: one for DVB-C2, one for DVB-T2 and one for ISDB-Tmm.

```
// Overload to be used for DVB-C2
DTAPI_RESULT DtMplpOutpChannel::SetModControl (
    [in] DtDvbC2Pars&   DvbC2Pars    // DVB-C2 modulation parameters
);
// Overload to be used for DVB-T2
DTAPI_RESULT DtMplpOutpChannel::SetModControl (
    [in] DtDvbT2Pars&   DvbT2Pars    // DVB-T2 modulation parameters
);
// Overload to be used for ISDB-Tmm
DTAPI_RESULT DtMplpOutpChannel::SetModControl (
    [in] DtIsdbTmmPars& IsdbTmmPars // ISDB-Tmm modulation parameters
);
```

### Parameters

*DvbC2Pars*

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

*DvbT2Pars*

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

*IsdbTmmPars*

ISDB-Tmm modulation parameters; see description of **class DtIsdbTmmPars**.

### 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 <b>DTAPI_TXCTRL_IDLE</b> ; 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

## DtMplpOutpChannel::WriteMplp

Write data to a multi-PLP modulator FIFO.

```
DTAPI_RESULT DtMplpOutpChannel::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 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 <b>DtMplpOutpChannel::SetModControl</b> 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 <b>DTAPI_TXCTRL_IDLE</b>
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

### Remarks

The data buffer can be any buffer in user space. The data is only written when the transmit-control state is **DTAPI\_TXCTRL\_HOLD** or **DTAPI\_TXCTRL\_SEND** (see **DtOutpChannel::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 multi-PLP modulator FIFOs carrying transport packets (data type **TS188** or **TS204**) have data to contribute. The contribution of multi-PLP modulator FIFOs carrying GSE-packets (data type **GSE**) is optional.

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 (data type **TS188** or **TS204**) is empty.

## DtMplpOutChannel::WriteMplpPacket

Write a GSE-packet to a multi-PLP modulator FIFO.

```
DTAPI_RESULT DtMplpOutChannel::WriteMplpPacket(
    [in] int    FifoIdx,           // FIFO index
    [in] char*  pPacket,          // Packet to be written to the FIFO
    [in] int    PacketSize        // Size of the packet
);
```

### Parameters

*FifoIndex*

Specifies the FIFO index.

*pPacket*

Pointer to the GSE-packet to be written to the multi-PLP modulator FIFO.

Syntax	#bits	Mnemonic
GsePacket() {		
<b>protocol_type</b>	<b>16</b>	<b>uimsbf</b>
if (GseLabelType == <b>NONE</b> ) {		
<b>reserved</b>	<b>48</b>	<b>bslbf</b>
} else if (GseLabelType == <b>3BYTE</b> ) {		
<b>label_3byte</b>	<b>24</b>	<b>bslbf</b>
<b>reserved</b>	<b>24</b>	<b>bslbf</b>
} else if (GseLabelType == <b>6BYTE</b> ) {		
<b>label_6byte</b>	<b>48</b>	<b>bslbf</b>
}		
for (i=0; i<N1; i++)		
<b>extension_header_byte</b>	<b>8</b>	<b>bslbf</b>
for (i=0; i<N2; i++)		
<b>data_byte</b>	<b>8</b>	<b>bslbf</b>
}		

*protocol\_type*

Protocol type carried in the packet, 16-bit field (2 bytes network order).

*label\_3byte, label\_6byte*

Label used for addressing, 0, 3 or 6 bytes, the address length is specified by *m\_GseLabelType*. The length of the label and the reserved bits is 6 byte.

*extension\_header\_byte*

Optional extension header bytes, the format depends on protocol type.

*data\_byte*

Packet data byte.

*PacketSize*

Size of the GSE-packet to be written to the multi-PLP modulator FIFO.

## 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 <code>DtMplpOutpChannel::SetModControl</code> parameters
DTAPI_E_INVALID_INP_TYPE	The data type of the FIFO is not GSE
DTAPI_E_INVALID_SIZE	The specified transfer size is negative or larger than the maximum packet size
DTAPI_E_IDLE	Cannot write data because transmission-control state is <code>DTAPI_TXCTRL_IDLE</code>
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

## Remarks

The data buffer can be any buffer in user space. The data is only written when the transmit-control state is `DTAPI_TXCTRL_HOLD` or `DTAPI_TXCTRL_SEND` (see `DtOutpChannel::SetTxControl()`), and sufficient space is available in the FIFO. `WriteMplpPacket()` 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 multi-PLP modulator FIFOs carrying transport packets (data type `TS188` or `TS204`) have data to contribute. The contribution of multi-PLP modulator FIFOs carrying GSE-packets (data type `GSE`) is optional.

For this reason the thread executing `WriteMplpPacket()` will sleep forever if `PacketSize` is greater than the number of free bytes in the MPLP FIFO and one of the other MPLP FIFOs (data type `TS188` or `TS204`) is empty.