DTAPI

CORE CLASSES

REFERENCE

Dec 2012

Table of Contents

struct DtCmPars

This structure specifies channel-modelling parameters. It's used to simulate the transmission distortions that may occur in the channel between a transmitter and a receiver.

```
struct DtCmPars
{
   bool   m_EnableAwgn;          // Enable white Gaussian noise injection
   double  m_Snr;                // Signal-to-noise ratio in dB
   bool   m_EnablePaths;         // Enable multi-path simulation
   std::vector<DtCmPath>  m_Paths;
};
```

Members

m_EnableAwgn
Enable Additive White Gaussian Noise (AWGN) injection.

m_Snr
Signal-to-noise ratio. The noise power is defined relative to an imaginative 0dB output signal of the modulator. This means that *m_Snr* is the real signal-to-noise ratio only if the accumulated power of the paths in *m_Paths* is 0dB.

m_EnablePaths
Enable the simulation of multiple transmission fading paths ("multi-path"). The common usage is to define one main path and several auxiliary paths for simulating echoes. If no paths are defined, a single 0dB path is used.

m_Paths
Vector of path parameters.

struct DtCmPath

This structure specifies the fading parameters for a single path in a multi-path simulation.

```
struct DtCmPath
{
  Type   m_Type;                // Type of path fading
  double  m_Attenuation;        // Attenuation in dB
  double  m_Delay;              // Delay in us
  double  m_Phase;              // Phase shift in degrees
  double  m_Doppler;            // Doppler frequency in Hz
};
```

Members

m_Type
Enumeration defining the type of path fading.

Value	Meaning
CONSTANT_DELAY	Constant delay/phase
CONSTANT_DOPPLER	Constant frequency shift
RAYLEIGH_JAKES	Rayleigh fading with Jakes power spectral density (mobile path model)
RAYLEIGH_GAUSSIAN	Rayleigh fading with Gaussian power spectral density (iono-spheric path model)

m_Attenuation
Attenuation in dB. The total attenuation of all paths should not exceed 0dB to avoid overflow of the channel simulator.

m_Delay
Delay in μs. The maximum delay for an 8MHz channel is 896μs.

m_Phase
Constant phase shift in degrees. Used for **CONSTANT_DELAY** and **CONSTANT_DOPPLER**; Don't care for other path types.

m_Doppler
Doppler frequency shift in Hz for all paths except **CONSTANT_DELAY**. The corresponding Speed in m/s is: Speed = $f_{doppler} * 3.10^8 / f_{RF}$.

struct DtConstelPoint

This structure describes a constellation point in a receiver constellation diagram.

```
struct DtConstelPoint
{
   int  m_X;                    // X-coordinate of the constellation point
   int  m_Y;                    // Y-coordinate of the constellation point
};
```

Members

m_X, m_Y
The X- and Y-coordinate of the described constellation point.

struct DtDemodParsAtsc

This structure describes the demodulation parameters for ATSC.

```
struct DtDemodParsAtsc
{
  int  m_Constellation;      // VSB constellation
};
```

Members

m_Constellation

The VSB constellation.

Value	Meaning	Symbol Rate (bd)	TS Rate (bps)
DTAPI_MOD_ATSC_VSB8	8-VSB	10,762,238	19,392,658
DTAPI_MOD_ATSC_VSB16	16-VSB	10,762,238	38,785,317

struct DtDemodParsDvbC2

This structure describes the demodulation parameters for DVB-C2.

```
struct DtDemodParsDvbC2
{
  int  m_Bandwidth;          // DVB-C2 Bandwidth (channel raster)
};
```

Members

m_Bandwidth

Channel raster of the network.

Value	Meaning
DTAPI_DVBC2_6MHZ	6 MHz
DTAPI_DVBC2_8MHZ	8 MHz

struct DtDemodParsDvbS

This structure describes the demodulation parameters for DVB-S.

```
struct DtDemodParsDvbS
{
   int  m_CodeRate;            // DVB-S code rate
   int  m_SpecInv;             // Spectral inversion (yes/no)
   int  m_SymRate;             // Symbol rate in baud
};
```

Members

m_CodeRate

DVB-S code rate

Value	Meaning
DTAPI_MOD_1_2	Code rate 1/2
DTAPI_MOD_2_3	Code rate 2/3
DTAPI_MOD_3_4	Code rate 3/4
DTAPI_MOD_4_5	Code rate 4/5
DTAPI_MOD_5_6	Code rate 5/6
DTAPI_MOD_6_7	Code rate 6/7
DTAPI_MOD_7_8	Code rate 7/8
DTAPI_MOD_CR_AUTO	Autodetect code rate
DTAPI_MOD_CR_UNK	Code rate is unknown

m_SpecInv

Spectral inversion

Value	Meaning
DTAPI_MOD_S_S2_SPECNONINV	No spectrum inversion detected
DTAPI_MOD_S_S2_SPECINV	Spectrum inversion detected
DTAPI_MOD_S_S2_SPECINV_AUTO	Autodetect spectrum inversion
DTAPI_MOD_S_S2_SPECINV_UNK	Spectrum inversion status is unknown

m_SymRate

The symbol rate (in bd). The value **DTAPI_MOD_SYMRATE_AUTO** specifies automatic detection of the symbol rate. The value **DTAPI_MOD_SYMRATE_UNK** indicates the symbol rate could not be detected.

struct DtDemodParsDvbS2

This structure describes the demodulation parameters for DVB-S.2.

```
struct DtDemodParsDvbS2
{
  int  m_CodeRate;          // DVB-S.2 code rate
  int  m_FecFrame;          // Long or short FEC-frames
  int  m_Pilots;            // Pilots (yes/no)
  int  m_SpecInv;           // Spectral inversion (yes/no)
  int  m_SymRate;           // Symbol rate in baud
};
```

Members

m_CodeRate

DVB-S.2 code rate

Value	Meaning
DTAPI_MOD_1_2	Code rate 1/2
DTAPI_MOD_1_3	Code rate 1/3
DTAPI_MOD_1_4	Code rate 1/4
DTAPI_MOD_2_3	Code rate 2/3
DTAPI_MOD_2_5	Code rate 2/5
DTAPI_MOD_3_4	Code rate 3/4
DTAPI_MOD_3_5	Code rate 3/5
DTAPI_MOD_4_5	Code rate 4/5
DTAPI_MOD_5_6	Code rate 5/6
DTAPI_MOD_6_7	Code rate 6/7
DTAPI_MOD_7_8	Code rate 7/8
DTAPI_MOD_8_9	Code rate 8/9
DTAPI_MOD_9_10	Code rate 9/10
DTAPI_MOD_CR_AUTO	Autodetect code rate
DTAPI_MOD_CR_UNK	Code rate is unknown

m_FecFrame

FEC-frame length

Value	Meaning
DTAPI_MOD_S2_SHORTFRM	Short FEC-frame
DTAPI_MOD_S2_LONGFRM	Long FEC-frame
DTAPI_MOD_S2_FRM_AUTO	Autodetect FEC-frame length
DTAPI_MOD_S2_FRM_UNK	FEC-frame length is unknown

m_Pilots

DVB-S.2 pilots

Value	Meaning
DTAPI_MOD_S2_NOPILOTS	Pilots disabled
DTAPI_MOD_S2_PILOTS	Pilots enabled
DTAPI_MOD_S2_PILOTS_AUTO	Autodetect pilots status
DTAPI_MOD_S2_PILOTS_UNK	Pilots status is unknown

m_SpecInv

Spectral inversion

Value	Meaning
DTAPI_MOD_S_S2_SPECNONINV	No spectrum inversion detected
DTAPI_MOD_S_S2_SPECINV	Spectrum inversion detected
DTAPI_MOD_S_S2_SPECINV_AUTO	Autodetect spectrum inversion
DTAPI_MOD_S_S2_SPECINV_UNK	Spectrum inversion status is unknown

m_SymRate

The symbol rate (in bd). The value **DTAPI_MOD_SYMRATE_AUTO** specifies automatic detection of the symbol rate. The value **DTAPI_MOD_SYMRATE_UNK** indicates the symbol rate could not be detected.

struct DtDemodParsDvbT

This structure describes the demodulation parameters for DVB-T.

```
struct DtDemodParsDvbT
{
   int  m_Bandwidth;          // Bandwidth
   int  m_CodeRate;           // Code rate
   int  m_Constellation;      // Constellation
   int  m_Guard;              // Guard interval
   int  m_Interleaving;       // Interleaving
   int  m_Mode;               // Transmission mode
};
```

Members

m_Bandwidth

Bandwidth.

Value	Meaning
DTAPI_DVBT_6MHZ	6 MHz
DTAPI_DVBT_7MHZ	7 MHz
DTAPI_DVBT_8MHZ	8 MHz

m_CodeRate

DVB-T code rate

Value	Meaning
DTAPI_MOD_1_2	Code rate 1/2
DTAPI_MOD_2_3	Code rate 2/3
DTAPI_MOD_3_4	Code rate 3/4
DTAPI_MOD_5_6	Code rate 5/6
DTAPI_MOD_7_8	Code rate 7/8
DTAPI_MOD_CR_AUTO	Autodetect code rate
DTAPI_MOD_CR_UNK	Code rate is unknown

m_Constellation

Constellation

Value	Meaning
DTAPI_MOD_DVBT_QPSK	QPSK
DTAPI_MOD_DVBT_QAM16	16-QAM
DTAPI_MOD_DVBT_QAM64	64-QAM
DTAPI_MOD_DVBT_CO_AUTO	Autodetect constellation
DTAPI_MOD_DVBT_CO_UNK	Constellation is unknown

m_Guard

Guard interval

Value	Meaning
DTAPI_MOD_DVBT_G_1_32	1/32
DTAPI_MOD_DVBT_G_1_16	1/16
DTAPI_MOD_DVBT_G_1_8	1/8
DTAPI_MOD_DVBT_G_1_4	1/4
DTAPI_MOD_DVBT_G_UNK	Autodetect guard interval
DTAPI_MOD_DVBT_G_UNK	Guard interval is unknown

m_Interleaving

Interleaving

Value	Meaning
DTAPI_MOD_DVBT_INDEPTH	In-depth interleaver (2k, 4k)
DTAPI_MOD_DVBT_NATIVE	Native interleaver
DTAPI_MOD_DVBT_IL_AUTO	Autodetect interleaving
DTAPI_MOD_DVBT_IL_UNK	Interleaving is unknown

m_Mode

Transmission mode

Value	Meaning
DTAPI_MOD_DVBT_2K	2k mode
DTAPI_MOD_DVBT_8K	8k mode
DTAPI_MOD_DVBT_MD_AUTO	Autodetect transmission mode
DTAPI_MOD_DVBT_MD_UNK	Transmission mode is unknown

struct DtDemodParsDvbT2

This structure describes the demodulation parameters for DVB-T2.

```
struct DtDemodParsDvbT2
{
  int  m_Bandwidth;          // Bandwidth
  int  m_T2Profile;          // DVB-T2 profile
};
```

Members

m_Bandwidth

Bandwidth.

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_T2Profile

DVB-T2 profile

Value	Meaning
DTAPI_DVBT2_PROFILE_BASE	Base profile
DTAPI_DVBT2_PROFILE_LITE	Lite profile

struct DtDemodParsIq

This structure describes the parameters for reception of I/Q-samples.

```
Struct DtDemodParsDvbT2
{
   int   m_Bandwidth;           // Signal bandwidth in Hz
   int   m_IqDemodType;         // Modulation type
   int   m_SampleRate;          // Sample rate in Hz
};
```

Members

m_Bandwidth

Bandwidth in Hz. The valid bandwidth values are: 1700000, 5000000, 6000000, 7000000, 8000000 and 10000000 Hz.

m_IqDemodType

The input signal's modulation type.

Value	Meaning
DTAPI_DEMOD_OFDM	OFDM modulated signal
DTAPI_DEMOD_QAM	QAM modulated signal

m_SampleRate

Sample rate in Hz.

struct DtDemodParsQam

This structure describes the demodulation parameters for QAM-A, B and C.

```
struct DtDemodParsQam
{
  int  m_Annex;              // ITU-T J.83 annex
  int  m_Interleaving;       // Interleaving; ignored vor Annex A and C
  int  m_SymRate;            // Symbol rate in baud
};
```

Members

m_Annex

ITU-T J.83 Annex.

Value	Meaning
DTAPI_MOD_J83_A	J.83 annex A (DVB-C)
DTAPI_MOD_J83_B	J.83 annex B (“American QAM”)
DTAPI_MOD_J83_C	J.83 annex C (“Japanese QAM”)

m_Interleaving

For J.83 Annex B, this parameter specifies the interleaving mode detected as specified in the table below. For Annex A and C this parameter is not used.

Value	CW	I	J	Burst protection 64-/256-QAM
DTAPI_MOD_QAMB_I128_J1D	0001	128	1	95 µs / 66 µs
DTAPI_MOD_QAMB_I64_J2	0011	64	2	47 µs / 33 µs
DTAPI_MOD_QAMB_I32_J4	0101	32	4	24 µs / 16 µs
DTAPI_MOD_QAMB_I16_J8	0111	16	8	12 µs / 8.2 µs
DTAPI_MOD_QAMB_I8_J16	1001	8	16	5.9 µs / 4.1 µs
DTAPI_MOD_QAMB_I128_J1	0000	128	1	95 µs / 66 µs
DTAPI_MOD_QAMB_I128_J2	0010	128	2	190 µs / 132 µs
DTAPI_MOD_QAMB_I128_J3	0100	128	3	285 µs / 198 µs
DTAPI_MOD_QAMB_I128_J4	0110	128	4	379 µs / 264 µs
DTAPI_MOD_QAMB_I128_J5	1000	128	5	474 µs / 330 µs
DTAPI_MOD_QAMB_I128_J6	1010	128	6	569 µs / 396 µs
DTAPI_MOD_QAMB_I128_J7	1100	128	7	664 µs / 462 µs
DTAPI_MOD_QAMB_I128_J8	1110	128	8	759 µs / 528 µs
DTAPI_MOD_QAMB_IL_AUTO	-	-	-	Autodetect interleaving mode
DTAPI_MOD_QAMB_IL_UNK	-	-	-	Interleaving mode is unknown

m_SymRate

The symbol rate (in bd). The value **DTAPI_MOD_SYMRATE_AUTO** specifies automatic detection of the symbol rate. The value **DTAPI_MOD_SYMRATE_UNK** indicates the symbol rate could not be detected.

struct DtDeviceDesc

This structure describes a DekTec device.

```
struct DtDeviceDesc
{
   int   m_Category;            // Device category (DTAPI_CAT_XXX)
   __int64  m_Serial;           // Unique serial number of the device
   int   m_PciBusNumber;        // PCI-bus number
   int   m_SlotNumber;          // PCI-slot number
   int   m_UsbAddress;          // USB address
   int   m_TypeNumber;          // Device type number
   int   m_SubType;             // Device subtype (0=none, 1=A, ...)
   int   m_DeviceId;            // Device ID
   int   m_VendorId;            // Vendor ID
   int   m_SubsystemId;         // Subsystem ID
   int   m_SubVendorId;         // Subsystem Vendor ID
   int   m_NumHwFuncs;          // #Hardware functions hosted by device
   int   m_HardwareRevision;    // Hardware revision (e.g. 302 = 3.2)
   int   m_FirmwareVersion;     // Firmware version
   int   m_FirmwareVariant;     // Firmware variant
   int   m_NumDtInpChan;        // Number of input channels
   int   m_NumDtOutpChan;       // Number of output channels
   int   m_NumPorts;            // Number of physical ports
   int   m_Ip[4];               // IP address (only for DTE-31XX)
   int   m_MacAddr[6];          // MAC address (only for DTE-31XX)
};
```

Members

m_Category

Code indicating the device category.

Value	Meaning
DTAPI_CAT_PCI	PCI or PCI-Express device
DTAPI_CAT_USB	USB-2 or USB-3 device
DTAPI_CAT_NW	Pseudo category that is used to refer to the network aspect of a device. This category value is used for getting the device driver version of the network driver and to refer to network related events.
DTAPI_CAT_IP	Network appliance: DTE-31XX
DTAPI_CAT_NIC	Non-DekTec network card (local NIC)

m_Serial

The serial number that uniquely identifies the device.

m_PciBusNumber, m_SlotNumber

For devices in category **DTAPI_CAT_PCI**, these integers identify the PCI bus and slot number in which the PCI card is installed. For other categories, the values are undefined.

m_UsbAddress

For devices in category **DTAPI_CAT_USB**, this number identifies the USB address of the device. For other categories, the value is undefined.

m_TypeNumber
This integer corresponds to the integer in the DekTec type identifier for the device, e.g. 2144 for the DTA-2144.

m_SubType
This integer identifies the subtype of the device.

Value	Meaning
-1	The driver could not establish a value for subtype[1].
0	Subtype is not applicable (no suffix to type number)
1	The type number is suffixed by 'A'
2	The type number is suffixed by 'B'
: :	etc.

Example: If *m_TypeNumber* is 2137 and *m_SubType* is 3, the full type number is DTA-2137C.

m_DeviceId, m_VendorId, m_SubsystemId, m_SubVendorId
Device ID, Vendor ID, Subsystem ID and Subsystem Vendor ID. Identification information of the device, as read from its PCI configuration-space registers.

m_NumHwFuncs
Number of hardware functions hosted by the device.

m_HardwareRevision
Hardware revision of the device, encoded as major hardware revision times 100 plus minor hardware revision. For example, *m_HardwareRevision*=102 corresponds to hardware revision r1.2, while *m_HardwareRevision*=310 corresponds to r3.10.

m_FirmwareVersion
Version number of the firmware loaded in the device.

m_FirmwareVariant
Variant of the firmware loaded on the device. This is not used in the current set of DekTec devices, but future devices may support multiple variants of the firmware each with different functionality.

m_NumDtInpChan
Number of input channels available on the device. For devices that have ports that are software programmable as input or output, the maximum number of input channels is used.
IP ports count as 1 input channel and 1 output channel.

m_NumDtOutpChan
Number of output channels available on the device. For devices that have ports that are software programmable as input or output, the maximum number of output channels is used.
IP ports count as 1 input channel and 1 output channel.

m_NumPorts
Number of physical ports available on the device. Doubly-buffered outputs count as a single port.

[1] This is an error condition that cannot occur for a correctly working board.

m_Ip

For devices in category **DTAPI_CAT_IP**, this member identifies the IP address of the device. Otherwise, the value of this member is undefined.

m_Mac

For devices in category **DTAPI_CAT_IP**, this member identifies the MAC address of the device. Otherwise, the value of this member is undefined.

struct DtDvbC2DemodPlpSigDataPlp

This structure specifies the DVB-C2 layer-1 signalling data for one physical layer pipe (PLP).

```
struct DtDvbC2DemodPlpSigDataPlp
{
   int  m_Id;                   // PLP ID: 0…255
   int  m_FecType;              // FEC type
   int  m_CodeRate;             // Code rate
   int  m_Modulation;           // Modulation type
   int  m_HdrCntr;              // Header counter
};
```

Members

m_Id

Unique identification of the PLP within a C2 system. 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 FEC-frames following the FEC-frame header.

Value	Meaning
0	1 FEC frame
1	2 FEC frames

Remarks

For type-1 data slices this structure contains the PLP signalling information from the layer-1 part-2 signalling data. For type-2 data slices this structure contains the PLP signalling information from the layer-1 part-1 signalling data (=FEC-frame header).

Unsupported fields are set to **DTAPI_STAT_UNSUP_INTITEM**.

struct DtDvbC2DemodPlpSigData

This structure specifies the DVB-C2 layer-1 signalling information for the physical layer pipes.

```
struct DtDvbC2DemodPlpSigData
{
   int  m_NumPlps;              // Number of PLPs
   std::vector<DtDvbC2DemodPlpSigDataPlp>  m_Plps;
};
```

Members

m_NumPlps
Specifies the number of physical layer pipes signalled in the DVB-C2 layer-1 signalling data.

m_Plps
A vector specifying the DVB-C2 layer-1 signalling data for the physical layer pipes (not necessarily for each detected PLP).

struct DtDvbC2DemodL1Part2Plp

This structure specifies the DVB-C2 layer-1part 2 signalling information for one physical layer pipe (PLP).

```
struct DtDvbC2DemodL1Part2Plp
{
   int  m_Id;                  // PLP ID: 0…255
   int  m_Bundled;             // PLP bundled (yes/no)
   int  m_Type;                // PLP type
   int  m_PayloadType;         // PLP payload type
   int  m_GroupId;             // PLP group ID
   int  m_Start;               // Start of first complete XFEC-frame
   int  m_FecType;             // FEC type
   int  m_Modulation;          // Modulation type
   int  m_CodeRate;            // Code rate
   int  m_PsiSiReproc;         // PSI/SI reprocessing is performed (yes/no)
   int  m_TsId;                // Transport stream ID
   int  m_OnwId;               // Original network ID
};
```

Members

m_Id

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

m_Bundled

If '1', the associated PLP is bundled with other PLP(s) within the current C2 system. All the bundled PLPs have the same PLP ID.

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_PayloadType

PLP payload type.

Value	Meaning
DTAPI_DVBC2_PAYLOAD_GFPS	Generic Fixed-length Packetized Stream
DTAPI_DVBC2_PAYLOAD_GCS	Generic Continuous Stream
DTAPI_DVBC2_PAYLOAD_GSE	Generic Stream Encapsulation
DTAPI_DVBC2_PAYLOAD_TS	Transport Stream

m_GroupId

Identifies the PLP group with which the PLP is associated.

m_Start
Start position of the first complete XFEC-frame of the PLP. Not set for type 2 data slices.

m_FecType
FEC type used by the PLP. Not set for type 2 data slices.

Value	Meaning
DTAPI_DVBC2_LDPC_16K	16K LDPC
DTAPI_DVBC2_LDPC_64K	64K LDPC

m_Modulation
Modulation used by the PLP. Not set for type 2 data slices.

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_CodeRate
Convolutional coding rate used by the PLP. Not set for type 2 data slices.

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_PsiSiReproc
If '1', indicates that PSI/SI has been reprocessed.

m_TsId, m_OnwId
If *m_PsiSiReproc* is set to '1', these members specify the transport stream ID and original network ID of the transport stream in the PLP. A receiver will use these fields if it can't rely on the PSI/SI.

Remarks

Unsupported fields are set to **DTAPI_STAT_UNSUP_INTITEM**.

struct DtDvbC2DemodL1Part2DSlice

This structure specifies the DVB-C2 layer-1 part-2 signalling information for one data slice.

```
struct DtDvbC2DemodL1Part2DSlice
{
  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
  int  m_ConstConfig;         // Constant data slice configuration(yes/no)
  int  m_LeftNotch;           // Left notch present (yes/no)
  int  m_NumPlps;             // Number of PLPs
  std::<vector<DtDvbC2DemodL1Part2Plp> m_Plps;
 };
```

Members

m_Id
Unique identification of the data slice within a C2 system.

m_TunePosition
Tune position of the associated data slice relative to the start frequency of the C2 system, in multiples of pilot carrier spacing.

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.

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.

m_TiDepth
Time interleaving depth within the associated data slice.

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

m_Type
Data slice type.

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

m_FecHdrType
FEC frame header type.

Value	Meaning
DTAPI_DVBC2_FECHDR_TYPE_ROBUST	Robust mode
DTAPI_DVBC2_FECHDR_TYPE_HEM	High efficiency mode

m_ConstConfig
If '1', indicates that the configuration of the associated data slice shall not change; otherwise, the configuration is assumed to be variable.

m_LeftNotch
If '1', indicates the presence of a left neighboured notch band.

m_NumPlps
Specifies the number of physical layer pipes signalled in the layer 1 signalling part 2 data.

m_Plps
A vector specifying the DVB-C2 layer 1 signalling part 2 data for the physical layer pipes (not necessarily for each detected PLP).

Remarks

Unsupported fields are set to **DTAPI_STAT_UNSUP_INTITEM**.

struct DtDvbC2DemodL1Part2

This structure specifies the DVB-C2 layer-1 part-2 signalling information.

```
struct DtDvbC2DemodL1Part2
{
   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
   int  m_C2FrameLength;      // Number of symbols per C2 frame
   int  m_L1P2ChangeCtr;      // Value of L1_PART2_CHANGE_COUNTER field
   int  m_ReservedTone;       // Reserved tones present (yes/no)

   // Data-slice parameters
   int  m_NumDSlices;         // Number of data slices
   std::vector< DtDvbC2DemodL1Part2DSlice>  m_DSlices;
   // Notches
   int  m_NumNotches;         // Number of notches
   std::vector< DtDvbC2NotchPars>  m_Notches;
};
```

Members

m_NetworkId
Network ID. Unique identification of the DVB-C2 network.

m_C2SystemId
C2 system ID. Unique identification of a C2 system.

m_StartFrequency
Start frequency of the C2 system by means of the distance from 0Hz in multiples of the carrier spacing (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.

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_C2FrameLength
The number of data symbols per C2 frame.

m_L1P2ChangeCtr
The value of the L1_PART2_CHANGE_COUNTER field, indicating the number of C2 frames ahead where the configuration will change.

m_ReservedTone
If '1', indicates one or more reserved tones (carriers) are used.

m_NumDSlices
Specifies the number of data slices signalled in the layer-1 signalling part-2 data.

m_DSlices
A vector specifying the layer-1 signalling part-2 data for the data slices (not necessarily for each detected data slice).

m_NumNotches
Specifies the number of notch bands signalled in the layer-1 signalling part-2 data.

m_Notches
A vector specifying the layer-1 signalling part-2 data for the notches (not necessarily for each detected notch).

Remarks

Unsupported fields are set to **DTAPI_STAT_UNSUP_INTITEM**.

struct DtDvbC2NotchPars

This structure specifies a DVB-C2 notch band. It is used in class **DtDvbC2Pars** and in structure **DtDvbC2DemodL1Part2Data**.

```
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 the 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 the 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 DtDvbC2StreamSel

This structure specifies the criteria to select a PLP from a DVB-C2 stream.

```
struct DtDvbC2StreamSelPars
{
   int  m_DSliceId;           // Data slice ID
   int  m_PlpId;              // Data PLP ID
   int  m_CommonPlpId;        // Common PLP ID
};
```

Members

m_DSliceId

Unique identification of the data slice within the DVB-C2 stream. Valid values are: 0 ... 255 and **DTAPI_DVBC2_DSLICE_ID_AUTO**. The latter value specifies automatic selection of the data slice. In this case the first PLP is selected.

m_PlpId

Unique identification of the data PLP within the DVB-C2 stream. The valid range is 0 ... 255 and **DTAPI_DVBC2_PLP_ID_AUTO**. The latter value specifies automatic selection of the data PLP.

m_CommonPlpId

Unique identification of the common PLP within the DVB-C2 stream. It will be combined with the selected data PLP. The valid values are: 0 ... 255, **DTAPI_DVBC2_PLP_ID_NONE** and **DTAPI_DVBC2_PLP_ID_AUTO.**

The value **DTAPI_DVBC2_PLP_ID_NONE** indicates that no common PLP is selected. The value **DTAPI_DVBC2_PLP_ID_AUTO** indicates automatic selection of the common PLP.

struct DtDvbT2AuxPars

This structure specifies the DVB-T2 auxiliary stream information from the layer-1 post signalling data.

```
struct DtDvbT2AuxPars
{
   int  m_AuxStreamType;      // Auxiliary stream type
   int  m_AuxPrivateConfg;    // Auxiliary stream info
};
```

Members

m_AuxStreamType
Type of the current auxiliary stream.

m_AuxPrivateConfig
Field for future use for signalling auxiliary streams.

struct DtDvbT2DemodL1PostPlp

This structure specifies the DVB-T2 layer-1 post signalling information for one physical layer pipe (PLP).

```
struct DtDvbT2DemodL1PostPlp
{
  int  m_Id;                  // PLP ID: 0…255
  int  m_Type;                // PLP type
  int  m_PayloadType;         // PLP payload type
  int  m_FfFlag;              // FF flag
  int  m_FirstRfIdx;          // First TFS RF channel where PLP occurs
  int  m_FirstFrameIdx;       // First frame index
  int  m_GroupId;             // PLP group ID
  int  m_CodeRate;            // Code rate
  int  m_Modulation;          // Modulation type
  int  m_Rotation;            // Constellation rotation (yes/no)
  int  m_FecType;             // FEC type
  int  m_NumBlocks;           // Maximum number of FEC blocks per IL frame
  int  m_FrameInterval;       // T2-frame interval
  int  m_TimeIlLength;        // Time interleaving length
  int  m_TimeIlType;          // Time interleaving type
  int  m_InBandAFlag;         // In-band A signalling information (yes/no)
  int  m_InBandBFlag;         // In-band B signalling information (yes/no)
  int  m_Reserved1;           // Reserved field1
  int  m_PlpMode;             // PLP mode
  int  m_Static;              // Static configuration
  int  m_StaticPadding;       // Static padding
 };
```

Members

m_Id

Unique identification of the PLP within a T2 system.

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_GFPS	Generic Fixed-length Packetized Stream
DTAPI_DVBT2_PAYLOAD_GCS	Generic Continuous Stream
DTAPI_DVBT2_PAYLOAD_GSE	Generic Stream Encapsulation
DTAPI_DVBT2_PAYLOAD_TS	Transport Stream

m_FfFlag
FF flag. This parameter is only meaningful for type-1 PLPs in a time-frequency-slicing (TFS) configuration.

m_FirstRfIdx
First TFS RF channel. This parameter is only meaningful for type-1 PLPs in a time-frequency-slicing (TFS) configuration.

m_FirstFrameIdx
First frame index.

m_GroupId
Identifies the PLP group with which the PLP is associated.

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	4/5, 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 '1', constellation rotation is used, otherwise not.

m_FecType
FEC type used by the PLP.

Value	Meaning
DTAPI_DVBT2_LDPC_16K	16K LDPC
DTAPI_DVBT2_LDPC_64K	64K LDPC; not allowed in T2 lite

m_NumBlocks
Maximum number of FEC blocks for this PLP.

m_FrameInterval
T2 frame interval for this PLP.

m_TimeIlLength
Time-interleaving length.
If *m_TimeIlType* is set to '0', this parameter specifies the number of TI blocks per interleaving frame.
If *m_TimeIlType* is set to '1', this parameter specifies the number of T2 frames to which each interleaving frame is mapped.

m_TimeIlType
Type of time interleaving used by the PLP.

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

m_InBandAFlag
If '1', the in-band A flag is set and in-band signalling information is inserted in this PLP.

m_InBandBFlag
If '1', the in-band B flag is set and in-band signalling information is inserted in this PLP. This is only useful if DVB-T2 V1.2.1 is selected.

m_Reserved1
Field reserved for future use. It is sometimes used for bias balancing.

m_PlpMode
Mode used for the current PLP.

Value	Meaning
DTAPI_DVBT2_PLP_MODE_NONE	Not specified, used if DVB-T2 V1.1.1 is selected.
DTAPI_DVBT2_PLP_MODE_NORMAL	Normal Mode
DTAPI_DVBT2_PLP_MODE_HEM	High Efficiency Mode

m_Static
If '1', the layer 1 post signalling fields only changes at a superframe boundaries otherwise the fields may change at any time.

m_StaticPadding
If '1', BB-frame padding is not used, otherwise BB-frame padding may be used.

Remarks

Unsupported fields are set to **DTAPI_STAT_UNSUP_INTITEM**.

struct DtDvbT2RfPars

This structure specifies the DVB-T2 time-frequency-slicing (TFS) RF-channel information from the layer-1 post signalling data.

```
struct DtDvbT2RfPars
{
  int  m_RfIdx;              // Index of the RF-frequency
  int  m_Frequency;          // Centre frequency in Hz
};
```

Members

m_RfIdx
The index of this RF-frequency within the loop.

m_Frequency
The centre frequency in Hz of the RF channel.

struct DtDvbT2DemodL1Data

This structure specifies the DVB-T2 layer-1 post signalling data.

```
struct DtDvbT2DemodL1Data
{
  // P1 information
  struct DtDvbT2DemodL1P1
  {
      bool  m_Valid;        // P1 found
      int   m_FftMode;      // FFT mode (or size)
      int   m_Miso;         // MISO used
      int   m_Fef;          // FEF used
      int   m_T2Profile;    // DVB-T2 profile
  } m_P1;
  // L1-Pre information
  struct DtDvbT2DemodL1Pre
  {
      bool  m_Valid;        // L1-pre found
      int   m_Type;         // Stream type within the super frame
      int   m_BwtExt;       // Bandwidth extension
      int   m_S1;           // S1 field value
      int   m_S2;           // S2 field value
      int   m_L1Repetition; // L1 repetition (yes/no)
      int   m_GuardInterval;// Guard interval
      int   m_Papr;         // PAPR reduction mode
      int   m_L1Modulation; // L1 modulation type
      int   m_L1CodeRate;   // L1 code rate
      int   m_L1FecType;    // L1 FEC type
      int   m_L1PostSize;   // Size of the L1-post in OFDM cells
      int   m_L1PostInfoSize; // Size of the L1-post information
      int   m_Rotation;     // Constellation rotation (yes/no)
      int   m_PilotPatern;  // Pilot pattern
      int   m_TxIdAvailability; // The TX-id
      int   m_CellId;       // Cell ID
      int   m_NetworkId;    // Network ID
      int   m_T2SystemId;   // T2 system ID
      int   m_NumT2Frames;  // Number of T2 frames in a super frame
      int   m_NumDataSyms;  // Number of data OFDM symbols per T2-frame
      int   m_RegenFlag;    // Regeneration count indicator
      int   m_L1PostExt;    // L1-post extensions (yes/no)
      int   m_NumRfChans;   // Number of RF channels
      int   m_CurrentRfIdx; // Current RF channel index
      int   m_T2Version;    // DVB-T2 specification version
      int   m_L1PostScrambling; // L1-post scrambling
      int   m_T2BaseLite;   // T2-Lite is used in a base profile
  } m_L1Pre;
  // L1-Post information
  struct DtDvbT2DemodL1Post
  {
      bool  m_Valid;        // L1-post found
      int   m_NumPlps;      // Number of PLPs
      int   m_NumAux;       // Number of auxiliary streams
      // TFS RF-channel frequencies
      std::vector< DtDvbT2DemodRfPars>  m_RfChanFeqs;
      int   m_FefType;      // FEF type (if FEF is used)
```

```
        int  m_FefLength;     // FEF length (if FEF is used)
        int  m_FefInterval;   // FEF interval (if FEF is used)

        // PLPs
        std::vector< DtDvbT2DemodL1PostPlp>  m_Plps;
        // Auxiliary stream signalling information
        std::vector< DtDvbT2DemodAuxPars>  m_AuxPars;
    } m_L1Pre;
};
```

Public members

m_P1.m_Valid
If true, P1signalling data has been successfully decoded and is valid.

m_P1.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_P1.m_Miso
If '1', MISO is used otherwise SISO.

m_P1.m_Fef
If '1', FEF-parts are used.

m_P1.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_L1Pre.m_Valid
If true, L1-pre signalling data has been successfully decoded and is valid.

m_L1Pre.m_Type
Stream types carried within the current T2 superframe.

Value	Meaning
DTAPI_DVBT2_TYPE_TS	Transport Stream only
DTAPI_DVBT2_TYPE_GS	Generic Stream only
DTAPI_DVBT2_TYPE_TS_GS	Mixed Transport Stream and Generic Stream

m_L1Pre.m_BwtExt
If '1', the extended carrier mode is used.

m_L1Pre.m_S1
S1 field.

m_L1Pre.m_S2
S2 field.

m_L1Pre.m_L1Repetition
If '1', L1 signalling is provided for the next frame.

m_L1Pre.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_L1Pre.m_Papr
The peak to average power reduction method.
For DVB-T2 version 1.1.1:

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

For DVB-T2 version 1.2.1 and higher:

Value	Meaning
DTAPI_DVBT2_PAPR_NONE	L1-ACE is used and TR is used on P2 symbols only
DTAPI_DVBT2_PAPR_ACE	L1-ACE and ACE only are used
DTAPI_DVBT2_PAPR_TR	L1-ACE and TR only are used
DTAPI_DVBT2_PAPR_ACE_TR	L1-ACE, ACE and TR are used

m_L1Pre.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_L1Pre.m_L1CodeRate

Convolutional coding rate used for the L1-post signalling block.

Value	Meaning
DTAPI_DVBT2_COD_1_2	1/2

m_L1Pre.m_L1FecType

FEC type used for the L1-post signalling block.

Value	Meaning
DTAPI_DVBT2_LDPC_16K	16K LDPC

m_L1Pre.m_L1PostSize

Size of the coded and modulated L1-post signalling data block, in OFDM cells.

m_L1Pre.m_L1PostInfoSize

Size of the information part of the L1-post signalling data block, in bits, including the extension field, if present, but excluding the CRC.

m_L1Pre.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_L1Pre.m_TxIdAvailability

Field for signalling the availability of transmitter identification signals within the current geographic cell.

m_L1Pre.m_CellId

Cell ID. Unique identification of a geographic cell in a DVB-T2 network.

m_L1Pre.m_NetworkId
Network ID. Unique identification of the DVB-T2 network.

m_L1Pre.m_T2SystemId
T2 system ID. Unique identification of the T2 system.

m_L1Pre.m_NumT2Frames
The number of T2 frames in a super frame. The valid range is 1 … 255.

m_L1Pre.m_NumDataSyms
The number of data OFDM symbols per T2 frame, excluding P1 and P2.

m_L1Pre.m_RegenFlag
Indicates the number of times the DVB-T2 signal has been regenerated.

m_L1Pre.m_L1PostExt
If '1', the L1-post extension field is present.

m_L1Pre.m_NumRfChans
The number of frequencies in the T2 system.

m_L1Pre.m_CurrentRfIdx
The index of the current RF channel within its TFS structure.

m_L1Pre.m_T2Version
Version of the DVB-T2 specification.

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_L1Pre.m_L1PostScrambling
If '1', L1-post signalling is scrambled.

m_L1Pre.m_T2BaseLite
If '1', T2 lite is used in a base profile component.

m_L1Post.m_Valid
If '1', L1-post signalling data has been successfully decoded and is valid.

m_L1Post.m_NumSubslices
The number of subslices per T2 frame for type-2 PLPs.

m_L1Post.m_NumPlps
Indicates the number of physical layer pipes in the T2 system.

m_L1Post.m_NumAux
Indicates the number of auxiliary streams.

m_L1Post.m_RfChanFreqs
A vector specifying the TFS RF-channel frequencies.

m_L1Post.m_FefType
Specifies the FEF type, only if FEF is used (i.e. *m_L1P.m_Fef='1'*).

m_L1Post.m_FefLength
The length of a FEF-part in number of T-units (= samples), only if FEF is used (i.e. *m_L1P.m_Fef='1'*).

m_L1Post.m_FefInterval
The number of T2 frames between two FEF parts, only if FEF is used (i.e. *m_L1P.m_Fef='1'*).

m_L1Post.m_Plps
A vector specifying the DVB-T2 L1-post signalling data for the physical layer pipes (not necessarily for each detected PLP).

m_L1Post.m_AuxPars
A vector specifying the auxiliary signalling information.

Remarks

Unsupported fields are set to **DTAPI_STAT_UNSUP_INTITEM**.

struct DtDvbT2ParamInfo

This structure contains the DVB-T2 "derived" parameters which can be computed from the main DVB-T2 parameters. **`DtDvbT2Pars::GetParamInfo`** and **`DtDvbT2Pars::OptimisePlpNumBlocks`** set the members in this structure.

```
struct DtDvbT2ParamInfo
{
   int  m_TotalCellsPerFrame; // Total number of cells per frame
   int  m_L1CellsPerFrame;    // Total #L1 signalling cells per frame
   int  m_AuxCellsPerFrame;   // Total #auxiliary stream cells per frame
   int  m_BiasBalCellsPerFrame;
                              // Total number of L1 bias balancing cells
   int  m_BiasBalCellsMax;    // Maximum #L1 bias balancing cells per P2
   int  m_DummyCellsPerFrame; // Total number of cells lost per frame
   int  m_SamplesPerFrame;    // Total number of samples per frame
};
```

Members

m_TotalCellsPerFrame
Total number of cells per frame.

m_L1CellsPerFrame
Total number of cells per frame used for L1 signaling.
The overhead is *m_L1CellsPerFrame* / *m_TotalCellsPerFrame*.

m_AuxCellsPerFrame
Total number of auxiliary stream cells per frame. This member is currently only used for Tx signalling if it is enabled.

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. It is computed for the first frame in case no NDP is used.
The dummy cells overhead = *m_DummyCellsPerFrame* / *m_TotalCellsPerFrame*.

m_SamplesPerFrame
Total number of samples per frame.

struct DtDvbT2PlpPars

This structure specifies the DVB-T2 modulation parameters for one physical layer pipe (PLP). It is used in class **DtDvbT2Pars**, in an array of structures that specify the parameters for each PLP.

```
struct DtDvbT2PlpPars
{
  bool  m_Hem;                // High Efficiency Mode: yes/no
  bool  m_Npd;                // Null Packet Deletetion: yes/no
  int   m_Issy;               // ISSY mode. See DTAPI_DVBT2_ISSY_XXX
  int   m_IssyBufs;           // ISSY BUFS
  int   m_IssyTDesign;        // T-design value for TTO generation
  int   m_CompensatingDelay;  // Additional delay before TS data is sent
  int   m_TsRate;             // If 0 compute rate from PLP parameters
  int   m_Id;                 // PLP ID: 0…255
  int   m_GroupId;            // PLP group ID: 0…255
  int   m_Type;               // PLP type: DTAPI_DVBT2_PLP_TYPE_XXX
  int   m_CodeRate;           // PLP code rate: DTAPI_DVBT2_COD_XXX
  int   m_Modulation;         // PLP modulation: DTAPI_DVBT2_BPSK/…
  bool  m_Rotation;           // Constellation rotation: yes/no
  int   m_FecType;            // FEC type: 0=16K, 1=64K
  int   m_FrameInterval;      // T2-frame interval for this PLP: 1…255
  int   m_FirstFrameIdx;      // First frame index: 0…m_FrameInterval-1
  int   m_TimeIlLength;       // Time interleaving length: 0…255
  int   m_TimeIlType;         // Interleaving type: DTAPI_DVBT2_IL_XXX
  bool  m_InBandAFlag;        // In band A signaling information: yes/no
  bool  m_InBandBFlag;        // In band B signaling information: yes/no
  bool  m_NumBlocks;          // Number of FEC blocks per IL frame

  // IDs of the other PLPs in In Band Signaling
  int   m_NumOtherPlpInBand;  // Number of other PLPs in m_OtherPlpInBand
  int   m_OtherPlpInBand[DTAPI_DVBT2_NUM_PLP_MAX-1];

  // The parameters below are only meaningful for type 1 PLPs in TFS case
  bool  m_FfFlag;              // FF-flag
  int   m_FirstRfIdx;          // First TFS RF channel: 0…NumRf-1
};
```

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 (NPD) is active, otherwise it is not 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
Value of the ISSY BUFS parameter.

m_IssyTDesign
T-design value for TTO generation. Use 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 (*m_FirstFrameIdx*) and the first output bit of the transport stream.

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

m_TsRate
Transport stream rate. If 0, the rate is computed from the PLP parameters. This is only possible if no NPD is used.

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

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

m_Type
PLP type.

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

m_CodeRate
Convolutional coding rate used by the PLP.

Value	Meaning
DTAPI_DVBT2_COD_1_2	1/2
DTAPI_DVBT2_COD_3_5	3/5
DTAPI_DVBT2_COD_2_3	2/3
DTAPI_DVBT2_COD_3_4	3/4
DTAPI_DVBT2_COD_4_5	4/5, not for T2 lite
DTAPI_DVBT2_COD_5_6	4/5, 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, otherwise not.

m_FecType

FEC type used by the PLP.

Value	Meaning
DTAPI_DVBT2_LDPC_16K	16K LDPC
DTAPI_DVBT2_LDPC_64K	64K LDPC; not allowed in T2 lite

m_FrameInterval

T2-frame interval for this PLP. The valid range is 1 … 255.

m_FirstFrameIdx

First frame index. The valid range is 0 … *m_FrameInterval*-1.

m_TimeIlLength

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

m_TimeIlType

Type of time interleaving used by the PLP.

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

m_InBandAFlag

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

m_InBandBFlag

If true, the in-band B flag is set and in-band signalling information is inserted in this PLP. This is only useful if DVB-T2 V1.2.1 is selected.

m_NumBlocks

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

m_NumOtherPlpInBand
Number of other PLPs in the in-band signalling.

m_OtherPlpInBand[DTAPI_DVBT2_NUM_PLP_MAX-1]
IDs of the other PLPs in the in-band signalling.

m_FfFlag
FF flag.
This parameters is only meaningful for type-1 PLPs in the TFS case.

m_FirstRfIdx
First TFS RF channel. The valid range is 0 … Number of RF channels - 1.
This parameters is only meaningful for type-1 PLPs in the TFS case.

struct DtDvbT2StreamSel

This structure specifies the criteria to select a PLP from a DVB-T2 stream.

```
struct DtDvbC2StreamSelPars
{
   int  m_PlpId;               // Data PLP ID
   int  m_CommonPlpId;         // Common PLP ID
};
```

Members

m_PlpId

Unique identification of the data PLP within the DVB-T2 stream. The valid range is 0 … 255 and **DTAPI_DVBT2_PLP_ID_AUTO**. The value **DTAPI_DVBT2_PLP_ID_AUTO** specifies automatic selection of the PLP. In this case the first PLP is selected.

m_CommonPlpId

Unique identification of the common PLP within the DVB-T2 stream. It will be combined with the selected data physical layer pipe. The valid values for *m_CommonPlpId* are: 0 … 255, **DTAPI_DVBT2_PLP_ID_NONE** and **DTAPI_DVBT2_PLP_ID_AUTO**.
The value **DTAPI_DVBT2_PLP_ID_NONE** specifies that no common PLP is used. The value **DTAPI_DVBT2_PLP_ID_AUTO** specifies automatic selection of the common PLP.

struct DtHwFuncDesc

Structure describing a hardware function.

```
struct DtHwFuncDesc
{
  DtDeviceDesc  m_DvcDesc;    // Device descriptor
  int  m_ChanType;            // Channel type (OR-able)
  DtCaps  m_Flags;            // Capability flags (OR-able)
  int  m_IndexOnDvc;          // Relative index of hardware function
  int  m_Port;                // Physical port number (1…#ports)
  int  m_Ip[4];               // IP address (TS-over-IP functions only)
  int  m_MacAddr[6];          // MAC address (TS-over-IP functions only)
};
```

Members

m_DvcDesc

Device descriptor of the device that hosts this hardware function.

m_ChanType

This member variable identifies the channel type of the hardware function. Channel types **DTAPI_CHAN_INPUT** and **DTAPI_CHAN_OUTPUT** may be OR-ed together. The channel-object column identifies the channel object that can be attached to this hardware function for interaction with the hardware.

Channel types **DTAPI_CHAN_DBLBUF**, **DTAPI_CHAN_DISABLED** and **DTAPI_CHAN_LOOPTHR** do not have an associated channel object because no direct interaction is possible.

Value	Channel Object	Meaning
DTAPI_CHAN_DISABLED	n.a.	Channel is disabled
DTAPI_CHAN_INPUT	**DtInpChannel**	Input channel
DTAPI_CHAN_OUTPUT	**DtOutpChannel**	Output channel
DTAPI_CHAN_DBLBUF	n.a.	The hardware function is a double-buffered copy of another hardware function
DTAPI_CHAN_LOOPTHR	n.a.	The hardware function is a loop-through copy of another hardware function

For TS-over-IP channels both **DTAPI_CHAN_INPUT** and **DTAPI_CHAN_OUTPUT** are set.

On the DTA-2137 the hardware function for physical port #2 will be disabled if port #1 is configured for APSK operation. In the same way, the hardware function for port #1 is disabled when port #2 is configured for APSK operation.

m_Flags

Capability flags that provide further information about the hardware function.
The available capabilities are listed in DTAPI.h as **DTAPI_CAP_xxx**.

m_IndexOnDvc

This integer identifies a specific hardware function when the device hosts multiple hardware functions with the same channel type and stream type.
If the function occurs only once, *m_IndexOnDvc* = 0. If the device supports the function twice, indices are 0 and 1; etc.

m_Port
This integer identifies the physical port number associated with this function.
The general rule on PCI cards is that the top-most port is #1, the one below that #2, etc., with the following exceptions:
- The Ethernet port on the DTA-160 and the DTA-2160 is port #4;
- Double-buffered outputs like on the DTA-140 count as a single port.

m_Ip
IP address of the hardware function. This field is only valid for functions with capability **DTAPI_CAP_IP**.

m_Mac
MAC address of the hardware function. This field is only valid for functions with capability **DTAPI_CAP_IP**.

Remarks

This structure is used by ::**DtapiHwFuncScan** to return a description of a hardware function.

The channel type of bi-directional ASI/SDI ports is *either* **DTAPI_CHAN_INPUT** *or* **DTAPI_CHAN_OUTPUT**. Method **DtDevice::SetIoConfig** can be used to change the direction. The next time ::**DtapiHwFuncScan** is called, the channel type in the hardware-function descriptor will be updated to reflect the last-programmed direction.

struct DtIsdbsLayerPars

This structure specifies the ISDB-S modulation for one hierarchical layer. This structure is used in class **DtIsdbsPars**, in an array of four structures for layer 1…4.

```
struct DtIsdbsLayerPars
{
   int  m_NumSlots;            // Number of slots
   int  m_ModCod;              // Modulation method and code rate
};
```

Members

m_NumSlots

The number of slots per frame used for this hierarchical layer.

m_ModCod

Modulation type used for this hierarchical layer.

Value	Meaning
DTAPI_ISDBS_MODCOD_ BPSK_1_2	BPSK 1/2
DTAPI_ISDBS_MODCOD_ QPSK_1_2	QPSK 1/2
DTAPI_ISDBS_MODCOD_ QPSK_2_3	QPSK 2/3
DTAPI_ISDBS_MODCOD_ QPSK_3_4	QPSK 3/4
DTAPI_ISDBS_MODCOD_ QPSK_5_6	QPSK 5/6
DTAPI_ISDBS_MODCOD_ QPSK_7_8	QPSK 7/8
DTAPI_ISDBS_MODCOD_ 8PSK_2_3	8PSK 2/3
DTAPI_ISDBS_MODCOD_ NOT_ALLOC	This layer is not used

struct DtIsdbtLayerPars

This structure specifies the ISDB-T modulation parameters for one hierarchical layer. It is used in class **DtIsdbtPars**, in an array of three structures for layer A, B and C.

```
struct DtIsdbtLayerPars
{
   int  m_NumSegments;         // Number of segments
   int  m_Modulation;          // Modulation type
   int  m_CodeRate;            // Code rate
   int  m_TimeInterleave;      // Time interleaving
   // Derived:
   int  m_BitRate;             // Bit rate in bps
};
```

Members

m_NumSegments

Number of segments used in this layer.

m_Modulation

Modulation type applied to the segments in this layer.

Value	Meaning
DTAPI_ISDBT_MOD_DQPSK	DQPSK
DTAPI_ISDBT_MOD_QPSK	QPSK
DTAPI_ISDBT_MOD_QAM16	16-QAM
DTAPI_ISDBT_MOD_QAM64	64-QAM

m_CodeRate

Convolutional coding rate applied to the segments in this layer.

Value	Meaning
DTAPI_ISDBT_RATE_1_2	1/2
DTAPI_ISDBT_RATE_2_3	2/3
DTAPI_ISDBT_RATE_3_4	3/4
DTAPI_ISDBT_RATE_5_6	5/6
DTAPI_ISDBT_RATE_7_8	7/8

m_TimeInterleave

Encoded length of time interleaving.

The table below defines the mapping of *m_TimeInterleave* to parameter I in the time-interleaving process.

Value	Mode 1	Mode 2	Mode 3
0	0	0	0
1	4	2	1
2	8	4	2
3	16	8	4

m_BitRate

Bit rate in bits-per-second, assuming this is a 6-MHz channel. This is a "derived" parameter, which is set to a value by calling **DtIsdbtPars::ComputeRates**.

Remarks

The ISDB-T modulator uses the sum of *m_NumSegments* over layer A/B/C to set the total number of segments. This enables the usage of broadcast type **BTYPE_TV** for 1-segment modulation.

struct DtRawIpHeader

Structure placed in front of all IP Packets in receive mode **DTAPI_RXMODE_IPRAW**.

```
struct DtRawIpHeader
{
  unsigned short  m_Tag;       // 0x44A0h = 'D'160
  unsigned short  m_Length;    // IP packet length
  unsigned int  m_TimeStamp; // IP packet arrival/transmit timestamp
};
```

Members

m_Tag

16-bit tag that marks the beginning of a **DtRawIpHeader** structure. The value of this field is fixed to: 0x44A0.

m_Length

16-bit integer that indicates the number of bytes (i.e. size of the IP packet) following directly after the **DtRawIpHeader** structure.

m_TimeStamp

A 32-bit timestamp, taken from the internal 54-MHz system clock on the device, indicating the arrival time of the IP packet following this structure.

struct DtStatistic

This structure represents a single statistic. It is used for modulators and receivers to pass measurements and statistics information.

```
struct DtStatistic
{
  DTAPI_RESULT  m_Result;      // Result of retrieving the statistic
  int  m_StatisticId;          // DTAPI_STAT_XXX
  int  m_ValueType;            // Value types supported for statistics
  union {
    bool  m_ValueBool;         // Value if value type is STAT_VT_BOOL
    double  m_ValueDouble;     // Value if value type is STAT_VT_DOUBLE
    int  m_ValueInt;           // Value if value type is STAT_VT_INT
    void*  m_pValue;           // Pointer if value type is
                               // STAT_VT_DVBC2_L1P2, STAT_VT_DVBC2_PLPSIG
                               // or STAT_VT_DVBT2_L1
  };
};
```

Members

m_Result

If the user queries one or more measurements with **DtInpChannel::GetStatistics**, this member is used to return the result code for retrieving the measurement from the receiver hardware.

m_StatisticId

Identifies the statistic:

Statistic	Type	Meaning	Supported by
DTAPI_STAT_AGC1	int	Returns the AGC1 value	2135, 2136, 2137, 2139
DTAPI_STAT_AGC2	int	Returns the AGC2 value	2135, 2136, 2137, 2138, 2139
DTAPI_STAT_BADPCKCNT	int	Count of uncorrected packets since last call. Returns 0 if receiver not locked.	2135, 2136, 2139, 235
DTAPI_STAT_BER_BCH	double	Outer BCH bit error rate	2137, 2138
DTAPI_STAT_BER_LDPC	double	Inner LDPC bit error rate	2137, 2138
DTAPI_STAT_BER_POSTVIT	double	Post-Viterbi bit error rate	2135, 2137, 235
DTAPI_STAT_BER_PRERS	double	Pre-Reed-Solomon bit error rate	2135, 2136, 2137, 2138, 235
DTAPI_STAT_BER_PREVIT	double	Pre-Viterbi bit error rate	2135, 2137, 2138, 235
DTAPI_STAT_CARRIER_LOCK	bool	Overall lock status	2136, 2138, 2139, 236
DTAPI_STAT_CNR	int / double	Carrier-over-noise ratio in units of	2135, 2137,

DTAPI_STAT_SNR		0.1dB (int) or dB (double).	2138, 235, 236
DTAPI_STAT_DVBC2_L1P2DATA	struct	DVB-C2 layer-1 part 2 signalling data	2138
DTAPI_STAT_DVBC2_PLPSIGDATA	struct	DVB-C2 layer-1 PLP signalling data	2138
DTAPI_STAT_DVBT2_L1DATA	struct	DVB-T2 layer-1 signalling data	2138
DTAPI_STAT_FEC_LOCK	bool	FEC decoding lock status	2135, 2137, 235
DTAPI_STAT_LOCK	bool	Overall lock status	All receivers
DTAPI_STAT_MER	int / double	MER in units of 0.1dB (int) or dB (double)	2135, 2136, 2137, 2138, 235
DTAPI_STAT_PACKET_LOCK	bool	Packet-level lock status	2137
DTAPI_STAT_PER	double	Packet-error rate	2138
DTAPI_STAT_RFLVL_CHAN	int / double	RF level for channel bandwidth in units of 0.1dBmV (int) or dBmV (double)	2131, 2135, 2136, 2137, 2138, 2139, 234, 235, 236
DTAPI_STAT_RFLVL_NARROW	int / double	RF level for a narrow bandwidth in units of 0.1dBmV (int) or dBmV (double)	234, 235, 236
DTAPI_STAT_RFLVL_UNCALIB	int / double	Uncorrected RF level reported by the tuner in units of 0.1dBmV (int) or dBmV (double)	2131, 2137, 2139
DTAPI_STAT_RS	int	Reed-Solomon error count. Set to -1 if receiver is not locked **Note**: See remarks section for usage instructions	2137
DTAPI_STAT_SPECTRUMINV	bool	Spectrum inversion	2137, 2138
DTAPI_STAT_TEMP_TUNER	int	Tuner temperature	2131, 2139
DTAPI_STAT_VIT_LOCK	bool	Viterbi lock status	2135, 2137, 235
DTAPI_STAT_VIT_LOCK	bool	Viterbi lock status	2135, 2137, 235

m_ValueType

Identifies the type of the value according to the following table:

Value	Meaning
STAT_VT_BOOL	The value type is bool
STAT_VT_DOUBLE	The value type is double
STAT_VT_DVBC2_L1P2	The value type is pointer to **DtDvbC2DemodL1Part2Data**
STAT_VT_DVBC2_PLPSIG	The value type is pointer to **DtDvbC2DemodL1PlpSigData**
STAT_VT_DVBT2_L1	The value type is pointer to **DtDvbT2DemodL1Data**

STAT_VT_INT	The value type is int
STAT_VT_UNDEFINED	The value is not valid yet

m_ValueBool, m_ValueDouble, m_ValueInt, m_pValue

The value of the statistic. *m_ValueType* determines which variable is used.

Remarks

The **DTAPI_STAT_RS** statistic is available only if the hardware has been put in a special mode using the **SetErrorStats** method. Refer to **DtInpChannel::SetErrorStats** for a full description.

struct DtTransmitter

This structure describes a transmitter. Used by **DtInpChannel::BlindScan** to return the transmitters found by scanning a frequency band.

```
struct DtTransmitter
{
   __int64  m_FreqHz;          // Center frequency of the transmitter
   int  m_ModType;             // Modulation type
   int  m_SymbolRate;          // Symbol rate
};
```

Members

FreqHz

Centre frequency of the transmitter in Hertz.

ModType

Type of modulation used in the transmitted signal. See **DtInpChannel::GetDemodControl**, section Modulation Types, for a list of applicable values.

SymbolRate

The symbol rate of the transmitted signal.

struct DtTsIpPars

Structure for specifying or reading back parameters related to the transmission of TS-over-IP streams.

The parameters in this structure are used when starting transmission or reception of a transport stream over IP to or from a unicast or multicast destination.

```
struct DtTsIpPars
{
   u_char  m_Ip[4];            // IP address
   u_short  m_Port;            // Port number
   u_char  m_SrcFltIp[4];      // Source filter: IP address
   u_short  m_SrcFltPort;      // Source filter: port number
   int  m_TimeToLive;          // Time-to-live (TTL) setting for IP Tx
   int  m_NumTpPerIp;          // #TPs per IP packet
   int  m_Protocol;            // Protocol: DTAPI_PROTO_UDP/RTP
   int  m_DiffServ;            // Differentiated services
   int  m_FecMode;             // Error correction mode
   int  m_FecNumRows;          // 'D' = #rows in FEC matrix
   int  m_FecNumCols;          // 'L' = #columns in FEC matrix
   int  m_Flags;               // Optional control/status flags
};
```

Members (for Rx)

m_Ip
IP address from which to receive IP packets. If the IP address is in the multicast range, DTAPI automatically joins and drops membership of the multicast group.

m_Port
Port number on which to receive IP packets. Destination port number 0 is not allowed.
When the protocol is RTP, the port number shall be even.

m_SrcFltIp
Source-specific multicast: source IP address.
Relevant for multicast reception only: In this case *m_SrcFltIp* can be set to a specific IP address for listening to a single source. Set to 0.0.0.0 if the source may be any IP address.

m_SrcFltPort
Source-specific multicast: source: port number associated with *m_SrcFltIp*.
m_SrcPort may be set to a specific source port number, or to 0 for accepting IP packets from any source port.

m_TimeToLive
Not used.

m_NumTpPerTp
Not used.
Read back: Number of Transport Packets (TPs) stored in one IP packet in the incoming stream.

m_Protocol
Protocol expected for the encapsulation of Transport Packets.
Write: Only automatic detection (value **DTAPI_PROTO_AUTO)** is allowed.

Read back: set to the protocol that has been detected.

Value	Meaning
DTAPI_PROTO_UDP	UDP
DTAPI_PROTO_RTP	RTP
DTAPI_PROTO_AUTO	Automatically detect UDP or RTP

m_DiffServ
Not used.

m_FecMode
Error-correction mode.

Value	Meaning
DTAPI_FEC_DISABLE	Don't try to apply error correction
DTAPI_FEC_2D	Try to apply error correction with the FEC streams received on port numbers *m_Port*+2 and *m_Port*+4

m_FecNumRows, m_FecNumCols
Not used.

m_Flags
Optional control/status flags field. Currently this field is not used and should be set to zero.

Members (for Tx)

m_Ip, m_Port
IP address used for transmission, specified as 4 bytes, and associated destination port number. When the protocol is RTP, the port number shall be even.

m_SrcFltIp, m_SrcFltPort
Not used for specifying Tx parameters.
Read back: The host's IP address is stored in *m_SrcFltIp* and the selected source port number in *m_SrcFltPort*.

m_TimeToLive
Time-To-Live (TTL) value to be used for transmission. When *m_TimeToLive* is 0, a default value is used.

m_NumTpPerIp
Number of Transport Packets (TPs) stored in one IP packet. The valid range is 1…7.

m_Protocol
Protocol expected for encapsulation of Transport Packets.

Value	Meaning
DTAPI_PROTO_UDP	UDP
DTAPI_PROTO_RTP	RTP

m_DiffServ
Value to be put in the *Differentiated Services* field (formerly *Service Type*) in the IP header.

m_FecMode

Error-correction mode.

Value	Meaning
DTAPI_FEC_DISABLE	No FEC
DTAPI_FEC_2D	RFC2733 parity FEC with 2D extensions as described in *Code of Practice #3*

m_FecNumRows, m_FecNumCols

Number of rows and columns in the FEC matrix. In the *COP #3* these parameters are called *D* and *L* respectively. The following restrictions apply to *L* and *D*:

$4 \leq D \leq 20, \quad 1 \leq L \leq 20 \quad \text{and} \quad L*D \leq 100$

m_Flags

Optional control/status flags field. Currently this field is not used and should be set to zero.

struct DtTsIpStat

Structure for retrieving IP statistics from drivers. The counters will be reset after read.

```
struct DtTsIpStat
{
   unsigned int  m_TotNumIpPackets;
   unsigned int  m_LostIpPacketsBeforeFec;
   unsigned int  m_LostIpPacketsAfterFec;
   unsigned int  m_LostIpPacketsBeforeSort;
   unsigned int  m_LostIpPacketsAfterSort;
};
```

Members

m_TotNumIpPackets
Total number of IP packets received.

m_LostIpPacketsBeforeFec
Number of IP packets lost before FEC.

m_LostIpPacketsAfterFec
Number of IP packets lost after FEC.

m_LostIpPacketsBeforeSort
Number of IP packets lost before RTP sorting.

m_LostIpPacketsAfterSort
Number of IP packets lost after RTP sorting.

DtCmmbPars

Structure describing parameters for CMMB modulation.

```
struct DtCmmbPars
{
  int  m_Bandwidth;          // CMMB channel bandwidth
  int  m_TsRate;             // CMMB TS rate in bps
  int  m_TsPid;              // PID of the CMMB stream.
  int  m_AreaId;             // Area ID
  int  m_TxId;               // Transmitter ID
};
```

Public members

m_Bandwidth
The bandwidth of the channel.

Value	Meaning
DTAPI_CMMB_BW_2MHZ	2 MHz
DTAPI_CMMB_BW_8MHZ	8 MHz

m_TsRate
The rate in bits per second of the input transport stream.

m_TsPid
The PID of the CMMB stream in the transport stream.

m_AreaId
The area ID. The valid range is 0 … 127.

m_TxId
The transmitter ID. The valid range is 0 … 127.

Remarks

If the CMMB modulation is selected, the data written to the Transmit FIFO shall be in the format of CMMB PMS data packets.

DtCmmbPars::RetrieveTsRateFromTs

Retrieve the TS rate from a 188-byte transport stream with CMMB PMS data packets and store the results in the **DtCmmbPars** object calling this function.

```
DTAPI_RESULT DtCmmbPars::RetrieveTsRateFromTs(
  [in] char*  pBuffer,            // Buffer with transport stream
  [in] int    NumBytes,           // Number of bytes in buffer
);
```

Function Arguments

pBuffer

Buffer containing CMMB PMS data packets from which to retrieve the TS rate.

NumBytes

Number of transport-stream bytes in the buffer (at least 3MB).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	TS rate has be retrieved successfully
DTAPI_E_INSUF_LOAD	The buffer contains insufficient data to determine the TS rate.
DTAPI_E_INVALID_TSTYPE	The buffer does not contain a transport stream consisting of CMMB PMS data packets.

Remarks

DtDemodPars

The **DtDemodPars** class encapsulates demodulation parameters.

```
class DtDemodPars;
```

DtDemodPars::CheckValidty

Check demodulation parameters for validity.

```
DTAPI_RESULT DtDemodPars::CheckValidity(
);
```

Function Arguments

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The demodulation parameters have been set successfully
DTAPI_E_INVALID_BANDWIDTH	Invalid value for bandwidth field
DTAPI_E_INVALID_CONSTEL	Invalid value for constellation field
DTAPI_E_INVALID_GUARD	Invalid value for guard-interval field
DTAPI_E_INVALID_INTERLVNG	Invalid value for interleaving field
DTAPI_E_INVALID_J83ANNEX	Invalid value for J.83 annex
DTAPI_E_INVALID_MODPARS	Invalid demodulation parameters
DTAPI_E_INVALID_PILOTS	Invalid value for pilots
DTAPI_E_INVALID_RATE	Invalid value for convolutional rate or FEC code rate
DTAPI_E_INVALID_SYMRATE	Invalid value for symbol rate
DTAPI_E_INVALID_T2PROFILE	Invalid value for DVB-T2 profile
DTAPI_E_INVALID_TRANSMODE	Invalid value for transmission-mode field

Remarks

DtDemodPars::GetModType

Returns the modulation type. The value **DTAPI_MOD_TYPE_UNK** indicates that no valid modulation type has been set.

```
int DtDemodPars::GetModType(
);
```

Function Arguments

Result

Remarks

DtDemodPars::SetModType

Initializes the **DtDemodPars** object for the specified modulation type.

```
DTAPI_RESULT DtDemodPars::SetModType(
  [in] int  ModType,              // Modulation type
);
```

Function Arguments

ModType

Modulation type for which the **DtDemodPars** object is initialized.

Value	Meaning	Encapsulated demodulation parameters	Access method in DtDemodPars
DTAPI_MOD_ATSC	ATSC VSB	DtDemodParsAtsc	Atsc()
DTAPI_MOD_DVBC2	DVB-C2	DtDemodParsDvbC2	DvbC2()
DTAPI_MOD_DVBS_QPSK	DVB-S, QPSK	DtDemodParsDvbS	DvbS()
DTAPI_MOD_DVBS2_8PSK	DVB-S.2, 8PSK	DtDemodParsDvbS2	DvbS2()
DTAPI_MOD_DVBS2_16APSK	DVB-S.2, 16APSK		
DTAPI_MOD_DVBS2_32APSK	DVB-S.2, 32APSK		
DTAPI_MOD_DVBS2_QPSK	DVB-S.2, QPSK		
DTAPI_MOD_DVBT	DVB-T	DtDemodParsDvbT	DvbT()
DTAPI_MOD_DVBT2	DVB-T2	DtDemodParsDvbT2	DvbT2()
DTAPI_MOD_IQDIRECT	IQ-Samples	DtDemodParsIq	Iq()
DTAPI_MOD_QAM16	16-QAM	DtDemodParsQam	Qam()
DTAPI_MOD_QAM32	32-QAM		
DTAPI_MOD_QAM64	64-QAM		
DTAPI_MOD_QAM128	128-QAM		
DTAPI_MOD_QAM256	256-QAM		
DTAPI_MOD_QAM_AUTO	Autodetect QAM		

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The modulation type has been set successfully
DTAPI_E_INVALID_MODTYPE	Invalid modulation type

Remarks

IDtDemodEvent

Call back interface for demodulation controller events (e.g. tuning frequency has changed).

IDtDemodEvent::TuningParsHaveChanged

This event method is called when the tuning frequency has been changed.

```
void IDtDemodEvent::TuningParsHaveChanged(
  [in] __int64  OldFreqHz     // Old tuning frequency in hertz
  [in] int      OldModType    // Old modulation type
  [in] int      OldParXtra[3] // Old extra modulation parameters
  [in] __int64  NewFreqHz     // New tuning frequency in hertz
  [in] int      NewModType    // New modulation type
  [in] int      NewParXtra[3] // New extra modulation parameters

);
```

Function Arguments

OldFreqHz

The old tuning frequency (in Hz). A value of -1 indicates no valid frequency was previously set.

OldModType, OldParXtra[3]

Old modulation parameters. A value of -1 indicates no previous modulation parameters have been set. Refer to **DtInpChannel::SetDemodControl** for the possible values of these parameters

NewFreqHz

The new tuning frequency (in Hz).

NewModType, NewParXtra[3]

New modulation parameters. Refer to **DtInpChannel::SetDemodControl** for the possible values of these parameters

IDtDemodEvent::TuningFreqHasChanged

This event method is called when the tuning frequency has been changed.

```
void IDtDemodEvent::TuningFreqHasChanged(
  [in] __int64  OldFreqHz    // Old tuning frequency in hertz
  [in] __int64  NewFreqHz    // New tuning frequency in hertz
);
```

Function Arguments

OldFreqHz

The old tuning frequency (in Hz). A value of -1 indicates no valid frequency was previously set.

NewFreqHz

The new tuning frequency (in Hz).

DtIsdbsPars

Structure describing parameters for ISDB-S modulation.

```
struct DtIsdbsPars
{
  bool  m_DoMux;              // Hierarchical multiplexing yes/no
  int   m_Emergency;          // Switch-on control for emergency broadcast
  int   m_RelTs2TsId[8];      // TS# to TS-ID mapping
  int   m_Slot2RelTs[48];     // Slot# to TS# mapping
  DtIsdbsLayerPars  m_LayerPars[4];
};
```

Members

m_DoMux
If true, perform hierarchical multiplexing in accordance with the ISDB-S parameters as defined explicitly in this class.
If false, the ISDB-S modulation parameters are specified indirectly by the TMCC information in the transport stream. The current DTAPI supports "raw" mode only: 188-byte transport packets with TMCC encoded in the SYNC bytes of the packets.

m_Emergency
Flag that indicates whether the switch-on control flag for emergency broadcast should be turned on: 0 = off, 1 = on.

m_RelTs2TsId[8]
The transport-stream identifier (TSID) for transport stream i. In the current version of the DTAPI just a single TS is supported (only *m_RelTs2TsId[0]* is relevant).

m_Slot2RelTs[8]
For each of the 48 TMCC slots, this array specifies the index of the transport stream. The current version of the DTAPI supports a single TS only, and therefor this array is not used (each slot contains the one and only TS).

m_LayerPars
Modulation parameters for hierarchical layers 1 (element 0) … 4 (element 3).

Remarks

DtIsdbsPars::CheckValidity

Check ISDB-S parameters for validity.

```
DTAPI_RESULT DtIsdbsPars::CheckValidity(
);
```

Function Arguments

Result

DTAPI_RESULT	Meaning
DTAPI_OK	ISDB-S modulation parameters are valid
DTAPI_E_INVALID_PARS	ISDB-T parameters are invalid

Remarks

DtIsdbtPars

Structure describing parameters for ISDB-T modulation.

```
struct DtIsdbtPars
{
  bool  m_DoMux;              // Hierarchical multiplexing yes/no
  bool  m_FilledOut;          // Members have been given a value
  int   m_ParXtra0;           // #Segments, bw, sample rate, sub-channel
  int   m_BType;              // Broadcast type
  int   m_Mode;               // Transmission mode
  int   m_Guard;              // Guard interval
  int   m_PartialRx;          // Partial reception
  int   m_Emergency;          // Switch-on control for emergency broadcast
  int   m_IipPid;             // PID used for multiplexing IIP packet
  DtIsdbtLayerPars  m_LayerPars[3];    // Layer-A/B/C parameters
  std::map<int, int>  m_Pid2Layer;     // PID-to-layer map
  int   m_LayerOther;         // Other PIDs are mapped to this layer
  int   m_Virtual13Segm;      // Virtual 13-segment mode
  // Derived:
  bool  m_Valid;              // The parameter set is valid
  int   m_TotalBitrate;       // Bitrate of entire stream
};
```

Public Members

m_DoMux

If true, perform hierarchical multiplexing in accordance with the ISDB-T parameters as defined explicitly in this class.
If false, the ISDB-T modulation parameters are specified indirectly by the TMCC information in the 16 extra bytes of the 204-byte packets.

m_FilledOut

This member has significance only if hierarchical multiplexing is on. In that case it indicates whether member variables *m_BType*, *m_Mode*, ... up to and including *m_LayerOther* have been given a value.
Method **RetrieveParsFromTs** will set *m_FilledOut* to true if it has succeeded in finding a valid set of parameters in the transport stream. Alternatively, an application can set *m_FilledOut* to true itself if it has filled out the ISDB-T parameters in the **DtIsdbtPars** object.

m_ParXtra0

Extra parameter encoding bandwidth, sample rate and number of segments. This parameter is encoded like ParXtra0 in **SetModControl** with *ModType* **DTAPI_MOD_ISDBT**.

m_BType

Broadcast type.

Value	Meaning
DTAPI_ISDBT_BTYPE_TV	TV broadcast; Can be used with any number of segments
DTAPI_ISDBT_BTYPE_RAD1	1-segment radio broadcast; Total #segments must be 1
DTAPI_ISDBT_BTYPE_RAD3	3-segment radio broadcast; Total #segments must be 3

m_Mode

Transmission mode.

Value	Meaning
1	Mode 1: 2k
2	Mode 2: 4k
3	Mode 3: 8k

m_Guard

Guard-interval length.

Value	Meaning
DTAPI_ISDBT_GUARD_1_32	1/32
DTAPI_ISDBT_GUARD_1_16	1/16
DTAPI_ISDBT_GUARD_1_8	1/8
DTAPI_ISDBT_GUARD_1_4	1/4

m_PartialRx

Flag that indicates whether layer A is used for partial reception: 0 = no partial reception, 1 = partial reception on.

m_Emergency

Flag that indicates whether the switch-on control flag for emergency broadcast should be turned on: 0 = off, 1 = on.

m_IipPid

PID value used for multiplexing the IIP packet.

m_LayerPars

Modulation parameters for hierarchical layers A (element 0), B (1) and C (2).

m_Pid2Layer

Map that specifies the hierarchical layer, or layers, to which an elementary stream is to be mapped. The key in the map is the PID of the elementary stream. The value stored in the map is an OR of one or more flags listed in the table below. A value of 0 indicates that the elementary stream is to be dropped.

Value	Meaning
DTAPI_ISDBT_LAYER_A	Map elementary stream to layer A
DTAPI_ISDBT_LAYER_B	Map elementary stream to layer B
DTAPI_ISDBT_LAYER_C	Map elementary stream to layer C

m_LayerOther
Map streams with PIDs not in *m_Pid2Layer* to this layer.

m_Valid
The ISDB-T parameter set is valid. This is a "derived" parameter, which is set to a value by **DtIsdbtPars::CheckValidity**.

m_Virtual13Segm
Use virtual 13 segment mode. The number of segments in layer B is "faked" to be 12.

m_TotalBitrate
Bitrate in bps of the entire stream. The bitrate includes the 16 dummy bytes per packet that contain the ISDB-T information.

Remarks

DtIsdbtPars::CheckValidity

Check ISDB-T parameters for validity. A boolean result (valid/not valid) is stored in the invoking object, in flag *m_Valid*.

```
DTAPI_RESULT DtIsdbtPars::CheckValidity(
  [out] int&  ResultCode          // Result of validity check
);
```

Function Arguments

ResultCode

Value	Meaning
DTAPI_ISDBT_OK	ISDB-T parameters are valid
DTAPI_ISDBT_E_BANDWIDTH	Illegal value for bandwidth field in *m_ParXtra0*
DTAPI_ISDBT_E_BTYPE	Illegal value for broadcast type (*m_BType*)
DTAPI_ISDBT_E_GUARD	Illegal value for guard-interval length (*m_Guard*)
DTAPI_ISDBT_E_MODE	Illegal value for transmission mode (*m_Mode*)
DTAPI_ISDBT_E_NSEGM	Number of segments is not equal to 1, 3 or 13, or number of segments is invalid for the current broadcast type, or number of segments encoded in *m_ParXtra0* does not match number of segments specified in *m_LayerPars*
DTAPI_ISDBT_E_NOTFILLED	Member *m_FilledOut* is false, indicating that not all ISDB-T parameters have been initialized
DTAPI_ISDBT_E_PARTIAL	'Partial Reception' is selected but number of segments in layer A is not 1
DTAPI_ISDBT_E_SRATE	Illegal value for sample rate field in *m_ParXtra0*
DTAPI_ISDBT_E_SUBCHANNEL	Invalid sub-channel number. For 1-segment radio: 1 ≤ sub channel ≤ 40 For 1-segment radio: 4 ≤ sub channel ≤ 37

Result

DTAPI_RESULT	Meaning
DTAPI_OK	ISDB-T modulation parameters have been found valid
DTAPI_E_INVALID_PARS	ISDB-T parameters are invalid. *ResultCode* is set to a value indicating the reason why the ISDB-T parameters are not valid

Remarks

This routine assumes that **DtIsdbtPars::ComputeRates** has been called so that the rate variables in the **DtIsdbtPars** object have been set to the correct value.

DtIsdbtPars::ComputeRates

Compute the bit rate per hierarchical layer and store the results in the object calling this function.

```
DTAPI_RESULT DtIsdbtPars::ComputeRates();
```

Function Arguments

Result

DTAPI_RESULT	Meaning
DTAPI_OK	ISDB-T rates have been computed successfully
DTAPI_E_INVALID_ARG	One of the ISDB-T parameters, or the combination of parameters is invalid.

Remarks

DtIsdbtPars::RetrieveParsFromTs

Retrieve modulation parameters from a 204-byte transport stream with TMCC information and store the results in the **DtIsdbtPars** object calling this function.

```
DTAPI_RESULT DtIsdbtPars::RetrieveParsFromTs(
  [in] char*  pBuffer,            // Buffer with transport stream
  [in] int    NumBytes,           // Number of bytes in buffer
);
```

Function Arguments

pBuffer

Buffer containing transport-stream packets from which to retrieve the ISDB-T parameters.

NumBytes

Number of transport-stream bytes in the buffer.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	ISDB-T modulation parameters have been recovered successfully
DTAPI_E_INVALID_TSTYPE	The buffer does not contain a transport stream consisting of 204-byte packets with TMCC information
DTAPI_E_INSUF_LOAD	The buffer contains insufficient data to recover all ISB-T modulation parameters

Remarks

DtSdi

The **DtSdi** class contains helper methods for processing SDI data.

```
class DtSdi;
```

DtSdi::ConvertFrame

This method can be used to convert an SDI frame from one data format to another, e.g. from 10-bit uncompressed to Huffman compressed.

```
DTAPI_RESULT DtSdi::ConvertFrame(
     [in] unsigned int*  pInFrame,     // Buffer with input frame
 [in/out] int&  InFrameSize,           // [in] Size of input frame
                                       // [out] Number of bytes used
     [in] int  InFrameFormat,          // Format of input frame
     [in] unsigned int*  pOutFrame,    // Buffer for output frame
 [in/out] int&  OutFrameSize,          // [in] Size of output frame
                                       // [out] Number of bytes returned
     [in] int  OutFrameFormat          // Format of output frame
);
```

Function Arguments

pInFrame

Buffer containing the frame to be converted. The buffer address shall be 32-bit aligned.

InFrameSize

As an input argument *InFrameSize* indicates the number of bytes in the input frame buffer. The input buffer should comprise at least one complete frame, including any stuff-bytes required to achieve 32-bit alignment. Furthermore, *InFrameSize* must be a multiple of 4.

As an output argument *InFrameSize* indicates how many bytes of the input frame buffer have been used.

InFrameFormat

Specifies the format of the frame-data in the input frame buffer.

Value	Meaning
DTAPI_SDI_FULL	Complete SDI frame, including SAV/ EAV, horizontal and vertical blanking periods
DTAPI_SDI_ACTVID	Only the active video part of the SDI frame

The format can optionally be combined (OR-ed) with the following flags:

Value	Meaning
DTAPI_SDI_HUFFMAN	The frame is compressed with lossless Huffman compression
DTAPI_SDI_625	The frame contains 625 lines
DTAPI_SDI_525	The frame contains 525 lines
DTAPI_SDI_8B	8-bit data samples: every 32-bit word contains four 8-bit samples
DTAPI_SDI_10B	Packed 10-bit samples: eight 10-bit samples are encoded in ten bytes
DTAPI_SDI_16B	One 10-bit sample per 16-bit word. Every 32-bit word in the frame buffer contains two 10-bit samples

pOutFrame

Buffer to receive the converted frame. The buffer address shall be 32-bit aligned

OutFrameSize

As an input argument *OutFrameSize* indicates the size of the output frame buffer. The output buffer should be large enough to receive one complete frame and should be 32-bit aligned.
As an output argument *OutFrameSize* indicates the size of the converted output frame (i.e. number of bytes returned). The returned size includes any stuffing bytes added to the end of the frame to achieve 32-bit alignment.

OutFrameFormat

Specifies the desired format of the data format for the output frame (please refer to the *InFrameFormat* parameter for a description of the available formats).
NOTE: not every input format can be converted to every output format (e.g. it is not possible to convert between 525-line and 625-line frames)

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Frame has successfully been converted
DTAPI_E_INVALID_BUF	The frame input or output buffer pointer is invalid (e.g. NULL pointer or not 32-bit aligned)
DTAPI_E_INCOMP_FRAME	The input buffer does not contain a complete frame
DTAPI_E_OUTBUF_ TOO_SMALL	The output buffer is too small for receiving the converted frame
DTAPI_E_UNSUP_CONV	The requested conversion is not supported

Remarks

::DtapiCheckDeviceDriverVersion

Check whether the versions of the device drivers are compatible with the current version of the DTAPI library.

```
DTAPI_RESULT  ::DtapiCheckDeviceDriverVersion(void);
DTAPI_RESULT  ::DtapiCheckDeviceDriverVersion(int DeviceCategory);
```

Function Arguments

DeviceCategory

Device category (**DTAPI_CAT_PCI** or **DTAPI_CAT_USB**) of device driver for which to check the version.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The device drivers are compatible with the current version of the DTAPI library.
DTAPI_E_DRIVER_INCOMP	Version of at least one of the device drivers is incompatible with the DTAPI library. The device drivers need to be upgraded.
DTAPI_E_NO_DEVICE	Device-driver version cannot be queried because no DekTec device of the given category is available in the system, or the DekTec devices are disabled in the Windows device manager, or the DekTec device driver for the category is not installed.

Remarks

These functions can only check the driver version if a PCI or USB device is installed. DTE devices use the standard network drivers and no specific DekTec driver that could be incompatible.

::DtapiDeviceScan

Scan DekTec devices in the system.

```
DTAPI_RESULT ::DtapiDeviceScan(
  [in] int  NumEntries,            // #Device entries in DvcDescArr
 [out] int&  NumEntriesResult,     // #Devices found or #entries required
 [out] DtDeviceDesc*  DvcDescArr,  // Device descriptor array
  [in] bool  InclDteDvcs=false     // Include DTE-31XX devices
);
```

Function Arguments

NumEntries

Specifies the size, in number of **DtDeviceDesc** entries, of the caller-supplied *pDvcDesc* array.

NumEntriesResult

Output argument that receives the number of devices found and described in *DvcDescArr*. The value of this argument can be greater than *NumEntries*; in this case **DtapiDeviceScan** returns **DTAPI_E_BUF_TOO_SMALL**.

DvcDescArr

Pointer to a caller-supplied array of **DtDeviceDesc** entries to receive the device descriptions.

InclDteDvcs

Include DekTec DTE-31XX devices in the scan. This requires scanning the network.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Scan has completed successfully and the *DvcDescArr* array was large enough to contain all device descriptions.
DTAPI_E_BUF_TOO_SMALL	The number of device-description entries in *DvcDescArr* is too small. The number of entries required is returned in *NumEntriesResult*.

Remarks

DtapiDeviceScan scans the PCI and USB bus(es) in the current system and returns all DekTec devices found.

If *InclDteDvcs* is set to **true**, **DtapiDeviceScan** also scans the network for DekTec DTE-31XX devices. This will take extra time.

This function may have to be called twice. The first time, *NumEntries* should be set to a best-guess maximum value. If the result status is **DTAPI_E_BUF_TOO_SMALL**, the application should free the current array of **DtDeviceDesc** entries, allocate a new array with the number of entries returned in *NumEntriesResult*, and call **DtapiDeviceScan** again.

::DtapiDtDeviceDesc2String

Create a descriptive string from a device descriptor, e.g. "DTA-2145 in Slot 0".

```
DTAPI_RESULT ::DtapiDtDeviceDesc2String(
  [in] DtDeviceDesc*  pDvcDesc,    // Device descriptor
  [in] int  StringType,            // Type of string to create
 [out] char*  pString              // String buffer
  [in] int  StringLength           // Size of the string buffer
);
DTAPI_RESULT ::DtapiDtDeviceDesc2String(
  [in] DtDeviceDesc*  pDvcDesc,    // Device descriptor
  [in] int  StringType,            // Type of string to create
 [out] wchar_t*  pString           // String buffer
  [in] int  StringLength           // Size of the string buffer
);
```

Function Arguments

pDvcDesc

Pointer to the hardware function descriptor used as input to create our string description.

StringType

Defines the type of string to create. Can be any of the values defined in the table below. The values should be prefixed by **DTAPI_DVC2STR_**.

Value	Example	Meaning
TYPE_NMB	"DTA-2145"	Device type number
TYPE_AND_LOC	"DTA-2145 in slot 5"	Device type number and location

pString

Pointer to the buffer that receives the descriptive string.

StringLength

Size of the provided buffer (including space for '\0' termination). If the size specified here is too short, the generated string will be clipped and no error is returned. A size of 64 characters should suffice for all strings created with this function.

Results

DTAPI_RESULT	Meaning
DTAPI_OK	Successfully created a string
DTAPI_E_INVALID_BUF	An invalid buffer pointer is supplied for *pDvcDesc* or *pString*

::DtapiDtHwFuncDesc2String

Create a descriptive string for a hardware function from a **DtHwFuncDesc** structure, e.g. "DTA-2145 in slot 5" or "DVB-ASI".

```
DTAPI_RESULT ::DtapiDtHwFuncDesc2String(
  [in] DtHwFuncDesc*  pHwFunc,     // Hardware function descriptor
  [in] int  StringType,            // Type of string to create
 [out] char*  pString              // String buffer
  [in] int  StringLength           // Size of the string buffer
);
DTAPI_RESULT ::DtapiDtHwFuncDesc2String(
  [in] DtHwFuncDesc*  pHwFunc,     // Hardware function descriptor
  [in] int  StringType,            // Type of string to create
 [out] wchar_t*  pString           // String buffer
  [in] int  StringLength           // Size of the string buffer
);
```

Function Arguments

pHwFunc

Pointer to the hardware function descriptor used as input to create our string description.

StringType

Defines the type of string to create. Can be any of the values defined in the table below. The values should be prefixed by **DTAPI_HWF2STR_**.

Value	Example	Meaning
TYPE_NMB	"DTA-2111"	Device type number
TYPE_AND_PORT	"DTA-145 port 1"	Device type number and port number
TYPE_AND_LOC	"DTA-2144 in slot 5"	Device type number and location
ITF_TYPE	"DVB-ASI"	Physical interface type
ITF_TYPE_SHORT	"ASI"	Physical interface type – short descriptive string

pString

Pointer to the buffer that receives the descriptive string

StringLength

Size of the provided buffer (including space for '\0' termination). If the size specified here is too short, the generated string will be clipped and no error is returned. A size of 64 characters should suffice for all strings created with this function.

Results

DTAPI_RESULT	Meaning
DTAPI_OK	Successfully created a string
DTAPI_E_INVALID_BUF	An invalid buffer pointer is supplied for *pHwFunc* or *pString*

::DtapiGetDeviceDriverVersion

Get device-driver version information for the PCI device driver (*Dta32/Dta64*) or USB device driver *Dtu32/Dtu64*).

```
DTAPI_RESULT  ::DtapiGetDeviceDriverVersion(
  [in] int  DeviceCategory,       // Device-driver category (PCI/USB)
 [out] int&  Major,               // Major version number
 [out] int&  Minor,               // Minor version number
 [out] int&  BugFix,              // Bug fix version number
 [out] int&  Build                // Build number
);
```

Function Arguments

DeviceCategory

Argument specifying the device category:

Value	Meaning
DTAPI_CAT_PCI	PCI-bus device; Query version of *Dta32* or *Dta64* device driver.
DTAPI_CAT_USB	USB device; Query version of *Dtu32* or *Dtu64* device driver.

DriverVersionMajor

Major version number of the device driver. This number is incremented for major functional upgrades of the device driver.

DriverVersionMinor

The minor version number is incremented for small functional increments of the device driver.

DriverVersionBugFix

The bug-fix version number is incremented when a bug in the device driver has been fixed, without further functional enhancements to the driver.

DriverVersionBuild

Build number.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Device-driver version has been retrieved successfully.
DTAPI_E_NO_DEVICE	Device-driver version cannot be queried because no DekTec device with the specified device category is installed.
DTAPI_E_NOT_SUPPORTED	The specified device category does not use a device driver with a version that can be queried.

Remarks

::DtapiGetDtapiServiceVersion

Get version of the DTAPI service.

```
DTAPI_RESULT :: DtapiGetDtapiServiceVersion(
 [out] int&  SvcMajor,             // Major version number
 [out] int&  SvcMinor,             // Minor version number
 [out] int&  SvcBugFix,            // Bug fix version number
 [out] int&  SvcBuild              // Build number
);
```

Function Arguments

SvcMajor
Major version number of the DTAPI service.

SvcMinor
The minor version number is incremented for small functional increments of the DTAPI service.

SvcBugFix
This number is incremented when a bug in the DTAPI service has been fixed, without functional enhancements.

SvcBuild
Build number.

Result

DTAPI_RESULT	Meaning
DTAPI_E_CONNECT_TO_SERVICE	Failed to connect to the DTAPI service.
DTAPI_OK	Version numbers have been retrieved successfully.

Remarks

The main version of DTAPI is the one retrieved with **::DtapiGetVersion**. The version of the DTAPI service is required in specific circumstances only.

::DtapiGetVersion

Get version of the DTAPI library.

```
DTAPI_RESULT ::DtapiGetVersion(
 [out] int&  Major,                // Major version number
 [out] int&  Minor,                // Minor version number
 [out] int&  BugFix,               // Bug fix version number
 [out] int&  Build                 // Build number
);
```

Function Arguments

Major

Major version number of the DTAPI library. This number is incremented for major functional upgrades of the DTAPI.

Minor

The minor version number is incremented for small functional increments of the DTAPI.

BugFix

This number is incremented when a bug in the DTAPI library has been fixed, without functional enhancements.

Build

Build number.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Version numbers have been retrieved successfully.

Remarks

This function always succeeds.

The PCI and USB device driver have their own version number, which is independent from the DTAPI library version.

::DtapiHwFuncScan

Scan hardware functions hosted by DekTec devices.

```
DTAPI_RESULT ::DtapiHwFuncScan(
  [in] int  NumEntries,            // #Function entries in pHwFuncs
  [out]int&  NumEntriesResult,     // #Functions found or #entries required
  [out]DtHwFuncDesc*  pHwFuncs,    // Hardware-function descriptions
   [in] bool  InclDteDvcs=false    // Include DTE-31XX devices
);
```

Function Arguments

NumEntries

Specifies the size, in number of **DtHwFuncDesc** entries, of the caller-supplied *pHwFuncs* array.

NumEntriesResult

Output argument that receives the number of hardware functions found and described in *pHwFuncs*. The value of this argument can be greater than *NumEntries*; in this case **DtapiHwFuncScan** returns **DTAPI_E_BUFFER_TOO_SMALL**.

pHwFuncs

Pointer to a caller-supplied array of **DtHwFuncDesc** entries to receive the hardware-function descriptors. A NULL pointer may be supplied, but only if *NumEntries* is 0.

InclDteDvcs

Include DekTec DTE-31XX devices in the scan. This requires scanning the network.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Scan has completed successfully and the *pHwFuncs* array was large enough to contain all function descriptions.
DTAPI_E_BUF_TOO_SMALL	The number of function-description entries in *pHwFuncs* is too small. The number of entries required is returned in *NumEntriesResult*.
DTAPI_E_INVALID_BUF	A NULL pointer was supplied to *pHwFuncs* while *NumEntries* is greater than 0.

Remarks

DtapiHwFuncScan scans the PCI and USB bus(es) in the current system and returns all hardware functions hosted by DekTec devices. Each device may implement multiple hardware functions.

If *InclDteDvcs* is set to **true**, the **DtapiHwFuncScan** also scans the network for DekTec DTE-31XX devices. This will take extra time.

This function may have to be called twice. The first time, *NumEntries* should be set to a best-guess maximum value. If the result status is **DTAPI_E_BUF_TOO_SMALL**, the application should free the current array of **DtHwFuncDesc** entries, allocate a new array with the number of entries returned in *NumEntriesResult*, and call **DtapiHwFuncScan** again.

Another method is to start with setting *pHwFuncs* to NULL and *NumEntries* to 0. The number of required hardware-function entries will be returned, after which *pHwFuncs* can be allocated with the right size and **DtapiHwFuncScan** called again.

The hardware-function descriptors are always retrieved in the same order. Hardware functions hosted by the same device are grouped together. Within a group of hardware functions hosted by a particular device, functions of the same type are grouped together. These sequencing rules enable application programs to easily create function lists in a 'logical' order.

::DtapiInitDtTsIpParsFromIpString

Initializes the IP and source-IP address members of a TS-over-IP parameters structure.

```
DTAPI_RESULT ::DtapiInitDtTsIpParsFromIpString(
 [out] DtTsIpPars&  TsIpPars,      // TS-over-IP parameters to initialize
  [in] const char*  pIp,           // String with destination IP address
  [in] const char*  pSrcIp         // String with source IP address
);
DTAPI_RESULT ::DtapiInitDtTsIpParsFromIpString(
 [out] DtTsIpPars&  TsIpPars,      // TS-over-IP parameters to initialize
  [in] const wchar_t*  pIp,        // String with destination IP address
  [in] const wchar_t*  pSrcIp      // String with source IP address
);
```

Function Arguments

TsIpPars

TS-over-IP parameters structure to initialize.

pIp

Pointer to a string that holds the IP address (e.g. "127.0.0.1") to be used as destination IP address. If this pointer is NULL the IP address "0.0.0.0" will be used.

pSrcIp

Pointer to a string that holds the IP address (e.g. "192.168.0.1") to be used as source IP address. If this pointer is NULL the IP address "0.0.0.0" will be used.

Results

DTAPI_RESULT	Meaning
DTAPI_OK	This method cannot fail

Remarks

This method only initializes the `m_Ip` and `m_SrcIp` members in the TS-over-IP parameters structure, it will leave the other members untouched.

::DtapiModPars2SymRate

Compute symbol rate from transport-stream rate and modulation parameters.

```
DTAPI_RESULT ::DtapiModPars2TsRate(
 [out] int&  SymRate          // Computed symbol rate
  [in] int   ModType,         // Modulation type
  [in] int   ParXtra0,        // Extra parameter #0
  [in] int   ParXtra1,        // Extra parameter #1
  [in] int   ParXtra2,        // Extra parameter #2
  [in] int   TsRate           // transport-stream rate
);
```

Function Arguments

SymRate

The symbol rate in baud computed from transport-stream rate and modulation parameters.

ModType, ParXtra0, ParXtra1, ParXtra2

Set of modulation parameters from which the symbol rate is computed. Refer to **DtOutpChannel::SetModControl** on page 236 for more details about these parameters.

TsRate

transport-stream rate in bps.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Symbol rate has been computed successfully
Other result values	Error in modulation parameters, please refer to **DtOutpChannel::SetModControl**

Remarks

::DtapiModPars2TsRate

Compute transport-stream rate from modulation parameters. There are two overloads:

DtapiModPars2TsRate(int&, int, int, int, int, int)
To be used for all modulation modes except DVB-T2.

DtapiModPars2TsRate(int&, DtDvbT2Pars&)
This second overload is specifically intended for DVB-T2 modulation. The modulation parameters are defined in **DtDvbT2Pars**.

```
DTAPI_RESULT ::DtapiModPars2TsRate(
 [out] int&  TsRate          // Computed transport-stream rate
  [in] int   ModType,        // Modulation type
  [in] int   ParXtra0,       // Extra parameter #0
  [in] int   ParXtra1,       // Extra parameter #1
  [in] int   ParXtra2,       // Extra parameter #2
  [in] int   SymRate=-1      // Optional symbol rate
);
DTAPI_RESULT ::DtapiModPars2TsRate(
 [out] int&  TsRate          // Computed transport-stream rate
  [in] DtDvbT2Pars  T2Pars   // DVB-T2 modulation parameters
);
```

Function Arguments

TsRate
The transport-stream rate in bps computed from modulation parameters.

ModType, ParXtra0, ParXtra1, ParXtra2
Set of modulation parameters from which the transport-stream rate is computed. Refer to **DtOutpChannel::SetModControl** on page 236 for more details about these parameters.

T2Pars
DVB-T2 modulation parameters from which the transport-stream rate of PLP0 is computed; see description of **class DtDvbT2Pars**.

SymRate
Symbol rate in baud. This argument is only required for modulation modes that are dependent on a symbol rate: DVB-C, DVB-S and DVB-S.2.
For other modulation modes the transport-stream rate is uniquely determined by *ModType*, *ParXtra0*, *ParXtra1* and *ParXtra2*.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Successfully derived a TS-rate from the modulation parameters
DTAPI_E_SYMRATE_REQD	Conversion requires a symbol rate but none is specified
Other result values	Error in modulation parameters, please refer to **DtOutpChannel::SetModControl**

Remarks

::DtapiPower2Voltage

Convert from dBm to dBmV.

```
DTAPI_RESULT ::DtapiPower2Voltage(
  [in] int   dBm,              // Level in dBm
 [out] int&  dBmV,             // Converted level in dBmV
  [in] bool  Is50Ohm=false     // 50 Ohm (true) or 75 Ohm (false)
);
```

Function Arguments

dBm

Input level in dBm that is to be converted.

dBmV

Converted level expressed in dBmV.

Is50Ohm

Indicates whether conversion has to be applied for a 50-Ω impedance (true) or a 75-Ω impedance (false).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Successfully converted the level in dBm to a level in dBmV

Remarks

::DtapiResult2Str

Convert **DTAPI_RESULT** value to a string.

```
const char* ::DtapiResult2Str(
  [in] DTAPI_RESULT  DtapiResult  // DTAPI_RESULT value to be converted
);
```

Function Arguments

DtapiResult

DTAPI_RESULT value to be converted to a string.

Result

Remarks

For ease of use, this function doesn't return a **DTAPI_RESULT** but returns the string directly.

::DtapiVoltage2Power

Convert from dBmV to dBm.

```
DTAPI_RESULT ::DtapiVoltage2Power(
  [in] int   dBmV,              // Level in dBmV
 [out] int&  dBm,               // Converted level in dBm
  [in] bool  Is50Ohm=false      // 50 Ohm (true) or 75 Ohm (false)
);
```

Function Arguments

dBmV

Input level in dBmV that is to be converted.

dBm

Converted level expressed in dBm.

Is50Ohm

Indicates whether conversion has to be applied for a 50-Ω impedance (true) or a 75-Ω impedance (false).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Successfully converted the level in dBmV to a level in dBm

Remarks

DtDevice::AttachToIpAddr

Attach device object to the device hardware, based on the IP address of a DTE-31XX device.

```
DTAPI_RESULT DtDevice::AttachToIpAddr(
  [in] unsigned char  Ip[4]  // IP address
);
```

Function Arguments

Ip
IP address of the DTE-31XX to attach to.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Device object has been attached successfully to the hardware
DTAPI_E_ATTACHED	Device object is already attached to device hardware
DTAPI_E_NO_DEVICE	No DekTec devices found (at all)
DTAPI_E_NO_SUCH_DEVICE	A DTE-31XX with the IP address cannot be found

Remarks

AttachToIpAddr is non-intrusive. No initialization actions are performed.

This method can only be applied to DTE-31XX devices.

DtDevice::AttachToSerial

Attach device object to the device hardware, based on the serial number of the device.

```
DTAPI_RESULT DtDevice::AttachToSerial(
  [in] __int64  Serial       // Serial number
);
```

Function Arguments

Serial

Serial number of the DekTec device to attach to.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Device object has been attached successfully to the hardware
DTAPI_E_ATTACHED	Device object is already attached to device hardware
DTAPI_E_DRIVER_INCOMP	Version of device driver is incompatible with the DTAPI version, device driver needs to be upgraded
DTAPI_E_NO_DEVICE	No DekTec devices found (at all)
DTAPI_E_NO_SUCH_DEVICE	The device with the specified serial number could not be found

Remarks

AttachToSerial is non-intrusive. No initialization actions are performed.

DtDevice::AttachToSlot

Attach device object to a PCI Bus device, based on PCI-bus number and slot number.

```
DTAPI_RESULT DtDevice::AttachToSlot(
  [in] int  PciBusNumber,    // PCI-bus number
  [in] int  SlotNumber       // PCI-slot number
);
```

Function Arguments

PciBusNumber, SlotNumber

PCI-bus number and slot number of the DekTec device to attach to.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Device object has been attached successfully to the hardware
DTAPI_E_ATTACHED	Device object is already attached to a PCI card
DTAPI_E_NO_DTA_CARD	No DekTec PCI cards found (at all)
DTAPI_E_NO_SUCH_DEVICE	No DekTec DTA-1xx PCI card found in the specified slot, or the slot is empty
DTAPI_E_DRIVER_INCOMP	Version of device driver is incompatible with the DTAPI version, device driver needs to be upgraded

Remarks

AttachToSlot is non-intrusive. No initialization actions are performed.

This method cannot be applied to USB devices. Use **AttachToSerial** or **AttachToType** instead.

DtDevice::AttachToType

Attach device object to the device hardware, based on the type number of the device.

```
DTAPI_RESULT DtDevice::AttachToType(
  [in] int  TypeNumber,      // Type number of the device to look for
  [in] int  DeviceNo=0       // Relative device number
);
```

Function Arguments

TypeNumber

Integer value representing the type number of the device to attach to. The integer corresponds to the number in the hardware's type string, e.g. 2160 for the DTA-2160 or 245 for DTU-245.

DeviceNo

If the system contains multiple devices of the same type, this index number distinguishes between the various devices. *DeviceNo* of the first device is 0, the next device 1, and so on.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Device object has been attached successfully to the hardware
DTAPI_E_ATTACHED	Device object is already attached to device hardware
DTAPI_E_NO_DEVICE	No DekTec devices found (at all)
DTAPI_E_NO_SUCH_DEVICE	No device with type *Typenumber* is found in this system, or the number of devices of this type is less-or-equal than *DeviceNo*
DTAPI_E_DRIVER_INCOMP	Version of device driver is incompatible with the DTAPI version, device driver needs to be upgraded

Remarks

AttachToType is non-intrusive. No initialization actions are performed.

DtDevice::Detach

Detach device object from device hardware.

```
DTAPI_RESULT DtDevice::Detach(
      void
);
```

Function Arguments

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Device object has been detached successfully from the device hardware
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware, so it cannot be detached

Remarks

Event subscribers will be automatically unsubscribed.

DtDevice::FlashDisplay

Flash the LCD display of a DTE-31XX a number of times. With this function, a configuration tool can alert a user to a particular DTE-31XX unit.

```
DTAPI_RESULT DtDevice::FlashDisplay(
  [in] int  NumFlashes=5,    // Number of flashes
  [in] int  OnTime=100,      // Time (ms) display is on
  [in] int  OffTime=100      // Time (ms) display is off
);
```

Function Arguments

NumFlashes

Number of times to flash the display.

OnTime, OffTime

Time in ms the LCD display has to be on/off for one flash.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	A 'flash display' command has been sent to the DTE-31XX
DTAPI_E_NOT_ATTACHED	Device object is not attached to a DTE-31XX
DTAPI_E_NOT_SUPPORTED	The device is not a DTE-31XX, or a DTE-31XX without a display

Remarks

DtDevice::GetAttribute

Get the value of an attribute for a port of this device.

```
DTAPI_RESULT DtDevice::GetAttribute(
  [in] int&   Port,           // Physical port number (1…#ports)
  [in] int&   AttrId,         // Attribute identifier
 [out] int&   AttrValue       // Returned attribute value
);
```

Function Arguments

Port

Physical port number of the port for which to retrieve the attribute.

AttrId

Identifies the attribute that is to be retrieved.

The attributes below apply to modulators only.

Value	Meaning
DTAPI_ATTR_LEVEL_MAX	Maximum output level in tenth of a dBm. For OFDM modes, the maximum output level is 3dB less.
DTAPI_ATTR_LEVEL_RANGE	Output-level range in tenth of a dB.
DTAPI_ATTR_RFFREQ_ABSMAX	Absolute maximum output frequency in MHz supported by the hardware. Frequencies between **RFFREQ_MAX** and **RFFREQ_ABSMAX** may work, but are not guaranteed to work.
DTAPI_ATTR_RFFREQ_ABSMIN	Absolute minimum output frequency in MHz supported by the hardware. Frequencies between **RFFREQ_ABSMIN** and **RFFREQ_MIN** may work, but are not guaranteed to work.
DTAPI_ATTR_RFFREQ_MAX	Maximum supported output frequency in MHz.
DTAPI_ATTR_RFFREQ_MIN	Minimum supported output frequency in MHz.
DTAPI_ATTR_SAMPRHW_ABSMAX	Absolute maximum sample rate supported by the hardware. The device hardware may work for sample rates between **SAMPRHW_MAX** and **SAMPRHW_ABSMAX**, but performance specifications are not guaranteed. If **SAMPRHW_MAX** equals **SAMPRHW_ABSMAX**, sample rate conversion is used for sample rates above **SAMPRHW_MAX**.
DTAPI_ATTR_SAMPRHW_ABSMIN	Absolute minimum sample rate supported by the hardware. The device hardware may work for sample rates between **SAMPRHW_ABSMIN** and **SAMPRHW_MIN**, but performance specifications are not guaranteed. If **SAMPRHW_MIN** equals **SAMPRHW_ABSMIN**, sample rate conversion is used for sample rates below **SAMPRHW_MIN**.
DTAPI_ATTR_SAMPRHW_HARDLIM	
DTAPI_ATTR_SAMPRHW_MAX	Maximum sample rate for which performance specifications are guaranteed.
DTAPI_ATTR_SAMPRHW_MIN	Minimum sample rate for which performance specifications are

	guaranteed.
DTAPI_ATTR_SAMPRATE_ABSMAX	Absolute maximum sample rate that is supported. The signal must be band-limited, otherwise signal frequencies between **SAMPRHW_MAX**/2 and **SAMPRATE_MAX**/2 are muted and may even fold back (alias) into the main band.
DTAPI_ATTR_SAMPRATE_ABSMIN	Absolute minimum sample rate that is supported.
DTAPI_ATTR_SAMPRATE_MAX	Maximum sample rate for which performance specifications are guaranteed.
DTAPI_ATTR_SAMPRATE_MIN	Minimum sample rate for which performance specifications are guaranteed.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The attribute value has been retrieved successfully
DTAPI_E_NO_SUCH_PORT	Invalid port number for this device
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware
DTAPI_E_NOT_SUPPORTED	The attribute is not supported for this device

Remarks

DtDevice::GetDescriptor

Get device descriptor.

```
DTAPI_RESULT DtDevice::GetDescriptor(
 [out] DtDeviceDesc&  DvcDesc      // Device descriptor
);
```

Function Arguments

DvcDesc

Output argument that receives the device descriptor.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The device descriptor has been retrieved successfully
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware

Remarks

DtDevice::GetDeviceDriverVersion

Get device-driver version information.

```
DTAPI_RESULT DtDevice::GetDeviceDriverVersion(
 [out] int&  Major,          // Major version number
 [out] int&  Minor,          // Minor version number
 [out] int&  BugFix,         // Buf-fix version number
 [out] int&  Build           // Build number
);
```

Function Arguments

Major

Major version number of the device driver used to access the device hardware. This number is incremented for major functional upgrades of the device driver.

Minor

The minor version number is incremented for small functional increments of the device driver.

BugFix

The bug-fix version number is incremented when a bug in the device driver is fixed, without functional enhancements.

Build

Build number.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Version numbers have been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware

Remarks

This function cannot be used to obtain hardware versioning information. Use **VpdRead** instead.

DtDevice::GetDisplayName

Get the name displayed on the LCD status display of a DTE-31XX.

```
DTAPI_RESULT DtDevice::GetDisplayName(
 [out] char*  pName          // Displayed name
);
DTAPI_RESULT DtDevice::GetDisplayName(
 [out] wchar_t*  pName       // Displayed name
);
```

Function Arguments

pName

Pointer to the character array that receives the displayed name. The character array must be allocated before calling **GetDisplayName**. The DTAPI limits the maximum length of a name including the null-terminator to 16 characters, so a 16-char array suffices.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Version numbers have been retrieved successfully
DTAPI_E_NOT_SUPPORTED	The device does not have a status display
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware

Remarks

DtDevice::GetFirmwareVersion

Get version number of the firmware loaded on the device.

```
DTAPI_RESULT DtDevice::GetFirmwareVersion(
 [out] int&  FirmwareVersion
);
```

Function Arguments

FirmwareVersion

Single number that identifies the version of the FPGA- and/or embedded software on the device.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Firmware version has been retrieved successfully.
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver.
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware.

Remarks

DtDevice::GetIoConfig

Get the I/O configuration of the specified port.

```
DTAPI_RESULT DtInpChannel::GetIoConfig(
  [in] int  Port,             // Physical port number
  [in] int  Group,            // I/O configuration group
 [out] int&  Value            // I/O configuration value
);
DTAPI_RESULT DtInpChannel::GetIoConfig(
  [in] int  Port,             // Physical port number
  [in] int  Group,            // I/O configuration group
 [out] int&  Value,           // I/O configuration value
 [out] int&  SubValue         // I/O configuration subvalue
);
DTAPI_RESULT DtInpChannel::GetIoConfig(
  [in] int  Port,             // Physical port number
  [in] int  Group,            // I/O configuration group
 [out] int&  Value,           // I/O configuration value
 [out] int&  SubValue,        // I/O configuration subvalue
 [out] __int64&  ParXtra0     // Extra parameter #0
);
DTAPI_RESULT DtInpChannel::GetIoConfig(
  [in] int  Port,             // Physical port number
  [in] int  Group,            // I/O configuration group
 [out] int&  Value,           // I/O configuration value
 [out] int&  SubValue,        // I/O configuration subvalue
 [out] __int64&  ParXtra0,    // Extra parameter #0
 [out] __int64&  ParXtra1     // Extra parameter #1
);
DTAPI_RESULT DtInpChannel::GetIoConfig(
 [in,out] DtIoConfig& IoCfg // I/O configuration parameters in a struct
);
```

Function Arguments

Port

Physical port number.

Group, Value, SubValue, ParXtra0, ParXtra1

I/O configuration parameters, see **SetIoConfig**.

IoCfg

I/O configuration data structure containing all I/O configuration parameters in one struct.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	I/O configuration has been read successfully
DTAPI_E_NO_SUCH_PORT	Invalid port number for this device
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware
DTAPI_E_NOT_SUPPORTED	The method is applied to either another device rather than a DTA-2137 or an invalid port number is given
DTAPI_E_FIRMW_INCOMP	The firmware is incompatible, please upgrade the firmware
DTAPI_E_SERVICE_INCOMP	The DTAPI service needs to be updated
DTAPI_E_INVALID_MODE	For *IoConfig* **DTAPI_IOCONFIG_S2LOOPMODE**: The configuration of the corresponding ASI port is not in loop-though mode

Remarks

On the DTA-2137, a receiver port becomes disabled when the other receiver port is configured with **DTAPI_IOCONFIG_INPUT_APSK**.

DtDevice::GetNwSpeed

Get the speed of the network link.

```
DTAPI_RESULT DtDevice::GetNwSpeed(
  [in] int  Port,              // Physical port number
 [out] bool&  Enable,          // Network port enabled yes/no
 [out] int&  Speed             // I/O configuration value
);
```

Function Arguments

Port

Physical port number.

Enable

Output argument that is set to *true* if the network port is enabled.

Speed

Current link speed.

Value	Meaning
DTAPI_NWSPEED_NOLIN	Link not connected
DTAPI_NWSPEED_10MB_HALF	10Mbps, half duplex
DTAPI_NWSPEED_10MB_FULL	10Mbps, full duplex
DTAPI_NWSPEED_100MB_HALF	100Mbps, half duplex
DTAPI_NWSPEED_100MB_FULL	100Mbps, full duplex
DTAPI_NWSPEED_100MB_FULL	100Mbps, full duplex
DTAPI_NWSPEED_1GB_MASTER	1Gbps in master mode
DTAPI_NWSPEED_1GB_SLAVE	1Gbps in slave mode

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Network speed has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware
DTAPI_E_NOT_SUPPORTED	The device is not a network port

Remarks

DtDevice::GetRefClkCnt

Get a sample of the reference-clock counter on the device.

```
DTAPI_RESULT DtDevice::GetRefClkCnt(
 [out] int&  RefClkCnt       // Sample of reference-clock counter
);
DTAPI_RESULT DtDevice::GetRefClkCnt(
 [out] int&  RefClkCnt,      // Sample of reference-clock counter
 [out] int&  RefClkFreqHz    // Clock frequency of the reference clock
);
```

Function Arguments

RefClkCnt
Sample of the 32-bit reference clock counter.

RefClkFreqHz
Clock frequency of the reference clock in Hz.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Sample has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware
DTAPI_E_NOT_SUPPORTED	The device does not support retrieval of a sample of the reference-clock counter

Remarks

This method is supported on the following devices:

Device Type Number	Clock Frequency
DTA-105	54.0MHz
DTA-107(S2)	25.0MHz
DTA-110(T)	25.0MHz
DTA-115	54.0MHz
DTA-120	40.5MHz (FW Version ≥ 4)
DTA-122	27.0MHz (FW Version ≥ 4)
DTA-124	40.5Mhz (FW Version 0) / 54MHz (FW Version ≥ 1)
DTA-140	40.5MHz (FW Version ≥ 1)
DTA-145	54.0MHz

DTA-160	54.0MHz
DTA-545	40.5Mhz (FW Version 0) / 54MHz (FW Version ≥ 1)
DTA-2135	54.0MHz
DTA-2144	54.0MHz
DTA-2145	54.0MHz
DTE-3100	54.0MHz
DTE-3120	54.0MHz

Some devices (e.g. DTU-225 and DTU-245) that have a reference-clock counter, do not allow access to their reference-clock counter (i.e. this method will return **`DTAPI_E_NOT_SUPPORTED`**). For these devices it is possible to determine the running frequency of their on-board reference-clock counter via the **`DtDevice::GetRefClkFreq`** method.

DtDevice::GetRefClkFreq

Get the frequency of the on-board reference clock.

```
DTAPI_RESULT DtDevice::GetRefClkFreq(
 [out] int&  RefClkFreqHz     // Clock frequency of the reference clock
);
```

Function Arguments

RefClkFreqHz

Clock frequency of the reference clock in Hz.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Sample has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware
DTAPI_E_NOT_SUPPORTED	The device does not support getting of the reference-clock frequency

Remarks

Amongst other purposes the on-board reference-clock counter is used for assigning arrival timestamps to the incoming data (see **DtInpChannel::SetRxMode**). By calling this method one can determine the running frequency of the reference-clock counter used for assigning the arrival timestamps.

Next to the devices mentioned in the description of the **DtDevice::GetRefClkCnt** method, this method supports the following devices:

Device Type Number	Clock Frequency
DTU-225	48Mhz (FW Version < 5) / 54MHz (FW Version ≥ 5)
DTU-245	48Mhz (FW Version < 5) / 54MHz (FW Version ≥ 5)

DtDevice::GetUsbSpeed

Get the speed (e.g. full or high speed) of the USB bus.

```
DTAPI_RESULT DtDevice::GetUsbSpeed(
 [out] int&   UsbSpeed
);
```

Function Arguments

UsbSpeed

Current speed of the USB bus the device is connected to.

Value	Meaning
DTAPI_USB_FULL_SPEED	USB bus operates at full speed (max. 12Mbps)
DTAPI_USB_HIGH_SPEED	USB bus operates at high speed (max. 480Mbps)

Result

DTAPI_RESULT	Meaning
DTAPI_OK	USB speed has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware
DTAPI_E_NOT_SUPPORTED	The device does not support the getting of the USB speed

Remarks

Use this method to determine if the USB device is connected to a USB bus operating in full or high speed mode. A USB bus operating at full speed usually indicates that the DTU-2XX device is connected to a USB-1 bus[2]. High speed is only supported by USB-2 buses.

USB "full speed" limits the maximum input/output bit-rate supported by the DTU-2XX devices to 8Mbps. To be able to use the DTU-2XX for bit-rates higher than 8Mbps a USB-2 bus operating at high speed should be used.

This method is only supported by the DTU-2XX devices.

[2] USB-2 buses can operate in full-speed mode for backward compatibility reasons (support for USB-1 devices)

DtDevice::GetVcxoState

Get the state of the on-board VCXO.

```
DTAPI_RESULT DtDevice::GetVcxoState(
 [out]bool&  Enable,        // VCXO is enabled or disabled?
 [out]int&   Lock,          // Current genlock state
 [out]int&   VcxoClkFreqHz  // Measured VCXO frequency in Hz
);
```

Function Arguments

Enable

Indicates whether the VCXO is enabled or disabled.

Lock

Current genlock state.

Value	Meaning
DTAPI_GENLOCK_INLOCK	The VCXO is genlocked to a SDI reference signal
DTAPI_GENLOCK_NOLOCK	The VCXO is not genlocked to a SDI signal

VcxoClkFreqHz

Measured VCXO frequency in Hz.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	State has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_SUPPORTED	The device does not support this function

Remarks

DtDevice::HwFuncScan

Scan hardware functions hosted by this device.

```
DTAPI_RESULT DtDevice::HwFuncScan(
  [in] int  NumEntries,            // #Function entries in pHwFuncs
  [out]int&  NumEntriesResult,     // #Functions found or #entries required
  [out] DtHwFuncDesc*  pHwFuncs    // Hardware-function descriptions
);
```

Function Arguments

NumEntries

Specifies the size, in number of **DtHwFuncDesc** entries, of the caller-supplied *pHwFuncs* array.

NumEntriesResult

Output argument that receives the number of hardware functions found and described in *pHwFuncs*. The value of this argument can be greater than *NumEntries* (when **DtapiHwFuncScan** returns **DTAPI_E_BUFFER_TOO_SMALL**).

pHwFuncs

Pointer to a caller-supplied array of **DtHwFuncDesc** entries to receive the hardware-function descriptors. A NULL pointer may be supplied, but only if *NumEntries* is 0.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Scan has completed successfully and the *pHwFuncs* array was large enough to contain all function descriptions.
DTAPI_E_BUF_TOO_SMALL	The number of function-description entries in *pHwFuncs* is too small. The number of entries required is returned in *NumEntriesResult*.
DTAPI_E_INVALID_BUF	A NULL pointer was supplied to *pHwFuncs* while *NumEntries* is greater than 0.

Remarks

This function is the equivalent of **::DtapiHwFuncScan** for a single device.

DtDevice::HwFuncScan function may have to be called twice. The first time, *NumEntries* should be set to a best-guess maximum value. If the result status is **DTAPI_E_BUF_TOO_SMALL**, the application should free the current array of **DtHwFuncDesc** entries, allocate a new array with the number of entries returned in *NumEntriesResult*, and call **DtapiHwFuncScan** again.

Another method is to start with setting *pHwFuncs* to NULL and *NumEntries* to 0. The number of required hardware-function entries will be returned, after which *pHwFuncs* can be allocated with the right size and **DtapiHwFuncScan** called again.

DtDevice::LedControl

Take direct control of the device's general-status LED, or let the hardware drive the LED.

```
DTAPI_RESULT DtDevice::LedControl(
  [in] int  LedControl       // DTAPI_LED_XXX
);
```

Function Arguments

LedControl

Controls status of the LED.

Value	Meaning
DTAPI_LED_HARDWARE	Hardware drives the LED (default after power up)
DTAPI_LED_OFF	LED is forced to off-state
DTAPI_LED_GREEN	LED is forced to green-state
DTAPI_LED_RED	LED is forced to red-state
DTAPI_LED_YELLOW	LED is forced to yellow-state

Result

DTAPI_RESULT	Meaning
DTAPI_OK	LED setting has been accepted
DTAPI_E_INVALID_MODE	The specified LED-control value is invalid
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware
DTAPI_E_NOT_SUPPORTED	The device does not have a general-status LED

Remarks

When a device object is detached from the device hardware, all direct-control settings are released (LED control is reset to **DTAPI_LED_HARDWARE**).

The DTA-120, DTA-122 and DTA-140 each have a single LED, which can be controlled by either this method (**DtDevice::LedControl**) or by **DtInpChannel::LedControl**. If both methods are applied in parallel, **DtDevice::LedControl** has precedence over **DtInpChannel::LedControl**.

DtDevice::SetDisplayName

Set the name on the LCD status display of a DTE-31XX device.

```
DTAPI_RESULT DtDevice::SetDisplayName(
  [in] char*  pName          // Displayed name
);
DTAPI_RESULT DtDevice::SetDisplayName(
  [in] wchar_t*  pName       // Displayed name
);
```

Function Arguments

pName

Null-terminated character string specifying the name to be displayed on the LCD status display of the device.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Version numbers have been retrieved successfully
DTAPI_E_NOT_SUPPORTED	The device does not have a status display
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware

Remarks

DtDevice::SetIoConfig

Configure a physical port. On Windows, the I/O configuration is persisted in the registry, and automatically reloaded after a reboot. On Linux, applications have to implement their own persistency.

```
DTAPI_RESULT DtOutpChannel::SetIoConfig(
  [in] int  Port,             // Physical port number
  [in] int  Group,            // I/O configuration group
  [in] int  Value,            // I/O configuration value
  [in] int  SubValue=-1,      // I/O configuration subvalue
  [in] __int64  ParXtra0=-1, // Extra parameter #0
  [in] __int64  ParXtra1=-1  // Extra parameter #1
);
DTAPI_RESULT DtInpChannel::SetIoConfig(
      [in]  int  Port,       // Physical port number
 [in,out] DtIoConfig& IoCfg // I/O configuration parameters in a struct
);
```

Function Arguments

Port

Physical port number.

Group

Specifies the I/O configuration 'group'.

The *Value* argument for the groups below is an *enumeration*:

Value	Meaning
DTAPI_IOCONFIG_IODIR	I/O direction
DTAPI_IOCONFIG_IOSTD	I/O standard
DTAPI_IOCONFIG_RFCLKSEL	RF clock source selection
DTAPI_IOCONFIG_SPICLKSEL	Parallel port clock source selection
DTAPI_IOCONFIG_SPIMODE	Parallel port mode
DTAPI_IOCONFIG_SPISTD	Parallel port I/O standard
DTAPI_IOCONFIG_TSRATESE	Transport-stream rate selection

The *Value* argument for the groups below is a *boolean*:

Value	Meaning
DTAPI_IOCONFIG_FAILSAFE	Put port in failsafe mode. *ParXtra0* is the time out. DTA-145/2145, port #2: If the watchdog triggers, the signal on port #1 will be connected to port #2 through a relay
DTAPI_IOCONFIG_GENLOCKED	The port will lock the SDI output timing to the genlock input. If the application fails to write data to the channel in time, black frames are inserted to maintain synchronisation
DTAPI_IOCONFIG_GENREF	Use port as a genlock reference input (for SDI ports)
DTAPI_IOCONFIG_SWS2APSK	Enable DVB-S2 reception in 16-APSK or 32-APSK mode. DTA-2137: If this option is enabled, the board will operate in

	single-channel mode; Without this option, two channels are available

Value, SubValue

Specifies the configuration option to be set.

Boolean I/O Configuration (**FAILSAFE**, **GENLOCKED**, **GENREF**, **SWS2APSK**)

Value	Meaning
DTAPI_IOCONFIG_TRUE	Turn I/O capability on
DTAPI_IOCONFIG_FALSE	Turn I/O capability off

Configuration *values* for **IODIR** (I/O direction)

Value	Meaning
DTAPI_IOCONFIG_DISABLED	Port is disabled and cannot be used
DTAPI_IOCONFIG_INPUT	Use port as input
DTAPI_IOCONFIG_OUTPUT	Use port as output

Configuration *subvalues* for **IODIR**, **IOCONFIG_INPUT**

SubValue	Meaning
DTAPI_IOCONFIG_INPUT	Standard input
DTAPI_IOCONFIG_SHAREDANT	Antenna input is shared with port *ParXtra0*

Configuration *subvalues* for **IODIR**, **IOCONFIG_OUTPUT**

SubValue	Meaning
DTAPI_IOCONFIG_OUTPUT	Standard output
DTAPI_IOCONFIG_DBLBUF	Make port a doubly-buffered copy of port *ParXtra0*
DTAPI_IOCONFIG_LOOPS2L3	DTA-2137: Read BBFRAMEs and encapsulate inL3 frames
DTAPI_IOCONFIG_LOOPS2TS	DTA-2137: Loop through a sub stream from a DVB-S2 Multiple Input Stream, based on ISI. The selected stream can be an MPEG2 transport stream or any other Input Stream. *ParXtra0* specifies the Input Stream Identifier (ISI)
DTAPI_IOCONFIG_LOOPTHR	Make port a loop-through copy of input port *ParXtra0*

Configuration *values* for **IOSTD** (I/O standard)

Value	Meaning
DTAPI_IOCONFIG_3GSDI	Use port in 3G-SDI mode
DTAPI_IOCONFIG_ASI	Use port in DVB-ASI mode
DTAPI_IOCONFIG_DEMOD	Use port as demodulator
DTAPI_IOCONFIG_HDSDI	Use port in HD-SDI mode

DTAPI_IOCONFIG_IFADC	Use port in IFADC mode (DTA-2135 only)
DTAPI_IOCONFIG_IP	Use port in TS-over-IP mode
DTAPI_IOCONFIG_MOD	Use port as modulator
DTAPI_IOCONFIG_SDI	Use port in SD-SDI mode
DTAPI_IOCONFIG_SPI	Use port in DVB-SPI mode
DTAPI_IOCONFIG_SPISDI	Use port in SDI-over-SPI mode (DTA-2142)

Configuration *values* for **RFCLKSEL** (RF clock source selection)

Value	Meaning
DTAPI_IOCONFIG_RFCLKINT	Use the internal reference for the RF clock
DTAPI_IOCONFIG_RFCLKEXT	Use an external clock reference for the RF clock

Configuration *values* for **SPICLKSEL** (SPI clock source selection)

DTA-2142 only

Value	Meaning
DTAPI_IOCONFIG_SPICLKINT	Use the internal reference for the parallel port clock
DTAPI_IOCONFIG_SPICLKEXT	Use an external reference for the parallel port clock

Note: The SPI clock reference is not used in parallel-port mode **SPIDVBMODE**. In that mode, the TS-rate clock is used, as selected by **TSRATESEL**.

Configuration *values* for **SPIMODE** (parallel port mode)

DTA-2142, *DTA-102* (**SPIDVBMODE** only)

Value	Meaning
DTAPI_IOCONFIG_SPIDVBMODE	Use standard SPI mode
DTAPI_IOCONFIG_SPIFIXEDCLK	Set parallel-port clock to a fixed frequency and use DValid as strobe to signal if a data word is valid or not
DTAPI_IOCONFIG_SPISER8B	Serial 8-bit mode
DTAPI_IOCONFIG_SPISER10B	Serial 10-bit mode

Configuration *values* for **SPISTD** (parallel port physical I/O standard)

DTA-2142, *DTA-102* (**DTAPI_IOCONFIG_SPILVDS1** only)

Value	Meaning
DTAPI_IOCONFIG_SPILVDS1	Configures interface for M-LVDS Type 1 operation mode
DTAPI_IOCONFIG_SPILVDS2	Configures interface for M-LVDS Type 2 operation mode
DTAPI_IOCONFIG_SPILVTTL	Configures interface for LVTLL operation mode

Configuration *values* for **TSRATESEL**

Value	Meaning
DTAPI_IOCONFIG_INTTSRATE	Use internal clock for TS-rate generation
DTAPI_IOCONFIG_EXTTSRATE	Use external clock source as the TS-rate
DTAPI_IOCONFIG_EXTRATIO	Derive TS-rate from an fixed external clock source with a fixed ratio
DTAPI_IOCONFIG_LOCK2INP	Lock TS rate output port to the TS rate of an input port

ParXtra0, ParXtra1

Extra parameters for some of the I/O configuration operations.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Channel type has been set successfully
DTAPI_E_ATTACHED	Cannot change I/O configuration because a channel object is attached to this port
DTAPI_E_DEV_DRIVER	Driver was not able to set the I/O configuration
DTAPI_E_FIRMW_INCOMP	The firmware is incompatible
DTAPI_E_INVALID_ISI	An invalid value for ISI is specified. The valid range is 0…255
DTAPI_E_INVALID_MODE	Invalid setting for I/O configuration
DTAPI_E_NO_SUCH_PORT	Invalid port number for this device
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware
DTAPI_E_NOT_SUPPORTED	The I/O configuration option is not supported
DTAPI_E_SERVICE_INCOMP	The DTAPI service needs to be updated

Remarks

The I/O configuration of a port can only be changed when the underlying device is attached to a **DtDevice** object.

DtDevice::SetNwSpeed

Set the speed of a network link.

```
DTAPI_RESULT DtDevice::SetNwSpeed(
  [in] int  Port,             // Physical port number
  [in] bool  Enable,          // Network port enabled yes/no
  [in] int  Speed             // I/O configuration value
);
```

Function Arguments

Port

Physical port number.

Enable

Enable (*true*) or disable (*false*) the network port.

Speed

Link speed.

Value	Meaning
DTAPI_NWSPEED_AUTO	Automatically set network link speed
DTAPI_NWSPEED_10MB_HALF	10Mbps, half duplex
DTAPI_NWSPEED_10MB_FULL	10Mbps, full duplex
DTAPI_NWSPEED_100MB_HALF	100Mbps, half duplex
DTAPI_NWSPEED_100MB_FULL	100Mbps, full duplex
DTAPI_NWSPEED_100MB_FULL	100Mbps, full duplex
DTAPI_NWSPEED_1GB_MASTER	1Gbps in master mode
DTAPI_NWSPEED_1GB_SLAVE	1Gbps in slave mode

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Network speed has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_ARG	Device object is not attached to device hardware
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware
DTAPI_E_NOT_SUPPORTED	The device is not a network port

Remarks

DtDevice::VpdDelete

Delete a Vital-Product Data (VPD) item from the VPD read/write section in the serial EEPROM on the device.

```
DTAPI_RESULT DtDevice::VpdDelete(
  [in] const char*  pTag     // Identifies VPD item, e.g. "Y0"
);
DTAPI_RESULT DtDevice::VpdDelete(
  [in] const wchar_t*  pTag  // Identifies VPD item, e.g. L"Y0"
);
```

Function Arguments

pTag

Null-terminated character string identifying the VPD item to be deleted.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	VPD item has been deleted successfully
DTAPI_E_EEPROM_READ	A read operation from the serial EEPROM did not succeed
DTAPI_E_EEPROM_WRITE	The write operation to the serial EEPROM did not succeed
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware
DTAPI_E_NOT_FOUND	The VPD item could not be found
DTAPI_E_READ_ONLY	An attempt was made to delete a read-only VPD item

Remarks

If a VPD item with the specified keyword already exists, that item is overwritten, unless it is a read-only item. In the latter case, **DTAPI_E_READ_ONLY** is returned.

The size of the VPD read/write segment is 256 bytes. Writing to the serial EEPROM is a relatively slow operation.

DtDevice::VpdRead

Read a Vital-Product Data (VPD) item from the EEPROM on the device.

```
DTAPI_RESULT DtOutpChannel::VpdTag(
  [in] const char* pTag,      // Identifies the VPD item, e.g. "SN"
 [out] char*  pVpdItem        // String read from EEPROM
);
DTAPI_RESULT DtOutpChannel::VpdTag(
  [in] const wchar_t*  pTag, // Identifies the VPD item, e.g. L"SN"
 [out] wchar_t*  pVpdItem    // String read from EEPROM
);
```

Function Arguments

pTag

Null-terminated character string identifying the VPD item to be read. The keyword must consist of either two characters, or it should be the special string "VPDID".
The table below lists standard keywords supported by DekTec devices. Next to these standard keywords, custom VPD keywords can be created with **VpdWrite**.

Value	Meaning
"VPDID"	Pseudo value to retrieve the *VPD ID String*, e.g. "DTA-100 DVB-ASI-C Output 0...150 Mbps".
"CL"	*Customer ID*
"EC"	*Engineering Change level*. Identifies the hardware revision level of the device, e.g. "Rev 1".
"MN"	*Manufacture ID* DekTec-internal code identifying the manufacturer of the hardware.
"PD"	*Production Date*, e.g. "2003.07"
"PN"	*Part Number*, e.g. "DTU-225"
"SN"	*Serial Number* E.g. "4225266001".
"XT"	*Crystal stability* E.g. "5ppm@25C;15ppm", which means a frequency stability of ±5 ppm at room temperature and a stability of ±15 ppm over the full temperature range and including aging.

pVpdItem

String retrieved from the EEPROM. The character array must be allocated before calling **VpdRead**. The DTAPI limits the maximum length of a VPD item to 63 characters, so a 64-char array suffices.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	VPD item has been read successfully
DTAPI_E_EEPROM_FORMAT	The data format in the serial EEPROM is not VPD compliant
DTAPI_E_EEPROM_READ	A read operation from the serial EEPROM did not succeed
DTAPI_E_KEYWORD	The keyword is neither two characters, nor "VPDID"
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware
DTAPI_E_NOT_FOUND	The VPD item could not be found

Remarks

If one of the standard keywords ("CL", "EC", …) has been specified and the method returns **DTAPI_E_NOT_FOUND**, the serial EEPROM has been tampered.

DtDevice::VpdWrite

Write a Vital-Product Data (VPD) item to the VPD read/write section in the serial EEPROM on the device.

```
DTAPI_RESULT DtDevice::VpdWrite(
  [in] const char*  pTag,     // Identifies VPD item, e.g. "Y1"
  [in] char*  pVpdItem        // String to be written to the EEPROM
);
DTAPI_RESULT DtDevice::VpdWrite(
  [in] const char*  pTag,     // Identifies VPD item, e.g. L"Y1"
  [in] wchar_t*  pVpdItem     // String to be written to the EEPROM
);
```

Function Arguments

pTag

Null-terminated character string identifying the VPD item to be written. The keyword must consist of two characters (the "VPDID" item cannot be written).

pVpdItem

String to be written to the EEPROM. The maximum size of a VPD item is 63 characters.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	VPD item has been written successfully
DTAPI_E_EEPROM_FULL	The serial EEPROM has not enough free space available for writing the new VPD item
DTAPI_E_EEPROM_READ	A read operation from the serial EEPROM did not succeed
DTAPI_E_EEPROM_WRITE	The write operation to the serial EEPROM did not succeed for another reason
DTAPI_E_KEYWORD	The keyword does not consist of two characters
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware
DTAPI_E_READ_ONLY	An attempt was made to overwrite a read-only VPD item
DTAPI_E_TOO_LONG	The length of the VPD item is too long (>63 characters)

Remarks

If a VPD item with the specified keyword already exists, that item is overwritten, unless it is a read-only item. In the latter case, **DTAPI_E_READ_ONLY** is returned.

For system-specific use, the VPD specification in the *PCI Local Bus Specification Rev 2.2* recommends keywords of the form "Yx", with the second character one of '0' … '9', 'B' … 'Z'. Keyword "YA" is defined as the *system-asset identifier* provided by the system owner. Keywords of the form "Vx" are reserved for use by DekTec.

The size of the VPD read/write segment is 256 bytes. Write operations to the serial EEPROM are relatively slow.

DtInpChannel

Class representing an input channel for receiving the following formats:
- MPEG-2 transport stream over ASI, SPI or IP;
- Serial Digital Interface (SDI);
- Demodulators (receivers).

```
class DtInpChannel;
```

Operations on input channels require *exclusive access* to the hardware. Just a single input channel object can attach to the hardware and that object gets exclusive access.

Demodulators are a special case, because some operations on demodulators do not require exclusive access. Therefore, for demodulators only, an input channel can attach to receiver hardware non-exclusively. In this mode, operations that require exclusive access like reading transport-stream data return an error. However, operations like tuning and getting measurements can be executed, possibly in parallel from multiple processes. This enables application scenarios in which one application tunes the receiver and receives transport stream data, while another application reads MER, constellation points and other measurement data.

DtInpChannel::AttachToPort

Attach the input-channel object to a specific physical port. Attachment can be exclusive (default) or shared. Demodulators are the only type of input channels that support shared access for a subset of functions (tuning and measurement functions).

```
DTAPI_RESULT DtInpChannel::AttachToPort(
  [in] DtDevice*  pDtDvc,          // Device object
  [in] int  Port,                  // Physical port number (1…#ports)
  [in] bool  Exclusive=true,       // Request exclusive access yes/no
  [in] bool  ProbeOnly=false       // Just check whether channel is in use
);
```

Function Arguments

pDtDvc

Pointer to the device object that represents a DekTec device. The device object must have been attached to the device hardware.

Port

Physical port number. The channel object is attached to this port. The port number of the top-most port is 1, except on the DTA-160 or DTA-2160, on which the top-most Ethernet port is port #4.

Exclusive

If *false*, request shared access. If *true*, request exclusive access. This is the default. For demodulators, and for demodulators only, this argument may be set to *false*. This way, multiple input channel objects in multiple processes can access the receiver and demodulator-specific functions that access the receiver using I2C calls can be called simultaneously, e.g. **GetConstellationPoints** and **Tune**. Most generic input-channel functions still require exclusive access to access device registers and will return an error (**DTAPI_E_EXCL_ACCESS_REQD**) if the channel was attached non-exclusively.

ProbeOnly

Probe whether the channel is in use, but do not actually attach.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Channel object has been attached successfully to the port
DTAPI_OK_FAILSAFE	For devices with a failsafe relay only. Attachment in failsafe mode was successfully completed. The application shall call **SetFailsafeAlive** on a regular basis to prevent the release of the failsafe relay. Failure to do so will physically connect this input port to the buddy output port. This is not an error code; It is intended to make the application aware of failsafe mode.
DTAPI_E_ATTACHED	Channel object is already attached
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_DEVICE	Pointer *pDtDvc* is not valid or the device object is not attached to a hardware device

DTAPI_E_EXCL_MANDATORY	Shared access (*Exclusive*=*false*) was requested, but this is not supported by the input channel. Only demodulators support shared access.
DTAPI_E_IN_USE	Another channel object is already attached to this port
DTAPI_E_NO_DT_INPUT	*Port* is not an input
DTAPI_E_NO_SUCH_PORT	Invalid port number for this device
DTAPI_E_OUT_OF_MEM	TS-over-IP: Receive FIFO cannot be allocated

Remarks

DtInpChannel::BlindScan

Scan the L-band for valid transponders. This method is currently only supported by the DTA-2137.

```
DTAPI_RESULT DtInpChannel::BlindScan(
  [in] int  NumEntries,            // #Transponder entries in pResults
 [out] int&  NumEntriesResult,     // #Transponders found or #entries req.
 [out] DtTransmitter* pResults,    // Transmitter descriptions
  [in] __int64  FreqHzSteps,       // Optional parameter to select stepsize
  [in] __int64  StartFreqHz,       // Optional parameter to select starting
                                   // frequency
  [in] __int64  EndFreqHz          // Optional parameter to select ending
                                   // frequency
);
```

Function Arguments

NumEntries

Specifies the size of the caller-supplied array *pResults* in number of **DtTransmitter** entries.

NumEntriesResult

Output argument that receives the number of transponders found. The value can be greater than *NumEntries* (when **BlindScan** returns **DTAPI_E_BUFFER_TOO_SMALL**).

pResults

Pointer to a caller-supplied array of **DtTransmitter** entries to receive the transmitter descriptors.

FreqHzSteps

This optional argument specifies the step size of the blind scan in Hertz.

StartFreqHz

This optional argument specifies the starting frequency of the blind scan in Hertz.

EndFreqHz

This optional argument specifies the ending frequency of the blind scan in Hertz.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The blind scan succeeded successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The blind scan is not supported by the attached hardware
DTAPI_E_INVALID_BUF	The number of transponder-description entries in *pResults* is too small. The number of entries required is returned in *NumEntriesResult*.
DTAPI_E_INVALID_FREQ	The range of frequencies specified with *FreqHzSteps*, *StartFreqHz* and *EndFreqHz* is incompatible (too low or too high) with the tuner on the attached hardware

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive*=*true*).

The **`BlindScan`** method is currently only supported by the DTA-2137.

The **`BlindScan`** method operates with the current LNB settings. In order to scan the frequency band for multiple local oscillator frequency's and/or polarization modes, the desired LNB setting needs to be applied prior to calling **`BlindScan`**.

Depending on the starting frequency, the ending frequency and the step size, **`BlindScan`** will take a few minutes to complete. To reduce the amount of time that **`BlindScan`** needs to complete the scan, the step size can be increased or the frequency band could be narrowed.

DtInpChannel::ClearFifo

Clear contents of the receive FIFO and set receive control to **DTAPI_RXCTRL_IDLE**. Clear the overflow flag (**DTAPI_RX_FIFO_OVF**).

```
DTAPI_RESULT DtInpChannel::ClearFifo();
```

Function Arguments

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Receive FIFO has been cleared
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

The effects of **ClearFifo** are equivalent to **Reset(DTAPI_FIFO_RESET)**.

Calling **ClearFifo()** will clear the receive-FIFO-overflow flag (**DTAPI_FIFO_OVF**) and set the receive-control state to **DTAPI_RXCTRL_IDLE**.

DtInpChannel::ClearFlags

Clear *latched* status flag(s).

```
DTAPI_RESULT DtInpChannel::ClearFlags(
  [in] int  Latched          // Latched status flags to be cleared
);
```

Function Arguments

Latched

Latched status flag(s) to be cleared. Multiple flags can be cleared with one method call by OR-ing the bit positions to be cleared. The following flags are latched and can be cleared:

Value	Meaning
DTAPI_RX_FIFO_OVF	See **GetFlags**
DTAPI_RX_SYNC_ERR	,, ,,
DTAPI_RX_RATE_OVF	,, ,,
DTAPI_RX_TARGET_ERR	,, ,,

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Flag(s) have been cleared successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

Some status flags that are queried with **GetFlags** are not latched and therefore cannot be cleared.

The latched status flags are also automatically reset after attaching and after **Reset**.

DtInpChannel::Detach

Detach input-channel object from hardware function and free associated resources.

```
DTAPI_RESULT DtInpChannel::Detach(
  [in] int  DetachMode       // How to detach
);
```

Function Arguments

DetachMode

Specifies how the channel object is detached from the hardware function.
If *DetachMode* is 0, the object is detached without further action. Other modes are defined below.

Value	Meaning
DTAPI_INSTANT_DETACH	Clear the contents of the receive FIFO and set receive control to **DTAPI_RXCTRL_IDLE**

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Channel object has been detached successfully from the hardware function
DTAPI_E_INVALID_FLAGS	An invalid detach flag was specified
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function, so it cannot be detached

Remarks

DtInpChannel::DetectIoStd

For ASI/SDI input channels: Detect whether the input signal is ASI or SDI.

```
DTAPI_RESULT DtInpChannel::DetectIoStd(
 [out] int&  Value           // DTAPI_IOCONFIG_ASI or DTAPI_IOCONFIG_SDI
 [out] int&  SubValue        // For SDI: specific SDI standard
);
```

Function Arguments

Value

Indicates whether a DVB-ASI or an SDI signal was received.

Value	Meaning
DTAPI_IOCONFIG_ASI	A valid DVB-ASI signal is being received
DTAPI_IOCONFIG_SDI	A valid SDI signal is being received

SubValue

If an SDI signal was received, *SubValue* receives the specific SDI standard, for example **DTAPI_IOCONFIG_525I59_94**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The I/O standard was detected successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

This routine may take considerable time to complete the detection process.

DtInpChannel::GetConstellationPoints

Get an array of constellation points.

```
DTAPI_RESULT DtInpChannel::GetConstellationPoints(
  [in] int  NumPoints,            // Number of points to get
 [out] DtConstelPoint*  pPoints   // Array with constellation points
);
```

Function Arguments

NumPoints

Specifies the number of constellation points to be read. The caller-supplied *pPoints* array must be able to accommodate at least *NumPoints* entries. A typical number of constellation points to read are 32.

pPoints

Pointer to a caller-supplied array of **DtConstelPoint** entries to receive the constellation points. The table below indicates the valid ranges for the constellation point x- and y-axis per device.

Device	Valid Range for X, Y	#Bits
DTU-234	0 ... 255	8
DTU-235	0 ... 1023	10
DTA-2135	0 ... 1023	10
DTA-2136	0 ... 65535	16
DTA-2137	0 ... 255	16

Result

DTAPI_RESULT	Meaning
DTAPI_OK	A valid set of constellation points has been returned
DTAPI_E_INVALID_BUF	The *pPoints* pointer is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	This functionality is not supported by the hardware or DTAPI

Remarks

DtInpChannel::GetDemodControl

Get modulation parameters as detected by the demodulator.

```
DTAPI_RESULT DtInpChannel::GetDemodControl (
 [out] DtDemodPars*  pDemodPars    // Receives the demodulation parameters
);
DTAPI_RESULT DtInpChannel::GetDemodControl(
 [out] int&  ModType,              // Modulation type
 [out] int&  ParXtra0,             // Extra parameter #0
 [out] int&  ParXtra1,             // Extra parameter #1
 [out] int&  ParXtra2              // Extra parameter #2
);
```

Function Arguments

pDemodPars

Receives the demodulation parameters. The user must have allocated the **DtDemodPars** structure. See class **DtDemodPars** for possible demodulation parameters.

ModType, ParXtra0, ParXtra1, ParXtra2

Modulation parameters as detected by the demodulator. See the tables on the following pages for a detailed specification of each parameter, per DekTec board type and firmware version.

Detailed Parameter Descriptions

Page	Modulation Type
140	Overview
141	ATSC
142	DVB-S
143	DVB-S.2
145	DVB-T
147	QAM

Page	Modulation Type

Modulation Types

ModType

Detected modulation type. The value **DTAPI_MOD_TYPE_UNK** indicates that the modulation type could not be detected.

L-Band

ModType	Meaning	Available on
DTAPI_MOD_DVBS_QPSK	DVB-S, QPSK	DTA-2137
DTAPI_MOD_DVBS2_8PSK	DVB-S.2, 8-PSK	DTA-2137
DTAPI_MOD_DVBS2_16APSK	DVB-S.2, 16-APSK	DTA-2137 (only in APSK-mode)
DTAPI_MOD_DVBS2_32APSK	DVB-S.2, 32-APSK	DTA-2137 (only in APSK-mode)
DTAPI_MOD_DVBS2_QPSK	DVB-S.2, QPSK	DTA-2137

VHF / UHF

ModType	Meaning	Available on
DTAPI_MOD_ATSC	ATSC VSB	DTU-234, DTA-236
DTAPI_MOD_DVBT	DVB-T	DTU-235, DTA-2135
DTAPI_MOD_QAM16	16-QAM	ITU-T J.83 Annex A/C: DTU-236, DTA-2136 ITU-T J.83 Annex B: DTU-234, DTU-236, DTA-2136
DTAPI_MOD_QAM32	32-QAM	
DTAPI_MOD_QAM64	64-QAM	
DTAPI_MOD_QAM128	128-QAM	
DTAPI_MOD_QAM256	256-QAM	

Modulation Mode: ATSC

ModType

ModType	Meaning	Available on
DTAPI_MOD_ATSC	ATSC	DTU-234, DTA-236

ParXtra0

Extra modulation parameter #0 specifies the VSB constellation.

ParXtra0	Meaning	Symbol Rate (bd)	TS Rate (bps)
DTAPI_MOD_ATSC_VSB8	8-VSB	10,762,238	19,392,658
DTAPI_MOD_ATSC_VSB16	16-VSB	10,762,238	38,785,317
DTAPI_MOD_ATSC_VSB_UNK	Constellation is unknown		
DTAPI_MOD_ATSC_VSB_MSK	AND-mask for ATSC constellation field		

ParXtra1, ParXtra2

Not used in ATSC modulation.

Modulation Mode: DVB-S

ParXtra0

Extra modulation parameter #0

ParXtra0	Meaning
DTAPI_MOD_1_2	Code rate 1/2
DTAPI_MOD_2_3	Code rate 2/3
DTAPI_MOD_3_4	Code rate 3/4
DTAPI_MOD_4_5	Code rate 4/5
DTAPI_MOD_5_6	Code rate 5/6
DTAPI_MOD_6_7	Code rate 6/7
DTAPI_MOD_7_8	Code rate 7/8
DTAPI_MOD_CR_UNK	Code rate is unknown

ParXtra1

Extra modulation parameter #1 encodes spectral inversion yes/no

Spectral Inversion

ParXtra1	Meaning
DTAPI_MOD_S_S2_SPECNONINV	No spectrum inversion detected
DTAPI_MOD_S_S2_SPECINV	Spectrum inversion detected
DTAPI_MOD_S_S2_SPECINV_UNK	Spectrum inversion status is unknown
DTAPI_MOD_S_S2_SPECINV_MSK	AND-mask for this field

ParXtra2

The detected symbol rate (in bd). The value **DTAPI_MOD_SYMRATE_UNK** indicates the symbol rate could not be detected.

Modulation Mode: DVB-S.2

ParXtra0

Extra modulation parameter #0 encodes the code rate.

ParXtra0	Meaning
DTAPI_MOD_1_2	Code rate 1/2
DTAPI_MOD_1_3	Code rate 1/3
DTAPI_MOD_1_4	Code rate 1/4
DTAPI_MOD_2_3	Code rate 2/3
DTAPI_MOD_2_5	Code rate 2/5
DTAPI_MOD_3_4	Code rate 3/4
DTAPI_MOD_3_5	Code rate 3/5
DTAPI_MOD_4_5	Code rate 4/5
DTAPI_MOD_5_6	Code rate 5/6
DTAPI_MOD_6_7	Code rate 6/7
DTAPI_MOD_7_8	Code rate 7/8
DTAPI_MOD_8_9	Code rate 8/9
DTAPI_MOD_9_10	Code rate 9/10
DTAPI_MOD_CR_UNK	Code rate is unknown

ParXtra1

Extra modulation parameter #1 encodes spectral inversion yes/no, pilots yes/no and long/short FEC frame.

Spectral Inversion

See Spectral Inversion section for DVB-S

Pilots

Value	Meaning
DTAPI_MOD_S2_NOPILOTS	Pilots disabled
DTAPI_MOD_S2_PILOTS	Pilots enabled
DTAPI_MOD_S2_PILOTS_UNK	Pilots status is unknown
DTAPI_MOD_S2_PILOTS_MSK	AND-mask for this field

Long or Short FECFRAME

Value	Meaning
DTAPI_MOD_S2_SHORTFRM	Short FECFRAME (16.200 bits)
DTAPI_MOD_S2_LONGFRM	Long FECFRAME (64.800 bits)

DTAPI_MOD_S2_FRM_UNK	Frame size is unknown
DTAPI_MOD_S2_FRM_MSK	AND-mask for this field

ParXtra2

The detected symbol rate in baud. The value **DTAPI_MOD_SYMRATE_UNK** indicates the symbol rate could not be detected.

Modulation Mode: DVB-T

ParXtra0

Extra modulation parameter #0 is the code rate.

Value	Meaning
DTAPI_MOD_1_2	Code rate 1/2
DTAPI_MOD_2_3	Code rate 2/3
DTAPI_MOD_3_4	Code rate 3/4
DTAPI_MOD_5_6	Code rate 5/6
DTAPI_MOD_7_8	Code rate 7/8
DTAPI_MOD_CR_UNK	Code rate is unknown

ParXtra1

Extra modulation parameter #1 is the OR of values for the following fields: Bandwidth, Constellation, Guard Interval, Interleaving and Transmission Mode.

Bandwidth

Value	Meaning
DTAPI_MOD_DVBT_6MHZ	6 MHz
DTAPI_MOD_DVBT_7MHZ	7 MHz
DTAPI_MOD_DVBT_8MHZ	8 MHz
DTAPI_MOD_DVBT_BW_UNK	Bandwidth is unknown
DTAPI_MOD_DVBT_BW_MSK	AND mask

Constellation

Value	Meaning
DTAPI_MOD_DVBT_QPSK	QPSK
DTAPI_MOD_DVBT_QAM16	16-QAM
DTAPI_MOD_DVBT_QAM64	64-QAM
DTAPI_MOD_DVBT_CO_UNK	Constellation is unknown
DTAPI_MOD_DVBT_CO_MSK	AND mask

Guard Interval

Value	Meaning
DTAPI_MOD_DVBT_G_1_32	1/32
DTAPI_MOD_DVBT_G_1_16	1/16
DTAPI_MOD_DVBT_G_1_8	1/8
DTAPI_MOD_DVBT_G_1_4	1/4

DTAPI_MOD_DVBT_CO_UNK	Guard interval is unknown
DTAPI_MOD_DVBT_GU_MSK	AND mask

Interleaving

Value	Meaning
DTAPI_MOD_DVBT_INDEPTH	In-depth interleaver (2k, 4k)
DTAPI_MOD_DVBT_NATIVE	Native interleaver
DTAPI_MOD_DVBT_IL_UNK	Interleaving is unknown
DTAPI_MOD_DVBT_IL_MSK	AND mask

Transmission Mode

Value	Meaning
DTAPI_MOD_DVBT_2K	2k mode
DTAPI_MOD_DVBT_8K	8k mode
DTAPI_MOD_DVBT_MD_UNK	Transmission mode is unknown
DTAPI_MOD_DVBT_MD_MSK	AND mask

ParXtra2

Not used for DVB-T

Modulation Mode: QAM

ModType

The QAM constellation is encoded in *ModType* according to the following table.

ModType	Meaning	Available on
DTAPI_MOD_QAM16	16-QAM	ITU-T J.83 Annex A/C: DTU-236, DTA-2136 ITU-T J.83 Annex B: DTU-234, DTU-236, DTA-2136
DTAPI_MOD_QAM32	32-QAM	
DTAPI_MOD_QAM64	64-QAM	
DTAPI_MOD_QAM128	128-QAM	
DTAPI_MOD_QAM256	256-QAM	

ParXtra0

Extra modulation parameter #0 is the ITU-T J.83 Annex.

ITU-T J.83 Annex	Meaning	Available on
DTAPI_MOD_J83_A	J.83 annex A (DVB-C)	DTA-236, DTA-2136
DTAPI_MOD_J83_B	J.83 annex B ("American QAM")	DTU-234, DTA-236, DTA-2136
DTAPI_MOD_J83_C	J.83 annex C ("Japanese QAM")	DTA-236, DTA-2136
DTAPI_MOD_J83_UNK	Annex is unknown	

ParXtra1

For J.83 Annex B, this parameter specifies the interleaving mode detected as specified in the table below. For Annex A and C this parameter is not used.

Value	CW	I	J	Burst protection 64-/256-QAM
DTAPI_MOD_QAMB_I128_J1D	0001	128	1	95 µs / 66 µs
DTAPI_MOD_QAMB_I64_J2	0011	64	2	47 µs / 33 µs
DTAPI_MOD_QAMB_I32_J4	0101	32	4	24 µs / 16 µs
DTAPI_MOD_QAMB_I16_J8	0111	16	8	12 µs / 8.2 µs
DTAPI_MOD_QAMB_I8_J16	1001	8	16	5.9 µs / 4.1 µs
DTAPI_MOD_QAMB_I128_J1	0000	128	1	95 µs / 66 µs
DTAPI_MOD_QAMB_I128_J2	0010	128	2	190 µs / 132 µs
DTAPI_MOD_QAMB_I128_J3	0100	128	3	285 µs / 198 µs
DTAPI_MOD_QAMB_I128_J4	0110	128	4	379 µs / 264 µs
DTAPI_MOD_QAMB_I128_J5	1000	128	5	474 µs / 330 µs
DTAPI_MOD_QAMB_I128_J6	1010	128	6	569 µs / 396 µs
DTAPI_MOD_QAMB_I128_J7	1100	128	7	664 µs / 462 µs
DTAPI_MOD_QAMB_I128_J8	1110	128	8	759 µs / 528 µs

DTAPI_MOD_QAMB_IL_UNK	-	-	-	Interleaving mode is unknown

ParXtra2

The detected symbol rate (in bd). The value **DTAPI_MOD_SYMRATE_UNK** indicates the symbol rate could not be detected.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The modulation parameters have been returned successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	This function is not supported for the underlying hardware

Remarks

The detected modulation parameters returned by this method should only be considered to be valid if the **DtInpChannel** indicates that receiver- and FEC-lock has been achieved.

DtInpChannel::GetDescriptor

Get hardware function descriptor for this input channel.

```
DTAPI_RESULT DtInpChannel::GetDescriptor(
 [out] DtHwFuncDesc&  HwFuncDesc  // Hardware function descriptor
);
```

Function Arguments

HwFuncDesc

Output argument that receives the hardware function descriptor.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The hardware function descriptor been retrieved successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

DtInpChannel::GetFifoLoad

Get the current load of the channel's receive FIFO.

```
DTAPI_RESULT DtInpChannel::GetFifoLoad(
  [out]int&  FifoLoad        // Load of receive FIFO
);
```

Function Arguments

FifoLoad

This output argument is set to the number of bytes in the receive FIFO.

Result

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

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

The value retrieved with this method call approximates the load of the Receive FIFO. Some additional data bytes may be buffered on the device.

If a transfer is in progress and/or the device receives data, then every call to **GetFifoLoad** may return a different value.

DtInpChannel::GetFlags

Get current and latched value of the input channel's status flags.

```
DTAPI_RESULT DtInpChannel::GetFlags(
 [out] int&  Flags,          // Status flags
 [out] int&  Latched         // Latched status flags
);
```

Function Arguments

Flags

Output argument that is set to the current value of the input-channel status flags. Each status flag is represented by one bit. Multiple status flags can be true simultaneously. If none of the status flags is true, *Flags* is set to zero.

Value	Meaning
DTAPI_RX_FIFO_OVF	A receive FIFO overflow condition has occurred. The data in the receive FIFO was not read fast enough to a system buffer.
DTAPI_RX_SYNC_ERR	A synchronisation error has occurred in the synchronisation stage of the input channel. This error cannot occur in packet mode **DTAPI_RXMODE_RAW**.
DTAPI_RX_RATE_OVF	Data is entering the receive FIFO faster than 150Mbps (DTA-122 only)
DTAPI_RX_TARGET_ERR	The target adapter signals a fault (DTA-122 only)
DTAPI_RX_LINK_ERR	The communication link with the device is broken (DTE-31XX devices only)
DTAPI_RX_DATA_ERR	Data is lost during transfer to a system buffer (DTE-31XX devices only)

Latched

Output argument that is set to the latched value of the status flags: On a '0' to '1' transition of a status flag, the corresponding bit in *Latched* is set to '1'. The bit remains set until cleared explicitly by one of the following DTAPI-calls: **ClearFlags**, **AttachToPort** or **Reset**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Status flags have been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

DtInpChannel::GetIoConfig

Get the I/O configuration of the physical port attached to the input channel. This is the same function as **DtDevice::GetIoConfig** applied to the physical port corresponding to this channel.

```
DTAPI_RESULT DtInpChannel::GetIoConfig(
  [in] int  Group,            // I/O configuration group
 [out] int&  Value            // I/O configuration value
);
DTAPI_RESULT DtInpChannel::GetIoConfig(
  [in] int  Group,            // I/O configuration group
 [out] int&  Value,           // I/O configuration value
 [out] int&  SubValue         // I/O configuration subvalue
);
DTAPI_RESULT DtInpChannel::GetIoConfig(
  [in] int  Group,            // I/O configuration group
 [out] int&  Value,           // I/O configuration value
 [out] int&  SubValue,        // I/O configuration subvalue
 [out] __int64&  ParXtra0     // Extra parameter #0
);
DTAPI_RESULT DtInpChannel::GetIoConfig(
  [in] int  Group,            // I/O configuration group
 [out] int&  Value,           // I/O configuration value
 [out] int&  SubValue,        // I/O configuration subvalue
 [out] __int64&  ParXtra0,    // Extra parameter #0
 [out] __int64&  ParXtra1     // Extra parameter #1
);
```

Function Arguments

Group, Value, SubValue, ParXtra0, ParXtra1

I/O configuration parameters, see **DtDevice::GetIoConfig**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	I/O configuration has been retrieved successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
Other result codes	See **DtDevice::GetIoConfig**

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

DtInpChannel::GetIpPars

Get TS-over-IP parameters of the received stream as detected by the driver.

```
DTAPI_RESULT DtInpChannel::GetIpPars(
  [in] DtTsIpPars*  pTsIpPars   // Receives the TS-over-IP parameters
);
```

Function Arguments

pTsIpPars

Receives the TS-over-IP parameters. The user must have allocated the **DtTsIpPars** structure.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The TS-over-IP parameters have been retrieved successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	Channel is not a TS-over-IP channel

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

DtInpChannel::GetIpStat

Get IP statistics from the network driver.

```
DTAPI_RESULT DtInpChannel::GetIpStat(
  [in] DtTsIpStat*  pTsIpStat   // Receives the IP parameters
);
```

Function Arguments

pTsIpStat

Receives the IP statistics. The user must have allocated the **DtTsIpStat** structure.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The TS-over-IP statistics have been retrieved successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	Channel is not a TS-over-IP channel

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

DtInpChannel::GetMaxFifoSize

Get the maximum size of the channel's receive FIFO.

```
DTAPI_RESULT DtInpChannel::GetMaxFifoSize(
  [out]int&  MaxFifoSize     // Maximum size of FIFO in bytes
);
```

Function Arguments

MaxFifoSize

Maximum size of the receive FIFO in number of bytes.

Result

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

Remarks

DtInpChannel::GetRxClkFreq

DTA-2142 only. Get the frequency of the DVB-SPI clock.

```
DTAPI_RESULT DtInpChannel::GetRxClkFreq(
  [out]int&  RxClkFreq       // Frequency of the DVB-SPI clock in Hertz
);
```

Function Arguments

MaxFifoSize

Output argument that is set to a measurement of the DVB-SPI clock frequency.

Result

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

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive*=*true*).

DtInpChannel::GetRxControl

Get the current value of receive control.

```
DTAPI_RESULT DtInpChannel::GetRxControl(
 [out] int&  RxControl       // Receive control
);
```

Function Arguments

RxControl

This argument is set to the current value of receive control: **DTAPI_RXCTRL_IDLE** or **DTAPI_RXCTRL_RCV**.

Refer to **SetRxControl** for a description of the receive control values.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Receive control has been successfully retrieved
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive*=*true*).

Transport Streams – In receive modes **DTAPI_RXMODE_ST188**, **DTAPI_RXMODE_STMP2** and **DTAPI_RXMODE_ST204**, receive control is synchronised to packet boundaries. For example, if **SetRxControl** is used to change the control state from **DTAPI_RXCTRL_RCV** to **DTAPI_RXCTRL_IDLE**, and **GetRxControl** is called immediately thereafter, then **DTAPI_RXCTRL_RCV** may be returned. Only when a new packet enters the Receive FIFO, the value returned by **GetRxControl** becomes **DTAPI_RXCTRL_IDLE**.

SDI – Receive-control state is synchronised to the vertical sync.

In receive mode **DTAPI_RXMODE_STRAW**, method **GetRxControl** always returns the receive-control state set by **SetRxControl**.

DtInpChannel::GetRxMode

Get the current value of receive mode.

```
DTAPI_RESULT DtInpChannel::GetRxMode(
 [out] int&  RxMode          // Receive mode
);
```

Function Arguments

RxControl

This argument is set to the current value of receive mode.
Refer to **SetRxMode** for a description of the receive modes.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Receive mode has been successfully retrieved
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive*=*true*).

DtInpChannel::GetStatistics

Get statistics information from demodulator. This function gets an array of **DtStatistic** structures so that multiple statistics can be retrieved in one call.

```
DTAPI_RESULT DtInpChannel::GetStatistics(
  [in] int  Count,                  // Number of statistics
 [out] DtStatistic*  pStatistics    // Array with statistics
);
```

Function Arguments

Statistic

The integer or double value of the requested statistic

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Statistics information has been read successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORT	Requested statistic is not supported by the hardware or the statistic is not supported in the requested return type (i.e. integer or double).
DTAPI_E_INVALID_MODE	The **ErrorStatsMode** is not set to the corresponding mode.
DTAPI_E_INVALID_MODTYPE	The received signal does

Remarks

DtInpChannel::GetStatus

Get status information from the input channel. If a device does not support a certain feature, the corresponding status variable is set to **DTAPI_NOT_SUPPORTED** (-1).

```
DTAPI_RESULT DtInpChannel::GetStatus(
 [out] int&  PacketSize,     // Packet size
 [out] int&  NumInv,         // #Invalid bytes per packet (DTA-122)
 [out] int&  ClkDet,         // Clock- or carrier detected
 [out] int&  AsiLock,        // DVB-ASI PLL locked (ASI inputs)
                             // SDI genlocked (SDI genlock inputs)
 [out] int&  RateOk,         // Input rate above minimum
 [out] int&  AsiInv          // Input-invert status (ASI inputs)
);
```

Function Arguments

PacketSize

MPEG mode: Size of incoming MPEG-2 transport packets.

Value	Meaning
DTAPI_PCKSIZE_188	188-byte packets at the transport-stream input
DTAPI_PCKSIZE_204	204-byte packets at the transport-stream input
DTAPI_PCKSIZE_INV	No MPEG-2 compliant packets found at the transport-stream input

SDI mode: SDI video standard of incoming stream

Value	Meaning
DTAPI_SDIMODE_525	525-line video mode input
DTAPI_SDIMODE_625	625-line video mode input
DTAPI_SDIMODE_INV	No valid SDI signal detected on the input

NumInv

Defined for DVB-SPI input channels (DTA-122) only: Number of "invalid" bytes (DVALID input signal is '0') per packet.

Value	Meaning
DTAPI_NUMINV_NONE	No invalid bytes
DTAPI_NUMINV_16	16 invalid bytes per packet
DTAPI_NUMINV_OTHER	Other number of invalid bytes per packet
DTAPI_NOT_SUPPORTED	Device does not support this parameter (not DTA-122)

ClkDet

For DVB-SPI input channels, this output argument indicates whether a receive clock of sufficient frequency is detected at the SPI input.

For DVB-ASI and SDI input channels, this output argument acts as a *Carrier Detect* signal.

Value	Meaning
DTAPI_CLKDET_OK	DVB-SPI : Receive clock detected DVB-ASI, SDI : Carrier detected TS-over-IP : IP traffic detected in the last second
DTAPI_CLKDET_FAIL	DVB-SPI : No receive clock detected, or receive-clock rate is too low DVB-ASI, SDI : No carrier detected TS-over-IP : No IP traffic in the last second

AsiLock

For DVB-ASI input channels, this output argument indicates whether the DVB-ASI clock signal can be recovered reliably.

Value	Meaning
DTAPI_ASI_INLOCK	PLL is locked to the incoming DVB-ASI input signal
DTAPI_ASI_NOLOCK	Clock signal cannot be recovered from the input signal
DTAPI_NOT_SUPPORTED	Hardware function does not support this parameter

For ports configured as SDI genlock input port, this output argument indicates whether the genlock circuitry is locked to the provided SDI genlock signal.

Value	Meaning
DTAPI_GENLOCK_INLOCK	The SDI genlock circuitry is locked to the incoming SDI input signal
DTAPI_GENLOCK_NOLOCK	The SDI genlock circuitry is NOT locked to the incoming SDI input signal
DTAPI_NOT_SUPPORTED	Hardware function does not support this parameter

RateOk

Defined for DVB-ASI input channels only: Output argument that indicates whether the transport rate at the DVB-ASI input is sufficiently high for further processing. When this parameter is set to **DTAPI_INPRATE_LOW**, the most likely cause is an "empty" DVB-ASI signal (stuffing symbols only).

Value	Meaning
DTAPI_INPRATE_OK	The DVB-ASI input rate is sufficient
DTAPI_INPRATE_LOW	The DVB-ASI input rate is too low (<900 bps)
DTAPI_NOT_SUPPORTED	Hardware function does not support this parameter

AsiInv

Defined for DVB-ASI input channels only: This argument indicates whether the input circuitry is currently inverting the DVB-ASI input signal. This is most useful when polarity control has been set to **DTAPI_POLARITY_AUTO**; In the other polarity-control settings, *AsiInv* just echoes the value of argument *PolarityControl* in the call to **PolarityControl**.

Value	Meaning
DTAPI_ASIINV_NORMAL	Polarity of DVB-ASI input signal is normal (not inverted)
DTAPI_ASIINV_INVERT	Polarity of DVB-ASI signal is inverted
DTAPI_NOT_SUPPORTED	Device does not support this parameter

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Status information has been read successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

DtInpChannel::GetStreamSelection

Get the selection parameters for the currently selected PLP.

```
DTAPI_RESULT DtInpChannel::GetStreamSelection(
 [out] DtDvbC2StreamSelPars&  StreamSel    // PLP selection parameters
);
DTAPI_RESULT DtInpChannel::GetStreamSelection(
 [out] DtDvbT2StreamSelPars&  StreamSel    // PLP selection parameters
);
```

Function Arguments

StreamSel

Output argument that receives the selection parameters.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The selection parameters have been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_MODE	The current demodulation type does not correspond to the type of the PLP selection parameters
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The device does not support DVB-C2 or DVB-T2

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive*=true).

DtInpChannel::GetTargetId

Get the target-adapter identifier (DTA-122 only).

```
DTAPI_RESULT DtInpChannel::GetTargetId(
 [out] int&  Present,        // Target adapter present?
 [out] int&  TargetId        // Target-adapter identifier
);
```

Function Arguments

Present

Output argument that indicates whether a target adapter has been detected.

Value	Meaning
DTAPI_NO_CONNECTION	Nothing is connected to the input connector of the DTA-122.
DTAPI_DVB_SPI_SOURCE	A standard DVB-SPI source is connected to the DTA-122.
DTAPI_TARGET_PRESENT	A target adapter is present.
DTAPI_TARGET_UNKNOWN	The device is busy assessing the situation on the input connector.

TargetId

Output argument that is set to an integer value that uniquely identifies the target adapter. Please refer to the DTA-122 documentation for a list of available target adapters.

TargetId is assigned a value only if *Present* is **DTAPI_TARGET_PRESENT**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Target ID has been retrieved successfully.
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver.
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function.
DTAPI_E_NOT_SUPPORTED	The input channel does not support target adapters (DVB-ASI input channels: DTA-120/140, DTU-225).

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive*=*true*).

The DTA-122 does not recognize the DTA-102 as a standard DVB-SPI source (*Present* is set to **DTAPI_NO_CONNECTION**), unless the ground pins on the DVB-SPI cable are connected together. This is due to the target-adapter detection circuitry.

DtInpChannel::GetTsRateBps

Get a measurement of the input's transport stream rate.

```
DTAPI_RESULT DtInpChannel::GetTsRateBps(
 [out] int&  TsRate          // transport-stream rate in bps
);
```

Function Arguments

TsRate

Measurement of the current transport stream rate, expressed in bits per second. This rate does not take into account 'extra' bytes beyond the 188 MPEG-2 defined bytes.
If the channel's receive mode is **DTAPI_RXMODE_STRAW**, the value returned by this method is equal to the raw input bit rate, this is the rate at which valid data enters the device.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	transport-stream rate has been read successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

In all receive modes except **DTAPI_RXMODE_STRAW** this method strictly applies the definition of transport-stream rate in the MPEG-2 Systems specification. This rate is based on 188-byte packets. If the packet size is not 188 bytes, a conversion factor is used.

Example: When 204-byte packets enter the system, the raw input rate is divided by 204/188.

DtInpChannel::GetTunerFrequency

Get current tuner frequency.

```
DTAPI_RESULT DtInpChannel::GetTunerFrequency(
 [out] __int64&  FreqHz      // Frequency in hertz
);
```

Function Arguments

FreqHz

Current tuning frequency (in Hz)

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The tuner frequency has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The device does not include a tuner

Remarks

DtInpChannel::GetViolCount

Get number of code violations on a DVB-ASI input channel.

```
DTAPI_RESULT DtInpChannel::GetViolCount(
 [out] int&  ViolCount       // Code-violation counter
);
```

Function Arguments

ViolCount

Total number of DVB-ASI code violations since power-up of a DVB-ASI input channel.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Code violation count has been read successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

A code violation is a bit error that leads to an illegal 8B/10B code (the line code used by DVB-ASI). Bit errors may be caused by poor cable quality, or by an input cable that is too long.

The value of this counter is updated about 20 times per second. The counter is only incremented and never reset. When the largest positive 32-bit integer value (2^{31-1}) has been reached, the counter wraps around to the largest 32-bit negative integer value (-2^{31}).

Connecting or disconnecting the cable to/from a DVB-ASI input channel may cause a massive amount of code violations. This is “normal” behaviour, caused by the locking process of the DVB-ASI input circuitry.

DtInpChannel::I2CLock

Lock the I2C bus for exclusive access.

```
DTAPI_RESULT DtInpChannel::I2Clock(
  [in] int  Timeout          // Maximum time to wait for lock
);
```

Function Arguments

Timeout

Maximum time (in ms) to wait for the I2C lock. The value -1 indicates an infinite wait time.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	I2C lock has successfully been obtained
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to device hardware
DTAPI_E_NOT_SUPPORTED	Locking of the I2C bus is not supported on this device
DTAPI_E_TIMEOUT	Lock could not be obtained within the specified timeout

Remarks

It is recommended to never hold the lock for longer than 2 seconds as locking the I2C bus for prolonged periods of time can result in serious degradation of the performance of the device and can even result in loss of functionality.

DtInpChannel::I2CRead

Read data from the I2C bus.

```
DTAPI_RESULT DtInpChannel::I2CRead(
  [in] int   DvcAddr,          // I2C device address
 [out] char*  pBuffer,         // Buffer for receiving the data
  [in] int   NumBytesToRead   // #Bytes to read
);
```

Function Arguments

DvcAddr

Device address of the targeted I2C device.

The I2C device address consists out of 1 transfer direction bit + 7 address bits. This method ignores the transfer bit (LSB) and only used the 7 address bits. Valid values for the device address are: 0x00h-0xFF

pBuffer

Pointer to a buffer for receiving the I2C bytes.

The buffer must be caller-allocated and have a size of at least `NumBytesToRead`.

NumBytesToRead

Number of bytes to read.

Maximum allowed number of bytes to read is 512.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Sample has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to device hardware
DTAPI_E_INVALID_BUF	Invalid buffer pointer provided
DTAPI_E_INVALID_SIZE	Invalid number of bytes to read specified (i.e. >512 bytes)
DTAPI_E_NOT_SUPPORTED	The device does not support reading of the on board I2C bus

Remarks

The **I2CRead** method is intended for direct low-level access to the on board I2C resources.

DtInpChannel::I2CUnlock

Release the lock (i.e. exclusive access) on the I2C bus.

```
DTAPI_RESULT DtInpChannel::I2CUnlock(void);
```

Result

DTAPI_RESULT	Meaning
DTAPI_OK	I2C lock has successfully been released
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to device hardware
DTAPI_E_NOT_SUPPORTED	Unlocking of the I2C bus is not supported on this device

Remarks

DtInpChannel::I2CWrite

Write data to the I2C bus.

```
DTAPI_RESULT DtInpChannel::I2CWrite(
  [in] int  DvcAddr,          // I2C device address
  [in] char*  pBuffer,        // Buffer with bytes to write
  [in] int  NumBytesToWrite // #Bytes to write
);
```

Function Arguments

DvcAddr

Device address of the targeted I2C device
The I2C device address consists out of 1 transfer direction bit + 7 address bits. This method ignores the transfer bit (LSB) and only used the 7 address bits. Valid values for the device address are: 0x00h-0xFF

pBuffer

Pointer to a buffer with the bytes to write.

The buffer must have a size of at least *NumBytesToWrite*.

NumBytesToWrite

Number of bytes to write.

Maximum allowed number of bytes to write is 512.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Sample has been retrieved successfully.
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver.
DTAPI_E_NOT_ATTACHED	Channel object is not attached to device hardware.
DTAPI_E_INVALID_BUF	Invalid buffer pointer provided
DTAPI_E_INVALID_SIZE	Invalid number of bytes to write specified (i.e. >512 bytes)
DTAPI_E_NOT_SUPPORTED	The device does not support writing to the on board I2C bus.

Remarks

The **I2CWrite** method is intended for direct low-level access to the on board I2C resources.

DtInpChannel::I2CWriteRead

Lock the I2C bus followed by a write and/or read action on the I2C bus and finally release the lock.

```
DTAPI_RESULT DtInpChannel::I2CWrite(
  [in] int   DvcAddrWrite,    // I2C device address to write to
  [in] char*  pBufferWrite,  // Buffer with bytes to write
  [in] int   NumBytesToWrite,// #Bytes to write
  [in] int   DvcAddrRead,     // I2C device address to read from
 [out] char*  pBufferRead,   // Buffer for receiving the data
  [in] int   NumBytesToRead   // #Bytes to read
);
```

Function Arguments

DvcAddrWrite

Device address of the targeted I2C device to write to.
The I2C device address consists out of 1 transfer direction bit + 7 address bits. This method ignores the transfer bit (LSB) and only used the 7 address bits. Valid values for the device address are: 0x00h-0xFF

pBufferWrite

Pointer to a buffer with the bytes to write.
The buffer must have a size of at least *NumBytesToWrite*.

NumBytesToWrite

Number of bytes to write.
Maximum allowed number of bytes to write is 512.

DvcAddrRead

Device address of the targeted I2C device to read from.
The I2C device address consists out of 1 transfer direction bit + 7 address bits. This method ignores the transfer bit (LSB) and only used the 7 address bits. Valid values for the device address are: 0x00h-0xFF

pBufferRead

Pointer to a buffer for receiving the I2C bytes.
The buffer must be caller-allocated and have a size of at least *NumBytesToRead*.

NumBytesToRead

Number of bytes to read.
Maximum allowed number of bytes to read is 512.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Sample has been retrieved successfully.
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver.
DTAPI_E_NOT_ATTACHED	Channel object is not attached to device hardware.
DTAPI_E_INVALID_BUF	Invalid buffer pointer provided
DTAPI_E_INVALID_SIZE	Invalid number of bytes to write specified (i.e. >512 bytes)
DTAPI_E_NOT_SUPPORTED	The device does not support writing to the on board I2C bus.

Remarks

The **I2CWriteRead** method is intended for direct low-level access to the on board I2C resources.

DtInpChannel::LedControl

Take direct control of input-status LED, or let hardware drive the LED.

```
DTAPI_RESULT DtInpChannel::LedControl(
  [in] int  LedControl       // DTAPI_LED_XXX
);
```

Function Arguments

LedControl

Controls the LED.

Value	Meaning
DTAPI_LED_HARDWARE	Hardware drives the LED (default after power-up)
DTAPI_LED_OFF	LED is forced to off-state
DTAPI_LED_GREEN	LED is forced to green-state
DTAPI_LED_RED	LED is forced to red-state
DTAPI_LED_YELLOW	LED is forced to yellow-state

Result

DTAPI_RESULT	Meaning
DTAPI_OK	LED setting has been accepted
DTAPI_E_INVALID_MODE	The specified LED-control value is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

Detaching the input channel releases any direct-control setting that might have been applied to the LEDs (LED control is reset to **DTAPI_LED_HARDWARE**).

Some devices have a single LED, which can be controlled by either **DtDevice::LedControl** or by **DtInpChannel::LedControl**. If both methods are used at the same time, then **DtDevice::LedControl** takes precedence over **DtInpChannel::LedControl**.

DtInpChannel::LnbEnable

For satellite receivers (DTA-2137, DTE-3137): Enable or disable the LNB controller.

```
DTAPI_RESULT DtInpChannel::LnbEnable(
  [in] bool  Enable          // Enable/disable controller
);
```

Function Arguments

Enable

If set true, the LNB controller will be enabled. If false the LNB controller is disabled.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	LNB controller has successfully been enabled or disabled
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INTERNAL	Unexpected internal DTAPI error encountered
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	This method is not supported by the underlying hardware

Remarks

DtInpChannel::LnbEnableTone

For satellite receivers (DTA-2137, DTE-3137): Enable or disable the 22kHz tone on the LNB.

```
DTAPI_RESULT DtInpChannel::LnbEnableTone(
  [in] bool  Enable          // Enable/disable 22kHz tone
);
```

Function Arguments

Enable

Enable (=true) or disable (=false) generation of 22 kHz tone.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The 22kHz tone has successfully been enabled or disabled
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INTERNAL	Unexpected internal DTAPI error encountered
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	This method is not supported by the underlying hardware

Remarks

Before calling this method the on-board LNB controller must have been enabled using **DtInpChannel::LnbEnable** method. If the LNB controller is disabled this method will fail.

DtInpChannel::LnbSendBurst

For satellite receivers (DTA-2137, DTE-3137): Transmit a tone burst of type A or B.

```
DTAPI_RESULT DtInpChannel::LnbSendBurst(
  [in] int  BurstType        // Burst type
);
```

Function Arguments

BurstType

Controls the burst type.

Value	Meaning
DTAPI_LNB_BURST_A	Burst type A
DTAPI_LNB_BURST_B	Burst type B

Result

DTAPI_RESULT	Meaning
DTAPI_OK	LNB burst has successfully been sent
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INTERNAL	Unexpected internal DTAPI error encountered
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	This method is not supported by the underlying hardware

Remarks

Before calling this method the on-board LNB controller must have been enabled using **DtInpChannel::LnbEnable** method. If the LNB controller is disabled this method will fail.

DtInpChannel::LnbSendDiseqcMessage

For satellite receivers (DTA-2137, DTE-3137): Send a DiSEqC message. There are two overloads: one with and one without capture of the DiSEqC reply.

```
DTAPI_RESULT DtInpChannel::LnbSendDiseqcMessage(
 [in] const unsigned char*  pMsgOut     // The message
 [in] int  NumBytesOut        // Size of output message
);
DTAPI_RESULT DtInpChannel::LnbSendDiseqcMessage(
 [in] const unsigned char*  pMsgOut
                              // Buffer with message
 [in] int  NumBytesOut        // Size of output buffer
 [in] unsigned char* pMsgIn   // Buffer for reply
 [in/out] int&  NumBytesIn    // Size of reply buffer / reply message
);
```

Function Arguments

pMsgOut

Pointer to buffer with the message to send. The maximum allowed message size is 8.

NumBytesOut

Number of bytes in the message buffer.

pMsgIn

Pointer to buffer in which the reply message is stored. The maximum reply size is 8 bytes.

NumBytesIn

As input argument this argument specifies the size of the reply buffer. As output argument this argument returns the number of bytes in the reply message.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	LNB message was successfully sent
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INTERNAL	Unexpected internal DTAPI error encountered
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	This method is not supported by the underlying hardware

Remarks

Before calling this method the on-board LNB controller must have been enabled using the **LnbEnable** method. If the LNB controller is disabled this method will fail.

DtInpChannel::LnbSetVoltage

For satellite receivers (DTA-2137, DTE-3137): Set the LNB voltage.

```
DTAPI_RESULT DtInpChannel::LnbSetVoltage(
  [in] int  Level            // Voltage level
);
```

Function Arguments

Level

Controls the LNB voltage.

Value	Meaning
DTAPI_LNB_13V	LNB voltage is 13V
DTAPI_LNB_14V	LNB voltage is 14V
DTAPI_LNB_18V	LNB voltage is 18V
DTAPI_LNB_19V	LNB voltage is 19V

Result

DTAPI_RESULT	Meaning
DTAPI_OK	LNB voltage has successfully been set
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INTERNAL	Unexpected internal DTAPI error encountered
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	This method is not supported by the underlying hardware

Remarks

The LNB voltage settings will only have effect if the LNB controller has been enabled using the **LnbEnable** method.

DtInpChannel::PolarityControl

Control the automatic polarity-detection circuitry of a DVB-ASI input channel.

```
DTAPI_RESULT DtInpChannel::PolarityControl(
  [in] int  PolarityControl  // Polarity-control setting
);
```

Function Arguments

PolarityControl

This argument controls inversion of the DVB-ASI signal.

Value	Meaning
DTAPI_POLARITY_AUTO	Automatically detect and correct the polarity
DTAPI_POLARITY_NORMAL	'Normal' operation: do not invert the DVB-ASI signal
DTAPI_POLARITY_INVERT	Invert DVB-ASI signal

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Polarity setting has been accepted
DTAPI_E_INVALID_MODE	The specified polarity-control value is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	Device is not a DVB-ASI device, or hardware does not support control of the polarity-detection process

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive*=*true*).

The DVB-ASI signal is sensitive to polarity. Without corrective measures, an inverted DVB-ASI signal – which may be caused by inverting distribution amplifiers – may be decoded incorrectly by a standard DVB-ASI receiver.

Automatic detection of DVB-ASI signal polarity (setting **DTAPI_POLARITY_AUTO**) can be successfully applied only when it is known a priori that the input signal is DVB/MPEG-2 compliant. For non MPEG-2 applications, *PolarityControl* should be set to **DTAPI_POLARITY_NORMAL**, or the input signal may be distorted badly due to periodic inversion.

Old revisions of the DTU-225 do not support these functions: These devices always operate as if *PolarityControl* is set to **DTAPI_POLARITY_NORMAL**.

DtInpChannel::Read

Read data bytes from the input channel.

```
DTAPI_RESULT DtInpChannel::Read(
  [in] char*  pBuffer,       // Buffer to store data
  [in] int  NumBytesToRead   // #Bytes to be read
);
DTAPI_RESULT DtInpChannel::Read(
  [in] char*  pBuffer,       // Buffer to store data
  [in] int  NumBytesToRead   // #Bytes to be read
  [in] int  TimeOut          // Maximum time to wait for data
);
```

Function Arguments

pBuffer

Pointer to the buffer into which the data bytes from the input channel will be written. The pointer must be aligned to a 32-bit word boundary, except for IP input channels for which there are no alignment restrictions.

NumBytesToRead

Transfer size: Number of bytes to be read from the input channel. The value of *NumBytesToWrite* must be a multiple of four, except for IP input channels, which can accept any positive value.

TimeOut

Transfer timeout: specifies the maximum time (in ms) to wait for the requested amount of data. This method will fail if the data cannot be read within the specified period. A value of 0 indicates that no time out applies and -1 specifies an infinite timeout.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Read operation has been completed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_BUF	The buffer is not aligned to a 32-bit word boundary
DTAPI_E_INVALID_SIZE	The specified transfer size is negative or not a multiple of 4
DTAPI_E_INVALID_TIMEOUT	Invalid timeout period specified
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_TIMEOUT	Read operation failed. Requested number of bytes could not be returned within the specified timeout period

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

Read returns when *NumBytesToRead* bytes have been transferred into the buffer. The thread executing **Read** will sleep until sufficient data has entered the receive FIFO to complete the transfer. If either the input signal disappears or receive control is **DTAPI_RXCTRL_IDLE**, the **Read** call may sleep for an indefinite period of time (the thread 'hangs').

There are two ways to avoid such a 'hanging' thread:

- Before calling **`Read`**, check the FIFO load. Read an amount of data that is less than or equal to the FIFO load;
- Use **`Read`** with a time out.

The first method should be used if maximum performance is required. The second method is easier to use at the expense of some extra CPU cycles.

DtInpChannel::ReadFrame

Read a single SDI frame from the input channel.

```
DTAPI_RESULT DtInpChannel::ReadFrame(
  [in] unsigned int*  pFrame,     // Buffer to receive the frame
[in/out] int&  FrameSize,         // [in] Size of frame buffer
                                  // [out] Number of bytes returned
  [in] int  Timeout               // Maximum time to wait for a frame
);
```

Function Arguments

pFrame

Buffer to receive the SDI frame. Must be 32-bit aligned. NOTE: the format (e.g. 8-bit/10-bit, compressed/uncompressed, etc.) of the data returned in the frame buffer depends on the active receive-mode

FrameSize

As an input argument this argument indicates the size of the frame buffer. The frame buffer should be large enough to receive a complete frame and must be 32-bit aligned.
As an output argument this argument indicates the number of bytes returned in the frame buffer. The returned number of bytes includes any stuff bytes added to the end of the frame to achieve 32-bit alignment.

Timeout

Maximum amount of time in ms to wait for a complete frame. This method will fail if a frame cannot be returned within the specified period.
The value of this argument must larger than 0 or -1 to specify an infinite timeout. The default value is -1 (infinite).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Read operation has been completed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_BUF	The buffer is not aligned to a 32-bit word boundary
DTAPI_E_BUF_TOO_SMALL	The frame buffer is too small for receiving a complete frame
DTAPI_E_INVALID_TIMEOUT	Invalid timeout period specified
DTAPI_E_NOT_SDI_MODE	The channel is not in SDI mode (see **SetRxMode** page 196)
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_TIMEOUT	Read operation failed. Could not return a complete frame within the specified timeout period

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

If an infinite timeout has been specified, this method will block until a complete frame has been received from the hardware.

DtInpChannel::RegisterDemodCallback

Register a callback function for handling demodulator events.

```
DTAPI_RESULT DtInpChannel::RegisterDemodCallback(
  [in] IDtDemodEvent*  pEvent,     // Event handler
  [in] __int64  Events=-1,         // Events to register for
);
```

Function Arguments

pIEvent

Pointer to a callback function for handling demodulator events. Use NULL to stop handling events.

Events

Events to register for. The table below contains the supported events. Multiple event constants can be OR-ed together to register for multiple events. To register for all demodulator events use the value -1.

Event	Description
DTAPI_EV_TUNE_PARS_HAVE_CHANGED	Tuning parameters have changed (**SetDemodControl**, **SetTunerFrequency** or **Tune** was called)
DTAPI_EV_TUNE_FREQ_HAS_CHANGED	Tuning frequency has changed (**SetTunerFrequency** or **Tune** was called)

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Callback has been registered or deregistered successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	Registering an event interface is not supported

Remarks

DtInpChannel::Reset

Reset input channel.

```
DTAPI_RESULT DtInpChannel::Reset(
  [in] int  ResetMode
);
```

Function Arguments

ResetMode

Specifies which part of the hardware and software stack is reset. The following values are defined (values cannot be OR-ed together):

Value	Meaning
DTAPI_FIFO_RESET	Reset (clear) the Receive FIFO: • Data transfers are halted instantaneously • All data pending in the Receive FIFO is discarded • Receive-control state is reset to **DTAPI_RXCTRL_IDLE** • Receive-FIFO overflow flag is cleared
DTAPI_FULL_RESET	Full input-channel reset: • All actions for **DTAPI_FIFO_RESET**, plus: • Synchronisation-error flag (**DTAPI_RX_SYNC_ERR**) is cleared • State machines in the device hardware are reset

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Input channel has been reset
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_MODE	The value specified for *ResetMode* is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

An input-channel reset operation does not affect the following settings:
- Receive mode and insert-time-stamp flag (refer to **DtInpChannel::SetRxMode**)
- Polarity control of DVB-ASI inputs (refer to **DtInpChannel::PolarityControl**)

DtInpChannel::SetAdcSampleRate

DTA-2135 only. Set sample rate for ADC input channels. The ADC sample-rate determines the rate at which samples are taken from the down-converted RF signal.

```
DTAPI_RESULT DtInpChannel::SetAdcSampleRate(
  [in] int  SampleRate       // ADC sample rate in Hz
);
```

Function Arguments

`SampleRate`

ADC sample-rate according to the table below.

Value	Meaning
DTAPI_ADCCLK_OFF	*Clock is off*
DTAPI_ADCCLK_27M	*27Mhz Clock*
DTAPI_ADCCLK_20M647	*20.647059 Clock*[3]
DTAPI_ADCCLK_13M5	*13.5Mhz Clock*

Result

DTAPI_RESULT	Meaning
DTAPI_OK	ADC sample-rate has been changed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_MODE	The specified receive mode is invalid or incompatible with the input channel
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	Current device is not supported by this function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

Only the first channel of the DTA-2135 provides access the down-converted RF signal.

The IF frequency of the DTA-2135 is 36.167Mhz. Since the available sample-rates are all well below the Nyquist rate the signal is under sampled. Please refer to sampling theory on details how to recover the signal.

[3] The exact frequency is 27 * 13 / 17 =20.647059Mhz

DtInpChannel::SetAntPower

For receivers: Turn antenna power on or off.

```
DTAPI_RESULT DtInpChannel::SetAntPower(
  [in] int  AntPower          // Antenna power on/off
);
```

Function Arguments

AntPower

Power state according to the table below.

Value	Meaning
DTAPI_POWER_OFF	No power is applied. The antenna needs to be self-powered
DTAPI_POWER_ON	Power (+5V, 30mA) is applied to the external antenna through the antenna connector(s) of the channel

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Power state has been changed successfully
DTAPI_E_INVALID_MODE	The specified antenna power mode value is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The device does not have a provision for antenna power

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

After **Attach** and after **Reset**, antenna power is turned off.

DtInpChannel::SetDemodControl

Set the demodulation parameters.

```
DTAPI_RESULT DtInpChannel::SetDemodControl(
  [in] DtDemodPars*  pDemodPars    // Demodulation parameters
);

DTAPI_RESULT DtInpChannel::SetDemodControl(
  [in] int  ModType,               // Modulation type
  [in] int  ParXtra0,              // Extra parameter #0
  [in] int  ParXtra1,              // Extra parameter #1
  [in] int  ParXtra2               // Extra parameter #2
);
```

Function Arguments

pDemodPars

Pointer to a structure containing the demodulation parameters. See class **DtDemodPars** for possible demodulation parameters.

ModType

Expected type of modulation of the input signal. See **GetDemodControl** for a list of applicable values, with the exception of the **DTAPI_MOD_TYPE_UNK** value, which cannot be used in the **SetDemodControl** method.

ParXtra0, ParXtra1, ParXtra2

Additional parameters further defining the demodulation process. See **GetDemodControl** for a list of applicable values, with the exception of the **DTAPI_MOD_XXX_UNK** values, which cannot be used in the **SetDemodControl** method.

Many of the additional parameters can be automatically detected by the demodulator hardware and therefore it is not always required to fully define the demodulation parameters. However, providing more detail will help the demodulator to achieve signal lock faster, as it will not have to autodetect them.

The tables below show the limitations with respect to automatic detection:

Automatic detection

ModType

The modulation type must always be set explicitly and using automatic detection is not allowed.

ParXtra0, ParXtra1, ParXtra2

The table below lists which parameters can be automatically detected for each of the modulation standards:

Modulation Mode: ATSC		
ParXtra0	Constellation	No automatic detection supported (constellation must be specified)
ParXtra2	Symbol rate	Automatically detected by definition

Modulation Mode: DVB-S		
ParXtra0	Code rate	DTAPI_MOD_CR_AUTO
ParXtra1	Spectral inversion	DTAPI_MOD_S_S2_SPECINV_AUTO
ParXtra2	Symbol rate	DTAPI_MOD_SYMRATE_AUTO
Modulation Mode: DVB-S.2		
ParXtra0	Code rate	DTAPI_MOD_CR_AUTO
ParXtra1	Spectral inversion	Automatically detected by definition
	Pilots	DTAPI_MOD_S2_PILOTS_AUTO
	FEC frame size	DTAPI_MOD_S2_FRM_AUTO
ParXtra2	Symbol rate	DTAPI_MOD_SYMRATE_AUTO
Modulation Mode: DVB-T		
ParXtra0	Code rate	DTAPI_MOD_CR_AUTO
ParXtra1	Bandwidth	No automatic detection supported (bandwidth must be specified)
	Constellation	DTAPI_MOD_DVBT_CO_AUTO
	Guard interval	DTAPI_MOD_DVBT_GU_AUTO
	Interleaving	DTAPI_MOD_DVBT_IL_AUTO
	Tx mode	DTAPI_MOD_DVBT_MD_AUTO
Modulation Mode: QAM		
ParXtra0	J.83 Annex	No automatic detection supported (Annex must be specified)
ParXtra1 (QAM-B)	Interleaving	Automatically detected by definition
ParXtra2	Symbol rate	DTAPI_MOD_SYMRATE_AUTO Automatic detection of symbol rate is only supported by the DTA-2136 and DTA-2138; for other QAM enabled receivers the symbol rate must be specified

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The modulation parameters have been set successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_BANDWIDTH	Invalid value for bandwidth field
DTAPI_E_INVALID_CONSTEL	Invalid value for constellation field
DTAPI_E_INVALID_FHMODE	Invalid value for frame-header mode field
DTAPI_E_INVALID_GUARD	Invalid value for guard-interval field
DTAPI_E_INVALID_INTERLVNG	Invalid value for interleaving field
DTAPI_E_INVALID_J83ANNEX DTAPI_E_INVALID_ROLLOFF	Invalid value for J.83 annex
DTAPI_E_INVALID_MODE	Modulation type is incompatible with demodulator

DTAPI_E_INVALID_MODPARS	Invalid demodulation parameters
DTAPI_E_INVALID_PILOTS	Pilots cannot be specified in C=1 mode
DTAPI_E_INVALID_RATE	Invalid value for convolutional rate or FEC code rate
DTAPI_E_INVALID_SYMRATE	Invalid value for symbol rate
DTAPI_E_INVALID_T2PROFILE	Invalid value for DVB-T2 profile
DTAPI_E_INVALID_TRANSMODE	Invalid value for transmission-mode field
DTAPI_E_INVALID_USEFRAMENO	Invalid value for use-frame-numbering field
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	This function is not supported for the underlying hardware

Remarks

DtInpChannel::SetErrorStatsMode

Set the way error statistics are gathered for the specified type of modulation. This method is currently only supported by the DTA-2137.

```
DTAPI_RESULT DtInpChannel::SetErrorStatsMode(
  [in] int  ModType,          // Type of modulation
  [in] int  Mode              // The desired error statistics mode
);
```

Function Arguments

ModType

Type of modulation for which the given error statistics mode is set.

Value	Meaning
DTAPI_MOD_DVBS_QPSK	DVB-S, QPSK
DTAPI_MOD_DVBS2_8PSK	DVB-S.2, 8-PSK
DTAPI_MOD_DVBS2_16APSK	DVB-S.2, 16-APSK
DTAPI_MOD_DVBS2_32APSK	DVB-S.2, 32-APSK
DTAPI_MOD_DVBS2_QPSK	DVB-S.2, QPSK

Mode

Desired error-statistics mode.

Value	Meaning
DTAPI_ERRORSTATS_BER	(Default for each type of modulation) Bit error rate
DTAPI_ERRORSTATS_RS	Reed-Solomon error count

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The error-statistics mode has been changed successfully
DTAPI_E_INVALID_MODTYPE	The specified type of modulation is not valid
DTAPI_E_INVALID_PARS	The combination of *ModType* and *Mode* is not valid

Remarks

Reed-Solomon error counts (error-statistics mode **DTAPI_ERRORSTATS_RS**) can only be used for DVB-S.

DtInpChannel::SetIoConfig

Set the I/O configuration of the physical port attached to the input channel. This is the same function as **DtDevice::SetIoConfig** applied to the physical port corresponding to this channel.

```
DTAPI_RESULT DtInpChannel::SetIoConfig(
  [in] int   Group,                 // I/O configuration group
  [in] int   Value,                 // I/O configuration value
  [in] int   SubValue=-1,           // I/O configuration subvalue
  [in] __int64   ParXtra0=-1,       // Extra parameter #0
  [in] __int64   ParXtra1=-1        // Extra parameter #1
);
```

Function Arguments

Group, Value, SubValue, ParXtra0, ParXtra1

I/O configuration parameters, see **DtDevice::SetIoConfig**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	I/O configuration has been set successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
Other result codes	See **DtDevice::SetIoConfig**

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

DtInpChannel::SetIpPars

Set parameters for the reception of TS-over-IP streams.

```
DTAPI_RESULT DtInpChannel::SetIpPars(
  [in] DtTsIpPars*  pTsIpPars   // TS-over-IP parameters
);
```

Function Arguments

SetIpPars

New parameter set to be applied. Please refer to the **DtTsIpPars** page for a description of the parameters.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	TS-over-IP parameters have been applied successfully
DTAPI_E_IN_USE	Function Arguments cannot be changed because the channel is busy. The receive-control state should be switched back to idle first
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

SetIpPars should be called at least once after attaching to the hardware but before setting the receive-control state to **DTAPI_RXCTRL_RCV**.

After the initial call to **SetIpPars**, parameters can be changed again, but only when the receive-control state is **DTAPI_RXCTRL_IDLE**.

When the destination IP address is a multicast IP address, DTAPI automatically joins the multicast group upon the first invocation of **SetIpPars**. When this method is called again, membership of the old multicast group is dropped and, if required, the new multicast group is joined.

DtInpChannel::SetPower

DTA-122 only. Turn on/off power for a target adapter attached to the DTA-122.

```
DTAPI_RESULT DtInpChannel::SetPower(
  [in] int  Power            // Power state
);
```

Function Arguments

Power

Power state according to the table below.

Value	Meaning
DTAPI_POWER_OFF	No power is applied. The 25-pin sub-D connector is compatible with DVB-SPI
DTAPI_POWER_ON	Apply power (+5V) to pin 12 and 25 of the 25-pin sub-D connector

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Power state has been changed successfully
DTAPI_E_INVALID_MODE	The specified power-mode value is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The device does not support a power connection for target adapters

Remarks

After **Attach** and after **Reset**, power is turned off.

DtInpChannel::SetRxControl

Set receive control.

```
DTAPI_RESULT DtInpChannel::SetRxControl(
  [in] int  RxControl        // Receive control
);
```

Function Arguments

RxControl

New receive control value according to the table below.

Value	Meaning
DTAPI_RXCTRL_IDLE	The input stream input is "disconnected" from the receive FIFO: Incoming transport packets are not stored in the receive FIFO.
DTAPI_RXCTRL_RCV	Normal operation. Incoming transport packets are stored in the receive FIFO.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Receive-control state has been changed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_MODE	The specified receive-control state is invalid or incompatible with the attached hardware function
DTAPI_E_NO_IP_PARS	For TS-over-IP channels: receive-control state cannot be set to **DTAPI_RXCTRL_RCV** because TS-over-IP parameters have not been specified
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

If receive control is set to **DTAPI_RXCTRL_RCV**, but the application does not read data from the receive FIFO, then the receive FIFO will quickly overflow.

Calling **AttachToPort**, **Reset** or **ClearFifo** will initialize receive control to **DTAPI_RXCTRL_IDLE**.

DtInpChannel::SetRxMode

Set receive mode.

```
DTAPI_RESULT DtInpChannel::SetRxMode(
  [in] int  RxMode            // Receive mode
);
```

Function Arguments

RxMode

Receive mode according to the table below.

Value	Meaning
DTAPI_RXMODE_ST188	*188-byte mode* Always store 188-byte packets in the receive FIFO. When the input contains 204-byte packets, the 16 trailing bytes are dropped. Input data without 188- or 204-byte packet structure is dropped.
DTAPI_RXMODE_ST204	*204-byte mode* Always store 204-byte packets in the receive FIFO. When the input contains 188-byte packets, 16 zero bytes are appended. Input data without 188- or 204-byte packet structure is dropped.
DTAPI_RXMODE_STMP2	*MPEG-2 mode* Store 188- or 204-byte packets in the receive FIFO without modification. Input data without 188- or 204-byte packet structure is dropped.
DTAPI_RXMODE_STRAW	*Raw mode* No notion of packets. All incoming valid data bytes are stored in the Receive FIFO. For DVB-ASI input channels, this mode is incompatible with **DTAPI_POLARITY_AUTO**! Please refer to the Remarks section.
DTAPI_RXMODE_STTRP	*Transparent mode* All incoming data bytes are stored in the receive FIFO. The data is aligned to packet boundaries if valid packets are detected. This format includes a trailer that contains information about the detected packet size, sync status and valid data bytes within the packet.
DTAPI_RXMODE_STL3	*L.3 Baseband frame mode* (DTA-2137 only). No notion of transport stream packets. The entire DVB-S2 baseband frame is passed with the addition of an L.3 Header.

The following modes are valid for SDI capable channels only.

Value	Meaning
DTAPI_RXMODE_SDI_FULL	*Full frame mode* Store all SDI data (i.e. complete frames).
DTAPI_RXMODE_SDI_ACTVID	*Active video mode* Store only the active video part of each SDI frame. This mode should only be used in combination with Huffman compression (i.e. with **DTAPI_RXMODE_SDI_HUFFMAN** flag)!

The following mode is valid for TS-over-IP reception only.

Value	Meaning
DTAPI_RXMODE_IPRAW	*Raw IP mode* Store unprocessed IP packets in the buffer. If error correction is requested, store FEC streams too. Each IP packet returned by a call the **DtInpChannel::Read** will be preceded by a **DtRawIpHeader** structure.

The receive mode can be optionally combined (OR-ed) with the following flag:

Value	Meaning
DTAPI_RXMODE_TIMESTAMP32	*Time-stamped mode (32-bit timestamps)* Insert a 32-bit timestamp before each packet. The timestamp is a sample of the system clock counter on the device. This flag may not be specified in raw mode (**DTAPI_RXMODE_STRAW**) or any of the SDI modes.
DTAPI_RXMODE_TIMESTAMP64	*Time-stamped mode (64-bit timestamps)* (hardware shall support **DTAPI_CAP_TIMESTAMP64)**. Insert a 64-bit timestamp before each packet. The timestamp is a sample of the system clock counter on the device. This flag may not be specified in raw mode (**DTAPI_RXMODE_STRAW**) or any of the SDI modes.
DTAPI_RXMODE_SDI	*SDI mode* Operate in SDI mode; Otherwise ASI mode will be used. This flag is already OR-ed into **DTAPI_RXMODE_SDI_FULL** and **DTAPI_RXMODE_SDI_ACTVID**.
DTAPI_RXMODE_SDI_10B	*10-bit SDI samples* Provide 10-bit SDI samples. If this flag is omitted the SDI data will be delivered as 8-bit samples.
DTAPI_RXMODE_SDI_HUFFMAN	*Huffman compression* Compress the SDI using Huffman compression.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Receive mode has been changed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_MODE	The specified receive mode is invalid or incompatible with the input channel
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	Current device is not supported by this function

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive=true*).

In receive mode **DTAPI_RXMODE_STRAW** ("raw" mode), the input channel does not care about the packet structure of the incoming data: All data bytes are stored in the input buffer. For DVB-ASI input channels, raw mode can only work reliably if *polarity control* is set to **DTAPI_POLARITY_NORMAL** or **DTAPI_POLARITY_INVERT**. If polarity control is set to **DTAPI_POLARITY_AUTO**, disaster may be the result: Automatic polarity detection assumes that the input has a valid packet structure. If such a structure cannot be found, the device tries again with the input signal inverted. In raw mode, such inversion may occur periodically and severely corrupt the input data!

For the DTA-122 and DTA-2142 DVB-SPI ports, packet synchronisation in modes **DTAPI_RXMODE_ST188** and **DTAPI_RXMODE_ST204** is based on the PSYNC signal, not on the value of the first byte of the packet: The value of DATA at a PSYNC pulse is stored in the input buffer, even if the value is not 0x47.

Timestamps are stored in little-endian format: the first byte contains the least-significant 8 bits, the fourth byte the most-significant 8 bits. 32-bit timestamps can be read by code like this:

```
unsigned int TimeStamp = *(unsigned int*) PtrInCharBuffer;
```

DtInpChannel::SetStreamSelection

Select a PLP in a DVB-C2 or DVB-T2 stream.

```
DTAPI_RESULT DtInpChannel::SetStreamSelection(
  [in] DtDvbC2StreamSelPars&  StreamSel    // PLP selection parameters
);
DTAPI_RESULT DtInpChannel::SetStreamSelection(
  [in] DtDvbT2StreamSelPars&  StreamSel    // PLP selection parameters
);
```

Function Arguments

StreamSel

Specification of the PLP selection criteria.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The PLP has been selected successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_DSLICE_ID	Invalid data slice identifier
DTAPI_E_INVALID_MODE	The current demodulation type does not correspond to the type of the PLP selection parameters
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The device does not support DVB-C2 or DVB-T2
DTAPI_E_PLP_ID	Invalid physical layer pipe identifier

Remarks

This function requires exclusive access (**AttachToPort** was called with *Exclusive*=*true*).

If the specified PLP is not available another PLP may be selected.

DtInpChannel::SetTunerFrequency

Set tuner frequency.

```
DTAPI_RESULT DtInpChannel::SetTunerFrequency(
  [in] __int64&  FreqHz      // Frequency in hertz
);
```

Function Arguments

FreqHz

Desired tuning frequency (in Hz). The table below specifies the valid range and the step size with which the RF rate can be specified. *FreqHz* is rounded to the nearest RF frequency compatible with the frequency resolution.

Device	Valid Range	Step Size
DTU-234	53,000,000 - 865,000,000 Hz	-
DTU-235	50,000,000 - 860,000,000 Hz	-
DTU-236	44,000,000 - 865,000,000 Hz	-
DTA-2135	50,000,000 - 860,000,000 Hz	-
DTA-2136	54,000,000 - 1002,000,000 Hz	-
DTA-2137	950,000,000 - 2150,000,000 Hz	-

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The tuner frequency has been set successfully
DTAPI_E_INVALID_FREQ	The specified frequency is incompatible (too low or too high) with the tuner
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The channel does not include a tuner

Remarks

DtInpChannel::Tune

Tunes the demodulator to a frequency using the specified demodulation parameters. This function basically combines the functionality of the **SetDemodControl** and **SetTunerFrequency** methods.

```
DTAPI_RESULT DtInpChannel::Tune(
  [in] __int64&  FreqHz            // Frequency in hertz
  [in] int  ModType,               // Modulation type
  [in] int  ParXtra0,              // Extra parameter #0
  [in] int  ParXtra1,              // Extra parameter #1
  [in] int  ParXtra2               // Extra parameter #2
);

DTAPI_RESULT DtInpChannel::Tune(
  [in] __int64&  FreqHz            // Frequency in hertz
  [in] DtDemodPars*  pDemodPars    // Demodulation parameters
);
```

Function Arguments

FreqHz

Desired tuning frequency (in Hz). See **SetTunerFrequency** for the allowed values.

ModType, ParXtra0, ParXtr1, ParXtra2

'Old style' demodulation parameters to use while tuning. Refer to **SetDemodControl** for more details about these parameters.

pDemodPars

'New style' demodulation parameters to use while tuning. Refer to **SetDemodControl** for more details about these parameters.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The tuner frequency has been set successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_BANDWIDTH	Invalid value for bandwidth field
DTAPI_E_INVALID_CONSTEL	Invalid value for constellation field
DTAPI_E_INVALID_FHMODE	Invalid value for frame-header mode field
DTAPI_E_INVALID_FREQ	The specified frequency is incompatible (too low or too high) with the tuner
DTAPI_E_INVALID_GUARD	Invalid value for guard-interval field
DTAPI_E_INVALID_INTERLVNG	Invalid value for interleaving field
DTAPI_E_INVALID_J83ANNEX DTAPI_E_INVALID_ROLLOFF	Invalid value for J.83 annex
DTAPI_E_INVALID_MODE	Modulation type is incompatible with demodulator
DTAPI_E_INVALID_MODPARS	Invalid demodulation parameters
DTAPI_E_INVALID_PILOTS	Pilots cannot be specified in C=1 mode

DTAPI_E_INVALID_RATE	Invalid value for convolutional rate or FEC code rate
DTAPI_E_INVALID_SYMRATE	Invalid value for symbol rate
DTAPI_E_INVALID_T2PROFILE	Invalid value for DVB-T2 profile
DTAPI_E_INVALID_TRANSMODE	Invalid value for transmission-mode field
DTAPI_E_INVALID_USEFRAMENO	Invalid value for use-frame-numbering field
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The channel does not include a tuner

DtOutpChannel

Class representing an output channel for transmitting the following formats:
- MPEG-2 transport stream over ASI, SPI or IP
- Serial Digital Interface (SDI)

```
class DtOutpChannel;
```

DtOutpChannel::AttachToPort

Attach the output-channel object to a specific physical port.

```
DTAPI_RESULT DtOutpChannel::AttachToPort(
  [in] DtDevice*  pDtDvc,          // Device object
  [in] int  Port,                  // Physical port number (1…#ports)
  [in] bool  ProbeOnly=false       // Just check whether channel is in use
);
```

Function Arguments

pDtDvc

Pointer to the device object that represents a DekTec device. The device object must have been attached to the device hardware.

Port

Physical port number. The channel object is attached to this port. The port number of the top-most port is 1, except on the DTA-160 and DTA-2160, on which the top-most Ethernet port is port #4.

ProbeOnly

Probe whether the channel is in use, but do not actually attach.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Channel object has been attached successfully to the port
DTAPI_OK_FAILSAFE	Channel object has been attached successfully to the hardware function. The application shall call the **SetFailsafeAlive** method on a regular basis to prevent the release of the failsafe relay, which will physically connect the input port to the output port. This is not an error code; It is intended to make the application aware of failsafe mode.
DTAPI_E_ATTACHED	Channel object is already attached
DTAPI_E_DEVICE	Pointer *pDtDvc* is not valid or the device object is not attached to a hardware device
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_IN_USE	Another channel object is already attached to this port
DTAPI_E_NO_DT_OUTPUT	*Port* is not an output
DTAPI_E_NO_SUCH_PORT	Invalid port number for this device
DTAPI_E_OUT_OF_MEM	TS-over-IP: Receive FIFO cannot be allocated

Remarks

DtOutpChannel::ClearFifo

Clear contents of the transmit FIFO and set transmit control to **IDLE**. Clears the output channel's status flags: transmit-FIFO-underflow flag (**DTAPI_TX_FIFO_UFL**) and transmit-synchronisation-error flag (**DTAPI_TX_SYNC_ERR**).

```
DTAPI_RESULT DtOutpChannel::ClearFifo(
      void
);
```

Function Arguments

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Transmit FIFO has been cleared
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

The effects of **ClearFifo** are equivalent to **Reset(DTAPI_FIFO_RESET)**.

DtOutpChannel::ClearFlags

Clear *latched* status flag(s).

```
DTAPI_RESULT DtOutpChannel::ClearFlags(
 [int] int  Latched          // Latched status flags to be cleared
);
```

Function Arguments

Latched

Latched status flag(s) to be cleared. Multiple flags can be cleared with one function call by OR-ing the bit positions to be cleared. The following flags are latched and can be cleared:

Value	Meaning
DTAPI_TX_CPU_UFL	See **GetFlags**
DTAPI_TX_DMA_UFL	,, ,,
DTAPI_TX_FIFO_UFL	,, ,,
DTAPI_TX_READBACK_ERR	,, ,,
DTAPI_TX_SYNC_ERR	,, ,,
DTAPI_TX_TARGET_ERR	,, ,,
DTAPI_TX_LINK_ERR	,, ,,
DTAPI_TX_DATA_ERR	,, ,,

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Flag(s) have been successfully cleared
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

Some status flags that are queried with **GetFlags** are not latched and therefore cannot be cleared.

The latched status flags are also automatically reset after attaching and after **Reset**. A call to **ClearFifo** clears **DTAPI_TX_FIFO_UFL** and **DTAPI_TX_SYNC_ERR**.

DtOutpChannel::Detach

Detach output channel object from a hardware function. Frees resources allocated for the output channel.

```
DTAPI_RESULT DtOutpChannel::Detach(
  [in] int  DetachMode       // How to detach
);
```

Function Arguments

DetachMode

Specifies how the channel object should detach from the hardware function. If *DetachMode* is 0, the object is detached without further action. A number of flags listed below are defined to detach from the hardware function in a specific way. The flags can be OR-ed together to their combine behaviour, with some exceptions as listed in the table.

Value	Meaning
DTAPI_INSTANT_DETACH	Clear the contents of the transmit FIFO and detach without waiting until pending data in the FIFO has been transmitted. This flag may not be combined with **DTAPI_WAIT_UNTIL_SENT**.
DTAPI_WAIT_UNTIL_SENT	Sleep until all pending data in the transmit FIFO has been transmitted. If this flag is combined with other flags, the wait is executed before the action associated with the other flags. This flag may not be combined with **DTAPI_INSTANT_DETACH**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Channel object has been detached successfully from the hardware function
DTAPI_E_INVALID_FLAGS	An invalid combination of detach flags was specified
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function, so it cannot be detached

Remarks

For ASI channels, if packet stuffing is turned on, the output channel keeps transmitting null packets after detaching.

Detach may take a long time if **DTAPI_WAIT_UNTIL_SENT** is specified while the FIFO still contains data and transmit control is **IDLE**.

DtOutpChannel::GetAttribute

Get the value of an attribute for the port to which this channel is attached.

```
DTAPI_RESULT DtOutpChannel::GetAttribute(
  [in]int&  AttrId,          // Attribute identifier
 [out]int&  AttrValue        // Returned attribute value
);
```

Function Arguments

AttrId

Identifies the attribute that is to be retrieved. Please refer to **DtDevice::GetAttribute** for a list of attributes that can be retrieved.

AttrValue

Output argument that receives the attribute value.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The attribute value has been retrieved successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The attribute is not supported for this port

Remarks

DtOutpChannel::GetDescriptor

Get hardware function descriptor for this output channel.

```
DTAPI_RESULT DtOutpChannel::GetDescriptor(
 [out] DtHwFuncDesc&  HwFuncDesc  // Hardware function descriptor
);
```

Function Arguments

HwFuncDesc

Output argument that receives the hardware function descriptor.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The hardware function descriptor been retrieved successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

DtOutpChannel::GetExtClkFreq

Get an estimate of the frequency of the external clock (DTA-102 and DTA-2142 only).

```
DTAPI_RESULT DtOutpChannel::GetExtClkFreq(
 [out] int&  ExtClkFreq      // Measurement of external-clock frequency
);
```

Function Arguments

ExtClkFreq

Output argument that is set to a measurement of the frequency of the signal applied to the external-clock input. For an accurate estimate, the external clock signal must be present and stable for at least one second.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	External-clock frequency has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

The frequency returned is a *byte* rate that must be multiplied by 8 or 8*204/188 to obtain the corresponding transport-stream bitrate.

DtOutpChannel::GetFailsafeAlive

Get current status of the watchdog that controls the failsafe relay.

```
DTAPI_RESULT DtOutpChannel::GetFailsafeAlive(
[out] bool&  Alive          // Are we alive and kicking?
);
```

Function Arguments

Alive

Indicates the current status of the watchdog.
If *Alive* is true, all is fine and the board is operating as normal. The failsafe timeout has not expired and the relay is in normal operational mode: the input port is connected to the input channel and the output channel is connected to the output port.
If *Alive* is false, the watchdog timer has expired before **SetFailsafeAlive** was called. The input-to-output relay is switched to failsafe mode: the input port is connected directly to the output port.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The watchdog status has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_CONFIG	The channel is not configured to operate in failsafe mode
DTAPI_E_NOT_SUPPORTED	The channel is not capable of operating in failsafe mode
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

DtOutpChannel::GetFailsafeConfig

Get configuration info about failsafe mode.

```
DTAPI_RESULT DtOutpChannel::GetFailsafeConfig(
 [out] bool&  Enable,        // Failsafe enabled yes/no
 [out] int&   Timeout        // Watchdog timeout (in ms)
);
```

Function Arguments

Enable

Operation in failsafe mode has been enabled or disabled (see also **SetFailsafeConfig**).

Timeout

Current watchdog timeout period (in ms).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Failsafe configuration has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_CONFIG	The channel is not configured to operate in failsafe mode
DTAPI_E_NOT_SUPPORTED	The channel is not capable of operating in failsafe mode
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

DtOutpChannel::GetFifoLoad

Get the current load of the channel's transmit FIFO.

```
DTAPI_RESULT DtOutpChannel::GetFifoLoad(
 [out] int&  FifoLoad        // Load of transmit FIFO in number of bytes
);
```

Function Arguments

FifoLoad

This output argument is set to the number of bytes in the transmit FIFO.

Result

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

Remarks

If transmit control is **SEND**, then the value retrieved with **GetFifoLoad** may not be exact: it *approximates* the load of the transmit FIFO.

If a DMA transfer is in progress and/or the transmit control is **SEND**, then every call to **GetFifoLoad** may return a different value.

DtOutpChannel::GetFifoSize

Get the current size of the channel's transmit FIFO (**GetFifoSize**), or the maximum size supported by the channel (**GetFifoSizeMax**), or a typical size of the transmit FIFO that generally should work well (**GetFifoSizeTyp**).

```
DTAPI_RESULT DtOutpChannel::GetFifoSize(
 [out] int&  FifoSize        // Size of transmit FIFO in bytes
);
DTAPI_RESULT DtOutpChannel::GetFifoSizeMax(
 [out] int&  FifoSize        // Maximum size of transmit FIFO in bytes
);
DTAPI_RESULT DtOutpChannel::GetFifoSizeTyp(
 [out] int&  FifoSize        // Typical size of transmit FIFO in bytes
);
DTAPI_RESULT DtOutpChannel::GetMaxFifoSize(
 [out] int&  FifoSize        // Maximum size of transmit FIFO in bytes
);
```

Function Arguments

FifoSize

Current, maximum or typical size of the transmit FIFO in number of bytes.

Result

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

Remarks

The actual size of the transmit FIFO is adjustable with **SetFifoSize**.

DekTec hardware devices have a transmit FIFO that has a size of at least 8Mbytes.

For modulators GetFifoSizeMax returns the size of the hardware FIFO.

DtOutpChannel::GetFlags

Get current and latched value of the output channel's status flags.

```
DTAPI_RESULT DtOutpChannel::GetFlags(
 [out] int&  Status,          // Status flags
 [out] int&  Latched          // Latched status flags
);
```

Function Arguments

Status

Output argument that receives the current status of the output channel. Each status flag is represented by one bit. Multiple status flags can be set at the same time. If none of the status flags is asserted, *Status* is set to zero.

Value	Meaning
DTAPI_TX_CPU_UFL	Modulation error caused by low CPU performance
DTAPI_TX_DMA_UFL	Modulation error caused by low DMA performance
DTAPI_TX_FIFO_UFL	A transmit-FIFO underflow condition has occurred. Underflow detection is available in all transmit modes, including modes with null-packet stuffing switched on.
DTAPI_TX_MUX_OVF	Overflow in hierarchical multiplexing for ISDB-T
DTAPI_TX_READBACK_ERR	An output pin is forced to an erroneous signal level, e.g. because of a short-circuit (DTA-102 only)
DTAPI_TX_SYNC_ERR	Transmit-FIFO synchronisation error. The size of one or more packets mode does not match the transmit mode. This status flag is not used in transmit mode **DTAPI_TXMODE_RAW**.
DTAPI_TX_TARGET_ERR	The target adapter signals a fault (DTA-102 only)
DTAPI_RX_LINK_ERR	The communication link with the device is broken (DTE-31XX devices only)
DTAPI_RX_DATA_ERR	Data is lost during transfer to the device (DTE-31XX devices only)

Latched

Output argument that *latches* the value of the status flags: If a status flag has become '1', even for a very short moment, the corresponding bit in *Latched* is set to '1'. The bit remains set until it is cleared explicitly by **ClearFlag**, or cleared implicitly by one of the following DTAPI-calls: **ClearFifo**, **AttachToPort** or **Reset**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Status flags have been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

DtOutpChannel::GetIoConfig

Get the I/O configuration of the physical port attached to the output channel. This is the same function as **DtDevice::GetIoConfig** applied to the physical port corresponding to this channel.

```
DTAPI_RESULT DtOutpChannel::GetIoConfig(
  [in] int  Group,            // I/O configuration group
 [out] int&  Value            // I/O configuration value
);
DTAPI_RESULT DtOutpChannel::GetIoConfig(
  [in] int  Group,            // I/O configuration group
 [out] int&  Value,           // I/O configuration value
 [out] int&  SubValue         // I/O configuration subvalue
);
DTAPI_RESULT DtOutpChannel::GetIoConfig(
  [in] int  Group,            // I/O configuration group
 [out] int&  Value,           // I/O configuration value
 [out] int&  SubValue,        // I/O configuration subvalue
 [out] __int64&  ParXtra0     // Extra parameter #0
);
DTAPI_RESULT DtOutpChannel::GetIoConfig(
  [in] int  Group,            // I/O configuration group
 [out] int&  Value,           // I/O configuration value
 [out] int&  SubValue,        // I/O configuration subvalue
 [out] __int64&  ParXtra0,    // Extra parameter #0
 [out] __int64&  ParXtra1     // Extra parameter #1
);
```

Function Arguments

Group, Value, SubValue, ParXtra0, ParXtra1

I/O configuration parameters, see **DtDevice::GetIoConfig**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	I/O configuration has been retrieved successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
Other result codes	See **DtDevice::GetIoConfig**

DtOutpChannel::GetIpPars

Get IP-related parameters for this channel, as programmed with **SetIpPars**.

```
DTAPI_RESULT DtOutpChannel::GetIpPars(
  [in] DtTsIpPars*  pTsIpPars   // Receives the TS-over-IP parameters
);
```

Function Arguments

pTsIpPars

Receives the TS-over-IP parameters. The user must have allocated the **DtTsIpPars** structure.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The TS-over-IP parameters have been retrieved successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	Channel is not a TS-over-IP channel

Remarks

DtOutpChannel::GetModControl

For modulators: get current modulation-control parameters.

```
DTAPI_RESULT DtOutpChannel::GetModControl(
 [out] int&  ModType,         // Modulation type
 [out] int&  ParXtra0,        // Extra parameter #0
 [out] int&  ParXtra1,        // Extra parameter #1
 [out] int&  ParXtra2,        // Extra parameter #2
 [out] void*&  pXtraPars      // More extra parameters
);
```

Function Arguments

ModType

Output argument that receives the modulation type. See **SetModControl** for a list of applicable values.

ParXtra0, ParXtra1, ParXtra2

Extra modulation parameters. See **SetModControl** for a list of applicable values.

pXtraPars

Extra parameters that are stored in a struct (they do not fit in *ParXtra0* … *ParXtra2*). Extra parameters are used for the following modulation types CMMB, ISDB-S, ISDB-T, DVB-C2 and DVB-T2.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The modulation parameters have been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The output channel is not a modulator

Remarks

DtOutpChannel::GetOutputLevel

Get current level (in dBm) for outputs with an adjustable output level.

```
DTAPI_RESULT DtOutpChannel::GetOutputLevel(
 [out] int&  LeveldBm)       // Current level in units of 0.1dBm
);
```

Function Arguments

LeveldBm

Output level expressed in units of 0.1 dBm (e.g. -30 → -30×0.1 = -3dBm).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The output level has been retrieved successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The channel does not support a adjustable output level

Remarks

DtOutpChannel::GetRfControl

Get upconverter parameters for devices with on-board RF upconverter.

```
DTAPI_RESULT DtOutpChannel::GetRfControl(
  [in] __int64&  RfRate,      // RF frequency in Hz
 [out] int&  LockStatus       // Lock status of RF PLL
);
```

Function Arguments

RfRate

Output arguments that is set to the current carrier frequency as programmed into the RF up-converter, expressed in Hertz.
The RF frequency returned in *RfRate* may be different from the frequency programmed with **SetRfControl** because of rounding to the RF step size.

LockStatus

Output argument that is an OR of the following flags. In the normal operational state all RF PLLs are in lock. The DTA-111, DTA-112 and DTA-115 have three PLLs, the other cards have a single PLL. If *LockStatus* is zero, none of the PLLs is in lock.

Value	Meaning
DTAPI_RFPLL_LOCK1	The first RF PLL is in lock.
DTAPI_RFPLL_LOCK2	The second RF PLL is in lock (DTA-111/112/115)
DTAPI_RFPLL_LOCK3	The third RF PLL is in lock (DTA-111/112/115)

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The upconverter parameters have been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The device does not have an RF upconverter

Remarks

DtOutpChannel::GetSpiClk

DTA-2142 only. Get the DVB-SPI clock frequency in case the SPI channel is operating with a fixed clock (I/O configurations **SPIFIXEDCLK**, **SPISER8B**, **SPISER10B**).

```
DTAPI_RESULT DtOutpChannel::GetSpiClk(
  [in] Int&  SpiClk             // Fixed SPI clock
);
```

Function Arguments

SpiClk

Receives the frequency of the fixed DVB-SPI clock in Hertz.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The TS-over-IP parameters have been applied successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_INVALID_MODE	The SPI clock is not fixed (SPI mode is **SPIDVBMODE**)
DTAPI_E_NOT_SUPPORTED	Not a DVB-SPI channel

Remarks

DtOutpChannel::GetTargetId

Get the target-adapter identifier (DTA-102 only).

```
DTAPI_RESULT DtOutpChannel::GetTargetId(
 [out] int&  Present,        // Target adapter present?
 [out] int&  TargetId        // Target-adapter identifier
);
```

Function Arguments

Present

Output argument that indicates whether a target adapter has been detected.

Value	Meaning
DTAPI_NO_CONNECTION	Nothing is connected to the output connector of the DTA-102
DTAPI_DVB_SPI_SINK	A standard DVB-SPI sink is connected to the DTA-102
DTAPI_TARGET_PRESENT	A target adapter is present
DTAPI_TARGET_UNKNOWN	The system is busy assessing the situation on the output connector

TargetId

Output argument that is set to an integer value that uniquely identifies the target adapter. Please refer to the DTA-102 data sheet for a list of available target adapters.

A value is assigned to *TargetId* only if *Present* is **DTAPI_TARGET_PRESENT**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Target-adapter identifier has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The output channel does not support target adapters

Remarks

DtOutpChannel::GetTsRateBps

Get the current transport-stream rate.

```
DTAPI_RESULT DtOutpChannel::GetTsRateBps(
 [out] int&  TsRate          // transport-stream rate in bps
);
```

Function Arguments

TsRate

Output argument that is set to the current transport-stream rate expressed in bits per second. If an external clock is used, or the bitrate is locked to an input, then result code **DTAPI_E_INVALID_TSRATESEL** is returned.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	transport-stream rate has been read successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_ TSRATESEL	The current TS-rate selection I/O configuration does not allow retrieval of the transport-stream rate
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

For a discussion of transport-stream rate vs. transmit-clock rate, refer to **SetTsRateBps**.

DtOutpChannel::GetTxControl

Get the current value of transmit control.

```
DTAPI_RESULT DtOutpChannel::GetTxControl(
 [out] int&  TxControl       // Transmit control
);
```

Function Arguments

TxControl

Refer to **SetTxControl** for a description of the different values for transmit control.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Transmit-control state has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

DtOutpChannel::GetTxMode

Get the current transmit mode and null-packet stuffing mode.

```
DTAPI_RESULT DtOutpChannel::GetTxMode(
 [out] int&  TxMode,          // Transmit mode
 [out] int&  StuffMode        // Null-packet stuffing on/off
);
```

Function Arguments

TxMode, StuffMode

Refer to **SetTxMode** for a description of transmit-control modes.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Transmit mode has been retrieved successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

DtOutpChannel::LedControl

Take direct control of the channel's status LED, or let the hardware drive the LED.

```
DTAPI_RESULT DtOutpChannel::LedControl(
  [in] int  LedControl       // DTAPI_LED_XXX
);
```

Function Arguments

LedControl

Value that controls the status of the LED.

Value	Meaning
DTAPI_LED_HARDWARE	Hardware drives the LED (default after power up)
DTAPI_LED_OFF	LED is forced to off-state
DTAPI_LED_GREEN	LED is forced to green-state
DTAPI_LED_RED	LED is forced to red-state
DTAPI_LED_YELLOW	LED is forced to yellow-state

Result

DTAPI_RESULT	Meaning
DTAPI_OK	LED setting has been accepted
DTAPI_E_INVALID_MODE	The specified LED-control value is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The device does not have a general-status LED

Remarks

DtOutpChannel::Reset

Reset the output channel.

```
DTAPI_RESULT DtOutpChannel::Reset(
  [in] int  ResetMode        // DTAPI_FIFO_RESET or DTAPI_FULL_RESET
);
```

Function Arguments

ResetMode

Specifies which part of the hardware and software should be reset. The following values are defined (values cannot be OR-ed together):

Value	Meaning
DTAPI_FIFO_RESET	Reset the transmit FIFO: • Data transfers and packet transmission are halted instantaneously. • All data pending in the transmit FIFO is discarded. • Transmit-control state is reset to **DTAPI_TXCTRL_IDLE**. • Transmit-FIFO underflow flag is cleared.
DTAPI_FULL_RESET	Full reset: • All actions for **DTAPI_FIFO_RESET**, plus: • transport-stream rate is reset to zero (except for ISDB-T and OFDM modulation on DTA-110T)

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Output channel has been reset
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_MODE	The specified value for *ResetMode* is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

A potential side-effect of calling **Reset** is that the packet currently being transmitted is truncated. For one packet, the number of bytes between two consecutive SYNC bytes is less than the packet size. To avoid such a truncation, **ClearFifo** may be used.

DtOutpChannel::SetChannelModelling

Set channel-modelling parameters. This function may only be called while transmit control is **IDLE**.

```
DTAPI_RESULT DtOutpChannel::SetChannelModelling(
  [in] Bool   CmEnable,          // Enable/disable channel modelling
  [in] DtCmPars&   CmPars        // Channel modelling parameters
);
```

Function Arguments

CmEnable

Enable channel modelling. This parameter provides an easy way to turn off channel modelling entirely.

CmPars

Channel-modelling parameters. See description of struct **DtCmPars**.

Result

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

Remarks

DtOutpChannel::SetCustomRollOff

DTA-2107 only. Set the FIR-filter coefficients of the root-raised-cosine (RRC) channel filter to construct a custom roll-off factor. The user has to compute the filter coefficients himself.

```
DTAPI_RESULT DtOutpChannel::SetCustomRollOff(
  [in] Bool  Enable,            // Enable/disable custom roll-off filter
  [in] DtFilterPars&  Filter    // Custom roll-off filter parameters
);
```

Function Arguments

Enable

Enable or disable the channel filter with custom roll-off factor.

Filter

Filter coefficients to be programmed in the hardware.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Channel-modelling parameters have been applied successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The output channel does not support a custom roll-off filter

Remarks

The filter does not necessarily need to be an RRC filter. Any set of filter coefficients can be programmed.

DtOutpChannel::SetFailsafeAlive

Reset the watchdog timer for operation in failsafe mode. Failing to call this method within the time-out set with **SetFailsafeConfig()** will result in the release of the on-board relay so that the output port is connected directly with the input port.

```
DTAPI_RESULT DtOutpChannel::SetFailsafeAlive();
```

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Watchdog has been triggered successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_CONFIG	The channel is not configured to operate in failsafe mode
DTAPI_E_NOT_SUPPORTED	The channel is not capable of operating in failsafe mode
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

DtOutpChannel::SetFailsafeConfig

Configure failsafe mode.

```
DTAPI_RESULT DtOutpChannel::SetFailsafeConfig(
  [in] bool  Enable,          // Enable/disable operation in failsafe mode
  [in] int   Timeout=0        // Watchdog timeout (in ms)
);
```

Function Arguments

Enable

Enables/disables operation in failsafe mode.
If *Enable* is false, the output channel will operate like a standard output channel.
If *Enable* is true, the output channel will start operating in failsafe mode. The user application shall call **SetFailsafeAlive** repeatedly within the watchdog timeout period. If the user application is too late (e.g. because it has crashed), the watchdog times out and the failsafe relay is released so that the output is connected directly to the input.

Timeout

Specifies the watchdog timeout period in ms.
The timeout value can only be a multiple of 20ms. If the value is not a multiple of 20ms, it will be rounded downwards to the closest multiple of 20ms. Setting *Timeout* to zero indicates the parameter should be ignored (i.e. only the *Enable* parameter has a meaning)

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Output channel has been reset
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_CONFIG	The channel is not configured to operate in failsafe mode
DTAPI_E_NOT_SUPPORTED	The channel is not capable of operating in failsafe mode
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

DtOutpChannel::SetFifoSize

Set the size the transmit FIFO to a specified value (**SetFifoSize**), or to the maximum value supported by the channel (**SetFifoSizeMax**), or to a typical value (**SetFifoSizeTyp**).

The FIFO size can only be changed if transmit control is **IDLE**.

```
DTAPI_RESULT DtOutpChannel::SetFifoSize(
  [in] int  FifoSize          // Size of transmit FIFO in bytes
);
DTAPI_RESULT DtOutpChannel::SetFifoSizeMax(void);
DTAPI_RESULT DtOutpChannel::SetFifoSizeTyp(void);
```

Function Arguments

FifoSize

Requested size of the transmit FIFO in number of bytes.
FifoSize must be a multiple of 16 and may not exceed the maximum physical size of the transmit FIFO.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The size of the transmit FIFO has been set successfully
DTAPI_E_IN_USE	The FIFO size cannot be changed because transmission-control state is **DTAPI_TXCTRL_HOLD** or **DTAPI_TXCTRL_SEND**
DTAPI_E_INVALID_SIZE	The specified FIFO size is negative, zero, not a multiple of 16 or greater than the maximum size
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

The size of the Transmit FIFO determines the amount of packet data that is buffered on the device. It also determines the delay between transferring data to the device (with **Write**) and transmission of that data.

DtOutpChannel::SetIoConfig

Set the I/O configuration of the physical port attached to the output channel. This is the same function as **DtDevice::SetIoConfig** applied to the physical port corresponding to this channel.

```
DTAPI_RESULT DtOutpChannel::SetIoConfig(
  [in] int   Group,                  // I/O configuration group
  [in] int   Value,                  // I/O configuration value
  [in] int   SubValue=-1,            // I/O configuration subvalue
  [in] __int64   ParXtra0=-1,        // Extra parameter #0
  [in] __int64   ParXtra1=-1         // Extra parameter #1
);
```

Function Arguments

Group, Value, SubValue, ParXtra0, ParXtra1

I/O configuration parameters, see **DtDevice::SetIoConfig**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	I/O configuration has been set successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
Other result codes	See **DtDevice::SetIoConfig**

DtOutpChannel::SetIpPars

Set IP-related parameters for the transmission of a TS-over-IP stream. The IP parameters can only be set if transmit control is **IDLE**.

```
DTAPI_RESULT DtOutpChannel::SetIpPars(
  [in] DtTsIpPars*  pTsIpPars   // TS-over-IP parameters
);
```

Function Arguments

pTsIpPars

New parameter set to be applied. Please refer to the **DtTsIpPars** page for a description of the parameters.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The TS-over-IP parameters have been applied successfully
DTAPI_E_DST_MAC_ADDR	The IP parameters cannot be applied because MAC address of destination cannot be determined. Most likely the destination address currently is invalid
DTAPI_E_IN_USE	The parameters cannot be changed because the channel is busy. Transmit control shall be switched back to **IDLE** first
DTAPI_E_INVALID_ARG	The value of one of the TS-over-IP parameters is invalid
DTAPI_E_NO_LINK	The IP parameters cannot be applied because the link is down
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	Channel is not a TS-over-IP channel

Remarks

SetIpPars should be called before transmit control is set to **HOLD** or **SEND**.

After the initial call to **SetIpPars**, parameters can be changed again, but only when transmit control is **IDLE**.

DtOutpChannel::SetModControl

Set modulation-control parameters for modulator channels. There are six overloads, five for specific modulation types (CMMB, DVB_C2, DVB-T2, ISDB-S and ISDB-T) and one for the other modulation types.

The ISDB-T overload can be used to let DTAPI perform hierarchical multiplexing. For ISDB-T without hierarchical multiplexing the first overload of **SetModControl** can be used. In that case the input of the modulator shall already be multiplexed and consist of 204-byte TMCC encoded packets.

If **SetFifoSizeTyp** has been called, **SetModControl** may change the size of the transmit FIFO to an appropriate value for the selected modulation type.

```
// Overload #1 - To be used for all modulation modes except CMMB, DVB-C2,
//               DVB-T2, ISDB-S and ISDB-T with hierarchical multiplexing
DTAPI_RESULT DtOutpChannel::SetModControl(
  [in] int  ModType,              // Modulation type: DTAPI_MOD_XXX
  [in] int  ParXtra0,             // Extra parameter #0
  [in] int  ParXtra1,             // Extra parameter #1
  [in] int  ParXtra2              // Extra parameter #2
);
// Overload #2 - To be used for CMMB
DTAPI_RESULT DtOutpChannel::SetModControl(
  [in] DtCmmbPars&  CmmbPars      // CMMB modulation parameters
);
// Overload #3 - To be used for DVB-C2
DTAPI_RESULT DtOutpChannel::SetModControl(
  [in] DtDvbC2Pars&  DvbC2Pars    // DVB-C2 modulation parameters
);
// Overload #4 - To be used for DVB-T2
DTAPI_RESULT DtOutpChannel::SetModControl(
  [in] DtDvbT2Pars&  DvbT2Pars    // DVB-T2 modulation parameters
);
// Overload #5 - To be used for ISDB-S with hierarchical multiplexing
DTAPI_RESULT DtOutpChannel::SetModControl(
  [in] DtIsdbsPars&  IsdbsPars    // ISDB-S modulation parameters
);
// Overload #6 - To be used for ISDB-T with hierarchical multiplexing
DTAPI_RESULT DtOutpChannel::SetModControl(
  [in] DtIsdbtPars&  IsdbtPars    // ISDB-T modulation parameters
                                  // for hierarchical multiplexing
);
```

Function Arguments

ModType, ParXtra0, ParXtra1, ParXtra2

Modulation parameters. See the tables on the following pages for a detailed specification of each parameter, per DekTec board type and firmware version.

CmmbPars

CMMB modulation parameters; see description of **class DtCmmbPars**.

DvbC2Pars

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

DvbT2Pars

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

IsdbsPars

ISDB-S modulation parameters for hierarchical multiplexing; see description of **class DtIsdbsPars**.

Note: For ISDB-S, the current version of DTAPI supports multiplexing of a single TS only.

IsdbtPars

ISDB-T modulation parameters for hierarchical multiplexing; see description of **class DtIsdbtPars**.

Detailed Parameter Descriptions

Page	Modulation Type
238	Overview
239	ADTB-T
241	ATSC
63	CMMB
242	DTMB
244	DVB-S
245	DVB-S.2

Page	Modulation Type
246	DVB-T / DVB-H
Error! Bookmark not defined.	DVB-T2
248	IQ-DIRECT
71, 249	ISDB-S
73, 249	ISDB-T
252	QAM

Modulation Types

ModType

Modulation type:

L-Band

ModType	Meaning	Available on
DTAPI_MOD_DVBS_QPSK	DVB-S, QPSK	DTA-107, DTA-107S2
DTAPI_MOD_DVBS2_8PSK	DVB-S.2, 8-PSK	DTA-107S2
DTAPI_MOD_DVBS2_QPSK	DVB-S.2, QPSK	DTA-107S2
DTAPI_MOD_ISDBS	ISDB-S	DTA-107, DTA-107S2

VHF* / UHF

ModType	Meaning	Available on
DTAPI_MOD_ADTBT	ADTB-T	DTA-110T, DTA-115 (DTMB option must be installed)
DTAPI_MOD_ATSC	ATSC VSB	DTA-110T Fw > 4 DTA-115 Fw > 0
DTAPI_MOD_DMBTH	DMB-T/H	DTA-110T, DTA-115 (DTMB option must be installed)
DTAPI_MOD_DVBT	DVB-T / DVB-H	DTA-110T, DTA-115
DTAPI_MOD_IQDIRECT	Direct I/Q-samples transmission	DTA-112, DTA-117 DTA-115 Fw > 1 DTA-116 Fw > 0
DTAPI_MOD_QAM16	16-QAM	ITU-T J.83 Annex A/C: DTA-110, DTA-110T, DTA-115 ITU-T J.83 Annex B: DTA-110, DTA-110T Fw > 3 DTA-115 Fw > 0
DTAPI_MOD_QAM32	32-QAM	
DTAPI_MOD_QAM64	64-QAM	
DTAPI_MOD_QAM128	128-QAM	
DTAPI_MOD_QAM256	256-QAM	

* VHF available on DTA-115 only

Modulation Mode : ADTB-T

ParXtra0

Extra modulation parameter #1 is the OR of values for the following fields: Bandwidth, Constellation, FEC Code Rate, Frame Header Mode, Interleaver Mode, Pilots and Use Frame Numbering.

Bandwidth

Value	Meaning
DTAPI_MOD_DTMB_5MHZ	5 MHz
DTAPI_MOD_DTMB_6MHZ	6 MHz
DTAPI_MOD_DTMB_7MHZ	7 MHz
DTAPI_MOD_DTMB_8MHZ	8 MHz
DTAPI_MOD_DTMB_BW_MSK	AND mask

Constellation

Value	Meaning
DTAPI_MOD_DTMB_QAM4NR	4-QAM-NR; can only be used with FEC code rate 0.8
DTAPI_MOD_DTMB_QAM4	4-QAM
DTAPI_MOD_DTMB_QAM16	16-QAM
DTAPI_MOD_DTMB_QAM32	32-QAM; can only be used with FEC code rate 0.8
DTAPI_MOD_DTMB_QAM64	64-QAM
DTAPI_MOD_DTMB_CO_MSK	AND mask

FEC Code Rate

Value	Meaning
DTAPI_MOD_DTMB_0_4	FEC code rate 0.4: FEC(7488, 3008)
DTAPI_MOD_DTMB_0_6	FEC code rate 0.6: FEC(7488, 4512)
DTAPI_MOD_DTMB_0_8	FEC code rate 0.8: FEC(7488, 6016)
DTAPI_MOD_DTMB_RATE_MSK	AND mask

Frame Header Mode

Value	Meaning
DTAPI_MOD_DTMB_PN420	PN420: Frame header 1 (420 symbols 55.6μs)
DTAPI_MOD_DTMB_PN595	PN595: Frame header 2 (595 symbols 78.7μs)
DTAPI_MOD_DTMB_PN945	PN945: Frame header 3 (945 symbols 125μs)
DTAPI_MOD_DTMB_PN_MSK	AND mask

Interleaver Mode

Value	Meaning
`DTAPI_MOD_DTMB_IL_1`	Interleaver mode 1: B=54, M=240
`DTAPI_MOD_DTMB_IL_2`	Interleaver mode 2: B=54, M=720
`DTAPI_MOD_DTMB_IL_MSK`	AND mask

Pilots

Value	Meaning
`DTAPI_MOD_DTMB_NO_PILOTS`	No pilots
`DTAPI_MOD_DTMB_PILOTS`	Add pilots; Can be used in single-carrier mode only
`DTAPI_MOD_DTMB_PIL_MSK`	AND mask

Use Frame Numbering

Value	Meaning
`DTAPI_MOD_DTMB_NO_FRM_NO`	No frame numbering
`DTAPI_MOD_DTMB_USE_FRM_NO`	Use frame numbering
`DTAPI_MOD_DTMB_UFRM_MSK`	AND mask

Modulation Mode : ATSC

ModType

ModType	Meaning	Available on	
DTAPI_MOD_ATSC	ATSC	DTA-110T DTA-115	Fw > 4 Fw > 0

ParXtra0

Extra modulation parameter #0 specifies the VSB constellation.

ParXtra0	Meaning	Symbol Rate (bd)	TS Rate (bps)
DTAPI_MOD_ATSC_VSB8	8-VSB	10,762,238	19,392,658
DTAPI_MOD_ATSC_VSB16	16-VSB	10,762,238	38,785,317
DTAPI_MOD_ATSC_VSB_MSK	AND-mask for ATSC constellation field		

ParXtra1

This parameter specifies the number of taps of each phase of the root-raised cosine filter that is used to shape the spectrum of the output signal. The number of taps can have any value between 2 and 256 (the implementation is optimized for powers of 2). Specifying more taps improves the spectrum, but increases processor overhead.

The recommend number of taps is 64 taps; If insufficient CPU power is available, 32 taps produces acceptable results, too.

ParXtra2

Not used in ATSC modulation.

Modulation Mode : DTMB

ParXtra0

Extra modulation parameter #1 is the OR of values for the following fields: Bandwidth, Constellation, FEC Code Rate, Frame Header Mode, Interleaver Mode and Use Frame Numbering.

Bandwidth

Value	Meaning
DTAPI_MOD_DTMB_5MHZ	5 MHz
DTAPI_MOD_DTMB_6MHZ	6 MHz
DTAPI_MOD_DTMB_7MHZ	7 MHz
DTAPI_MOD_DTMB_8MHZ	8 MHz
DTAPI_MOD_DTMB_BW_MSK	AND mask

Constellation

Value	Meaning
DTAPI_MOD_DTMB_QAM4NR	4-QAM-NR; can only be used with FEC code rate 0.8
DTAPI_MOD_DTMB_QAM4	4-QAM
DTAPI_MOD_DTMB_QAM16	16-QAM
DTAPI_MOD_DTMB_QAM32	32-QAM; can only be used with FEC code rate 0.8
DTAPI_MOD_DTMB_QAM64	64-QAM
DTAPI_MOD_DTMB_CO_MSK	AND mask

FEC Code Rate

Value	Meaning
DTAPI_MOD_DTMB_0_4	FEC code rate 0.4: FEC(7488, 3008)
DTAPI_MOD_DTMB_0_6	FEC code rate 0.6: FEC(7488, 4512)
DTAPI_MOD_DTMB_0_8	FEC code rate 0.8: FEC(7488, 6016)
DTAPI_MOD_DTMB_RATE_MSK	AND mask

Frame Header Mode

Value	Meaning
DTAPI_MOD_DTMB_PN420	PN420: Frame header 1 (420 symbols 55.6μs)
DTAPI_MOD_DTMB_PN945	PN945: Frame header 3 (945 symbols 125μs)
DTAPI_MOD_DTMB_PN_MSK	AND mask

Interleaver Mode

Value	Meaning
DTAPI_MOD_DTMB_IL_1	Interleaver mode 1: B=54, M=240
DTAPI_MOD_DTMB_IL_2	Interleaver mode 2: B=54, M=720
DTAPI_MOD_DTMB_IL_MSK	AND mask

Use Frame Numbering

Value	Meaning
DTAPI_MOD_DTMB_NO_FRM_NO	No frame numbering
DTAPI_MOD_DTMB_USE_FRM_NO	Use frame numbering
DTAPI_MOD_DTMB_UFRM_MSK	AND mask

Modulation Mode : DVB-S

ParXtra0

Extra modulation parameter #0

ParXtra0	Meaning
DTAPI_MOD_1_2	Code rate 1/2
DTAPI_MOD_2_3	Code rate 2/3
DTAPI_MOD_3_4	Code rate 3/4
DTAPI_MOD_4_5	Code rate 4/5
DTAPI_MOD_5_6	Code rate 5/6
DTAPI_MOD_6_7	Code rate 6/7
DTAPI_MOD_7_8	Code rate 7/8

Modulation Mode : DVB-S.2

ParXtra0

Extra modulation parameter #0 encodes the code rate.

ParXtra0	Meaning
DTAPI_MOD_1_2	Code rate 1/2
DTAPI_MOD_1_3	Code rate 1/3
DTAPI_MOD_1_4	Code rate 1/4
DTAPI_MOD_2_3	Code rate 2/3
DTAPI_MOD_2_5	Code rate 2/5
DTAPI_MOD_3_4	Code rate 3/4
DTAPI_MOD_3_5	Code rate 3/5
DTAPI_MOD_4_5	Code rate 4/5
DTAPI_MOD_5_6	Code rate 5/6
DTAPI_MOD_6_7	Code rate 6/7
DTAPI_MOD_7_8	Code rate 7/8
DTAPI_MOD_8_9	Code rate 8/9
DTAPI_MOD_9_10	Code rate 9/10

ParXtra1

Extra modulation parameter #1 encodes pilots yes/no and long/short FEC frame.

Pilots

Value	Meaning
DTAPI_MOD_S2_NOPILOTS	Pilots disabled
DTAPI_MOD_S2_PILOTS	Pilots enabled
DTAPI_MOD_S2_PILOTS_MSK	AND-mask for this field

Long or Short FECFRAME

Value	Meaning
DTAPI_MOD_S2_SHORTFRM	Short FECFRAME (16.200 bits)
DTAPI_MOD_S2_LONGFRM	Long FECFRAME (64.800 bits)
DTAPI_MOD_S2_FRM_MSK	AND-mask for this field

ParXtra2

Physical layer scrambling initialization sequence “n”, aka “Gold code”.

Modulation Mode : DVB-T/DVB-H

ParXtra0

Extra modulation parameter #0 is the code rate.

DTAPI_MOD_1_2	Code rate 1/2
DTAPI_MOD_2_3	Code rate 2/3
DTAPI_MOD_3_4	Code rate 3/4
DTAPI_MOD_5_6	Code rate 5/6
DTAPI_MOD_7_8	Code rate 7/8

ParXtra1

Extra modulation parameter #1 is the OR of values for the following fields: Bandwidth, Constellation, Guard Interval, Interleaving, Transmission Mode and DVB-H-Signalling.

Bandwidth

Value	Meaning
DTAPI_MOD_DVBT_5MHZ	5 MHz
DTAPI_MOD_DVBT_6MHZ	6 MHz
DTAPI_MOD_DVBT_7MHZ	7 MHz
DTAPI_MOD_DVBT_8MHZ	8 MHz
DTAPI_MOD_DVBT_BW_MSK	AND mask

Constellation

Value	Meaning
DTAPI_MOD_DVBT_QPSK	QPSK
DTAPI_MOD_DVBT_QAM16	16-QAM
DTAPI_MOD_DVBT_QAM64	64-QAM
DTAPI_MOD_DVBT_CO_MSK	AND mask

Guard Interval

Value	Meaning
DTAPI_MOD_DVBT_G_1_32	1/32
DTAPI_MOD_DVBT_G_1_16	1/16
DTAPI_MOD_DVBT_G_1_8	1/8
DTAPI_MOD_DVBT_G_1_4	¼
DTAPI_MOD_DVBT_GU_MSK	AND mask

Interleaving

Value	Meaning
DTAPI_MOD_DVBT_INDEPTH	In-depth interleaver (2k, 4k)
DTAPI_MOD_DVBT_NATIVE	Native interleaver
DTAPI_MOD_DVBT_IL_MSK	AND mask

Transmission Mode

Value	Meaning
DTAPI_MOD_DVBT_2K	2k mode
DTAPI_MOD_DVBT_4K	4k mode (DVB-H)
DTAPI_MOD_DVBT_8K	8k mode
DTAPI_MOD_DVBT_MD_MSK	AND mask

Disable DVB-H Signalling Service Indication

Value	Meaning
DTAPI_MOD_DVBT_ENA4849	Enable DVB-H signalling indication bits s48 and s49. *Note*: If *ParXtra2* is set to -1, s48 and s49 are disabled, too
DTAPI_MOD_DVBT_DIS4849	Disable DVB-H signalling bits by setting TPS length field to 31, or 23 when *ParXtra2* is set to -1
DTAPI_MOD_DVBT_4849_MSK	AND mask

DVB-H Signalling – Service Indication s48

Value	Meaning
DTAPI_MOD_DVBT_S48_OFF	Time slicing is not used
DTAPI_MOD_DVBT_S48	At least one elementary stream uses Time Slicing
DTAPI_MOD_DVBT_S48_MSK	AND mask

DVB-H Signalling – Service Indication s49

Value	Meaning
DTAPI_MOD_DVBT_S49_OFF	MPE-FEC is not used
DTAPI_MOD_DVBT_S49	At least one elementary stream uses MPE-FEC
DTAPI_MOD_DVBT_S49_MSK	AND mask

ParXtra2

16-bit cell identifier (cell_id). If *ParXtra2* is set to -1, the cell identifier is disabled by setting the TPS length field to 23 (this disables the DVB-H Service Indication bits s48 and s49, too).

Modulation Mode : IQ-DIRECT

ParXtra0

Extra modulation parameter #0 specifies which interpolation method is used.

Interpolation Method

Value	Meaning
DTAPI_MOD_INTERPOL_OFDM	Use OFDM interpolation
DTAPI_MOD_INTERPOL_QAM	Use QAM interpolation

ParXtra1

Extra modulation parameter #1 specifies the sample rate used by hardware to clock out I and Q samples. The valid range is 5,000,000 … 9,200,000 samples per second.

Remarks

If the modulation mode IQ-DIRECT is selected, the data written to the Transmit FIFO shall be an array of I/Q sample pairs. The samples are signed 16-bit integer in I, Q order.

Modulation Mode : ISDB-S

If ISDB-S modulation is selected using **`SetModControl`**(**`DTAPI_MOD_ISDBS`**, -1, -1, -1), no further parameters are required and DTAPI expects an 188-byte transport stream with TMCC information encoded in the SYNC bytes. DTAPI will decode the TMCC information for obtaining the required modulation type and code rates.

Modulation Mode : ISDB-T

ParXtra0

Extra modulation parameter #0 is the OR of values for the following fields: Initial Total Number of Segments, Bandwidth, Sample Rate and Sub Channel.

Initial Total Number of Segments

Value	Meaning
DTAPI_ISDBT_SEGM_1	1 segment
DTAPI_ISDBT_SEGM_3	3 segments
DTAPI_ISDBT_SEGM_13	13 segments

The DTAPI needs the number of segments to initialise the modulator and to compute bit rates. When in operation, the ISDB-T modulator dynamically follows the number of segments encoded in the TMCC information.

Bandwidth

Value	Meaning
DTAPI_ISDBT_BW_5MHZ	5 MHz
DTAPI_ISDBT_BW_6MHZ	6 MHz
DTAPI_ISDBT_BW_7MHZ	7 MHz
DTAPI_ISDBT_BW_8MHZ	8 MHz
DTAPI_ISDBT_BW_MSK	AND mask

Sample Rate

Value	Meaning
DTAPI_ISDBT_SRATE_1_1	Use nominal sample rate (512/63 MHz for 6MHz)
DTAPI_ISDBT_SRATE_1_2	Use nominal sample rate divided by 2 (at most 6 segments)
DTAPI_ISDBT_SRATE_1_4	Use nominal sample rate divided by 4 (at most 3 segments)
DTAPI_ISDBT_SRATE_1_8	Use nominal sample rate divided by 8 (at most 1 segment)
DTAPI_ISDBT_SRATE_MSK	AND mask

This is the sample rate used by the hardware.

Sub Channel

Sub-channel number (0 … 41) of the centre segment of the spectrum.
WARNING: This parameter is only used for PRBS generation, not for actual frequency translation.

Value	Meaning
DTAPI_ISDBT_SUBCH_SHIFT	Bit position of bit 0 of sub-channel number
DTAPI_ISDBT_SUBCH_MSK	AND mask for encoded sub-channel field

ParXtra1, ParXtra2
Not used.

Remarks

SetModControl(DTAPI_MOD_ISDBT, int, int, int) can be used only for modulation of "TMCC-encoded" streams with 204-byte packets (last 16 bytes containing the TMCC information). The DTAPI is capable of hierarchical multiplexing too, but for using that the overload **SetModControl(DtIsdbtPars&)** has to be used.

The ISDB-T modulator does not use the Broadcast Type parameter to set the number of segments. This enables the usage of broadcast type **BTYPE_TV** for 1-segment modulation.

Modulation Mode : QAM

ModType

The QAM constellation is encoded in *ModType* according to the following table.

ModType	Meaning	Available on
DTAPI_MOD_QAM16	16-QAM	ITU-T J.83 Annex A/C: DTA-110, DTA-110T, DTA-115 ITU-T J.83 Annex B: DTA-110, DTA-110T Fw > 3 DTA-115 Fw > 0
DTAPI_MOD_QAM32	32-QAM	
DTAPI_MOD_QAM64	64-QAM	
DTAPI_MOD_QAM128	128-QAM	
DTAPI_MOD_QAM256	256-QAM	

ParXtra0

Extra modulation parameter #0 is the ITU-T J.83 Annex.

ITU-T J.83 Annex	Meaning	Available on
DTAPI_MOD_J83_A	J.83 annex A (DVB-C)	DTA-110, DTA-110T, DTA-115
DTAPI_MOD_J83_B	J.83 annex B ("American QAM")	DTA-110, DTA-110T Fw > 3 DTA-115 Fw > 0
DTAPI_MOD_J83_C	J.83 annex C ("Japanese QAM")	DTA-110, DTA-110T, DTA-115

ParXtra1

For J.83 Annex B, this parameter specifies the interleaving mode used as specified in the table below. For Annex A and C this parameter is not used.

Value	CW	I	J	Burst protection 64-/256-QAM
DTAPI_MOD_QAMB_I128_J1D	0001	128	1	95 µs / 66 µs
DTAPI_MOD_QAMB_I64_J2	0011	64	2	47 µs / 33 µs
DTAPI_MOD_QAMB_I32_J4	0101	32	4	24 µs / 16 µs
DTAPI_MOD_QAMB_I16_J8	0111	16	8	12 µs / 8.2 µs
DTAPI_MOD_QAMB_I8_J16	1001	8	16	5.9 µs / 4.1 µs
DTAPI_MOD_QAMB_I128_J1	0000	128	1	95 µs / 66 µs
DTAPI_MOD_QAMB_I128_J2	0010	128	2	190 µs / 132 µs
DTAPI_MOD_QAMB_I128_J3	0100	128	3	285 µs / 198 µs
DTAPI_MOD_QAMB_I128_J4	0110	128	4	379 µs / 264 µs
DTAPI_MOD_QAMB_I128_J5	1000	128	5	474 µs / 330 µs
DTAPI_MOD_QAMB_I128_J6	1010	128	6	569 µs / 396 µs
DTAPI_MOD_QAMB_I128_J7	1100	128	7	664 µs / 462 µs
DTAPI_MOD_QAMB_I128_J8	1110	128	8	759 µs / 528 µs

ParXtra2

Not used.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The modulation parameters have been set successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_IDLE	Transmit-control state is not DTAPI_TXCTRL_IDLE; The requested modulation parameters can only be set in idle state
DTAPI_E_INVALID_BANDWIDTH	Invalid value for bandwidth field
DTAPI_E_INVALID_CONSTEL	Invalid value for constellation field
DTAPI_E_INVALID_FHMODE	Invalid value for frame-header mode field
DTAPI_E_INVALID_GUARD	Invalid value for guard-interval field
DTAPI_E_INVALID_INTERLVNG	Invalid value for interleaving field
DTAPI_E_INVALID_J83ANNEX DTAPI_E_INVALID_ROLLOFF	Invalid value for J.83 annex
DTAPI_E_INVALID_MODE	Modulation type is incompatible with modulator
DTAPI_E_INVALID_PILOTS	Pilots cannot be specified in C=1 mode
DTAPI_E_INVALID_RATE	Invalid value for convolutional rate or FEC code rate
DTAPI_E_INVALID_TRANSMODE	Invalid value for transmission-mode field
DTAPI_E_INVALID_USEFRAMENO	Invalid value for use-frame-numbering field
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The device does not include a modulator

Remarks

Changing the modulation parameters may change the symbol rate!
This is because DTAPI automatically computes the symbol rate from the transport-stream rate (as set with **SetTsRateBps**) and the modulation parameters.

DtOutpChannel::SetOutputLevel

Set level for modulators with an adjustable output level.

```
DTAPI_RESULT DtOutpChannel::SetOutputLevel(
  [in] int  LeveldBm         // Output level in units of 0.1dBm
);
```

Function Arguments

LeveldBm

Output level expressed in units of 0.1dBm. For example, -30 maps to -30×0.1 = -3dBm. Most modulators do not support a granularity of 0.1dBm. In that case, the output level is rounded to the nearest value supported by the modulator.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The output level has been set successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The channel does not support a controllable output level

Remarks

DtOutpChannel::SetPower

DTA-102 only. Turn on/off power for a target adapter attached to the DTA-102.

```
DTAPI_RESULT DtOutpChannel::SetPower(
  [in] int  Power            // New power state
);
```

Function Arguments

Power

New power state according to the table below.

Value	Meaning
DTAPI_POWER_OFF	No power is applied. The 25-pin sub-D connector is compatible with DVB-SPI
DTAPI_POWER_ON	Apply power (+5V) to pin 12 and 25 of the 25-pin sub-D connector

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Power state has been changed successfully
DTAPI_E_INVALID_MODE	The specified power-mode value is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The hardware function does not support target adapters

Remarks

After **Attach** and after **Reset**, power is turned off.

DtOutpChannel::SetRfControl

Set upconverter parameters for devices with on-board RF upconverter.

```
DTAPI_RESULT DtOutpChannel::SetRfControl(
  [in] __int64  RfRate       // RF frequency in Hz
);
```

Function Arguments

RfRate

New carrier frequency for RF upconverter, specified in Hertz. *RfRate* is rounded to the nearest RF frequency compatible with the frequency resolution.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The carrier frequency has been changed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_RATE	The specified carrier frequency is incompatible (too low or too high) with the upconverter
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The device does not have an RF upconverter

Remarks

Changing the RF frequency takes some time to let the PLL settle at the new frequency.

DtOutpChannel::SetRfMode

Set special modes for devices with on-board RF upconverter.

```
DTAPI_RESULT DtOutpChannel::SetRfMode(
  [in] int  RfMode           // Special upconverter mode
);
```

Function Arguments

RfMode

New RF upconverter mode according to the table below.

Value	Meaning
DTAPI_UPCONV_CW	Generate carrier only
DTAPI_UPCONV_MUTE	Mute RF output signal
DTAPI_UPCONV_NORMAL	Normal mode

The RF-modes mentioned above can be OR-ed with the values specified in the table below:

Value	Meaning
DTAPI_UPCONV_SPECINV	Apply spectral inversion

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The new upconverter mode has been set successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_MODE	The specified upconverter mode is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The channel does not support the requested operation

Remarks

DtOutpChannel::SetSnr

Sets noise generation mode and signal-to-noise ratio for modulators with a hardware-based white-noise generator.

```
DTAPI_RESULT DtOutpChannel::SetSnr(
  [in] int  Mode,             // Noise generation mode
  [in] int  SNR               // Desired signal to noise-ratio
);
```

Function Arguments

Mode

Noise generation mode to be used.

Value	Meaning
DTAPI_NOISE_DISABLED	No noise generation
DTAPI_NOISE_WNG_HW	Use build-in hardware white noise generator

SNR

Desired signal-to-noise ratio, expressed in units of 0.1dB. For example, 250 = 250×0.1 = 25dB. The table below specifies the valid range for SNR, based on the used hardware and noise generation mode.

Device	Noise mode	Valid SNR range
DTA-107	**DTAPI_NOISE_WNG_HW**	0.0 … 35.9 dB

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Noise mode and SNR have been set successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_ARG	SNR value is invalid
DTAPI_E_INVALID_MODE	The specified noise generation mode is not valid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	Setting the SNR not supported

Remarks

DtOutpChannel::SetSpiClk

DTA-2142 only. Set the DVB-SPI clock frequency in case the SPI channel is operating with a fixed clock (I/O configurations **SPIFIXEDCLK**, **SPISER8B**, **SPISER10B**).

```
DTAPI_RESULT DtOutpChannel::SetSpiClk(
  [in] Int  SpiClk              // Fixed SPI clock
);
```

Function Arguments

SpiClk

Specifies the frequency of the fixed DVB-SPI clock in Hertz.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The TS-over-IP parameters have been applied successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_INVALID_MODE	The SPI clock is not fixed (SPI mode is **SPIDVBMODE**)
DTAPI_E_NOT_SUPPORTED	Not a DVB-SPI channel

Remarks

DtOutpChannel::SetTsRateBps

Set the channel's transport-stream rate, based on 188-byte transport packets.

```
DTAPI_RESULT DtOutpChannel::SetTsRateBps(
  [in] int  TsRate          // transport-stream rate in bps
);
```

Function Arguments

TsRate

New transport-stream rate (@188) specified in bits per second.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The transport-stream rate has been changed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_RATE	The specified transport-rate is invalid (negative) or incompatible (too high) with the attached hardware function
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	Setting the TS rate is not allowed

Remarks

For transmit modes that are not based on 188-byte packets, the transport-stream rate will be different from the line clock. For example, in modes **DTAPI_TXMODE_204** and **DTAPI_TXMODE_ADD16** the line clock rate is set to 204/188 times the specified transport-stream clock.

The transport-stream rate is usually set in the initialisation phase after **AttachToPort**. It is recommended to first set the transmit mode with **SetTxMode** before setting the TS rate with **SetTsRateBps**.

SetTsRateBps may also be used while packets are being transmitted. The DTA and DTU series of devices impose no constraints on the bit-rate step size or on the number of changes per second. Note however that bit-rate changes may lead to a (temporary) violation of the MPEG-2 Systems requirements on transport-streams.

The transport-stream rate may be set to zero. This effectively disables packet transmission and may stall the output channel.

DtOutpChannel::SetTsRateRatio

Set the ratio between transport-stream rate and external clock frequency. If the external clock frequency differs from the specified frequency, the transport stream rate will differ from the desired value by the same percentage.

```
DTAPI_RESULT DtOutpChannel::SetTsRateRatio(
  [in] int  TsRate,          // Desired transport stream rate
  [in] int  RefClk           // Frequency of external clock
);
```

Function Arguments

TsRate

Transport stream rate in bits per second.

RefClk

Frequency (in Hz) of the external reference clock applied to the device.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The transport-stream ratio has been changed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_TSRATESEL	The channel is not configured in **DTAPI_TSRATESEL_EXTRATIO** mode
DTAPI_E_NOT_SUPPORTED	The hardware does not have an external clock input

Remarks

A ratio can only be set if the transport-stream rate selection (**DTAPI_IOCONFIG_TSRATESEL**) I/O configuration is **DTAPI_TSRATESEL_EXTRATIO**.

DtOutpChannel::SetTxControl

Set transmit control.

```
DTAPI_RESULT DtOutpChannel::SetTxControl(
  [in] int  TxControl        // Transmit-control state
);
```

Function Arguments

TxControl

New value for transmit control according to the table below.

Value	Meaning
DTAPI_TXCTRL_IDLE	Packet transmission and DMA writes to the transmit FIFO are disabled.
DTAPI_TXCTRL_HOLD	Packet transmission is disabled, but DMA writes to the transmit FIFO are enabled.
DTAPI_TXCTRL_SEND	Normal operation. Both packet transmission and DMA writes are enabled.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Transmit control has been changed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INSUF_LOAD	For modulators: FIFO load is insufficient to start modulation
DTAPI_E_INVALID_LEVEL	The output level specified in **SetOutputLevel** is invalid for the attached hardware function
DTAPI_E_INVALID_MODE	The specified value for transmit control is invalid or incompatible with the attached hardware function
DTAPI_E_MODPARS_NOT_SET	For modulators: cannot start transmission because modulation parameters have not been set
DTAPI_E_MODTYPE_UNSUP	For modulators: modulation type is not supported
DTAPI_E_NO_IPPARS	For TS-over-IP channels: cannot start transmission because IP parameters have not been specified yet
DTAPI_E_NO_TSRATE	Cannot start transmission because TS rate has not been set
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

Setting transmit control can be used for controlled start-up and shutdown of the streaming process. If transmit control is **DTAPI_TXCTRL_HOLD**, the transmit FIFO can be pre-loaded with packets (using **Write**), while no packets are transmitted yet. Then, if enough *credit* has been built up in the transmit FIFO, transmit control is set to **DTAPI_TXCTRL_SEND**. This procedure prevents accidental Transmit-FIFO underflow in the start-up phase.

For TS-over-IP channels, transmit control values **`HOLD`** or **`SEND`** can only be entered if the IP transmission parameters have been specified using **`SetIpPars`**.

After **`AttachToPort`** and after **`Reset`**, transmit control is initialised to **`IDLE`**.

DtOutpChannel::SetTxMode

Set the transmit mode.

```
DTAPI_RESULT DtOutpChannel::SetTxMode(
  [in] int  TxMode,          // Transmit mode
  [in] int  StuffMode        // TS: Null-packet stuffing on/off
                             // SDI: Black-frame stuffing on/off
);
```

Function Arguments

TxMode

New transmit mode according to the table below.

Value	Meaning
DTAPI_TXMODE_188	*188-byte mode* Packets in the Transmit FIFO are assumed to be 188 bytes long. Packets are transmitted without modification.
DTAPI_TXMODE_192	*192-byte mode* (DTA-102 only) Packets in the Transmit FIFO are assumed to be 192 bytes long. The SYNC byte of every second packet may be modified (not 0x47). PSYNC is pulsed at the start of every 192 byte packet.
DTAPI_TXMODE_204	*204-byte mode* Packets in the Transmit FIFO are assumed to be 204 bytes long. Packets are transmitted without modification.
DTAPI_TXMODE_ADD16	*Add 16 bytes mode* Packets in the Transmit FIFO are assumed to be 188 bytes long. The device adds 16 placeholder bytes (0) to every packet.
DTAPI_TXMODE_MIN16	*Minus 16 bytes mode* Packets in the Transmit FIFO are assumed to be 204 bytes long. The device removes the last 16 bytes of each packet.
DTAPI_TXMODE_RAW	*Raw mode* No assumptions are made on packet structure. Bytes in the buffer are transmitted unmodified. Null-packet stuffing cannot be applied. This mode is not allowed for TS-over-IP channels.

The following modes are valid for SDI-capable channels only.

Value	Meaning
DTAPI_TXMODE_SDI_FULL	*Full SDI mode* The data in the Transmit FIFO is assumed to consist of complete SDI frames, including all synchronisation information.
DTAPI_TXMODE_SDI_ACTVID	*Active Video SDI mode* The data in the Transmit FIFO is assumed to be the active video part of SDI frames. The hardware adds blanking information to create a complete frame.

	This mode can only be used in combination with Huffman compression (i.e. with **DTAPI_TXMODE_SDI_HUFFMAN** flag). When using this mode with Huffman compression disabled, the behaviour of the output channel is undefined.

The following mode is valid for TS-over-IP transmission only.

Value	Meaning
DTAPI_TXMODE_IPRAW	*Raw IP mode* The Transmit FIFO is assumed to contain time-stamped IP packets that are transmitted unmodified. Each IP packet shall be preceded by a **DtRawIpHeader** structure.

For DVB-ASI output channels, *TxMode* can be OR-ed with following flags:

DTAPI_TXMODE_BURST	*Burst mode* The bytes making up a transport packet are sent in one burst, without K28.5 stuffing characters. If this flag is not specified, transmission of packet data is "continuous" (linear over time).
DTAPI_TXMODE_TXONTIME	*Transmit on timestamp* The MPEG-2 packets in the Transmit FIFO are assumed to be prefixed with a 32-bit timestamp (54MHz resolution) and will be transmitted at the times indicated by the timestamps. See §**Error! Reference source not found.** for more details.

For SDI-capable output channels *TxMode* can be OR-ed with the values specified in the table below:

Value	Meaning
DTAPI_TXMODE_SDI	*SDI mode* Indicates that the output channel shall operate in SDI mode. If this flag is not used the channel operates in ASI mode. This flag is already OR-ed into the SDI-related transmit modes mentioned in the table above.
DTAPI_TXMODE_SDI_10B	*10-bit SDI samples* Indicates that SDI data in the Transmit FIFO is assumed to consist of 10-bit SDI samples. If this flag is omitted, 8-bit samples is assumed.
DTAPI_TXMODE_SDI_525	*525-line mode* Indicates that the SDI data in the Transmit FIFO is assumed to be 525-lines SDI.
DTAPI_TXMODE_SDI_625	*625-line mode* Indicates that the SDI data in the Transmit FIFO is assumed to be 625-lines SDI.
DTAPI_TXMODE_SDI_HUFFMAN	*Huffman compression* Indicated that the SDI frame in the Transmit FIFO is assumed to be Huffman compressed.

For genlock-capable cards the output channel can be configured to operate in genlock mode (see **SetIOConfig**). When the genlock mode and the requested transmit mode conflict, **DTAPI_E_INVALID_MODE** error will be returned.

StuffMode

This parameter controls the behaviour of the output when there is no packet data available for transmission from the Transmit FIFO.
For channels in ASI transmission mode:
If *StuffMode* is '1' (On), the output is stuffed with null packets. The size of inserted null packets is matched to *TxMode*. Packet Stuffing is not supported if *TxMode* is **DTAPI_TXMODE_RAW**, because both packet size and packet boundaries are unknown (**DTAPI_E_MODE** error is returned.)
If *StuffMode* is '0' (Off), null-packet stuffing is not applied. If the Transmit FIFO underflows:
- For DVB-ASI outputs (DTA-100, DTA-140), the output is stuffed with K28.5 characters;
- For DVB-SPI outputs (DTA-102), DVALID is de-asserted.

For channels in SDI transmission mode (e.g. **DTAPI_TXMODE_SDI_FULL**) this parameter is ignored for non-genlock capable channels.
When the card is SDI genlock capable but the output channel is not configured to operate in genlock mode this parameter can be used to manually enable *black-frame stuffing*. When black-frame stuffing is enabled, the SDI output is stuffed with black frames when there is no packet data available in the Transmit FIFO.
Note that this parameter is ignored when the TX Channel is in SDI genlock mode, since the driver will automatically enable black-frame stuffing in this case.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Transmit mode has been changed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_MODE	The specified transmit mode is invalid or incompatible with the output channel
DTAPI_E_NO_GENREF	No genlock reference port has been configured
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_IDLE	For DTA-110T and DTA-160 only: transmit mode can only be changed when transmit-control state is **DTAPI_TXCLTRL_IDLE**

Remarks

Changing the transmit mode may change the transmit-clock rate! For example, if transmit mode is changed from **DTAPI_TXMODE_ADD16** to **DTAPI_TXMODE_188** the transmit-clock is multiplied by 188/204. The DTAPI keeps the transport-stream rate constant.

The transmit mode is usually set in the initialisation phase just after **AttachToSerial**, **AttachToSlot**, **AttachToType** or **Reset**. It is recommended to set the transmit mode before setting the transport-stream rate.

It is recommended to stop transmission and clear the Transmit FIFO with **ClearFifo** before changing transmit mode.

DtOutpChannel::SetTxPolarity

Set the polarity of the DVB-ASI output signal.

```
DTAPI_RESULT DtOutpChannel::SetTxPolarity(
  [in] int  TxPolarity       // Polarity (normal or inverted)
);
```

Function Arguments

TxPolarity

New polarity according to the table below.

Value	Meaning
DTAPI_TXPOL_NORMAL	Generate a 'normal' ASI signal
DTAPI_TXPOL_INVERTED	Generate an inverted ASI signal

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The polarity of the ASI signal has been changed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_MODE	The specified polarity is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORT	Setting the polarity is not support by this output channel

Remarks

This function can be used to test if an ASI receiver is capable of receiving both normal and inverted ASI signals.

DtOutpChannel::Write

Write data to the output channel. To avoid deadlock, the user shall meet certain preconditions as described in the Remarks section below.

```
DTAPI_RESULT DtOutpChannel::Write(
  [in] char*  pBuffer,        // Buffer with data to be written
  [in] int    NumBytesToWrite // Number of bytes to be written
);
```

Function Arguments

pBuffer

Pointer to the buffer containing the data to be written to the output channel. The pointer must be aligned to a 32-bit word boundary, except for IP output channels for which there are no alignment restrictions.

NumBytesToWrite

Number of bytes to be written to the output channel. The value of *NumBytesToWrite* must be a multiple of 4, except for IP output channels, which can accept any positive value.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Write operation has been completed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_IDLE	Cannot write data because transmit control is **IDLE**
DTAPI_E_INVALID_BUF	The buffer is not aligned to a 32-bit word boundary
DTAPI_E_INVALID_SIZE	The specified transfer size is negative or not a multiple of four
DTAPI_E_NO_TSRATE	For TS-over-IP channels: cannot write data because transport-stream rate has not been specified, or TS rate is too low
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

Preconditions:

- Transmit mode must be **HOLD** or **SEND**, otherwise **DTAPI_E_IDLE** is returned;
- If transmit mode is **HOLD**, the amount of data written may not overflow the transmit FIFO, or deadlock will be the result.

If transmit mode is **SEND**, it is *strongly recommended* that the amount of data written does not overflow the transmit FIFO. Under certain conditions, overflowing the transmit FIFO will work reliably without deadlock, but these conditions are hard to specify. Generally speaking it is safer for a user application to monitor the FIFO load and never write more data than can be contained in the FIFO.

Write returns when all data has been transferred to the transmit FIFO.