

# DTAPI

## | CORE DTAPI CLASSES

## REFERENCE

Sept 2013



## Table of Contents

<b>Structures</b> .....	<b>4</b>	<b>DtCmmbPars</b> .....	<b>91</b>
struct DtCmPars .....	4	DtCmmbPars::RetrieveTsRateFromTs .....	92
struct DtCmPath .....	5	<b>DtDemodPars</b> .....	<b>93</b>
struct DtConstelPoint .....	6	DtDemodPars.....	93
struct DtDabEnsembleInfo.....	7	DtDemodPars::CheckValidity .....	94
struct DtDabService .....	8	DtDemodPars::GetModType .....	95
struct DtDabServiceComp.....	9	DtDemodPars::SetModType .....	96
struct DtDabEtiStreamSelPars.....	11	<b>IDtDemodEvent</b> .....	<b>98</b>
struct DtDabStreamSelPars .....	12	IDtDemodEvent::TuningParsHaveChanged .....	98
struct DtDabSubChannel .....	13	IDtDemodEvent::TuningFreqHasChanged.....	99
struct DtDemodLdpcStats .....	15	<b>DtIsdbsPars</b> .....	<b>100</b>
struct DtDemodMaLayerData.....	16	DtIsdbsPars .....	100
struct DtDemodMaLayerStat.....	17	DtIsdbsPars::CheckValidity .....	101
struct DtDemodParsAtsc.....	18	<b>DtIsdbtPars</b> .....	<b>102</b>
struct DtDemodParsDab .....	19	DtIsdbtPars.....	102
struct DtDemodParsDvbC2 .....	20	DtIsdbtPars::CheckValidity .....	105
struct DtDemodParsDvbS .....	21	DtIsdbtPars::ComputeRates .....	106
struct DtDemodParsDvbS2 .....	22	DtIsdbtPars::RetrieveParsFromTs.....	107
struct DtDemodParsDvbT .....	24	<b>DtSdi</b> .....	<b>108</b>
struct DtDemodParsDvbT2 .....	26	DtSdi .....	108
struct DtDemodParsLq .....	27	DtSdi::ConvertFrame .....	109
struct DtDemodParsIsdbt .....	28	<b>DtStatistic</b> .....	<b>111</b>
struct DtDemodParsQam.....	29	struct DtStatistic .....	111
struct DtDemodPlpBlocks .....	31	DtStatistic::GetValue .....	115
struct DtDeviceDesc.....	32	<b>Global Functions</b> .....	<b>116</b>
struct DtDvbC2DemodPlpSigDataPlp.....	35	::DtapiCheckDeviceDriverVersion .....	116
struct DtDvbC2DemodPlpSigData .....	37	::DtapiDeviceScan .....	117
struct DtDvbC2DemodL1Part2Plp.....	38	::DtapiDtDeviceDesc2String .....	118
struct DtDvbC2DemodL1Part2DSlice.....	40	::DtapiDtHwFuncDesc2String .....	119
struct DtDvbC2DemodL1Part2 .....	42	::DtapiGetDeviceDriverVersion.....	121
struct DtDvbC2NotchPars .....	44	::DtapiGetDtapiServiceVersion .....	122
struct DtDvbC2StreamSelPars .....	45	::DtapiGetVersion.....	123
struct DtDvbTStreamSelPars .....	46	::DtapiHwFuncScan .....	124
struct DtDvbTTpsInfo .....	47	::DtapiInitDtTslpParsFromIpString .....	126
struct DtDvbT2AuxPars .....	50	::DtapiIoStd2VidStd .....	127
struct DtDvbT2DemodL1PostPlp .....	51	::DtapiModPars2SymRate .....	128
struct DtDvbT2RfPars .....	54	::DtapiModPars2TsRate.....	129
struct DtDvbT2DemodL1Data .....	55	::DtapiPower2Voltage .....	130
struct DtDvbT2ParamInfo.....	61	::DtapiRegisterCallback .....	131
struct DtDvbT2PlpPars .....	62	::DtapiResult2Str.....	132
struct DtDvbT2StreamSelPars .....	66	::DtapiUnregisterCallback.....	133
struct DtEventArgs .....	67	::DtapiVidStd2IoStd .....	134
struct DtHwFuncDesc.....	68	::DtapiVidStd2Str .....	135
struct DtIsdbsLayerPars .....	70	::DtapiVoltage2Power .....	136
struct DtIsdbtLayerData.....	71	<b>DtDevice</b> .....	<b>137</b>
struct DtIsdbtLayerPars.....	73	DtDevice::AttachToIpAddr.....	137
struct DtIsdbtParamData.....	75	DtDevice::AttachToSerial.....	138
struct DtIsdbtStreamSelPars.....	77	DtDevice::AttachToSlot.....	139
struct DtPar .....	78	DtDevice::AttachToType .....	140
struct DtRawIpHeader .....	79	DtDevice::Detach .....	141
struct DtRsDecStats .....	80	DtDevice::DetectIoStd .....	142
struct DtTransmitter .....	81	DtDevice::FlashDisplay .....	143
struct DtTslpPars.....	82	DtDevice::GetAttribute .....	144
struct DtTslpStat .....	87	DtDevice::GetDescriptor .....	146
struct DtT2MiStreamSelPars .....	88	DtDevice::GetDeviceDriverVersion.....	147
struct DtVitDecStats.....	89	DtDevice::GetDisplayName .....	148
<b>Callback functions</b> .....	<b>90</b>		
pDtEventCallback.....	90		
<b>DtCmmbPars</b> .....	<b>91</b>		

DtDevice::GetFirmwareVersion .....	149	DtInpChannel::Reset .....	229
DtDevice::GetGenlockState .....	150	DtInpChannel::SetAdcSampleRate .....	230
DtDevice::GetIoConfig .....	151	DtInpChannel::SetAntPower .....	231
DtDevice::GetNwSpeed .....	153	DtInpChannel::SetDemodControl .....	232
DtDevice::GetRefClkCnt .....	154	DtInpChannel::SetErrorStatsMode .....	235
DtDevice::GetRefClkFreq .....	156	DtInpChannel::SetIoConfig .....	236
DtDevice::GetUsbSpeed .....	157	DtInpChannel::SetIpPars .....	237
DtDevice::GetVcxoState .....	158	DtInpChannel::SetPars .....	239
DtDevice::HwFuncScan .....	159	DtInpChannel::SetPower .....	240
DtDevice::LedControl .....	160	DtInpChannel::SetRxControl .....	241
DtDevice::SetDisplayName .....	161	DtInpChannel::SetRxMode .....	242
DtDevice::SetIoConfig .....	162	DtInpChannel::SetStreamSelection .....	245
DtDevice::SetNwSpeed .....	165	DtInpChannel::SetTunerFrequency .....	246
DtDevice::VpdDelete .....	166	DtInpChannel::Tune .....	247
DtDevice::VpdRead .....	167	<b>DtOutChannel .....</b>	<b>249</b>
DtDevice::VpdWrite .....	169	DtOutChannel .....	249
<b>DtInpChannel .....</b>	<b>170</b>	DtOutChannel::AttachToPort .....	250
DtInpChannel .....	170	DtOutChannel::ClearFifo .....	251
DtInpChannel::AttachToPort .....	171	DtOutChannel::ClearFlags .....	252
DtInpChannel::BlindScan .....	173	DtOutChannel::Detach .....	253
DtInpChannel::ClearFifo .....	175	DtOutChannel::GetAttribute .....	254
DtInpChannel::ClearFlags .....	176	DtOutChannel::GetDescriptor .....	255
DtInpChannel::Detach .....	177	DtOutChannel::GetExtClkFreq .....	256
DtInpChannel::DetectIoStd .....	178	DtOutChannel::GetFailsafeAlive .....	257
DtInpChannel::GetConstellationPoints .....	179	DtOutChannel::GetFailsafeConfig .....	258
DtInpChannel::GetDemodControl .....	180	DtOutChannel::GetFifoLoad .....	259
DtInpChannel::GetDescriptor .....	190	DtOutChannel::GetFifoSize .....	260
DtInpChannel::GetFifoLoad .....	191	DtOutChannel::GetFlags .....	261
DtInpChannel::GetFlags .....	192	DtOutChannel::GetIoConfig .....	262
DtInpChannel::GetIoConfig .....	193	DtOutChannel::GetIpPars .....	263
DtInpChannel::GetIpPars .....	194	DtOutChannel::GetModControl .....	264
DtInpChannel::GetIpStat .....	195	DtOutChannel::GetOutputLevel .....	265
DtInpChannel::GetMaxFifoSize .....	196	DtOutChannel::GetRfControl .....	266
DtInpChannel::GetPars .....	197	DtOutChannel::GetSpiClk .....	267
DtInpChannel::GetRxClkFreq .....	198	DtOutChannel::GetTargetId .....	268
DtInpChannel::GetRxControl .....	199	DtOutChannel::GetTsRateBps .....	269
DtInpChannel::GetRxMode .....	200	DtOutChannel::GetTxControl .....	270
DtInpChannel::GetStatistics .....	201	DtOutChannel::GetTxMode .....	271
DtInpChannel::GetStatus .....	202	DtOutChannel::LedControl .....	272
DtInpChannel::GetStreamSelection .....	205	DtOutChannel::Reset .....	273
DtInpChannel::GetSupportedPars .....	206	DtOutChannel::SetChannelModelling .....	274
DtInpChannel::GetSupportedStatistics .....	207	DtOutChannel::SetCustomRollOff .....	275
DtInpChannel::GetTargetId .....	208	DtOutChannel::SetFailsafeAlive .....	276
DtInpChannel::GetTsRateBps .....	209	DtOutChannel::SetFailsafeConfig .....	277
DtInpChannel::GetTunerFrequency .....	210	DtOutChannel::SetFifoSize .....	278
DtInpChannel::GetViolCount .....	211	DtOutChannel::SetIoConfig .....	279
DtInpChannel::I2CLOCK .....	212	DtOutChannel::SetIpPars .....	280
DtInpChannel::I2CRead .....	213	DtOutChannel::SetModControl .....	282
DtInpChannel::I2CUnlock .....	214	DtOutChannel::SetOutputLevel .....	304
DtInpChannel::I2CWrite .....	215	DtOutChannel::SetPower .....	305
DtInpChannel::I2CWriteRead .....	216	DtOutChannel::SetRfControl .....	306
DtInpChannel::LedControl .....	218	DtOutChannel::SetRfMode .....	307
DtInpChannel::LnbEnable .....	219	DtOutChannel::SetSnr .....	308
DtInpChannel::LnbEnableTone .....	220	DtOutChannel::SetSpiClk .....	309
DtInpChannel::LnbSendBurst .....	221	DtOutChannel::SetTsRateBps .....	310
DtInpChannel::LnbSendDiseqcMessage .....	222	DtOutChannel::SetTsRateRatio .....	311
DtInpChannel::LnbSetVoltage .....	223	DtOutChannel::SetTxControl .....	312
DtInpChannel::PolarityControl .....	224	DtOutChannel::SetTxMode .....	314
DtInpChannel::Read .....	225	DtOutChannel::SetTxPolarity .....	318
DtInpChannel::ReadFrame .....	227	DtOutChannel::Write .....	319
DtInpChannel::RegisterDemodCallback .....	228		

Copyright © 2013 by DekTec Digital Video B.V.

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

## Structures

### struct 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  $\mu\text{s}$ . The maximum delay for an 8MHz channel is  $896\mu\text{s}$ .

*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:  $\text{Speed} = f_{\text{doppler}} * 3.10^8 / f_{\text{RF}}$ .

## 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 DtDabEnsembleInfo

This structure describes the DAB ensemble information used for the statistic `DTAPI_STAT_DAB_ENSEM_INFO`.

```
struct DtDabEnsembleInfo
{
    int    m_CountryId;           // Country identifier
    int    m_EnsembleReference;   // Identifier of the ensemble
    int    m_ExtCountryCode;      // Extended country code
    int    m_InterTableId;        // International table identifier
    std::wstring m_Label;         // Label identifying this ensemble
    int    m_LocalTimeOffset;     // Local time offset in half hours from UTC
    int    m_LtoUnique;           // Covers one(=0) or several(=1) time zones
    int    m_TransmissionMode;    // Transmission mode: 1..4
    std::vector< DtDabService> m_Services; // Services in this ensemble
    std::map<int, DtDabSubChannel> m_SubChannels; // Sub-channels
};
```

### Members

*m\_CountryId*

Country identification as defined in TS 101 756.

*m\_EnsembleReference*

The number of the ensemble allocated for use within a national area.

*m\_ExtCountryCode*

Extended country code as defined in TS 101 756.

*m\_InterTableId*

International table identifier as defined in TS 101 756.

*m\_Label*

Label associated with this ensemble.

*m\_LocalTimeOffset*

The Local Time Offset (LTO) for the ensemble. It is expressed in multiples of half hours in the range -12 hours to +12 hours.

*m\_TransmissionMode*

DAB transmission mode: 1...4.

*m\_Services*

A vector specifying the services in this ensemble.

*m\_SubChannels*

A map specifying the sub-channels in this ensemble. The key in the map is the sub-channel identifier.

## struct DtDabService

This structure describes a single service in the DAB ensemble. It is used in struct `DtDabEnsembleInfo`.

```
struct DtDabService
{
    int    m_CondAccessId;        // Conditional access identifier
    int    m_CountryId;          // Country identifier
    int    m_ExtCountryCode;      // Extended country code
    bool   m_IsLocal;            // True if local
    std::wstring m_Label;        // Label identifying this service
    int    m_ServiceReference;    // Identifier of the service
    std::vector<DtDabServiceComp> m_Components; // Service components
};
```

### Members

*m\_CondAccessId*

Access Control System (ACS) identifier used for the service.

*m\_CountryId*

Country identification as defined in TS 101 756.

*m\_ExtCountryCode*

Extended country code as defined in TS 101 756.

*m\_IsLocal*

Indicates whether the service is available over the whole, or only a partial area served by the ensemble, **false**: whole ensemble service area; **true**: partial ensemble service area.

*m\_Label*

Label associated with this service.

*m\_ServiceReference*

Indicates the number of the service.

*m\_Components*

A vector specifying the service components in this service.



## struct DtDabServiceComp

This structure describes a single DAB service component. It is used in struct **DtDabService**.

```
struct DtDabServiceComp
{
    int  m_AudioServiceCompType; // Audio service component type
    int  m_DataServiceCompType;  // Data service component type
    int  m_FidChannelId;         // Fast information data channel identifier
    bool m_HasCondAccess;        // True if access control applies
    int  m_IsPrimary;            // True if this is the primary component
    std::wstring m_Label;        // Label identifying this service component
    int  m_Language;             // Service component language
    int  m_SubChannelId;         // Subchannel identifier
    int  m_ServiceCompId;        // Service component identifier
    int  m_TransportMechanismId; // Transport mechanism identifier
};
```

### Members

*m\_AudioServiceCompType*

Type of the audio component if the transport mechanism is "MSC stream audio" else -1.

Value	Meaning
0	Foreground sound (MPEG 1/2 layer 2)
1	Background sound (MPEG 1/2 layer 2)
2	Multi-channel audio extension (MPEG 2 layer 2)
63	AAC audio (DAB+) (MPEG 4 HE AAC v2)
Other	Reserved

*m\_DataServiceCompType*

Type of the data service component as defined in TS 101 756 if the transport mechanism is "MSC stream data" else -1.

Value	Meaning
24	MPEG-2 Transport Stream (DMB)
Other	Specified in TS 101 756

*m\_FidChannelId*

Identifies the service component carried in the Fast Information Data Channel (FIDC) if the transport mechanism is "FIDC" else -1.

*m\_HasCondAccess*

Indicates whether access control applies to the service component. **false**: no access control or access control applies only to a part of the service component; **true**: access control applies to the whole of the service component.

*m\_IsPrimary*

Indicates whether the service component is the primary one, **false**: not primary (secondary); **true**: primary.

*m\_Label*

Label associated with this service component.

*m\_Language*

Indicates the language of the audio or data service component as defined in TS 101 756 or -1 if not available.

*m\_SubChannelId*

Identifies the sub-channel in which the service component is carried if the transport mechanism is "MSC stream audio" or "MSC stream data" else -1.

*m\_ServiceCompId*

Uniquely identifies the service component within the ensemble if the transport mechanism is "MSC packet data" else -1.

*m\_TransportMechanismId*

The transport mechanism used.

Value	Meaning
0	Main Service Channel (MSC) - Stream mode - audio
1	Main Service Channel (MSC) - Stream mode - data
2	Fast Information Data Channel (FIDC)
3	Main Service Channel (MSC) - Packet mode - data

## struct DtDabEtiStreamSelPars

This structure specifies the selection parameters for a DAB Ensemble Transport Interface (ETI) stream.

```
struct DtDabEtiStreamSelPars
{
    // No parameters required
};
```

### Members

**DtDabEtiStreamSelPars** structure has no members

### Remark

All DAB sub-channels are selected and output in a DAB Ensemble Transport Interface (ETI) stream.

## struct DtDabStreamSelPars

This structure specifies the criteria to select a sub-channel from a DAB stream.

```
struct DtDabStreamSelPars
{
    int    m_BitrateKbps;           // Bitrate in kbps
    int    m_ErrProtLevel;          // Error protection level
    int    m_ErrProtMode;           // Error protection mode
    int    m_ErrProtOption;         // Error protection option
    int    m_StartAddress;          // Start address in capacity units
    DtDabExtractionMode m_ExtractionMode; // DAB data extraction mode
};
```

### Members

*m\_BitrateKbps*

Specifies the bitrate of the channel in kbps. For UEP profile the valid range is: 32 ... 384 kbps and the bitrate must be a multiple of 8 kbps. For EEP profile the valid range is: 8 ... 2048 kbps and the bitrate must be a multiple of 8 kbps.

*m\_ErrProtLevel*

Error protection level, the valid range for the UEP profile is: 1 ... 4; the valid range for the EEP profile is: 1 ... 5.

*m\_ErrProtMode*

Error protection mode.

Value	Meaning
DTAPI_DAB_UEP	Unequal Error Protection (UEP)
DTAPI_DAB_EEP	Equal Error Protection (EEP)

*m\_ErrProtOption*

Protection level option for EEP: 0 or 1. Only meaningful for EEP profile.

*m\_StartAddress*

Specifies the address of the first capacity unit (CU) of the sub-channel. The valid range is: 0 ... 863.

*m\_ExtractionMode*

Value	Meaning
DAB_RAW	Raw DAB stream
DAB_EXTRACTION_AAC	AAC/DAB+ stream extraction
DAB_EXTRACTION_DMB	DMB stream extraction

## struct DtDabSubChannel

This structure describes a single DAB sub-channel. A DAB sub-channel contains the data for a single audio or data stream. Multiple service components can refer to the same sub-channel. **DtDabSubChannel** is used in struct **DtDabEnsembleInfo**.

```
struct DtDabSubChannel
{
    int  m_BitrateKbps;           // Bitrate in kbps
    int  m_ErrProtLevel;          // Error protection level
    int  m_ErrProtMode;           // Error protection mode
    int  m_ErrProtOption;         // Error protection option
    int  m_FecScheme;             // FEC scheme
    int  m_StartAddress;          // Start address in capacity units
    int  m_SubChannelId;          // Subchannel identifier
    int  m_SubChannelSize;        // Size of subchannel in capacity units
    int  m_UepTableIndex;         // Index in UEP table
    int  m_UepTableSwitch;        // UEP table switch
};
```

### Members

*m\_BitrateKbps*

The bitrate of the channel in kbps.

*m\_ErrProtLevel*

Error protection level, for UEP profile: 1 ... 4; for EEP profile: 1 ... 5.

*m\_ErrProtMode*

Error protection mode.

Value	Meaning
DTAPI_DAB_UEP	Unequal Error Protection (UEP)
DTAPI_DAB_EEP	Equal Error Protection (EEP)

*m\_ErrProtOption*

Protection level option, for EEP: 0 or 1. For UEP: -1.

*m\_FecScheme*

Indicates the forward error correction (FEC) scheme in use, 0: no FEC scheme applied; 1: FEC for MSC packet mode; other values are reserved for future use.

*m\_StartAddress*

The address of the first capacity unit (CU) of the sub-channel, range: 0 ... 863.

*m\_SubChannelId*

Identifies the sub-channel.

*m\_SubChannelSize*

The number of capacity units occupied by the sub-channel.

*m\_UepTableIndex*

Index which identifies one of the 64 options available for the sub-channel size and protection level as defined in EN 300 401 table 6.

*m\_UepTableSwitch*

Indicates whether the table index refers to table 6 or some other use, 0: table 6 is used; 1: reserved.

## struct DtDemodLdpcStats

This structure describes the LDPC information used for the statistic `DTAPI_STAT_LDPC_STATS`.

```
struct DtDemodLdpcStats
{
    __int64 m_FecBlocksCount; // Total number of FEC blocks
    __int64 m_UncorrFecBlocksCount; // Number of uncorrected FEC blocks
    __int64 m_FecBlocksCount1; // Number of FEC blocks since last read
    __int64 m_FecBlocksItCount; // Total number of LDPC iterations
    int m_FecBlocksItMin; // Maximum number of LDPC iteration
    int m_FecBlocksItMax; // Maximum number of LDPC iteration
    __int64 m_BchBitCount; // Total number of decoded data bits
    __int64 m_BchBitErrorCount; // Bit error count before LDPC
};
```

### Members

*m\_FecBlocksCount*

The total number of decoded FEC-blocks.

*m\_UncorrFecBlocksCount*

The total number of uncorrected FEC-blocks (not exact).

*m\_FecBlocksCount1*

The number of decoded FEC-blocks, reset when the statistic is read.

*m\_FecBlocksItCount*

The number of LDPC iterations, the average number of LDPC-iterations =  $m\_FecBlocksItCount / m\_FecBlocksCount1$ .

*m\_FecBlockCountItMin*

The minimum number of LDPC-iterations for a FEC-block, reset when the statistic is read.

*m\_FecBlockCountItMax*

The maximum number of LDPC-iterations for a FEC-block, reset when the statistic is read.

*m\_BchBitCount*

The total number of decoded data bits, including BCH bits.

*m\_BchBitErrorCount*

The bit error count before LDPC.

The BER before LDPC is approximately:  $m\_BchBitErrorCount / m\_BchBitCount$ , this is accurate if there are no uncorrected blocks ( $m\_UncorrFecBlocksCount = 0$ ).

## struct DtDemodMaLayerData

This structure describes the DVB-C2/T2 mode adaptation layer information used for the statistic **DTAPI\_STAT\_MA\_DATA**.

```
struct DtDemodMaLayerData
{
    bool    m_Hem;                // High Efficiency Mode: yes/no
    bool    m_Npd;                // Null Packet Deletion: yes/no
    int     m_Issy;                // ISSY mode. See DTAPI_DVBX2_ISSY_XXX
    int     m_IssyBufs;           // Current ISSY BUFS value
    int     m_IssyTto;            // Last ISSY TTO value (DVB-T2 only)
    int     m_IssyBufStat;        // Last ISSY BUFSTAT value (DVB-C2/S2 only)
};
```

### 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_DVBC2_ISSY_NONE / DTAPI_DVBT2_ISSY_NONE	No ISSY field is used
DTAPI_DVBC2_ISSY_SHORT/ DTAPI_DVBT2_ISSY_SHORT	2-byte ISSY field is used
DTAPI_DVBC2_ISSY_LONG / DTAPI_DVBT2_ISSY_LONG	3-byte ISSY field is used

*m\_IssyBufs*

Value of the ISSY BUFS parameter.

*m\_IssyTto*

Last value of ISSY TTO, DVB-T2 only.

*m\_IssyBufStat*

Last value of ISSY BUFSTAT, DVB-C2/S2 only.



## struct DtDemodMaLayerStat

This structure describes the DVB-C2/T2 mode adaptation layer statistics used for the statistic `DTAPI_STAT_MA_STATS`.

```
struct DtDemodMaLayerStats
{
    __int64 m_HdrCrc8ErrorCount;    // Number BBframe header CRC8 errors
    __int64 m_PckCrc8ErrorCount;    // Number packet CRC8 errors (m_Hem=0)
    __int64 m_FramingErrorCount;    // SYNCED/DFL/UPD consistency errors
    __int64 m_CommonPlpResyncCount; // Number data-PLP common-PLP resyncs
};
```

### Members

*m\_HdrCrc8ErrorCount*

Number of CRC8 errors for BB-frame headers.

*m\_PckCrc8ErrorCount*

Number of CRC8 errors for packets, only for HEM=0.

*m\_FramingErrorCount*

Number of consistency errors found in SYNCED, DFL and UPD fields.

*m\_CommonPlpResyncCount*

Number of times a resynchronization between data and common PLP was needed. It normally happens only in case of receive errors. This field is only updated in the corresponding data PLP.

## 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 DtDemodParsDab

This structure describes the demodulation parameters for DAB.

```
struct DtDemodParsDab
{
    // Empty
};
```

### Members

**DtDemodParsDab** structure has no members.

## 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_GU_AUTO	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 DtDemodParsIsdbt

This structure describes the demodulation parameters for ISDB-T.

```
struct DtDemodParsIsdbt
{
    int    m_Bandwidth;           // Bandwidth
    int    m_SubChannel;         // Sub-channel number
    int    m_NumberOfSegments;   // Initial total number of segments
};
```

### Members

*m\_Bandwidth*  
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

*m\_SubChannel*  
Sub-channel number (0 ... 41) of the centre segment of the spectrum. The default is 22.

*m\_NumberOfSegments*  
Number of Segments

Value	Meaning
DTAPI_ISDBT_SEGM_1	1 ISDB-T segment
DTAPI_ISDBT_SEGM_3	3 ISDB-T segments
DTAPI_ISDBT_SEGM_13	13 ISDB-T segments

## 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 $\mu$ s / 66 $\mu$ s
DTAPI_MOD_QAMB_I64_J2	0011	64	2	47 $\mu$ s / 33 $\mu$ s
DTAPI_MOD_QAMB_I32_J4	0101	32	4	24 $\mu$ s / 16 $\mu$ s
DTAPI_MOD_QAMB_I16_J8	0111	16	8	12 $\mu$ s / 8.2 $\mu$ s
DTAPI_MOD_QAMB_I8_J16	1001	8	16	5.9 $\mu$ s / 4.1 $\mu$ s
DTAPI_MOD_QAMB_I128_J1	0000	128	1	95 $\mu$ s / 66 $\mu$ s
DTAPI_MOD_QAMB_I128_J2	0010	128	2	190 $\mu$ s / 132 $\mu$ s
DTAPI_MOD_QAMB_I128_J3	0100	128	3	285 $\mu$ s / 198 $\mu$ s
DTAPI_MOD_QAMB_I128_J4	0110	128	4	379 $\mu$ s / 264 $\mu$ s
DTAPI_MOD_QAMB_I128_J5	1000	128	5	474 $\mu$ s / 330 $\mu$ s
DTAPI_MOD_QAMB_I128_J6	1010	128	6	569 $\mu$ s / 396 $\mu$ s
DTAPI_MOD_QAMB_I128_J7	1100	128	7	664 $\mu$ s / 462 $\mu$ s
DTAPI_MOD_QAMB_I128_J8	1110	128	8	759 $\mu$ s / 528 $\mu$ s
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.

## Remarks

For DTA-2136 and DTA-2139 J.83 annex A (DVB-C) QAM-128 is not supported

## struct DtDemodPlpBlocks

This structure describes the number of FEC-blocks for DVB-C2/T2 used for the statistic **DTAPI\_STAT\_PLP\_BLOCKS**.

```
struct DtDemodMaLayerStats
{
    int  m_NumBlocks;           // Last plp_num_blocks value
    int  m_NumBlocksMin;       // Minimum plp_num_blocks value
    int  m_NumBlocksMax;       // Maximum plp_num_blocks value
};
```

### Members

*m\_NumBlocks*

Last plp\_num\_blocks value.

*m\_NumBlocksMin*

Minimum plp\_num\_blocks value, set to -1 when the statistic is read.

*m\_NumBlocksMax*

Maximum plp\_num\_blocks value, reset when the statistic is read.

## 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
    unsigned char m_Ip[4];       // IPv4 address
    unsigned char m_IpV6[3][16]; // 3x IPv6 address
    unsigned char m_MacAddr[6];  // MAC address
};
```

### 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 <sup>1</sup> .
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.

<sup>1</sup> This is an error condition that cannot occur for a correctly working board.

*m\_NumPorts*

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

*m\_Ip*

For devices in category **DTAPI\_CAT\_IP** and **DTAPI\_CAT\_NIC**, this member identifies the IPv4 address of the device. Otherwise, the value of this member is undefined.

*m\_IpV6[3]*

For devices in category **DTAPI\_CAT\_IP** and **DTAPI\_CAT\_NIC**, this member identifies the IPv6 addresses of the device (if the device supports IPv6 and IPv6 is enabled). Otherwise, the values of this member are undefined.

*m\_Mac*

For devices in category **DTAPI\_CAT\_IP** and **DTAPI\_CAT\_NIC**, 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-1 part 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_TIDEPH_NONE	No time interleaving
DTAPI_DVBC2_TIDEPH_4	4 OFDM symbols
DTAPI_DVBC2_TIDEPH_8	8 OFDM symbols
DTAPI_DVBC2_TIDEPH_16	16 OFDM symbols



*m\_Type*

Data slice type.

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

*m\_FecHdrType*

FEC frame header type.

Value	Meaning
DTAPI_DVBC2_FECHDR_TYPE_ROBUST	Robust mode
DTAPI_DVBC2_FECHDR_TYPE_HEM	High efficiency mode

*m\_ConstConfig*

If '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 DtDvbC2StreamSelPars

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 DtDvbTStreamSelPars

This structure specifies the selection parameters for a DVB-T transport stream.

```
struct DtDvbTStreamSelPars
{
    // No parameters required
};
```

### Members

### Remarks

No additional parameters are required to select a DVB-T transport stream. Hierarchical DVB-T demodulation is not supported.

## struct DtDvbTTpsInfo

This structure describes the DVB-T Transmission Parameter Signalling (TPS) information used for the statistic `DTAPI_STAT_DVBT_TPS_INFO`.

```
struct DtDvbTTpsInfo
{
    int  m_LengthIndicator;    // TPS length indicator
    int  m_Constellation;      // Constellation
    int  m_HpCodeRate;         // High priority stream code rate
    int  m_LpCodeRate;         // Low priority stream code rate
    int  m_Guard;              // Guard interval
    int  m_Interleaving;       // Interleaving
    int  m_Mode;               // Transmission mode
    int  m_Hierarchy;          // Hierarchy information
    int  m_CellId;             // Cell identifier or -1 if not available
    int  m_HpS48S49;           // High priority S48 and S49
    int  m_LpS48S49;           // Low priority S48 and S49
    int  m_OddS50_S53;         // S50...S53 of the odd frames
    int  m_EvenS50_S53;        // S50...S53 of the even frames
};
```

### Members

*m\_LengthIndicator*  
TPS length indicator field.

*m\_Constellation*  
DVB-T constellation pattern.

Value	Meaning
<code>DTAPI_MOD_DVBT_QPSK</code>	QPSK
<code>DTAPI_MOD_DVBT_QAM16</code>	16-QAM
<code>DTAPI_MOD_DVBT_QAM64</code>	64-QAM

*m\_HpCodeRate, m\_LpCodeRate*  
High priority stream and low priority stream code rate.

<code>DTAPI_MOD_1_2</code>	Code rate 1/2
<code>DTAPI_MOD_2_3</code>	Code rate 2/3
<code>DTAPI_MOD_3_4</code>	Code rate 3/4
<code>DTAPI_MOD_5_6</code>	Code rate 5/6
<code>DTAPI_MOD_7_8</code>	Code rate 7/8

*m\_Guard*  
Guard interval.

Value	Meaning
<code>DTAPI_MOD_DVBT_G_1_32</code>	1/32
<code>DTAPI_MOD_DVBT_G_1_16</code>	1/16

DTAPI_MOD_DVBT_G_1_8	1/8
DTAPI_MOD_DVBT_G_1_4	1/4

#### *m\_Interleaving*

Interleaving.

Value	Meaning
DTAPI_MOD_DVBT_INDEPTH	In-depth interleaver (2k, 4k)
DTAPI_MOD_DVBT_NATIVE	Native interleaver

#### *m\_Mode*

Transmission mode.

Value	Meaning
DTAPI_MOD_DVBT_2K	2k mode
DTAPI_MOD_DVBT_4K	4k mode (DVB-H)
DTAPI_MOD_DVBT_8K	8k mode

#### *m\_Hierarchy*

Hierarchy information.

Value	Meaning
DTAPI_MOD_DVBT_HARCHY_N ONE	Non hierarchical
DTAPI_MOD_DVBT_HARCHY_A 1	Alpha = 1
DTAPI_MOD_DVBT_HARCHY_A 2	Alpha = 2
DTAPI_MOD_DVBT_HARCHY_A 4	Alpha = 4

#### *m\_CellId*

16-bit cell identifier (cell\_id) or -1 if not available.

#### *m\_HpS48S49, m\_LpS48S49*

High priority stream and low priority stream S48 and S49 bits for DVB-H or -1 if not available. The values may be OR-ed together.

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_S49_OFF	MPE-FEC is not used
DTAPI_MOD_DVBT_S49	At least one elementary stream uses MPE-FEC



*m\_OddS50\_S53, m\_EvenS50\_53*

Reserved bits of odd and even frames.

## 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_AuxPrivateConfig;     // 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_DVB_T2_PLP_TYPE_COMM	Common PLP
DTAPI_DVB_T2_PLP_TYPE_1	Data PLP type1
DTAPI_DVB_T2_PLP_TYPE_2	Data PLP type2

*m\_PayloadType*

PLP payload type.

Value	Meaning
DTAPI_DVB_T2_PAYLOAD_GFPS	Generic Fixed-length Packetized Stream
DTAPI_DVB_T2_PAYLOAD_GCS	Generic Continuous Stream
DTAPI_DVB_T2_PAYLOAD_GSE	Generic Stream Encapsulation
DTAPI_DVB_T2_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 T1 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
<b>DTAPI_DVBT2_IL_ONETOONE</b>	One interleaving frame corresponds to one T2 frame
<b>DTAPI_DVBT2_IL_MULTI</b>	One interleaving frame is carried in multiple T2 frames

#### *m\_InBandAFlag*

If '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
<b>DTAPI_DVBT2_PLP_MODE_NONE</b>	Not specified, used if DVB-T2 V1.1.1 is selected.
<b>DTAPI_DVBT2_PLP_MODE_NORMAL</b>	Normal Mode
<b>DTAPI_DVBT2_PLP_MODE_HEM</b>	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\_Pl.m\_Valid*

If true, P1 signalling data has been successfully decoded and is valid.

*m\_Pl.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\_Pl.m\_Miso*

If '1', MISO is used otherwise SISO.

*m\_Pl.m\_Fef*

If '1', FEF-parts are used.

*m\_Pl.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 Deletion: 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_TimeIILength;       // Time interleaving length: 0...255
    int     m_TimeIILType;        // 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 DtDvbT2StreamSelPars

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 DtEventArgs

This structure contains all information about the occurred event.

```
struct DtEventArgs
{
    int    m_HwCat;           // Hardware category: DTAPI_CAT_XXX
    __int64 m_Serial;         // Serial number of device causing the event
    int    m_Value1;          // Event value #1
    int    m_Value2;          // Event value #2
    void*   m_pContext;       // Opaque pointer passed when registered
};
```

### Members

*m\_HwCat*

Identifies the hardware category:

Parameter	Meaning
DTAPI_CAT_PCI	PCI or PCI-Express device
DTAPI_CAT_USB	USB-2 or USB-3 device
DTAPI_CAT_NW	Network device
DTAPI_CAT_IP	Network appliance: DTE-31xx
DTAPI_CAT_NIC	Non-DekTec network card

*m\_Serial*

Serial number of the device causing the event.

*m\_Value1*

Event specific value #1:

Parameter	Meaning
DT_EVENT_VALUE1_POWER_UP	A device power up has been occurred.
DT_EVENT_VALUE1_POWER_DOWN	A device power down has been occurred.
DT_EVENT_VALUE1_NO_LOCK	Internal PLLs have not achieved or lost lock.
DT_EVENT_VALUE1_LOCKED	Full clock-lock has been achieved.

*m\_Value2*

Event specific value #2. This member variable is not used and reserved for future use.

*m\_pContext*

Opaque pointer that is passed to **DtapiRegisterCallback** when subscribing to the event.

## 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)
    unsigned char m_Ip[4];             // IPv4 address (TS-over-IP only)
    unsigned char m_IpV6[3][16];      // IPv6 address (TS-over-IP only)
    unsigned char m_MacAddr[6];       // MAC address (TS-over-IP 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
<b>DTAPI_CHAN_DISABLED</b>	n.a.	Channel is disabled
<b>DTAPI_CHAN_INPUT</b>	<b>DtInpChannel</b>	Input channel
<b>DTAPI_CHAN_OUTPUT</b>	<b>DtOutpChannel</b>	Output channel
<b>DTAPI_CHAN_DBLBUF</b>	n.a.	The hardware function is a double-buffered copy of another hardware function
<b>DTAPI_CHAN_LOOPTHR</b>	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 must be disabled before port #1 is configured for APSK operation. Only physical port #1 can be 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`

IPv4 address of the hardware function. This field is only valid for functions with capability `DTAPI_CAP_IP`.

`m_IpV6[3]`

Array of IPv6 addresses of the hardware function. This field is only valid for functions with capability `DTAPI_CAP_IP`. The array contains the link-local, site-local and global IPv6 addresses associated to the hardware function. If an IPv6 address type is not available, an entry with all 0's is added to the end.

`m_MacAddr`

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 DtIsdbtLayerData

This structure specifies the ISDB-T modulation parameters for one hierarchical layer. It is used in class `DtIsdbtParamData` used for the statistic `DTAPI_STAT_ISDBT_PARSDATA`.

```
struct DtIsdbtLayerData
{
    int  m_NumSegments;           // Number of segments
    int  m_Modulation;            // Modulation type
    int  m_CodeRate;              // Code rate
    int  m_TimeInterleave;        // Time interleaving
};
```

### 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 *l* 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



## 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 *l* 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 DtIsdbtParamData

This structure specifies the ISDB-T modulation parameters. It is used in class **DtIsdbtParamData** used for the statistic **DTAPI\_STAT\_ISDBT\_PARSDATA**.

```
struct DtIsdbtPars
{
    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
    DtIsdbtLayerData m_LayerPars[3]; // Layer-A/B/C parameters
};
```

### Public Members

*m\_BType*

Broadcast type.

Value	Meaning
DTAPI_ISDBT_BTTYPE_TV	TV broadcast; Can be used with any number of segments
DTAPI_ISDBT_BTTYPE_RAD1	1-segment radio broadcast; Total #segments must be 1
DTAPI_ISDBT_BTTYPE_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\_LayerPars*

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

## Remarks

## struct DtIsdbtStreamSelPars

This structure specifies the selection parameters for an ISDB-T stream.

```
struct DtIsdbtStreamSelPars
{
    // Empty
};
```

### Members

**DtIsdbtStreamSelPars** structure has no members

### Remark

All layers are selected and output the same stream

## struct DtPar

This is a generic structure to represent a single parameter setting. It is used for setting and retrieving parameter settings.

```
struct DtPar
{
    DTAPI_RESULT m_Result;    // Result of retrieving the parameters
    int m_ParId;              // DTAPI_PAR_XXX
    int m_IdXtra[4];          // Extra identification parameters
    ParValueType m_ValueType; // Value types of the parameter
    union {
        bool m_ValueBool;    // Value if value type is PAR_VT_BOOL
        double m_ValueDouble; // Value if value type is PAR_VT_DOUBLE
        int m_ValueInt;       // Value if value type is PAR_VT_INT
        void* m_pValue;       // Pointer for complex types
    };
};
```

### Members

*m\_Result*

If the user queries one or more parameters with **GetPar**, this member is used to return the result code for retrieving the parameter.

*m\_ParId*

Identifies the parameter:

Parameter	Type	Meaning
DTAPI_PAR_DEMOD_THREADS	int	Number of threads used for the software demodulation; The default value is 4.
DTAPI_PAR_DEMOD_LDPC_MAX	int	Maximum number of iterations for DVB-C2/T2 LDPC-error correction; The default value is 50.
DTAPI_PAR_DEMOD_LDPC_AVG	int	Limit for the average number of iterations for DVB-C2/T2 LDPC-error correction; The default value is 16.
DTAPI_PAR_DEMOD_MER_ENA	bool	Enable (true) or disable MER calculation; The default value is 'true'.

*m\_IdXtra[4]*

Extra identification parameters. Not yet used.

*m\_ValueType*

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

Value	Meaning
PAR_VT_BOOL	The value type is bool
PAR_VT_DOUBLE	The value type is double
PAR_VT_INT	The value type is int
PAR_VT_UNDEFINED	The value is not valid yet

## 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 DtRsDecStats

This structure specifies the Reed-Solomon decoder statistics used for the statistic **DTAPI\_STAT\_RSDEC\_STATS**.

```
struct DtRsDecStats
{
    bool    m_Locked;           // Decoder is locked
    __int64 m_ByteSkipCount;    // Bytes skipped while looking for sync
    __int64 m_PacketCount;      // Decoded packets count
    __int64 m_UncorrPacketCount; // Uncorrected packets count
    __int64 m_ByteErrorCount;   // Byte error count
    __int64 m_BitErrorCount;    // Bit error count
};
```

### Members

*m\_Locked*

Indication whether the Reed-Solomon decoder is locked.

*m\_ByteSkipCount*

The total number of bytes skipped while looking for synchronisation.

*m\_PacketCount*

The total number of decoded transport stream packets.

*m\_UncorrPacketCount*

The total number of uncorrected transport stream packets.

*m\_ByteErrorCount*

The byte error count before Reed-Solomon.

*m\_BitErrorCount*

The bit error count before Reed-Solomon.



## 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 DtTslpPars

The parameters in this structure are used when starting transmission or reception of a TS-over-IP stream to or from a unicast or multicast destination. **DtTslpPars** is also used to read back TS-over-IP parameters using **DtIpChannel::GetIpPars**.

Both IPv4 and IPv6 are supported, as selected by the **DTAPI\_IP\_V4/DTAPI\_IP\_V6** flag in *m\_Flags*.

**DtTslpPars** supports network redundancy by setting *m\_Mode* to **DTAPI\_IP\_TX\_DBLBUF** (transmission on two links) or **DTAPI\_IP\_RX\_DBLBUF** (reception from two links). If one of these modes is used, the redundant link parameters have to be specified in the six struct members starting at *m\_Ip2*. See the description of the members for more information. Network redundancy is currently supported only by DekTec's DTA-2162 (Dual GigE port card for PCIe).

```
struct DtTslpPars
{
    // Main link
    unsigned char m_Ip[16];           // IP address (IPv4/IPv6)
    unsigned short m_Port;            // Port number
    unsigned char m_SrcFltIp[16];     // Source filter: IP address (IPv4/IPv6)
    unsigned short m_SrcFltPort;      // Source filter: port number
    int m_VlanId;                     // VLAN ID
    int m_VlanPriority;                // VLAN priority

    // Redundant link (mode DTAPI_IP_TX_DBLBUF, DTAPI_IP_RX_DBLBUF only)
    u_char m_Ip2[16];                 // IP address (IPv4/IPv6)
    u_short m_Port2;                  // Port number
    u_char m_SrcFltIp2[16];           // Source filter: IP address (IPv4/IPv6)
    u_short m_SrcFltPort2;            // Source filter: port number
    int m_VlanId2;                    // VLAN ID
    int m_VlanPriority2;               // VLAN priority

    int m_TimeToLive;                 // Time-to-live (TTL) 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;                      // Control flags
    int m_Mode;                       // Redundancy mode
};
```

### Members when used for IP reception (Rx)

*m\_Ip* (main link), *m\_Ip2* (redundant link)

IP address from which to receive IP packets, specified as 4 bytes for IPv4 and 16 bytes for IPv6. For unicast receive, all bytes can be set to 0. If the IP address is in the multicast range, **DTAPI** automatically joins and drops membership of the multicast group when required. The **DTAPI\_IP\_V4/DTAPI\_IP\_V6** flag in *m\_Flags* member selects between using the IPv4 and IPv6 protocol.

*m\_Port*, *m\_Port2*

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, m\_SrcFltIp2*

Source IP address for source-specific multicast.

Relevant only if the IP address is a multicast address. In this case *m\_SrcFltIp(2)* can be set to a specific IP address for listening to a single multicast source.

If *m\_SrcFltIp(2)* is set to 0.0.0.0 (or 16x 0 for IPv6), any-source multicast is used and the multicast sender may be any IP address.

*m\_SrcFltPort, m\_SrcFltPort2*

Source-specific multicast: source port number associated with *m\_SrcFltIp(2)*.

*m\_SrcPort(2)* may be set to a specific source port number, or to 0 for accepting IP packets from any source port.

*m\_VlanId, m\_VlanId2*

VLAN identifier (VID) as specified in IEEE 802.1Q. If set to 0, VLAN is not used. If set to a positive integer, only packets with the specified VLAN identifier are received.

*m\_VlanPriority, m\_VlanPriority2*

Not used for Rx.

*m\_TimeToLive*

Not used for Rx.

*m\_NumTpPerTp*

Not used.

Read back: Number of transport packets stored in one IP packet in the incoming stream.

*m\_Protocol*

Expected protocol.

Write: Only automatic detection (value **DTAPI\_PROTO\_AUTO**) is allowed.

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

Value	Meaning
<b>DTAPI_PROTO_UDP</b>	UDP
<b>DTAPI_PROTO_RTP</b>	RTP
<b>DTAPI_PROTO_AUTO</b>	Automatically detect UDP or RTP

*m\_DiffServ*

Not used.

*m\_FecMode*

Error-correction mode.

Value	Meaning
<b>DTAPI_FEC_DISABLE</b>	Do not apply error correction
<b>DTAPI_FEC_2D</b>	Apply error correction with the FEC streams received on port numbers <i>m_Port+2</i> and <i>m_Port+4</i>

*m\_FecNumRows, m\_FecNumCols*

Not used.

Read back: Number of rows and columns in the FEC matrix that has been detected. In SMPTE-2022 these parameters are called *D* and *L* respectively.

*m\_Flags*

Control/status flags. In the current version of DTAPI a single flag is supported, selecting between IPv4 and IPv6 addressing.

Value	Meaning
<b>DTAPI_IP_V4</b>	Use the IPv4 protocol. All IP addresses are 4 bytes long. This is the default when neither <b>DTAPI_IP_V4</b> nor <b>DTAPI_IP_V6</b> is specified.
<b>DTAPI_IP_V6</b>	Use the IPv6 protocol. All IP addresses are 16 bytes long.

*m\_Mode*

IP redundancy mode.

Value	Meaning
<b>DTAPI_IP_NORMAL</b>	Normal mode: non-redundant IP reception. The constructor of <b>DtTsIpPars</b> initializes <i>m_Mode</i> to this mode, so this is the default.
<b>DTAPI_IP_RX_DBLBUF</b>	Receive packets from two IP ports and merge the packets into a single logical stream. This mode can only be used when the port supports capability <b>CAP_IP_PAIR</b> and has an <u>odd</u> physical port number N. The port will be paired with IP port N+1. The redundant link parameters must have to be valid. The incoming streams on ports N and N+1 must be equal to each other, with the exception of the IP-address fields. Such streams can be generated by e.g. a DTA-2162 in <b>DTAPI_IP_TX_DBLBUF</b> mode.  This mode is not supported for IPv6 with the UDP protocol. If using the DBLBUF mode with IPv6 and UDP, unpredictable output is returned to the application. When using IPv6, only the RTP protocol is supported for receive.

**Members when used for IP transmission (Tx)***m\_Ip* (main link), *m\_Ip2* (redundant link)

IP destination address that will be used for transmission. *m\_Ip* is specified as 4 bytes for IPv4 and 16 bytes for IPv6. The **DTAPI\_IP\_V4/DTAPI\_IP\_V6** flag in *m\_Flags* member selects between using the IPv4 and IPv6 protocol.

*m\_Port*, *m\_Port2*

Destination port number. When the protocol is RTP, the port number shall be even.

*m\_SrcFltIp*, *m\_SrcFltIp2*

Not used for specifying Tx parameters.  
Read back: The host's IP address.

*m\_SrcFltPort*, *m\_SrcFltPort2*

Source port number used for transmission. This member is used only if the **DTAPI\_IP\_TX\_MANSRCPORT** flag in *m\_Flags* is set. If the **DTAPI\_IP\_TX\_MANSRCPORT** flag is not set, a free source port is assigned automatically.

Read back: The selected source port number.

*m\_VlanId, m\_VlanId2*

VLAN identifier (VID) as specified in IEEE 802.1Q. If set to 0, VLAN is not used. If set to a positive integer, the IP packets will be tagged with the specified VLAN identifier.

*m\_VlanPriority, m\_VlanPriority2*

Priority value which refers to the IEEE 802.1p Priority Code Point (PCP). It is used in the VLAN header as specified in IEEE 802.1Q.

*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 used 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 *Type Of Service*) in the IPv4 header or *Traffic Class* field in the IPv6 header. The valid range is 0...255. The default value is 0.

Bits	Field	Meaning
7..2	DSCP	Differentiated Services Code Point
1..0	ECN	Explicit Congestion Notification

*m\_FecMode*

Error-correction mode.

Value	Meaning
DTAPI_FEC_DISABLE	Do not add FEC packets
DTAPI_FEC_2D	Generate FEC packets as specified in SMPTE-2022

*m\_FecNumRows, m\_FecNumCols*

Number of rows and columns in the FEC matrix. In SMPTE-2022 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*

Control/status flags.

Value	Meaning
DTAPI_IP_V4	Use the IPv4 protocol. All IP addresses are 4 bytes long. This is the default when neither <b>DTAPI_IP_V4</b> nor <b>DTAPI_IP_V6</b> is specified (this is if <i>m_Flags</i> is 0)
DTAPI_IP_V6	Use the IPv6 protocol. All IP addresses are 16 bytes long.

<b>DTAPI_IP_TX_MANSRCPORT</b>	Use <i>m_SrcFltPort</i> and <i>m_SrcFltPort2</i> as the source port for transmitting packets. If this flag is not specified, a free source port is assigned automatically.
-------------------------------	--

*m\_Mode*  
IP redundancy mode.

Value	Meaning
<b>DTAPI_IP_NORMAL</b>	Normal mode: non-redundant IP transmission. The constructor of <b>DtTsIpPars</b> initializes <i>m_Mode</i> to this mode, so this is the default.
<b>DTAPI_IP_TX_DBLBUF</b>	Transmit TS-over-IP packets to two physical IP ports at once. Both streams will be equal to each other, with the exception of the IP-addressing fields. This mode can only be used when the port supports capability <b>CAP_IP_PAIR</b> and has an <u>odd</u> physical port number N. The redundant link is port N+1. The redundant link parameters must have to be valid.

## struct DtTsIpStat

Structure for retrieving IP statistics from drivers. The counters will not be reset after read and a counter wrap must be handled by the application.

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

### 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.

## struct DtT2MiStreamSelPars

This structure specifies the selection parameters for a T2-MI transport stream containing a complete DVB-T2 stream.

```
struct DtT2MiStreamSelPars
{
    int  m_T2MiOutPid;           // T2-MI output PID
    int  m_T2MiTsRate;          // T2-MI transport stream rate
};
```

### Members

*m\_T2MiOutPid*

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

*m\_T2MiTsRate*

Specifies the T2-MI transport-stream rate in bits per second. If set to '-1' a variable bitrate transport stream is created, else null packets are added to reach the specified rate. The maximum rate is 72 Mbps.

In case the specified transport-stream rate is too low, T2-MI overflows occur. The number of overflows can be retrieved using the **DTAPI\_STAT\_T2MI\_OVFS** statistic.

### Remarks

T2-MI transport stream selection cannot be combined with DVB-T2 stream (PLP) selection.



## struct DtVitDecStats

This structure specifies the Viterbi decoder statistics used for the statistic `DTAPI_STAT_VITDEC_STATS`.

```
struct DtVitDecStats
{
    __int64 m_BitCount;           // Input bit count
    __int64 m_BitErrorCount;     // Bit error count
};
```

### Members

*m\_BitCount*

The total number of decoded bits.

*m\_BitErrorCount*

The bit error count before Viterbi.

## Callback functions

### pDtEventCallback

Prototype of a callback function to be supplied by the user with `::DtapiRegisterCallback`. This callback function is invoked when an event has occurred.

```
typedef void (*pDtEventCallback) (  
    [out] int    Event           // Occurred event  
    [out] const DtEventArgs*    pArgs // Pointer to event arguments structure  
);
```

#### Parameters

*Event*

Identifies the event that has been occurred. See `::DtapiRegisterCallback` for a list of applicable values.

*pArgs*

Pointer to the `DtEventArgs` event argument structure with additional information about the event that has been occurred.

#### Remarks

The callback function may not block and the amount of processing should be kept as low as possible to avoid stalling the `DTAPI` or driver. In case significant processing time is required the data should be written to a temporary buffer and be processed in another thread.

## DtCmmbPars

### 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***

### **DtDemodPars**

The **DtDemodPars** class encapsulates demodulation parameters.

```
class DtDemodPars;
```

## DtDemodPars::CheckValidity

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_DAB	DAB	DtDemodParsDab	Dab()
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	I/Q samples	DtDemodParsIq	Iq()
DTAPI_MOD_ISDBT	ISDB-T	DtDemodParsIsdbt	Isdbt()
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

## 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 therefore 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

## 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_BTTYPE_TV	TV broadcast; Can be used with any number of segments
DTAPI_ISDBT_BTTYPE_RAD1	1-segment radio broadcast; Total #segments must be 1
DTAPI_ISDBT_BTTYPE_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
<code>DTAPI_ISDBT_LAYER_A</code>	Map elementary stream to layer A
<code>DTAPI_ISDBT_LAYER_B</code>	Map elementary stream to layer B
<code>DTAPI_ISDBT_LAYER_C</code>	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 <i>m_ParXtra0</i>
DTAPI_ISDBT_E_BTYPE	Illegal value for broadcast type ( <i>m_BType</i> )
DTAPI_ISDBT_E_GUARD	Illegal value for guard-interval length ( <i>m_Guard</i> )
DTAPI_ISDBT_E_MODE	Illegal value for transmission mode ( <i>m_Mode</i> )
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 <i>m_ParXtra0</i> does not match number of segments specified in <i>m_LayerPars</i>
DTAPI_ISDBT_E_NOTFILLED	Member <i>m_FilledOut</i> 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 <i>m_ParXtra0</i>
DTAPI_ISDBT_E_SUBCHANNEL	Invalid sub-channel number. For 1-segment radio: $1 \leq \text{sub channel} \leq 40$ For 1-segment radio: $4 \leq \text{sub channel} \leq 37$

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	ISDB-T modulation parameters have been found valid
DTAPI_E_INVALID_PARS	ISDB-T parameters are invalid. <i>ResultCode</i> 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***

### **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

## DtStatistic

### struct DtStatistic

This structure represents a single measurement or statistic value, mostly for RF receiver cards. It is used in `DtInpChannel::GetStatistics` to pass measurements and statistics information. Please use `DtInpChannel::GetSupportedStatistics` to obtain a list of statistics supported for a given input channel.

```
struct DtStatistic
{
    DTAPI_RESULT m_Result;           // Result of retrieving the statistic
    int m_StatisticId;               // DTAPI_STAT_XXX
    int m_IdXtra[4];                 // Extra identification parameters
    int m_ValueType;                 // Identifies the type of the value
    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 not boolean,
                                     // int or double
    };
};
```

### 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 statistic from the receiver hardware.

*m\_StatisticId*

Identifies the statistic:

Statistic	Type	Meaning
DTAPI_STAT_AGC1	int	Returns the AGC1 value
DTAPI_STAT_AGC2	int	Returns the AGC2 value
DTAPI_STAT_BADPCKCNT	int	Count of uncorrected packets since last call. Returns 0 if the receiver is not locked.
DTAPI_STAT_BER_POSTBCH	double	Post-BCH bit error rate
DTAPI_STAT_BER_POSTLDPC	double	Post-LDPC bit error rate. <i>m_IdXtra[0]</i> identifies: DVB-C2/T2 PLP.
DTAPI_STAT_BER_POSTVIT	double	Post-Viterbi bit error rate. <i>m_IdXtra[0]</i> identifies: DAB sub-channel's start-address or ISDB-T layer.
DTAPI_STAT_BER_PREBCH	double	Pre-BCH bit error rate. <i>m_IdXtra[0]</i> identifies: DVB-C2/T2 PLP.
DTAPI_STAT_BER_PRELDPC	double	Pre-LDPC bit error rate. <i>m_IdXtra[0]</i> identifies: DVB-C2/T2 PLP.
DTAPI_STAT_BER_PRERS	double	Pre-Reed-Solomon bit error rate. <i>m_IdXtra[0]</i> identifies: DAB sub-channel's start-

		address or ISDB-T layer.
DTAPI_STAT_BER_PREVIT	double	Pre-Viterbi bit error rate. m_IdXtra[0] identifies: DAB sub-channel's start-address or ISDB-T layer.
DTAPI_STAT_CARRIER_LOCK	bool	Carrier lock status
DTAPI_STAT_CNR	int	Carrier-over-noise ratio in units of 0.1 dB
DTAPI_STAT_DAB_ENSEM_INFO	struct	DAB ensemble information from the Fast Information Channel (FIC)
DTAPI_STAT_DVBC2_DSLICEDISC	int	DVB-C2 data slice discontinuity count
DTAPI_STAT_DVBC2_L1HDR_ERR	int	DVB-C2 L1-Preamble header error count
DTAPI_STAT_DVBC2_L1P2_ERR	int	DVB-C2 L1-Part 2 error count
DTAPI_STAT_DVBC2_L1P2DATA	struct	DVB-C2 layer-1 part 2 signalling data
DTAPI_STAT_DVBC2_PLPSIGDATA	struct	DVB-C2 layer-1 PLP signalling data
DTAPI_STAT_DVBT_TPS_INFO	struct	DVB-T transmission parameter signalling information
DTAPI_STAT_DVBT2_L1DATA	struct	DVB-T2 layer-1 signalling data
DTAPI_STAT_DVBT2_L1POST_ERR	int	DVB-T2 L1-Post error count
DTAPI_STAT_DVBT2_L1PRE_ERR	int	DVB-T2 L1-Post error count
DTAPI_STAT_ESN0	int	Energy per symbol to noise power spectral density in units of 0.1 dB
DTAPI_STAT_EBN0	int	Energy per bit to noise power spectral density ratio in units of 0.1 dB
DTAPI_STAT_FEC_LOCK	bool	FEC decoding lock status
DTAPI_STAT_FER_POSTBCH	double	Post-BCH frame error rate. m_IdXtra[0] identifies: DVB-C2/T2 PLP.
DTAPI_STAT_FREQ_SHIFT	double	Input frequency shift (Hz)
DTAPI_STAT_ISDBT_PARSDATA	struct	ISDB-T signalling data
DTAPI_STAT_LDPC_STATS	struct	LDPC error correction counters. m_IdXtra[0] identifies: DVB-C2/T2 PLP.
DTAPI_STAT_LINKMARGIN	int	Difference in dB between C/N of the received signal and the C/N at which the receiver cannot demodulate the signal any more in units of 0.1 dB
DTAPI_STAT_LOCK	bool	Overall lock status
DTAPI_STAT_MA_DATA	struct	DVB-C2/T2 mode adaptation data. m_IdXtra[0] identifies: DVB-C2/T2 PLP.
DTAPI_STAT_MA_STATS	struct	DVB-C2/T2 mode adaptation statistics. m_IdXtra[0] identifies: DVB-C2/T2 PLP.
DTAPI_STAT_MER	int	MER in units of 0.1 dB. m_IdXtra[0] identifies: DVB-C2/T2 PLP.
DTAPI_STAT_OCCUPIEDBW	double	Occupied bandwidth in MHz
DTAPI_STAT_PACKET_LOCK	bool	Packet-level lock status



DTAPI_STAT_PER	double	Packet-error rate. m_IdXtra[0] identifies: DAB sub-channel's start-address or ISDB-T layer.
DTAPI_STAT_PLP_BLOCKS	struct	DVB-T2 number of FEC blocks. m_IdXtra[0] identifies: DVB-T2 PLP.
DTAPI_STAT_RELOCKCNT	int	Receiver relock count
DTAPI_STAT_RFLVL_CHAN	int	RF level for channel bandwidth in units of 0.1 dBmV
DTAPI_STAT_RFLVL_CHAN_QS	int	Quick-scan RF level for channel bandwidth in units of 0.1 dBmV
DTAPI_STAT_RFLVL_NARROW	int	RF level for a narrow bandwidth in units of 0.1 dBmV
DTAPI_STAT_RFLVL_NARROW_QS	int	Quick-scan RF level for a narrow bandwidth in units of 0.1 dBmV
DTAPI_STAT_RFLVL_UNCALIB	int	Uncorrected RF level reported by the tuner in units of 0.1 dBmV
DTAPI_STAT_ROLLOFF	double	Roll-off factor (percentage)
DTAPI_STAT_RS	int	Reed-Solomon error count. Set to -1 if receiver is not locked <b>Note:</b> See remarks section for usage instructions
DTAPI_STAT_RSDEC_STATS	struct	Reed-Solomon decoder statistics. m_IdXtra[0] identifies: DAB sub-channel's start-address, DVB-C2/T2 PLP or ISDB-T layer.
DTAPI_STAT_SAMPRATE_OFFSET	double	Sample rate offset (ppm)
DTAPI_STAT_SNR	int	Signal-to-noise ratio in units of 0.1 dB
DTAPI_STAT_SPECTRUMINV	bool	Spectrum inversion
DTAPI_STAT_T2MI_OVFS	int	DVB-T2 T2-MI overflow count
DTAPI_STAT_TEMP_TUNER	int	Tuner temperature
DTAPI_STAT_VIT_LOCK	bool	Viterbi lock status
DTAPI_STAT_VITDEC_STATS	struct	Viterbi decoder statistics. m_IdXtra[0] identifies: DAB sub-channel's start-address, DVB-C2/T2 PLP or ISDB-T layer.

*m\_IdXtra[4]*

Extra identification attributes. See the statistics for possible values or can be set to -1 if not applicable (default value).

*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_DAB_ENSEM	The value type is pointer to <b>DtDabEnsembleInfo</b>
STAT_VT_DVBC2_L1P2	The value type is pointer to <b>DtDvbC2DemodL1Part2Data</b>

<b>STAT_VT_DVBC2_PLPSIG</b>	The value type is pointer to <b>DtDvbC2DemodL1PlpSigData</b>
<b>STAT_VT_DVBT_TPS_INFO</b>	The value type is pointer to <b>DtDvbTTpsInfo</b>
<b>STAT_VT_DVBT2_L1</b>	The value type is pointer to <b>DtDvbT2DemodL1Data</b>
<b>STAT_VT_INT</b>	The value type is int
<b>STAT_VT_ISDBT_PARS</b>	The value type is pointer to <b>DtIsdbtParamsData</b>
<b>STAT_VT_LDPC_STATS</b>	The value type is pointer to <b>DtDemodLdpcStats</b>
<b>STAT_VT_MA_DATA</b>	The value type is pointer to <b>DtDemodMaLayerData</b>
<b>STAT_VT_MA_STATS</b>	The value type is pointer to <b>DtDemodMaLayerStats</b>
<b>STAT_VT_PLP_BLOCKS</b>	The value type is pointer to <b>DtDemodPlpBlocks</b>
<b>STAT_VT_RS_STATS</b>	The value type is pointer to <b>DtRsDecStats</b>
<b>STAT_VT_VIT_STATS</b>	The value type is pointer to <b>DtVitDecStats</b>
<b>STAT_VT_UNDEFINED</b>	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.

In DVB-S2, the **DTAPI\_STAT\_BER\_PRELDPC** statistic is the bit error rate before the receiver has applied any error correction. For the DTA-2137 it is computed from the MER. This computation has been validated using DekTec's advanced demodulator simulation software (this software has been used amongst others in the DVB working groups for the definition of DVB-T2 and DVB-C2). The correspondence between theoretical and measured values is very good.

For the DTA-2137, the **DTAPI\_STAT\_BER\_ESNO** statistic is computed from the MER under the assumption that the noise distribution is Gaussian (AWGN channel), as under these circumstances  $E_s/N_0$  and MER are identical.

For the DTA-2137, the **DTAPI\_STAT\_BER\_EBNO** statistic is computed from the  $E_s/N_0$  and is valid for constant modulated (CCM) streams only.

## DtStatistic::GetValue

Get the value of the statistic.

```
DTAPI_RESULT DtStatistic::GetValue(int& Value);
DTAPI_RESULT DtStatistic::GetValue(double& Value);
DTAPI_RESULT DtStatistic::GetValue(bool& Value);

DTAPI_RESULT DtStatistic::GetValue(DtDabEnsembleInfo*& pValue);
DTAPI_RESULT DtStatistic::GetValue(DtDvbC2DemodL1Part2Data*& pValue);
DTAPI_RESULT DtStatistic::GetValue(DtDvbC2DemodL1PlpSigData*& pValue);
DTAPI_RESULT DtStatistic::GetValue(DtDvbTTpsInfo*& pValue);
DTAPI_RESULT DtStatistic::GetValue(DtDvbT2DemodL1Data*& pValue);
DTAPI_RESULT DtStatistic::GetValue(DtDemodLdpcStats*& pValue);
DTAPI_RESULT DtStatistic::GetValue(DtDemodMaLayerData*& pValue);
DTAPI_RESULT DtStatistic::GetValue(DtDemodMaLayerStats*& pValue);
DTAPI_RESULT DtStatistic::GetValue(DtDemodPlpBlocks*& pValue);
DTAPI_RESULT DtStatistic::GetValue(DtIsdbtParamsData*& pValue);
DTAPI_RESULT DtStatistic::GetValue(DtRsDecStats*& pValue);
DTAPI_RESULT DtStatistic::GetValue(DtVitDecStats*& pValue);
```

### Function Arguments

*Value*

Receives the value of the statistic. The type of Value must match the type of the statistic. Integers are not automatically converted to doubles or vice versa.

*pValue*

Receives a pointer to the value of the statistic.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Frame has successfully been converted
DTAPI_E_INVALID_TYPE	The value type of the overload does not match the type of the statistic

### Remarks

## Global Functions

### ::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
<b>DTAPI_OK</b>	Scan has completed successfully and the <i>DvcDescArr</i> array was large enough to contain all device descriptions.
<b>DTAPI_E_BUF_TOO_SMALL</b>	The number of device-description entries in <i>DvcDescArr</i> is too small. The number of entries required is returned in <i>NumEntriesResult</i> .

### 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
<b>TYPE_NMB</b>	"DTA-2145"	Device type number
<b>TYPE_AND_LOC</b>	"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
<b>DTAPI_OK</b>	Successfully created a string
<b>DTAPI_E_INVALID_BUF</b>	An invalid buffer pointer is supplied for <i>pDvcDesc</i> or <i>pString</i>

## ::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
<b>TYPE_NMB</b>	"DTA-2111"	Device type number
<b>TYPE_AND_PORT</b>	"DTA-145 port 1"	Device type number and port number if necessary to uniquely identify the hardware function. If both ports of a DTA-145 are configured as output this function will return "DTA-145 port 1", if port 1 is configured as input and port 2 as output this function will return "DTA-145".
<b>TYPE_AND_LOC</b>	"DTA-2144 in slot 5"	Device type number and location
<b>ITF_TYPE</b>	"DVB-ASI"	Physical interface type
<b>ITF_TYPE_SHORT</b>	"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 <i>pHwFunc</i> or <i>pString</i>



## ::DtapiGetDeviceDriverVersion

Get device driver version information.

```
DTAPI_RESULT  ::DtapiGetDeviceDriverVersion(
    [in] int    DeviceCategory,      // Device-driver category
    [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 <i>Dta32</i> or <i>Dta64</i> device driver.
DTAPI_CAT_USB	USB device; Query version of <i>Dtu32</i> or <i>Dtu64</i> device driver.
DTAPI_CAT_NW	Network device; Query version of <i>DtaNw32</i> or <i>DtaNw64</i> device driver
DTAPI_CAT_NWAP	VLAN device; Query version of <i>DtaNwAp32</i> or <i>DtaNwAp64</i> 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
<b>DTAPI_OK</b>	Scan has completed successfully and the <i>pHwFuncs</i> array was large enough to contain all function descriptions.
<b>DTAPI_E_BUF_TOO_SMALL</b>	The number of function-description entries in <i>pHwFuncs</i> is too small. The number of entries required is returned in <i>NumEntriesResult</i> .
<b>DTAPI_E_INVALID_BUF</b>	A NULL pointer was supplied to <i>pHwFuncs</i> while <i>NumEntries</i> 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.

## ::DtapiIoStd2VidStd

Convert *Value* and *SubValue* parameter values (as used in **DtDevice::SetIoConfig** calls) to a **DTAPI\_VIDSTD\_XXX** constant (as used in the Matrix API to identify video standards).

```
DTAPI_RESULT ::DtapiIoStd2VidStd(  
    [in] int& Value,           // I/O configuration value:  
                                // DTAPI_IOCONFIG_SDI/HDSDI/3GSDI  
    [in] int& SubValue,       // I/O configuration subvalue:  
                                // DTAPI_IOCONFIG_1080I50, ...  
    [out] int VidStd,         // DTAPI_VIDSTD_XXX: video standard  
);
```

### Function Arguments

*Value*, *SubValue*

I/O configuration value and sub value to be converted to a **DTAPI\_VIDSTD\_XXX** standard.

*VidStd*

Corresponding video standards. For example, value **DTAPI\_IOCONFIG\_HDSDI** and sub value **DTAPI\_IOCONFIG\_1080I50** are converted to **DTAPI\_VIDSTD\_1080I50**.

### Result

DTAPI_RESULT	Meaning
<b>DTAPI_E_INVALID_VIDSTD</b>	<i>Value</i> and <i>SubValue</i> do not correspond to a video standard
<b>DTAPI_OK</b>	Successfully converted I/O configuration value and sub value to the corresponding video standard

### Remarks

## ::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 282 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 <b>DtOutpChannel::SetModControl</b>

### 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 282 for more details about these parameters.

*T2Pars*

DVB-T2 modulation parameters from which the transport-stream rate of PLPO 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 <b>DtOutpChannel::SetModControl</b>

### 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

## ::DtapiRegisterCallback

Register to one or multiple events and define the callback function.

```
DTAPI_RESULT ::DtapiRegisterCallback (
    [in] pDtEventCallback Callback // Callback for occurred events
    [in] void* pContext             // Opaque pointer passed to callback
    [in] int EventTypes             // Events to subscribe to (OR-able)
    [out] void** pId                // Subscription identifier
);
```

### Function Arguments

*Callback*

Pointer to the user-provided callback function that will be called when an event occurred.

*pContext*

Opaque pointer that is passed to the callback function.

*EventTypes*

Specifies the events to subscribe to. The events may be OR-ed together.

Global events:

Event	Meaning
DT_EVENT_TYPE_ADD	A DekTec device has been inserted and is now available.
DT_EVENT_TYPE_REMOVE	A DekTec device has been removed and is no longer available.

Device events:

Event	Meaning
DT_EVENT_TYPE_POWER	A device power up or power down has been occurred.
DT_EVENT_TYPE_GENLOCK	The genlock state of the device has changed.

Network events:

Event	Meaning
DT_EVENT_IP_CHANGED	An IP-address has been changed.
DT_EVENT_ADMINST_CHANGED	The admin status has changed.

*pId*

Pointer to the event subscription identifier.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Callback function has been attached successfully to the event(s)
DTAPI_E_INVALID_ARG	The value of one of the arguments is invalid, and no specific other error code applies
DTAPI_E_OUT_OF_MEM	Event watcher cannot be allocated

## ::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.

## ::DtapiUnregisterCallback

Unregister to the one or multiple events.

```
DTAPI_RESULT ::DtapiUnregisterCallback(  
    [in] void*  pId          // Subscription identifier to unregister  
);
```

### Function Arguments

*pId*

Pointer to the event subscription identifier given by **DtapiRegisterCallback**.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Callback function has been successfully unregistered.
DTAPI_E_INVALID_ARG	The value of one of the arguments is invalid, and no specific other error code applies
DTAPI_E_NOT_INITIALIZED	No callback functions for events have been registered

## ::DtapiVidStd2IoStd

Convert a **DTAPI\_VIDSTD\_XXX** constant, used in the Matrix API to identify video standards, to *Value* and *SubValue* parameter values used in **DtDevice::SetIoConfig** calls.

```
DTAPI_RESULT ::DtapiVidStd2IoStd(
    [in] int VidStd,           // DTAPI_VIDSTD_XXX: video standard
    [out] int& Value,         // I/O configuration value:
                             // DTAPI_IOCONFIG_SDI/HDSDI/3GSDI
    [out] int& SubValue,      // I/O configuration subvalue:
                             // DTAPI_IOCONFIG_1080I50, ...
);
```

### Function Arguments

*VidStd*

Video standard to be converted: **DTAPI\_VIDSTD\_1080I50**, ...

*Value*, *SubValue*

Converted I/O configuration value and sub value. For example, **DTAPI\_VIDSTD\_1080I50** is converted to value **DTAPI\_IOCONFIG\_HDSDI** and sub value **DTAPI\_IOCONFIG\_1080I50**.

### Result

DTAPI_RESULT	Meaning
<b>DTAPI_E_INVALID_VIDSTD</b>	An invalid value for video standard was specified
<b>DTAPI_OK</b>	Successfully converted the video standard to I/O configuration <i>Value</i> and <i>SubValue</i> values

### Remarks

The return value is only valid for SDI ports. For SPISDI ports there is no corresponding function.

## ::DtapiVidStd2Str

Convert a **DTAPI\_VIDSTD\_XXX** constant to a human-readable string.

```
const char*  ::DtapiVidStd2Str(  
    [in] int  VidStd,           // DTAPI_VIDSTD_XXX: video standard  
);
```

### Function Arguments

*VidStd*

Video standard to be converted. For example, **DTAPI\_VIDSTD\_1080I50**, is converted to "DTAPI\_VIDSTD\_1080I50".

### 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

### 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 <i>Typenumber</i> is found in this system, or the number of devices of this type is less-or-equal than <i>DeviceNo</i>
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::DetectIoStd

Detect the video standard of the signal currently applied to an input port.

```
DTAPI_RESULT DtDevice::DetectIoStd (
    [in] int    Port,           // Physical port number (1...#ports)
    [out] int&   Value          // Detected video standard
    [out] int&   SubValue       // Detected video standard
);
```

### Parameters

*Port*

Physical port number of the port to detect the video standard for.

*Value/SubValue*

Returns the detected video standard. The values can be used via Refer to **SetIoConfig(Port, DTAPI\_IOCONFIG\_IOSTD, Value, SubValue)** to set the input video standard for a specific port. See **SetIoConfig()** for more information.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Call succeeded
DTAPI_E_NOT_SUPPORTED	Detection of video standard is not supported for current device
DTAPI_E_DEV_DRIVER	Unexpected driver error

### Remarks

This function is only supported by cards having the Matrix capability.

## 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 <b>RFFREQ_MAX</b> and <b>RFFREQ_ABSMAX</b> may work, but are not guaranteed to work.
DTAPI_ATTR_RFFREQ_ABSMIN	Absolute minimum output frequency in MHz supported by the hardware. Frequencies between <b>RFFREQ_ABSMIN</b> and <b>RFFREQ_MIN</b> 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 <b>SAMPRHW_MAX</b> and <b>SAMPRHW_ABSMAX</b> , but performance specifications are not guaranteed. If <b>SAMPRHW_MAX</b> equals <b>SAMPRHW_ABSMAX</b> , sample rate conversion is used for sample rates above <b>SAMPRHW_MAX</b> .
DTAPI_ATTR_SAMPRHW_ABSMIN	Absolute minimum sample rate supported by the hardware. The device hardware may work for sample rates between <b>SAMPRHW_ABSMIN</b> and <b>SAMPRHW_MIN</b> , but performance specifications are not guaranteed. If <b>SAMPRHW_MIN</b> equals <b>SAMPRHW_ABSMIN</b> , sample rate conversion is used for sample rates below <b>SAMPRHW_MIN</b> .
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.
<b>DTAPI_ATTR_SAMPRATE_ABSMAX</b>	Absolute maximum sample rate that is supported. The signal must be band-limited, otherwise signal frequencies between <b>SAMP_RHW_MAX/2</b> and <b>SAMPRATE_MAX/2</b> are muted and may even fold back (alias) into the main band.
<b>DTAPI_ATTR_SAMPRATE_ABSMIN</b>	Absolute minimum sample rate that is supported.
<b>DTAPI_ATTR_SAMPRATE_MAX</b>	Maximum sample rate for which performance specifications are guaranteed.
<b>DTAPI_ATTR_SAMPRATE_MIN</b>	Minimum sample rate for which performance specifications are guaranteed.

## Result

<b>DTAPI_RESULT</b>	<b>Meaning</b>
<b>DTAPI_OK</b>	The attribute value has been retrieved successfully
<b>DTAPI_E_NO_SUCH_PORT</b>	Invalid port number for this device
<b>DTAPI_E_NOT_ATTACHED</b>	Device object is not attached to device hardware
<b>DTAPI_E_NOT_SUPPORTED</b>	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::GetGenlockState

Detects the video standard currently applied to the specified reference source input port (set with `SetIoConfig`).

```
DTAPI_RESULT DtDevice::GetGenlockState(
    [out] int& State,           // Current state
    [out] int& RefVidStd       // Reference video standard
);
// OVERLOAD: Get just the state
DTAPI_RESULT DtDevice::GetGenlockState(
    [out] int& State,           // Current state
);
```

### Function Arguments

*State*

Returns the state of the on-board video clock generator.

Value	Meaning
DTAPI_GENL_NO_REF	No reference input signal is detect on the reference source input port or the specified reference video standard is inconsistent with the applied video standard.
DTAPI_GENL_LOCKING	A valid reference input signal is detected on the input and internal PLLs are locking to it.
DTAPI_GENL_LOCKED	Full clock-lock has been achieved.

*RefVidStd*

Returns the video standard set (with `SetIoConfig`) for the reference source. Refer to spreadsheet [CapList.xlsx](#), sheet **IO Capabilities** and group **IOSTD**, for a list of the possible values.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Genlock state has been returned.
DTAPI_E_NOT_SUPPORTED	This function is not supported by the current hardware.
DTAPI_E_DEV_DRIVER	Unexpected driver error.
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 (1...#ports)
    [in] int Group,         // I/O configuration group
    [out] int& Value        // I/O configuration value
);

DTAPI_RESULT DtInpChannel::GetIoConfig(
    [in] int Port,           // Physical port number (1...#ports)
    [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 (1...#ports)
    [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 (1...#ports)
    [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

*Port*

Physical port number.

*Group, Value, SubValue, ParXtra0, ParXtra1*

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

### 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 <i>IoConfig</i> <b>DTAPI_IOCONFIG_S2LOOPMODE</b> : The configuration of the corresponding ASI port is not in loop-through mode

## Remarks



## DtDevice::GetNwSpeed

Get the speed of the network link.

```
DTAPI_RESULT DtDevice::GetNwSpeed(
    [in] int Port,           // Physical port number (1...#ports)
    [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-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<sup>2</sup>. 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.

<sup>2</sup> 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
<b>DTAPI_OK</b>	Scan has completed successfully and the <i>pHwFuncs</i> array was large enough to contain all function descriptions.
<b>DTAPI_E_BUF_TOO_SMALL</b>	The number of function-description entries in <i>pHwFuncs</i> is too small. The number of entries required is returned in <i>NumEntriesResult</i> .
<b>DTAPI_E_INVALID_BUF</b>	A NULL pointer was supplied to <i>pHwFuncs</i> while <i>NumEntries</i> 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 that supports certain configurable “I/O capabilities”. An overview of the I/O capabilities and the DekTec devices supporting them can be found in spreadsheet [CapList.xlsx](#), sheet [IO Capabilities](#), which is part of the DTAPI user documentation.

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

Two overloads are defined, one to configure a single port and another to configure a number of ports in one operation. It is mandatory to use the latter overload if there are dependencies between the ports, so that the ports must be configured at the same time.

```
DTAPI_RESULT DtDevice::SetIoConfig(  
    [in,out] DtIoConfig* pIoConfigs,    // I/O configuration parameters  
    [in] int Count,                    // Number of elements in pIoConfigs  
);  
  
DTAPI_RESULT DtDevice::SetIoConfig(  
    [in] int Port,                    // Physical port number (1...#ports)  
    [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

*pIoConfigs, Count*

Array of I/O configuration parameters per port. *Count* is the number of array elements.

*Port*

Physical port number.

*Group, Value, SubValue, ParXtra0, ParXtra1*

Specifies the I/O configuration parameters. Refer to spreadsheet [CapList.xlsx](#), sheet [IO Capabilities](#) for details. In this sheet, *Cap* corresponds to *Value*, while *SubCap* corresponds to *SubValue*. Some additional explanation on a subset of the I/O configuration settings is provided below.

### Group BOOLIO

I/O capabilities in group **BOOLIO** have a Boolean *Value* argument.

Value	Meaning
DTAPI_IOCONFIG_TRUE	Enable the I/O capability
DTAPI_IOCONFIG_FALSE	Disable the I/O capability

### Boolean I/O Capability FAILSAFE

If **DTAPI\_IOCONFIG\_FAILSAFE** for an output port is set to **TRUE**, that port is put in “failsafe” mode, protected by a “watchdog”. This means that the user application has to call **DtOutpChannel::SetFailsafeAlive** repeatedly within the watchdog timeout period. If the user application is too late (e.g. due to a crash), the watchdog times out and the failsafe relay is released so that the output is connected directly to the input. Failsafe mode is implemented

on the DTA-145 and the DTA-2145. On these cards, if the watchdog triggers, the input signal on port #1 will be physically connected to port #2 through a relay.  
I/O configuration parameter *ParXtra0* specifies the watchdog time out in milliseconds.

### Boolean I/O Capability GENLOCKED

If set, the output 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 synchronization.

### Boolean I/O Capability GENREF

If set, the input port will act as an SDI genlock reference input. I/O configuration parameter *ParXtra0* specifies the expected video standard, e.g. **DTAPI\_IOCONFIG\_1080I50**.

### Boolean I/O Capability SWS2APSK

DTA-2137 only: Enable DVB-S2 reception in 16-APSK or 32-APSK mode.

This I/O capability can only be enabled on port 1; port 2 must be disabled. The board can only receive a single channel when this I/O capability is enabled. Without **SWS2APSK**, two channels are available.

### I/O Capability IODIR, OUTPUT for the DTA-2137

The DTA-2137 supports two specific I/O capabilities for looping through the received DVB-S2 stream on one of the ASI ports:

**DTAPI\_IOCONFIG\_LOOPS2L3** encapsulates DVB-S2 baseband frames (BBFRAMEs) in L3 frames and outputs them on the ASI port.

**DTAPI\_IOCONFIG\_LOOPS2TS** selects a sub stream from a DVB-S2 Multiple Input Stream, based on ISI (specified in *ParXtra1*). The selected stream can be an MPEG2 transport stream or any other input stream type. It is output on the ASI port.

## 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_ARG	The value of one of the arguments is invalid, and no specific other error code applies
DTAPI_E_INVALID_ISI	For <b>DTAPI_IOCONFIG_LOOPS2TS</b> only: 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 (1...#ports)
    [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 $\pm 5$ ppm at room temperature and a stability of $\pm 15$ 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***

### **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
<b>DTAPI_OK</b>	Channel object has been attached successfully to the port
<b>DTAPI_OK_FAILSAFE</b>	For devices with a failsafe relay only. Attachment in failsafe mode was successfully completed. The application shall call <b>SetFailsafeAlive</b> 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.
<b>DTAPI_E_ATTACHED</b>	Channel object is already attached
<b>DTAPI_E_DEV_DRIVER</b>	Unclassified failure in device driver
<b>DTAPI_E_DEVICE</b>	Pointer <i>pDtDvc</i> is not valid or the device object is not attached to a hardware device

<b>DTAPI_E_EXCL_MANDATORY</b>	Shared access ( <i>Exclusive=false</i> ) was requested, but this is not supported by the input channel. Only demodulators support shared access.
<b>DTAPI_E_IN_USE</b>	Another channel object is already attached to this port
<b>DTAPI_E_NO_DT_INPUT</b>	<i>Port</i> is not an input
<b>DTAPI_E_NO_SUCH_PORT</b>	Invalid port number for this device
<b>DTAPI_E_OUT_OF_MEM</b>	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
<b>DTAPI_OK</b>	The blind scan succeeded successfully
<b>DTAPI_E_NOT_ATTACHED</b>	Channel object is not attached to a hardware function
<b>DTAPI_E_NOT_SUPPORTED</b>	The blind scan is not supported by the attached hardware
<b>DTAPI_E_INVALID_BUF</b>	The number of transponder-description entries in <i>pResults</i> is too small. The number of entries required is returned in <i>NumEntriesResult</i> .
<b>DTAPI_E_INVALID_FREQ</b>	The range of frequencies specified with <i>FreqHzSteps</i> , <i>StartFreqHz</i> and <i>EndFreqHz</i> 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 <code>GetFlags</code>
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 or SPI input channels: Detect whether the input signal is ASI, SDI, SPI or SPISDI.

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

### Function Arguments

#### *Value*

Indicates whether a DVB-ASI, an SDI, a SPI or a SPISDI 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
DTAPI_IOCONFIG_SPI	A valid SPI signal is being received
DTAPI_IOCONFIG_SPISDI	A valid SPISDI signal is being received

#### *SubValue*

If an SDI signal was received, *SubValue* receives the specific SDI standard, for example **DTAPI\_IOCONFIG\_525I59\_94**.

If an SPISDI signal was received, *SubValue* receives the specific SPISDI standard, for example **DTAPI\_IOCONFIG\_SPI525I59\_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 <i>pPoints</i> 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
181	Overview
182	ATSC
183	DVB-S
184	DVB-S.2
186	DVB-T
188	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	Required Capability
<code>DTAPI_MOD_DVBS_QPSK</code>	DVB-S, QPSK	<code>DTAPI_CAP_RX_DVBS</code>
<code>DTAPI_MOD_DVBS2_8PSK</code>	DVB-S.2, 8-PSK	<code>DTAPI_CAP_RX_DVBS</code>
<code>DTAPI_MOD_DVBS2_16APSK</code>	DVB-S.2, 16-APSK	<code>DTAPI_CAP_RX_DVBS</code> + <code>DTAPI_CAP_S2APSK</code>
<code>DTAPI_MOD_DVBS2_32APSK</code>	DVB-S.2, 32-APSK	<code>DTAPI_CAP_RX_DVBS</code> + <code>DTAPI_CAP_S2APSK</code>
<code>DTAPI_MOD_DVBS2_QPSK</code>	DVB-S.2, QPSK	<code>DTAPI_CAP_RX_DVBS</code>

#### VHF / UHF

ModType	Meaning	Required Capability
<code>DTAPI_MOD_ATSC</code>	ATSC VSB	<code>DTAPI_CAP_RX_ATSC</code>
<code>DTAPI_MOD_DVBT</code>	DVB-T	<code>DTAPI_CAP_RX_DVBT</code>
<code>DTAPI_MOD_QAM16</code>	16-QAM	<code>DTAPI_CAP_RX_QAM_A</code> for QAM-A <code>DTAPI_CAP_RX_QAM_B</code> for QAM-B <code>DTAPI_CAP_RX_QAM_C</code> for QAM-C
<code>DTAPI_MOD_QAM32</code>	32-QAM	
<code>DTAPI_MOD_QAM64</code>	64-QAM	
<code>DTAPI_MOD_QAM128</code>	128-QAM	
<code>DTAPI_MOD_QAM256</code>	256-QAM	

## Modulation Mode: ATSC

### *ModType*

ModType	Meaning	Required Capability
DTAPI_MOD_ATSC	ATSC	DTAPI_CAP_RX_ATSC

### *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)



<b>DTAPI_MOD_S2_FRM_UNK</b>	Frame size is unknown
<b>DTAPI_MOD_S2_FRM_MSK</b>	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	Required Capability
DTAPI_MOD_QAM16	16-QAM	DTAPI_CAP_RX_QAM_A (QAM-A) DTAPI_CAP_RX_QAM_B (QAM-B) DTAPI_CAP_RX_QAM_C (QAM-C)
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	Required Capability
DTAPI_MOD_J83_A	J.83 annex A (DVB-C)	DTAPI_CAP_RX_QAM_A
DTAPI_MOD_J83_B	J.83 annex B ("American QAM")	DTAPI_CAP_RX_QAM_B
DTAPI_MOD_J83_C	J.83 annex C ("Japanese QAM")	DTAPI_CAP_RX_QAM_C
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 $\mu$ s / 66 $\mu$ s
DTAPI_MOD_QAMB_I64_J2	0011	64	2	47 $\mu$ s / 33 $\mu$ s
DTAPI_MOD_QAMB_I32_J4	0101	32	4	24 $\mu$ s / 16 $\mu$ s
DTAPI_MOD_QAMB_I16_J8	0111	16	8	12 $\mu$ s / 8.2 $\mu$ s
DTAPI_MOD_QAMB_I8_J16	1001	8	16	5.9 $\mu$ s / 4.1 $\mu$ s
DTAPI_MOD_QAMB_I128_J1	0000	128	1	95 $\mu$ s / 66 $\mu$ s
DTAPI_MOD_QAMB_I128_J2	0010	128	2	190 $\mu$ s / 132 $\mu$ s
DTAPI_MOD_QAMB_I128_J3	0100	128	3	285 $\mu$ s / 198 $\mu$ s
DTAPI_MOD_QAMB_I128_J4	0110	128	4	379 $\mu$ s / 264 $\mu$ s
DTAPI_MOD_QAMB_I128_J5	1000	128	5	474 $\mu$ s / 330 $\mu$ s
DTAPI_MOD_QAMB_I128_J6	1010	128	6	569 $\mu$ s / 396 $\mu$ s
DTAPI_MOD_QAMB_I128_J7	1100	128	7	664 $\mu$ s / 462 $\mu$ s
DTAPI_MOD_QAMB_I128_J8	1110	128	8	759 $\mu$ s / 528 $\mu$ s

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 **DtInpChannel1** 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 <b>DTAPI_RXMODE_RAW</b> .
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 <code>DtDevice::GetIoConfig</code>

### 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::GetPars

Get parameter settings from the demodulator. This function accepts and returns an array of **DtPar** structures so that multiple parameters can be retrieved in one call.

```
DTAPI_RESULT DtInpChannel::GetPars(  
    [in] int NumPars,           // Number of parameters  
    [in,out] DtPar* pPars      // Array with parameters  
);
```

### Function Arguments

*NumPars*

Specifies the size, in number of **DtPar** entries, of the caller-supplied *pPars* array.

*pPar*

Pointer to a caller-supplied array of **DtPar** structures specifying the requested parameters. The current values of the requested parameters are returned through this same array.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Parameter values have been read successfully
DTAPI_E_INVALID_BUF	Invalid buffer pointer is passed
DTAPI_E_INVALID_TYPE	Parameter type is incorrect
DTAPI_E_NOT_ATTACHED	demodulator is not attached

### 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
    [in out] DtStatistic* pStatistics // Array with statistics
);
```

### Function Arguments

*Count*

The number of requested statistics.

*pStatistic*

An array specifying the statistics to be retrieved in one call. After the call it holds the values of the requested statistics.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	The statistics have been read successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_MODE	The statistic is not compatible with <b>ErrorStatsMode</b>
DTAPI_E_INVALID_FOR_ACM	The requested statistic is not available for Adaptive Coding Modulation (ACM) in DVB-S2
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_LOCKED	The requested statistic cannot be retrieved because the receiver is not locked to the input signal
DTAPI_E_NOT_SUPPORTED	The requested statistic is not supported by the hardware or the requested return type is not supported

### 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
<b>DTAPI_ASIINV_NORMAL</b>	Polarity of DVB-ASI input signal is normal (not inverted)
<b>DTAPI_ASIINV_INVERT</b>	Polarity of DVB-ASI signal is inverted
<b>DTAPI_NOT_SUPPORTED</b>	Device does not support this parameter

### Result

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

### Remarks

## DtInpChannel::GetStreamSelection

Get the selection parameters for the currently selected DAB-stream, PLP, Layer or T2-MI stream.

```
DTAPI_RESULT DtInpChannel::GetStreamSelection(  
    [out] DtDabEtiStreamSelPars& StreamSel // DAB-ETI selection parameters  
);  
  
DTAPI_RESULT DtInpChannel::GetStreamSelection(  
    [out] DtDabStreamSelPars& StreamSel // DAB stream selection parameters  
);  
  
DTAPI_RESULT DtInpChannel::GetStreamSelection(  
    [out] DtDvbC2StreamSelPars& StreamSel // PLP selection parameters  
);  
  
DTAPI_RESULT DtInpChannel::GetStreamSelection(  
    [out] DtDvbT2StreamSelPars& StreamSel // PLP selection parameters  
);  
  
DTAPI_RESULT DtInpChannel::GetStreamSelection(  
    [out] DtT2MiStreamSelPars& StreamSel // T2-MI parameters  
);  
  
DTAPI_RESULT DtInpChannel::GetStreamSelection(  
    [out] DtIsdbtStreamSelPars& StreamSel // ISDB-T 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 selection parameters
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The device does not support DAB, DVB-C2, DVB-T2, T2-MI or ISDB-T

### Remarks

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

## DtInpChannel::GetSupportedPars

Get the parameters supported by the demodulator.

```
DTAPI_RESULT DtInpChannel::GetSupportedPars (
[in,out] int&  NumPars,      // Number of parameters
[out] DtPar*  pPars        // Array with supported parameters
);
```

### Function Arguments

*NumPars*

As an input argument it specifies the size, in number of **DtPar** entries, of the caller-supplied *pPars* array. As an output argument it receives the number of supported parameters.

*pPar*

Pointer to a caller-supplied array of **DtPar** to receive the requested parameters.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Parameter information has been read successfully
DTAPI_E_BUF_TOO_SMALL	The number of <b>DtPar</b> entries in <i>pPars</i> is too small
DTAPI_E_NOT_ATTACHED	Advanced demodulator object is not attached

## DtInpChannel::GetSupportedStatistics

Get the supported statistics from demodulator. This function gets an array of **DtStatistic** structures.

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

### Function Arguments

*Count*

As input it specifies the size, in number of **DtStatistic** entries, of the caller-supplied *pStatistics* array. As output it receives the number of supported statistics and described in *pStatistics*.

*pStatistic*

Pointer to a caller-supplied array of **DtStatistic** entries identifying the supported statistics.

### 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_BUFFER_TOO_SMALL	The number of statistic entries in <i>pStatistics</i> is too small.

### Remarks

## 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.

Internal use only: this function is exported for DekTec application usage.

## 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.

Internal use only: this function is exported for DekTec application usage.

## 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

Internal use only: this function is exported for DekTec application usage.

## 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.

Internal use only: this function is exported for DekTec application usage.

## 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.  
Internal use only: this function is exported for DekTec application usage.

## 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 <b>SetRxMode</b> page 242)
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 ( <b>SetDemodControl</b> , <b>SetTunerFrequency</b> or <b>Tune</b> was called)
DTAPI_EV_TUNE_FREQ_HAS_CHANGED	Tuning frequency has changed ( <b>SetTunerFrequency</b> or <b>Tune</b> 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: <ul style="list-style-type: none"> <li>• Data transfers are halted instantaneously</li> <li>• All data pending in the Receive FIFO is discarded</li> <li>• Receive-control state is reset to <b>DTAPI_RXCTRL_IDLE</b></li> <li>• Receive-FIFO overflow flag is cleared</li> </ul>
DTAPI_FULL_RESET	Full input-channel reset: <ul style="list-style-type: none"> <li>• All actions for <b>DTAPI_FIFO_RESET</b>, plus:</li> <li>• Synchronisation-error flag (<b>DTAPI_RX_SYNC_ERR</b>) is cleared</li> <li>• State machines in the device hardware are reset</li> </ul>

### 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 <i>ResetMode</i> 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 <sup>3</sup>
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.

<sup>3</sup> The exact frequency is  $27 * 13 / 17 = 20.647059\text{Mhz}$

## 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	<p>DTAPI_MOD_SYMRATE_AUTO</p> <p>Automatic detection of symbol rate is supported by the DTA-2136, DTA-2138, DTA-2139 and the DTU-234. The DTU-236 supports automatic detection of the symbol rate for QAM-B only.</p> <p>You can directly set the symbol rate the tuner should lock to for the DTA-2136, DTA-2139 and DTU-236. For QAM-A and QAM-C on the DTU-236 this is the only possibility since automatic detection isn't supported.</p>

## 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

For DTA-2136 and DTA-2139 J.83 annex A (DVB-C) QAM-128 is not supported

For DTA-2131 a low C/N and MER could be measured when using a DVB-T2 configuration including Pilot Pattern 8 (PP8).

## 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 <i>ModType</i> and <i>Mode</i> 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 <code>DtDevice::SetIoConfig</code>

### 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_BIND	Error binding port to IP address
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_INVALID_DIFFSERV	The m_Diffserv parameter is invalid
DTAPI_E_INVALID_FEC_MODE	The m_Fecmode parameter is invalid
DTAPI_E_INVALID_FLAGS	The m_Flags parameter is invalid
DTAPI_E_INVALID_MODE	The m_Mode parameter is invalid
DTAPI_E_INVALID_IP_ADDR	The m_Ip or m_Ip2 parameter is invalid
DTAPI_E_INVALID_PROTOCOL	The m_Protocol parameter is invalid
DTAPI_E_INVALID_SRCIP_ADDR	The m_SrcFltIp or m_SrcFltIp2 parameter is invalid
DTAPI_E_IPV6_NOT_SUPPORTED	IPv6 is not supported on this operating system
DTAPI_E_MULTICASTOIN	Error joining multicast address
DTAPI_E_NO_ADAPTER_IP_ADDR	The network IP address could not be retrieved. Check the network driver IP protocol settings
DTAPI_E_NO_LINK	The IP parameters cannot be applied because the link is down. Check network cable and speed settings
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NW_DRIVER	The IP address could not be retrieved from the network driver
DTAPI_E_NWAP_DRIVER	An error occurred using the Advanced Protocol Driver.
DTAPI_E_VLAN_NOT_FOUND	The VLAN with the given ID is not found

### 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::SetPars

Set parameter settings for the demodulator. This function accepts an array of **DtPar** structures so that multiple parameters can be set in one call.

```
DTAPI_RESULT DtInpChannel::SetPars (
    [in] int    Count,           // Number of parameters
    [in] DtPar* pPars           // Array with parameters
);
```

### Function Arguments

*NumPars*

Specifies the size, in number of **DtPar** entries, of the caller-supplied *pPars* array.

*pPar*

Pointer to a caller-supplied array of **DtPar** structures.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Parameter values have been set successfully
DTAPI_E_INVALID_BUF	Invalid buffer pointer is passed
DTAPI_E_INVALID_TYPE	Parameter type is incorrect
DTAPI_E_NOT_ATTACHED	Demodulator is not attached
DTAPI_E_NOT_IDLE	Demodulator is already started

### Remarks

Demodulation should not yet been started when this function is called.

## 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 <b>DTAPI_RXCTRL_RCV</b> 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 the receive mode for the input channel. It determines the conversions that will be applied to the input signal. Receive mode has to be set before setting receive control to `DTAPI_RXCTRL_RCV`.

**Note:** For ASI/SDI input channels, first configure the input port for ASI or SDI with `SetIoConfig`, group `IOSTD`. If an ASI-specific receive mode is applied to a channel that is configured for SDI (or vice versa), an error will be returned.

```
DTAPI_RESULT DtInpChannel::SetRxMode(
    [in] int RxMode           // Receive mode
);
```

### Function Arguments

*RxMode*

Receive mode according to the table below.

Value	Meaning
<code>DTAPI_RXMODE_ST188</code>	<i>Transport Stream - 188-byte mode</i> 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.
<code>DTAPI_RXMODE_ST204</code>	<i>Transport Stream - 204-byte mode</i> 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.
<code>DTAPI_RXMODE_STMP2</code>	<i>Transport Stream - MPEG-2 mode</i> Store 188- or 204-byte packets in the receive FIFO without modification. Input data without 188- or 204-byte packet structure is dropped.
<code>DTAPI_RXMODE_STRAW</code>	<i>Transport Stream - Raw mode</i> 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 <code>DTAPI_POLARITY_AUTO</code> ! Please refer to the Remarks section.
<code>DTAPI_RXMODE_STTRP</code>	<i>Transport Stream - Transparent mode</i> 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.
<code>DTAPI_RXMODE_STL3</code>	<i>L.3 Baseband frame mode</i> (DTA-2137/DTA-2137C only). No notion of transport stream packets. The entire DVB-S2 baseband frame is passed with the addition of an L.3 Header. Dummy frames are skipped. See DTAPI Manual – Overview and Data Formats.pdf for more details.

<b>DTAPI_RXMODE_STL3FULL</b>	<i>Full L.3 Baseband frame mode</i> Similar to <b>DTAPI_RXMODE_STL3</b> . Dummy frames are encoded with ModCod is '0'.
<b>DTAPI_RXMODE_RAWASI</b>	<i>Raw ASI symbols</i> (Ports needs CAP_RAWASI) The complete incoming data stream can be read as packed 10-bit symbols.

The following modes are valid for SDI capable channels only.

Value	Meaning
<b>DTAPI_RXMODE_SDI_FULL</b>	<i>Full frame mode</i> Store all SDI data (i.e. complete frames).
<b>DTAPI_RXMODE_SDI_ACTVID</b>	<i>Active video mode</i> Store only the active video part of each SDI frame. This mode should only be used in combination with Huffman compression (i.e. with <b>DTAPI_RXMODE_SDI_HUFFMAN</b> flag)!

The following mode is valid for TS-over-IP reception only.

Value	Meaning
<b>DTAPI_RXMODE_IPRAW</b>	<i>Raw IP mode</i> Store unprocessed IP packets in the buffer. If error correction is requested, store FEC streams too. Each IP packet returned by a call the <b>DtInpChannel::Read</b> will be preceded by a <b>DtRawIpHeader</b> structure.

The receive mode can be optionally combined (OR-ed) with the following flag:

Value	Meaning
<b>DTAPI_RXMODE_TIMESTAMP32</b>	<i>Time-stamped mode (32-bit timestamps)</i> 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 ( <b>DTAPI_RXMODE_STRAW</b> ) or any of the SDI modes.
<b>DTAPI_RXMODE_TIMESTAMP64</b>	<i>Time-stamped mode (64-bit timestamps)</i> (hardware shall support <b>DTAPI_CAP_TIMESTAMP64</b> ). 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 ( <b>DTAPI_RXMODE_STRAW</b> ) or any of the SDI modes.
<b>DTAPI_RXMODE_SDI</b>	<i>SDI mode</i> Operate in SDI mode; Otherwise ASI mode will be used. This flag is already OR-ed into <b>DTAPI_RXMODE_SDI_FULL</b> and <b>DTAPI_RXMODE_SDI_ACTVID</b> .
<b>DTAPI_RXMODE_SDI_10B</b>	<i>10-bit SDI samples</i> Provide 10-bit SDI samples. If both this flag and the 16B flag are omitted, 8-bit samples is assumed.

<b>DTAPI_RXMODE_SDI_16B</b>	<i>16-bit SDI samples</i> Provide 10-bit SDI samples packed in 16-bits. Only the 10 least significant bits of each 16-bit sample will be used. If both this flag and the 10B flag are omitted, 8-bit samples is assumed. 16B mode is only supported by cards having the CAP_MATRIX capability.
<b>DTAPI_RXMODE_SDI_HUFFMAN</b>	<i>Huffman compression</i> Compress the SDI using Huffman compression.

## Result

<b>DTAPI_RESULT</b>	<b>Meaning</b>
<b>DTAPI_OK</b>	Receive mode has been changed successfully
<b>DTAPI_E_DEV_DRIVER</b>	Unclassified failure in device driver
<b>DTAPI_E_INVALID_MODE</b>	The specified receive mode is invalid or incompatible with the input channel
<b>DTAPI_E_NOT_ATTACHED</b>	Channel object is not attached to a hardware function
<b>DTAPI_E_NOT_SUPPORTED</b>	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 transport stream: 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 DAB-stream, PLP in a DVB-C2 or DVB-T2 stream, T2-MI stream or ISDB-T layer

```
DTAPI_RESULT DtInpChannel::SetStreamSelection(
    [in] DtDabEtiStreamSelPars& StreamSel // DAB-ETI selection parameters
);
DTAPI_RESULT DtInpChannel::SetStreamSelection(
    [in] DtDabStreamSelPars& StreamSel // DAB-stream selection parameters
);
DTAPI_RESULT DtInpChannel::SetStreamSelection(
    [in] DtDvbC2StreamSelPars& StreamSel // PLP selection parameters
);
DTAPI_RESULT DtInpChannel::SetStreamSelection(
    [in] DtDvbT2StreamSelPars& StreamSel // PLP selection parameters
);
DTAPI_RESULT DtInpChannel::SetStreamSelection(
    [in] DtDvbT2MiStreamSelPars& StreamSel // T2-MI selection parameters
);
DTAPI_RESULT DtInpChannel::SetStreamSelection(
    [in] DtIsdbtStreamSelPars& StreamSel // ISDB-T 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 selection parameters
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The device does not support DAB, DVB-C2, DVB-T2, ISDB-T or T2-MI
DTAPI_E_PLP_ID	Invalid physical layer pipe identifier
DTAPI_E_LAYER_ID	Invalid ISDB-T layer identifier

### Remarks

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

If the specified PLP/Layer is not available another PLP/Layer 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, ParXtra1, 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

<b>DTAPI_E_INVALID_RATE</b>	Invalid value for convolutional rate or FEC code rate
<b>DTAPI_E_INVALID_SYMRATE</b>	Invalid value for symbol rate
<b>DTAPI_E_INVALID_T2PROFILE</b>	Invalid value for DVB-T2 profile
<b>DTAPI_E_INVALID_TRANSMODE</b>	Invalid value for transmission-mode field
<b>DTAPI_E_INVALID_USEFRAMENO</b>	Invalid value for use-frame-numbering field
<b>DTAPI_E_NOT_ATTACHED</b>	Channel object is not attached to a hardware function
<b>DTAPI_E_NOT_SUPPORTED</b>	The channel does not include a tuner



## ***DtOutpChannel***

### **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 <b>SetFailsafeAlive</b> 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 <i>pDtDvc</i> 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	<i>Port</i> 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 <b>GetFlags</b>
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  $8 \cdot 204 / 188$  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 <b>DTAPI_TXMODE_RAW</b> .
DTAPI_TX_TARGET_ERR	The target adapter signals a fault (DTA-102 only)
DTAPI_RX_LINK_ERR	Communication link with the device is broken (DTE-31XX only)
DTAPI_RX_DATA_ERR	Data is lost during transfer to the device (DTE-31XX 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

## DtOutputChannel::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 DtOutputChannel::GetIoConfig(  
    [in] int Group,           // I/O configuration group  
    [out] int& Value          // I/O configuration value  
);  
  
DTAPI_RESULT DtOutputChannel::GetIoConfig(  
    [in] int Group,           // I/O configuration group  
    [out] int& Value,         // I/O configuration value  
    [out] int& SubValue       // I/O configuration subvalue  
);  
  
DTAPI_RESULT DtOutputChannel::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 DtOutputChannel::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 <code>DtDevice::GetIoConfig</code>

## 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 <b>SPIDVBMODE</b> )
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
<b>DTAPI_OK</b>	transport-stream rate has been read successfully
<b>DTAPI_E_DEV_DRIVER</b>	Unclassified failure in device driver
<b>DTAPI_E_INVALID_TSRATESEL</b>	The current TS-rate selection I/O configuration does not allow retrieval of the transport-stream rate
<b>DTAPI_E_NOT_ATTACHED</b>	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: <ul style="list-style-type: none"><li>• Data transfers and packet transmission are halted instantaneously.</li><li>• All data pending in the transmit FIFO is discarded.</li><li>• Transmit-control state is reset to <b>DTAPI_TXCTRL_IDLE</b>.</li><li>• Transmit-FIFO underflow flag is cleared.</li></ul>
DTAPI_FULL_RESET	Full reset: <ul style="list-style-type: none"><li>• All actions for <b>DTAPI_FIFO_RESET</b>, plus:</li><li>• transport-stream rate is reset to zero (except for ISDB-T and OFDM modulation on DTA-110T)</li></ul>

### 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 <i>ResetMode</i> 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 <b>CmPars</b> 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 <b>DTAPI_TXCTRL_HOLD</b> or <b>DTAPI_TXCTRL_SEND</b>
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 <code>DtDevice::SetIoConfig</code>

## 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. Check if you can reach the destination IP address using the ping command on the console.
DTAPI_E_IN_USE	The parameters cannot be changed because the channel is busy. Transmit control shall be switched back to <b>IDLE</b> first
DTAPI_E_INVALID_ARG	The value of one of the TS-over-IP parameters is invalid
DTAPI_E_INVALID_FEC_MATRIX	The m_FecNumCols / m_FecNumRows are invalid
DTAPI_E_INVALID_FLAGS	The m_Flags parameter is invalid
DTAPI_E_INVALID_MODE	The m_Mode parameter is invalid
DTAPI_E_INVALID_PROTOCOL	The m_Protocol parameter is invalid or not valid for the current m_FecMode
DTAPI_E_IPV6_NOT_SUPPORTED	IPv6 is not supported on this operating system
DTAPI_E_NO_ADAPTER_IP_ADDR	The network IP address could not be retrieved. Check the network driver IP protocol settings
DTAPI_E_NO_LINK	The IP parameters cannot be applied because the link is down. Check network cable and speed settings
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
DTAPI_E_NW_DRIVER	The IP address could not be retrieved from the network driver. Check network driver / network connection.
DTAPI_E_NWAP_DRIVER	An error occurred using the Advanced Protocol Driver.
DTAPI_E_VLAN_NOT_FOUND	The VLAN with the given ID is not found

## 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** in document: *DTAPI Reference – DVB-C2+T2 Multi-PLP Extensions*.

#### *DvbT2Pars*

DVB-T2 modulation parameters; see description of `class DtDvbT2Pars` in document: *DTAPI Reference – DVB-C2+T2 Multi-PLP Extensions*.

#### *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
284	Overview
285	ADTB-T
287	ATSC
91	CMMB
288	DAB
289	DTMB
291	DVB-S
292	DVB-S.2

Page	Modulation Type
294	DVB-S.2 L.3 base-band frames
294	DVB-T / DVB-H
297	IQ-DIRECT
100, 298	ISDB-S
102, 299	ISDB-T
301	QAM
302	DVB-T2 T2-MI

The DVB-C2 and DVB-T2 parameters are described in a separate document: *DTAPI Reference – DVB-C2+T2 Multi-PLP Extensions*.

## Modulation Types

*ModType*

Modulation type:

### L-Band

ModType	Meaning	Required Capability
DTAPI_MOD_DVBS_QPSK	DVB-S, QPSK	DTAPI_CAP_TX_DVBS
DTAPI_MOD_DVBS2_16APSK	DVB-S.2, 16-APSK	DTAPI_CAP_TX_S2APSK
DTAPI_MOD_DVBS2_32APSK	DVB-S.2, 32-APSK	DTAPI_CAP_TX_S2APSK
DTAPI_MOD_DVBS2_8PSK	DVB-S.2, 8-PSK	DTAPI_CAP_TX_DVBS2
DTAPI_MOD_DVBS2_L3	DVB-S.2 L.3 baseband frame modulation	DTAPI_CAP_TX_DVBS2
DTAPI_MOD_DVBS2_QPSK	DVB-S.2, QPSK	DTAPI_CAP_TX_DVBS2
DTAPI_MOD_ISDBS	ISDB-S	DTAPI_CAP_TX_ISDBS

### VHF\* / UHF

ModType	Meaning	Required Capability
DTAPI_MOD_ADTBT	ADTB-T	DTAPI_CAP_TX_DTMF
DTAPI_MOD_ATSC	ATSC VSB	DTAPI_CAP_TX_ATSC
DTAPI_MOD_DAB	DAB+/DMB	DTAPI_CAP_TX_DAB
DTAPI_MOD_DMBTH	DMB-T/H	DTAPI_CAP_TX_DTMF
DTAPI_MOD_DVBT	DVB-T / DVB-H	DTAPI_CAP_TX_DVBT
DTAPI_MOD_IQDIRECT	Direct I/Q sample transmission	DTAPI_CAP_TX_IQ
DTAPI_MOD_QAM16	16-QAM	DTAPI_CAP_TX_QAM_A or DTAPI_CAP_TX_QAM_B or DTAPI_CAP_TX_QAM_C
DTAPI_MOD_QAM32	32-QAM	
DTAPI_MOD_QAM64	64-QAM	
DTAPI_MOD_QAM128	128-QAM	
DTAPI_MOD_QAM256	256-QAM	
DTAPI_MOD_T2MI	T2-MI modulation	DTAPI_CAP_TX_DVBT2

## 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 55.6 $\mu$ s)
DTAPI_MOD_DTMB_PN595	PN595: Frame header 2 (595 symbols 78.7 $\mu$ s)
DTAPI_MOD_DTMB_PN945	PN945: Frame header 3 (945 symbols 125 $\mu$ s)
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	Required Capability
DTAPI_MOD_ATSC	ATSC	DTAPI_CAP_TX_ATSC

### *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 : DAB**

The DAB modulator doesn't work with transport streams but instead expects ETI(NI, G.703) streams with a constant bitrate of 2.048Mbps.

*ParXtra0, ParXtra1, ParXtra2*

Not used in DAB 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 55.6 $\mu$ s)
DTAPI_MOD_DTMB_PN945	PN945: Frame header 3 (945 symbols 125 $\mu$ s)
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

### Roll-off

Value	Meaning
DTAPI_MOD_ROLLOFF_AUTO	Default roll-off
DTAPI_MOD_ROLLOFF_NONE	No roll-off
DTAPI_MOD_ROLLOFF_20	20% roll-off
DTAPI_MOD_ROLLOFF_25	25% roll-off
DTAPI_MOD_ROLLOFF_35	35% roll-off
DTAPI_MOD_ROLLOFF_MSK	AND-mask for this field

### *ParXtra2*

Physical layer scrambling initialization sequence “n”, aka “Gold code”.

### **Modulation Mode : DVB-S.2 L.3 Baseband Frame**

When DVB-S.2 L.3 baseband frame modulation is used **DTAPI** expects DVB-S2 baseband frames with an addition L.3 Header. See DTAPI Manual – Overview and Data Formats.pdf for more details. **DTAPI** decodes this information and performs the DVB-S2 modulation.

*ParXtra0*

Extra modulation parameter #0 specifies the symbol rate (in bd).

*ParXtra1, ParXtra2*

Not used.

## 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	1/4
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. <i>Note:</i> If <i>ParXtra2</i> 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 <i>ParXtra2</i> 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.

## 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	Required Capability
DTAPI_MOD_QAM16	16-QAM	DTAPI_CAP_TX_QAM_A or DTAPI_CAP_TX_QAM_B or DTAPI_CAP_TX_QAM_C
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	Required Capability
DTAPI_MOD_J83_A	J.83 annex A (DVB-C)	DTAPI_CAP_TX_QAM_A
DTAPI_MOD_J83_B	J.83 annex B ("American QAM")	DTAPI_CAP_TX_QAM_B
DTAPI_MOD_J83_C	J.83 annex C ("Japanese QAM")	DTAPI_CAP_TX_QAM_C

### *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 $\mu$ s / 66 $\mu$ s
DTAPI_MOD_QAMB_I64_J2	0011	64	2	47 $\mu$ s / 33 $\mu$ s
DTAPI_MOD_QAMB_I32_J4	0101	32	4	24 $\mu$ s / 16 $\mu$ s
DTAPI_MOD_QAMB_I16_J8	0111	16	8	12 $\mu$ s / 8.2 $\mu$ s
DTAPI_MOD_QAMB_I8_J16	1001	8	16	5.9 $\mu$ s / 4.1 $\mu$ s
DTAPI_MOD_QAMB_I128_J1	0000	128	1	95 $\mu$ s / 66 $\mu$ s
DTAPI_MOD_QAMB_I128_J2	0010	128	2	190 $\mu$ s / 132 $\mu$ s
DTAPI_MOD_QAMB_I128_J3	0100	128	3	285 $\mu$ s / 198 $\mu$ s
DTAPI_MOD_QAMB_I128_J4	0110	128	4	379 $\mu$ s / 264 $\mu$ s
DTAPI_MOD_QAMB_I128_J5	1000	128	5	474 $\mu$ s / 330 $\mu$ s
DTAPI_MOD_QAMB_I128_J6	1010	128	6	569 $\mu$ s / 396 $\mu$ s
DTAPI_MOD_QAMB_I128_J7	1100	128	7	664 $\mu$ s / 462 $\mu$ s
DTAPI_MOD_QAMB_I128_J8	1110	128	8	759 $\mu$ s / 528 $\mu$ s

### *ParXtra2*

Not used.

## Modulation Mode : T2MI

When T2-MI modulation is used **DTAPI** expects a 188-byte transport stream carrying T2-MI packets. **DTAPI** will decode the T2-MI packets and perform the DVB-T2 modulation.

*ParXtra0*

Extra modulation parameter #0 specifies the T2-MI transport-stream bitrate.

*ParXtra1*

Extra modulation parameter #1 is the OR of values for the following fields: First T2-MI Component PID value, Second T2-MI Component PID value and Multi-Profile.

### First T2-MI Component PID

PID-value of the first T2-MI component (0 ... 8190).

Value	Meaning
DTAPI_MOD_T2MI_PID1_SHIFT	Bit position of bit 0 of PID-value of the first T2-MI component
DTAPI__MOD_T2MI_PID1_MSK	AND mask for encoded PID field

### Second T2-MI Component PID

PID-value of the second T2-MI component (0 ... 8190). This field is only valid if multi-profile is enabled.

Value	Meaning
DTAPI_MOD_T2MI_PID2_SHIFT	Bit position of bit 0 of PID-value of the second T2-MI component
DTAPI__MOD_T2MI_PID2_MSK	AND mask for encoded PID field

### Multi-Profile

Value	Meaning
DTAPI_MOD_T2MI_MULT_DIS	Single profile, only first T2-MI component PID is used
DTAPI_MOD_T2MI_MULT_ENA	Multi-profile, both T2-MI components PIDs are used
DTAPI_MOD_T2MI_MULT_MSK	AND mask for encoded Multi-Profile field

*ParXtra2*

Extra modulation parameter #2 specifies the DVB-T2 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

## Result

DTAPI_RESULT	Meaning
DTAPI_OK	The modulation parameters have been set successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_IDLE	Transmit-control state is not <b>DTAPI_TXCTRL_IDLE</b> ; The requested modulation parameters can only be set in idle state
DTAPI_E_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  $-30 \times 0.1 = -3\text{dBm}$ .

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 =  $250 \times 0.1 = 25$ dB. 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 <b>SPIDVBMODE</b> )
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_MODPARS_NOT_SET	For modulators: cannot set transport-rate because modulation parameters have not been set
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**.

For modulators the modulation parameters have to be set with **SetModControl** first 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 <code>SetOutputLevel1</code> 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 for the output channel. It determines the conversions that will be applied to the data written to the output channel. Transmit mode has to be set while transmit mode is still idle, this is before setting transmit control to `DTAPI_TXCTRL_HOLD` or `DTAPI_TXCTRL_SEND`.

**Note:** For ASI/SDI input channels, first configure the output port for ASI or SDI with `SetIoConfig`, group `IOSTD`. If an ASI-specific transmit mode is applied to a channel that is configured for SDI (or vice versa), an error will be returned.

```
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.
DTAPI_TXMODE_RAWASI	Raw ASI symbols (Ports needs CAP_RAWASI) The bytes in the buffer are assumed to be 10-bit packed raw ASI symbols which are transmitted at 270Mb/s.

The following modes are valid for channels configured as SDI.

Value	Meaning
<b>DTAPI_TXMODE_SDI_FULL</b>	<i>Full SDI mode</i> The data in the Transmit FIFO is assumed to consist of complete SDI frames, including all synchronization information.
<b>DTAPI_TXMODE_SDI_ACTVID</b>	<i>Active Video SDI mode</i> 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 <b>DTAPI_TXMODE_SDI_HUFFMAN</b> flag). When using this mode with Huffman compression disabled, the behavior of the output channel is undefined.

The following mode is valid for TS-over-IP transmission only.

Value	Meaning
<b>DTAPI_TXMODE_IPRAW</b>	<i>Raw IP mode</i> The Transmit FIFO is assumed to contain time-stamped IP packets that are transmitted unmodified. Each IP packet shall be preceded by a <b>DtRawIpHeader</b> structure.

For DVB-ASI output channels, *TxMode* can be OR-ed with following flags:

<b>DTAPI_TXMODE_BURST</b>	<i>Burst mode</i> 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).
<b>DTAPI_TXMODE_TXONTIME</b>	<i>Transmit on timestamp</i> 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 DTAPI Manual – Overview and Data Formats.pdf for more details.

For output channels configured as SDI, *TxMode* can be OR-ed with the values specified in the table below:

Value	Meaning
<b>DTAPI_TXMODE_SDI_10B</b>	<i>10-bit SDI samples</i> Indicates that SDI data in the Transmit FIFO is assumed to consist of 10-bit SDI samples. If both this flag and the 16B flag are omitted, 8-bit samples is assumed.
<b>DTAPI_TXMODE_SDI_16B</b>	<i>16-bit SDI samples</i> Indicates that SDI data in the Transmit FIFO is assumed to consist of 10-bit SDI samples packed in 16-bits. Only the 10 least significant bits of each 16-bit sample are used. If both this flag and the 10B flag are omitted, 8-bit samples is assumed. 16B mode is only supported by cards having the CAP_MATRIX capability.

<b>DTAPI_TXMODE_SDI_HUFFMAN</b>	<i>Huffman compression</i> Indicates that the SDI frame in the Transmit FIFO is assumed to be Huffman compressed. Huffman compression is not supported by cards that support HD-SDI such as the DTA-2152.
---------------------------------	--

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 configured as ASI:

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 configured as SDI:

This parameter is ignored if genlock is not supported.

If genlock is supported, but the output channel is not configured to operate in genlock mode, this parameter enables *black-frame stuffing*: 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 transmit channel is in SDI genlock mode, since the driver will automatically enable black-frame stuffing in this case.

Black-frame stuffing is not supported by cards supporting the matrix model such as the DTA-2152.

## Result

DTAPI_RESULT	Meaning
<b>DTAPI_OK</b>	Transmit mode has been changed successfully
<b>DTAPI_E_DEV_DRIVER</b>	Unclassified failure in device driver
<b>DTAPI_E_INVALID_MODE</b>	The specified transmit mode is invalid or incompatible with the output channel
<b>DTAPI_E_NO_GENREF</b>	No genlock reference port has been configured
<b>DTAPI_E_NOT_ATTACHED</b>	Channel object is not attached to a hardware function
<b>DTAPI_E_NOT_IDLE</b>	For DTA-110T and DTA-160 only: transmit mode can only be changed when transmit-control state is <b>DTAPI_TXCTRL_IDLE</b>

## 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 <b>IDLE</b>
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.