

C++ API FOR

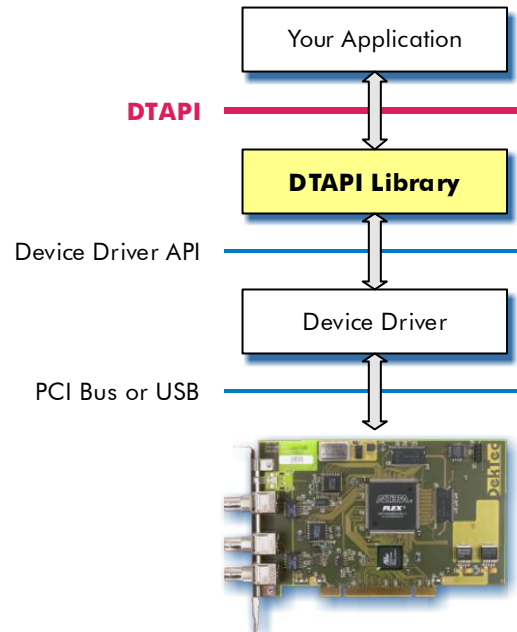
- DTA ADAPTERS FOR PCI
- DTU ADAPTERS FOR USB

- Uniform access to DekTec hardware
- Abstracts from hardware details like DMA, USB, Interrupts, VPD
- Object oriented, easy to use

FEATURES

- Common C++ interface to DekTec's line of digital-video devices
- Encapsulates the device-driver layer in an easy-to-use object-oriented interface
- Access to all hardware features from user-mode programs
- High efficiency; no need to resort to kernel-mode programming to achieve real-time operation
- Operates on top of the Windows WDM device driver for Windows 2000 / 2003 / XP
- Packaged in a C++ header file (to be included in your C++ project) and a library (to be linked to the application)

LAYERING



KEY ATTRIBUTES

Parameter	Value
Maximum number of parallel hardware functions per PC	75
DTAPI Version	4.4.1.121
Dta1xx Device-Driver Version	2.6.0.x / 2.2.0.x
Dtu2xx Device-Driver Version	3.0.0.x / 3.0.0.x

SUPPORTED ADAPTERS

INTERFACE TYPE	SUPPORTED ADAPTERS
PCI	DTA-100, DTA-102, DTA-105, DTA-107, DTA-110, DTA-110T, DTA-115, DTA-120, DTA-122, DTA-124, DTA-140, DTA-145, DTA-160
PCI Express	DTA-2135, DTA-2144, DTA-2145
USB	DTU-205, DTU-225, DTU-234, DTU-235, DTU-245
Ethernet	DTE-3100, DTE-3120

Copyright © 2000-2009 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 Digital Video assumes no responsibility for any errors that may appear in this material.

Table of Contents

1. General Description	7	DtDevice::LedControl.....	65
1.1. Object Model	7	DtDevice::SetDisplayName	66
1.2. Terminology	8	DtDevice::SetIoConfig.....	67
1.3. References.....	8	DtDevice::VpdDelete.....	70
2. Using DTAPI	10	DtDevice::VpdRead.....	71
2.1. Including and Linking DTAPI.....	10	DtDevice::VpdWrite	73
2.2. Using the Static Link Library.....	10	DtInpChannel.....	74
2.3. Using the .NET 2.0 Assembly	10	DtInpChannel::Attach (obsolete).....	75
2.4. Using the DLL	10	DtInpChannel::AttachToPort.....	77
2.5. Using the Linux version	11	DtInpChannel::ClearFifo	79
2.6. Attaching to a DtDevice.....	11	DtInpChannel::ClearFlags	80
2.7. Attaching to a Channel	12	DtInpChannel::Detach	81
2.8. Initialising a Channel	12	DtInpChannel::GetConstellationPoints	82
2.9. Streaming Data – Input	12	DtInpChannel::Equalise	83
2.10. Auxiliary ADC Data – Input.....	13	DtInpChannel::GetDemodControl	84
2.11. Streaming Data – Output	13	DtInpChannel::GetDemodStatus	85
2.12. Using VPD.....	14	DtInpChannel::GetFecErrorCounters	86
2.13. Self-Test Support.....	14	DtInpChannel::GetFifoLoad	87
2.14. Complete Example	14	DtInpChannel::GetFlags	88
3. DTAPI Classes and Methods	16	DtInpChannel::GetMaxFifoSize	90
3.1. Overview.....	16	DtInpChannel::GetReceiveByteCount	91
Struct DtConstelPoint	19	DtInpChannel::GetRfLevel.....	92
Struct DtDeviceDesc	20	DtInpChannel::GetRxControl.....	93
Struct DtHwFuncDesc.....	22	DtInpChannel::GetStatistics.....	94
Struct DtIsdbtLayerPars	25	DtInpChannel::GetStatus	96
Struct DtapiHwFunc (Obsolete)	27	DtInpChannel::GetTargetId	99
Struct DtRawIplHeader.....	28	DtInpChannel::GetTsRateBps	100
Struct DtTslpPars	29	DtInpChannel::GetTunerFrequency	101
::DtapiCheckDeviceDriverVersion	32	DtInpChannel::LedControl	102
::DtapiGetDeviceDriverVersion	33	DtInpChannel::PolarityControl	103
::DtapiGetVersion	35	DtInpChannel::Read	104
::DtapiDeviceScan.....	36	DtInpChannel::ReadDirect	106
::DtapiDtDeviceDesc2String	37	DtInpChannel::ReadFrame	107
::DtapiDtHwFuncDesc2String	38	DtInpChannel::Reset.....	108
::DtapiDvbT2ValidateParams.....	39	DtInpChannel::SetAdcSampleRate	110
::DtapiHwFuncScan.....	40	DtInpChannel::SetAntPower	111
::DtapiInitDtTslpParsFromIplString.....	42	DtInpChannel::SetDemodControl	112
::DtapiPciScan (Obsolete)	43	DtInpChannel::SetIplPars	114
::DtapiModPars2SymRate	44	DtInpChannel::SetLoopBackMode	115
::DtapiModPars2TsRate	45	DtInpChannel::SetPower	116
::DtapiResult2Str	46	DtInpChannel::SetRxControl.....	117
DtDevice::AttachToIplAddr.....	47	DtInpChannel::SetRxMode	118
DtDevice::AttachToSerial	48	DtInpChannel::SetTunerFrequency	121
DtDevice::AttachToSlot	49	DtInpChannel::TuneChannel	122
DtDevice::AttachToType.....	50	DtInpChannel::WriteLoopBackData	123
DtDevice::Detach	51	DtIsdbtPars.....	124
DtDevice::GetDescriptor	52	DtIsdbtPars::CheckValidity	127
DtDevice::GetDeviceDriverVersion	53	DtIsdbtPars::ComputeRates	128
DtDevice::GetDisplayName	54	DtIsdbtPars::RetrieveParsFromTs	129
DtDevice::GetFirmwareVersion	55	DtOutpChannel.....	130
DtDevice::GetIoConfig	56	DtOutpChannel::Attach	131
DtDevice::GetVcxoState	57	DtOutpChannel::AttachToPort.....	133
DtDevice::GetRefClkCnt	58	DtOutpChannel::ClearFifo	134
DtDevice::GetRefClkFreq	60	DtOutpChannel::ClearFlags	135
DtDevice::GetUsbSpeed	61	DtOutpChannel::Detach	136
DtDevice::HwFuncScan.....	62	DtOutpChannel::GetExtClkFreq	137
DtDevice::I2Cread	63	DtOutpChannel::GetFailsafeAlive.....	138
DtDevice::I2Cwrite	64	DtOutpChannel::GetFailsafeConfig.....	139
		DtOutpChannel::GetFifoLoad	140

DtOutpChannel::GetFifoSize	141
DtOutpChannel::GetFlags	142
DtOutpChannel::GetMaxFifoSize	144
DtOutpChannel::GetModControl	145
DtOutpChannel::GetOutputLevel	146
DtOutpChannel::GetRfControl	147
DtOutpChannel::GetTargetId	148
DtOutpChannel::GetTransmitByteCount	149
DtOutpChannel::GetTsRateBps	150
DtOutpChannel::GetTxControl	151
DtOutpChannel::GetTxMode	152
DtOutpChannel::ReadLoopBackData	153
DtOutpChannel::Reset	154
DtOutpChannel::SetFailsafeAlive	155
DtOutpChannel::SetFailsafeConfig	156
DtOutpChannel::SetFifoSize	157
DtOutpChannel::SetIpPars	158
DtOutpChannel::SetLoopBackMode	159
DtOutpChannel::SetModControl	160
DtOutpChannel::SetOutputLevel	179
DtOutpChannel::SetPower	180
DtOutpChannel::SetRfControl	181
DtOutpChannel::SetRfMode	182
DtOutpChannel::SetSNR	183
DtOutpChannel::SetTsRateBps	184
DtOutpChannel::SetTxControl	186
DtOutpChannel::SetTxMode	188
DtOutpChannel::SetTxPolarity	191
DtOutpChannel::Write	192
DtOutpChannel::WriteDirect	194
TsInpChannel	195
TsOutpChannel	196
DtLoop	197
DtLoop::AttachToInput	198
DtLoop::AttachToOutput	199
DtLoop::Detach	200
DtLoop::DetachFromInput	201
DtLoop::DetachFromOutput	202
DtLoop::IsStarted	203
DtLoop::SetStuffingMode	204
DtLoop::Start	205
DtSdi	206
DtSdi::ConvertFrame	207
4. Definition of data formats	209
4.1. 10-bit SDI format	209
4.2. 8-bit SDI format	210
4.3. Huffman-Compressed SDI format	211
4.4. Transparent Mode Packets	214
5. Ports and Hardware Functions per Device ...	216

DTAPI Revision History

Version	Date	Change Description
V4.4.1.121	2009.01.27	<ul style="list-style-type: none"> Updated library for 64-bit support Added support for DTA-2144 Added support for Genlock / Blackburststuffing (DTA-(2)145 / DTA-2144) Add DtlncChannel::ReadSubCh method
V4.3.0.115	2008.07.25	<ul style="list-style-type: none"> Support for tuning DTU-235 and getting RF statistics (level, constellation, BER, etc) from DTU-235 Add support for specifying sub-channel for 1-segment ISDB-T modulation
V4.2.1.114	2008.06.30	<ul style="list-style-type: none"> Fixed bug in DtOutpChannel::SetOutputLevel and DtOutpChannel::SetSNR for DTA-107 Solved link conflicts between DTAPI and applications that uses gSOAP Fixed bug which prevented the Linux version of the DTAPI to attach to Dtu2xx devices Support for transparent-packet receive mode for the DTA-160 IP port (FOR NOW: only available on Windows)
V4.2.0.112	2008.05.21	<ul style="list-style-type: none"> Fixed bug in DtlncChannel::SetTunerFrequency resulting in a failure to set the tuner frequency for a DTU-234 Support for new Dtu2xx Linux driver Add initial support for DTE-31XX devices (FOR NOW: only available on Windows)
V4.1.1.108	2008.02.27	<ul style="list-style-type: none"> Added section 2.5, with usage instruction for Linux, to document Support for DTA-2135 Fixed bug in DtOutpChannel::AttachToPort: for DTA-145/2145 DTAPI_E_NO_DT_OUTPUT was returned if port 2 was configured in fail-safe mode
V4.0.4.104	2008.01.07	<ul style="list-style-type: none"> In DTAPI.h file: renamed DtDeviceDesc2String to DtapiDtDeviceDesc2String Add support for setting output level of a DTA-107 with firmware version 3 or higher New routine for DtOutpChannel class: SetSNR DTAPINET: signed with strong name DTA-160 IP on Linux: fixed failure to resolve the MAC address with ARP if multiple Ethernet devices are present in the Linux PC. Updated definition of DtDevice::VpdDelete, DtDevice::VpdRead and DtDevice::VpdWrite methods (changed keyword parameter from char* to const char*)
V4.0.2.101	2007.12.13	<ul style="list-style-type: none"> Support for double-buffered and loop-through capabilities in DTA-145/160/2145 Support for transparent-packet mode and SDI time-stamping Updated description of capabilities (see Struct DtHwFuncDesc description) and supported capabilities per device in Table 7 Updated description of DtDevice::GetloConfig and DtDevice::SetloConfig methods Updated behavior of IP transmit if the IP address assigned to the DTA-160 IP port, changes during transmission (i.e. automatically set new IP address in source IP address field of all IP packets generated)
V4.0.0.99	2007.11.12	<ul style="list-style-type: none"> Merged Linux and Windows versions of DTAPI, so that both are based on the same source code. NOTE: In the Linux version the Dtu2xx devices are not supported. The Dtu2xx devices will be supported again from version 4.2.0.x
V3.8.0.91	2007.09.04	<ul style="list-style-type: none"> Add support for ADTB-T and ATSC modulation Bug fix for DVB-S2 modulation in 8-PSK mode with pilots on Support of modulation routines for DTA-115 firmware version 2 Changed name of DtapiModulationPars2TsRate into DtapiModPars2TsRate; Added parameter SymRate; Generalise to any modulation mode Add global function DtapiModPars2SymRate Support of 1-segment and 3-segment ISDB-T modulation in broadcast TV mode Add member m_FilledOut to DtIsdbtPars: must be set to true for correct operation
V3.7.0.90	2007.06.05	<ul style="list-style-type: none"> New global routine: DtapiModulationPars2TsRate

		<ul style="list-style-type: none"> New routines for DtInpChannel class: GetConstellationPoints, GetFecErrorCounters and GetRfLevel
V3.6.0.88	2007.05.16	<ul style="list-style-type: none"> Modified section 2.1 and added sections 2.2, 2.3 and 2.4, which provide details on using the different versions (static link library, .NET 2.0 assembly and DLL) of the DTAPI Add GetRefClkFreq method to DtDevice class Improved PCR handling algorithm in hierarchical ISDB-T multiplexing code Add overloaded version of DtInpChannel::Read method which includes a timeout parameter Add optional timeout parameter to DtInpChannel::ReadFrame Add support for ISDB-T 1-segment radio broadcast
V3.5.0.85	2007.03.30	<ul style="list-style-type: none"> Support for DTA-116 Fixes initialised problem with ISDB-T modulation
V3.4.2.84	2007.03.29	<ul style="list-style-type: none"> Support for DTMB modulation DtIsdbtPars: Support for specifying a default layer in ISDB-T multiplexing Updated documentation of DtInpChannel::GetDemodStatus
V3.4.0.81	2007.02.06	<ul style="list-style-type: none"> Support for DTA-107S2, DTA-115, DTA-145, DTA-2145 Support for hierarchical ISDB-T multiplexing New DtSdi class with SDI support functions New looping algorithm in DtLoop class. More robust against high jitter input streams (e.g. from IP) New global routine: DtapiDvbS2GetBitRate New routines for DtInpChannel class: GetDemodControl, GetDemodStatus, GetTunerFrequency, ReadFrame, SetTunerFrequency, SetDemodControl, SetTunerFrequency and TuneChannel New routines for DtOutpChannel class: GetFailsafeAlive, GetFailsafeConfig, GetOutputLevel, SetFailsafeAlive, SetFailsafeConfig and SetOutputLevel Add section describing the data formats by DekTec cards and in the DTAPI
V3.3.0.75	2006.10.06	<ul style="list-style-type: none"> Support for DTA-160 Rev4 Several minor bug-fixes related to configurable I/O support Support for new VPD control interface of Dta1xx driver
V3.2.0.71	2006.08.29	<ul style="list-style-type: none"> Support for DTA-105 Support for ISDB-T with the DTA-110T Performance enhancements for the DTA-160 IP transmit Support for RAW IP mode in the transmit-channel Several bug-fixes for IP FEC generation algorithm
V3.0.5.64	2006.05.22	<ul style="list-style-type: none"> New generic classes DtInpChannel and DtOutpChannel with SdiInpChannel, TslpInpChannel, SdiOutpChannel and DtOutpChannel as derived classes New concept (physical) Port to identify a specific port of a device; Addition of port-related fields to DtDeviceDesc and DtHwFuncDesc; New member functions DtInpChannel::AttachToPort and DtOutpChannel::AttachToPort Redefinition of a number of fields in DtHwFuncDesc for improved description of hardware capabilities New DtLoop class for looping Transport Streams from port to port New structure DtTslpPars and function DtOutpChannel::SetTslpPars for specifying TS-over-IP parameters Addition of several string conversion routines Support for 8-bit SDI on DTA-160, DTU-225 and DTU-245 Support for DTA-110T, DTA-160, DTU-234, DTU-245
V2.4.4.44	2005.07.11	<ul style="list-style-type: none"> Support for DTA-110 Fixed several errors in the example code included in this document
V2.3.0.39	2004.12.29	<ul style="list-style-type: none"> New routines: DtInpChannel::GetMaxFifoSize, DtInpChannel::GetReceiveByteCount, DtOutpChannel::GetTransmitByteCount, DtDevice::I2Cread and DtDevice::I2Cwrite

V2.1.2.36	2004.09.16	<ul style="list-style-type: none"> • Support for DTA-107 • Support for DTU-205
V2.0.0.30	2004.03.15	<ul style="list-style-type: none"> • Added support for DTU-225 • Added DtDevice class
V1.3.1.25	2003.02.07	<ul style="list-style-type: none"> • Support for time stamping on DTA-140, amongst others addition of time-stamp flag in DtInpChannel::SetRxMode • Add support for receive mode DTAPI_RXMODE_STMP2 • Add support for transmit mode DTAPI_TXMODE_MIN16 • Change of DtInpChannel::Read, so that entire receive FIFO can be read, even without input signal • New routines: PciCard::GetRefClkCnt DtInpChannel::ClearFifo, DtInpChannel::GetRxControl, DtInpChannel::WriteLoopBackData
V1.2.0.20	2002.11.12	<ul style="list-style-type: none"> • Added this change description section • New routine ::DtapiCheckDeviceDriverVersion
V1.1.0.19	2002.11.11	<ul style="list-style-type: none"> • Support for DTA-140
V1.0.2.6	2002.03.22	<ul style="list-style-type: none"> • New routine DtOutpChannel::ClearFifo to clear transmit FIFO without generation of a truncated packet

1. General Description

DTAPI is an acronym for DekTec Application Programming Interface. The **DTAPI** enables application programs to access the functions of DekTec Devices at a higher level of abstraction than would be possible using direct device-driver calls. Still, the **DTAPI** allows access to nearly all hardware features.

On the Windows™ platform, the **DTAPI** is composed of a header file, to be included by the application source code, and a library file, to be linked to the application's executable.

1.1. Object Model

The **DTAPI** is a collection of C++ classes, each representing a generic hardware function. Instances of a **DTAPI** class refer to a specific hardware function on a particular device. The hardware is controlled indirectly by invoking methods on the **DTAPI** objects.

Figure 1 illustrates a number of instantiated **DTAPI** objects in action. The application interacts with the **DTAPI** objects. The library methods implementing the **DTAPI** classes communicate with the hardware through the WDM device driver.

Each DekTec device is represented by precisely one **DtDevice** object. This object is the starting point for all interaction with a particular DekTec device. The following functions are supported:

- Instantiating channels for streaming.
- Getting versioning information on the device driver and firmware installed on the hardware.
- Reading and writing Vital Product Data (VPD), used for storing product-identification data. VPD is divided in a read-only part (e.g. *part number*) and a read/write part for custom identification information.

The **DtDevice** class is supplemented by global function **DtapiDtDeviceScan** to scan the PCI and USB buses and build an inventory of DekTec devices in the system.

Streaming functions are represented by "channel" objects. For the purpose of streaming

MPEG-2 Transport Streams, two channel classes are defined: **DtInpChannel** for reading a Transport Stream and **DtOutpChannel** for outputting a Transport Stream. A device may support multiple physical input and/or output streams, so in general a device can be represented by multiple channel objects.

Class **DtOutpChannel** supports the following functions:

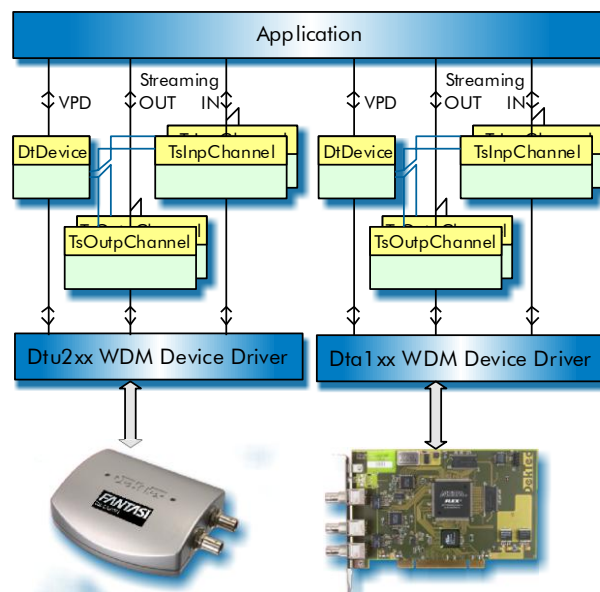


Figure 1. Example of DTAPI objects representing two devices.

- Attaching to (and detaching from) the hardware identified by a **DtDevice** object.
- Initialising and controlling the output channel, like setting bit rate and packet size, reading the load of the Transmit FIFO, etc.
- The core function: writing Transport-Stream data. This function encapsulates DMA details, so that the user is effectively shielded from the entire DMA process.
- Controlling special functions supported by the output channel, such as identification of the connected Target Adapter, applying power supply to the Target Adapter, measurement of external-clock frequency, etc.
- Functions to enable software testing of the Transmit FIFO on-board of the device.

Class **DtInpChannel** supports similar functions for an input stream:

- Attaching to (and detaching from) the hardware identified by a **DtDevice** object.

- Initialising and controlling the input channel, like setting packet size, reading bit rate, reading the load of the Receive FIFO, etc.
- Reading incoming Transport-Stream data and storing it in a buffer in host memory.
- Controlling special functions supported by the input channel, such as identification of the connected Target Adapter, applying power supply to the Target Adapter, etc.
- Functions to enable software testing of the Receive FIFO on-board of the device.

Note

- There is no one to one mapping between **DTAPI** functions and system calls to the device driver: **DTAPI** method calls may lead to zero, one or more system calls to the device driver.

1.2. Terminology

bslbf – Bit string, left bit first, where “left” is the order in which bit strings are written in this document. Bit strings are written as a string of 1s and 0s within single quote marks, e.g. ‘1000 0001’. Blanks within a bit string are for ease of reading and have no significance.

Bsrlb – Bit string right-to-left in bytes.

Channel object – Instance of a C++ class representing a single physical input or output stream. The application manipulates input or output streams by invoking methods on the channel object.

DtDevice – DekTec Device, generic designator for either PCI card or USB adapter.

DTA-xxx card – Any DekTec PCI card in the DTA series.

Dta1xx – Name of the WDM device driver for Windows-2000 / XP / 2003. This device driver is generic for DekTec PCI Cards.

DTAPI – DekTec Application Programming Interface.

DTU-xxx device – Any DekTec USB adapter in the DTU series.

Dtu2xx – Name of the WDM device driver for Windows-2000 / XP / 2003. This device driver is generic for DekTec USB Adapters.

(hardware) function – An elementary “capability” of a PCI card, e.g. outputting a Transport Stream in DVB-ASI format. A device can support multiple functions. Note that the interpretation of *function* is not necessarily equivalent to the *PCI-function* concept in the PCI rev. 2.2 specifications.

PCI card – The hardware card that contains the interface-adaptation and/or other processing hardware.

Target adapter – External hardware device that translates the DVB-SPI signals on the DTA-102 and DTA-122 to another physical format.

Uimsbf – Unsigned integer, most significant bit first.

USB Adapter – The USB-compatible hardware that contains the interface-adaptation and/or other processing hardware.

VPD – “Vital Product Data.” Information stored in a PCI device to uniquely identify the hardware and, potentially, software elements of the device. VPD is defined in *PCI Local Bus Specification Rev2.2*. DekTec devices store VPD in on-board serial EEPROMs. **DTAPI** supports methods to read and write VPD items.

1.3. References

- *ISO/IEC 13818-1, Information technology – Generic coding of moving pictures and associated audio information: Systems*, April 27th, 1995, also known as “MPEG-2 Systems” – Specification of the structure of a MPEG-2 Transport Stream.
- Recommendation ITU-R BT.656-4. Interfaces for digital component video signals in 525-line and 625-line television systems operating at the 4:2:2 level of recommendation ITU-R BT.601 (Part A), 1998
- *DTA-100, DVB-ASI Output Adapter for PCI Bus, Product Specification, DekTec Digital Video B.V., 2002.*
- *DTA-102, DVB-SPI Output Adapter for PCI Bus, Product Specification, DekTec Digital Video B.V., 2002.*

- DTA-107, QPSK Modulator / Upconverter for PCI Bus, Product Specification, DekTec Digital Video B.V., 2004.
- DTA-120, DVB-ASI Input Adapter for PCI Bus, Product Specification, DekTec Digital Video B.V., 2002.
- DTA-122, DVB-SPI Input Adapter for PCI Bus, Product Specification, DekTec Digital Video B.V., 2002.
- DTA-124, Quad ASI/SDI Input Adapter for PCI Bus, Product Specification, DekTec Digital Video B.V., 2004.
- DTA-140, DVB-ASI Input+Output Adapter for PCI Bus, Product Specification, DekTec Digital Video B.V., 2003.
- DTA-2135, Dual/Diversity DVB-T Receiver for PCI Express, Product Specification, DekTec Digital Video B.V., 2008.
- DTU-205, DVB-ASI/SDI Output Adapter for USB, Product Specification, DekTec Digital Video B.V., 2004.
- DTU-225, DVB-ASI/SDI Input Adapter for USB, Product Specification, DekTec Digital Video B.V., 2004.
- PCI Local Bus Specification, Revision 2.2, December 18, 1998 – Formal specification of the protocol, electrical, and mechanical features of the PCI bus.

2. Using DTAPI

This section discusses the actual usage of the **DTAPI** library. Code snippets are provided to illustrate key methods.

2.1. Including and Linking DTAPI

All **DTAPI** declarations and definitions are contained in one C++ header file: `"DTAPI.h"`. Each module that uses **DTAPI** functionality has to include this file.

The **DTAPI** implementation for Windows is available as static link library, a .NET 2.0 assembly or as a DLL.

See the following sections for more details on using the different version of the **DTAPI** on the Windows platform.

2.2. Using the Static Link Library

Two **DTAPI** versions are available for static linking: **DTAPI.lib** and **DTAPIMD.lib**.

DTAPI.lib has been compiled with the C/C++ Code-Generation Options in VC++ set to: "Use run-time library: Multithreaded." On the compiler command line this corresponds to the `/MT` option.

DTAPIMD.lib has been compiled with the C/C++ Code-Generation Options in VC++ set to: "Use run-time library: Multithreaded DLL." On the compiler command line this corresponds to the `/MD` option.

The correct version of the **DTAPI** library file will automatically be linked, because of a `pragma` directive in **DTAPI.h**.

Automatic linking can be disabled by defining `_DTAPI_DISABLE_AUTO_LINK` (using `#define` or in the Pre-processor Definitions).

So, to use static link library of the **DTAPI** follow these steps:

1. Copy **DTAPI.h** and either **DTAPI.lib** or **DTAPIMD.lib** to your project or to a standard location visible to VC++.
2. Add `#include "DTAPI.h"` to each file using **DTAPI** functions.

3. Compile your application using the Multi-threaded DLL (compiler switch `/MD`) or static (compiler switch `/MT`) version of the C runtime library.

NOTE: the static library files are available for VC6, VC7 and VC8. Be sure to use the version matching your Visual Studio C++ version.

2.3. Using the .NET 2.0 Assembly

DTAPINET.dll is a .NET 2.0 compatible assembly of the **DTAPI**. To use it you should perform the following steps:

1. Make sure the .NET 2.0 SDK has been installed on your system.
2. Copy **DTAPINET.dll** to your project or to a standard location visible to VC# (or other .NET IDE).
3. Add a reference to the **DTAPINET.dll** assembly to your project.
4. Add a `#using DTAPINET` statement to the beginning of each source file that uses the classes, methods, and or constants which are exported by the **DTAPINET** assembly.

2.4. Using the DLL

DTAPIDLL.dll is a dynamic link library version of the **DTAPI**. The corresponding import library (to which your project should link) is **DTAPIDLL.lib**.

Before using the DLL you must first define the `_USE_DTAPIDLL` symbol in the "Preprocessor Definitions" of your project's compiler settings.

DTAPIDLL.lib will automatically be linked because of a `pragma` directive in **DTAPI.h**.

To use the DLL you should perform the following tasks:

1. Copy **DTAPI.h**, **DTAPIDLL.lib** and **DTAPIDLL.dll** to your project or to a standard location visible to VC++.
2. Add a definition of the `_USE_DTAPIDLL` symbol to the "Preprocessor Definitions" section in your project's compiler settings.
3. Add `#include "DTAPI.h"` to each file using **DTAPI** functions.

2.5. Using the Linux version

To use the DTAPI in a Linux application, please perform the following steps:

1. Make sure the **DTAPI.h** and **DTAPI.o** files are located in a path reachable for you project.
2. Add `#include "DTAPI.h"` to each file using **DTAPI** functions.
3. Make sure you project links to the **DTAPI.o** library file.
4. The DTAPI requires the **pthread** library, therefore also make sure your project links with **pthread** library.

For an example makefile please refer to the make files for the DtPlay and DtRecord projects as included in the LinuxSDK distribution.

NOTE: the DTAPI library file is available for different GCC versions Please refer too the `"/LinuxSDK/DTAPI/Bin/"` directory.

2.6. Attaching to a DtDevice

Programs based on **DTAPI** first have to instantiate a **DtDevice** object and "attach" it to a hardware card. This can be accomplished in several ways.

Method **AttachToType** is convenient when the DekTec device's type number is known and the system contains a single PCI card or USB adapter of the given type¹.

```
DtDevice Dvc;
if (Dvc.AttachToType(122) != DTAPI_OK) {
    // No DTA-122 in the system ...
    // Error-handling code
}
```

Figure 2. Attaching a **DtDevice** object to the hardware PCI card based on type number.

Another, method **AttachToSerial** can be used if the unique serial number of the device is known.

```
DtDevice Dvc;
if (Dvc.AttachToSerial(410000001) !=
DTAPI_OK) {
    // No card with serial 410000001
    connected...
    // Error-handling code
}
```

Figure 3. Attaching a **DtDevice** object to the hardware based on serial number.

Alternatively, method **AttachToSlot** can be used if the physical location (PCI Bus#, Slot#) of a PCI card in the system is known.

```
DtDevice Dvc;
if (Dvc.AttachToSlot(1, 3) != DTAPI_OK)
{
    // No card in slot 3 on PCI bus 1 ...
    // Error-handling code
}
```

Figure 4. Attaching a **DtDevice** object to the hardware PCI card based on PCI-Bus and Slot number.

A more sophisticated application creates an inventory of available **DTAPI**-compatible PCI cards and USB Adapters and lets the user configure which device is to be used. Alternatively, the software applies an automatic selection criterion, e.g. the first device with a Transport-Stream output. Such schemes can be built with the help of global functions **DtapiHwFuncScan** and **DtapiDeviceScan**.

```
DtHwFuncDesc Funcs[10];
int f, nFuncs;
::DtapiHwFuncScan(10, nFuncs, Funcs);

for (f=0; f<nFuncs; f++) {
    if (Funcs[f].m_ChanType &
        DTAPI_TS_OUTPUT)
        break;
}
if (f == nFuncs) { // No output card }

DtDevice Dvc;
Dvc.AttachToSerial(
    Funcs[f].m_DvcDesc.m_Serial);
```

Figure 5. Selecting the first **DtDevice** with a Transport-Stream output on board.

After all operations have been completed, a **DtDevice** object should be detached from the hardware with method **Detach**.

¹ Or when usage of the first device of the given type is acceptable.

2.7. Attaching to a Channel

Before you can stream data into or out of a DekTec Device, two objects must have been instantiated and attached to the hardware:

- **DtDevice** object (§2.6);
- **DtOutpChannel** or **DtInpChannel** object.

A channel object can be attached to the hardware with the channel's **AttachToPort** member function. The first parameter of this function, a pointer to the **DtDevice** object, identifies the hardware device. The second parameter identifies the port number.

Example code to attach an output channel is shown in Figure 6. After attachment, the size of the channel's Transmit FIFO is initialised to its maximum value.

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

DtOutpChannel TsOut;
if (TsOut.AttachToPort(&Dvc, 1) !=
    DTAPI_OK) {
    // Error-handling code
}
TsOut.SetFifoSizeMax();
```

Figure 6. Attaching a **DtOutpChannel** object to the hardware.

Input channels are attached to the hardware in precisely the same way, except that the size of the Receive FIFO cannot be programmed.

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

2.8. Initialising a Channel

A number of **DTAPI** methods are available to initialise a channel. Refer to the reference pages later in this document for a description of the semantics.

DtOutpChannel

Method **Reset** clears the contents of the Transmit FIFO and provides other initialisation services.

Method **SetTsRateBps** changes the output bit rate.

SetTxMode sets packet size, burst- or continuous mode, and other special modes.

DtInpChannel

Input channels also support a **Reset** method, which clears the Receive FIFO.

Method **SetRxMode** sets the packet size of packets stored in the Receive FIFO.

2.9. Streaming Data – Input

This section considers the actual streaming of data from an external source to a MPEG-2 Transport-Stream processing application. The core of an elementary input-streaming program is shown in Figure 7. This code assumes:

- **DtDevice** object **Dvc** and **DtInpChannel** object **TsIn** have been attached to the hardware;
- The Receive FIFO is empty and receive mode has been initialised;
- **ProcessTsData(Buf, NumBytes)** is the user-supplied function that processes **NumBytes** new Transport-Stream data bytes in **Buf**.
- **StopCondition()** is a user-supplied function that returns true if sufficient data has been read.

```
// PRE: Dvc and TsIn attached
char Buf[BUFSIZE];

// Start reading Transport Stream
TsIn.SetRxControl(DTAPI_RXCTRL_RCV);

// Main loop
while (!StopCondition()) {
    TsIn.Read(Buf, BUFSIZE);
    ProcessTsData(Buf, BUFSIZE);
}
```

Figure 7. Processing data from an external Transport-Stream source.

The code is straightforward. First receive mode is set to **Rcv**, which starts reading of incoming data into the Receive FIFO. Then the main loop alternates between reading a buffer load of data and processing the data, until the stop condition becomes true.

The following factors should be considered to achieve optimal results:

- The buffer size (constant **BUFSIZE**) should not be too small. Every data transfer from Receive FIFO to the buffer in host memory

incurs non-negligible overhead: a system call invokes the device driver, which for PCI cards, builds a DMA scatter/gather list for every transfer.

A reasonable minimum buffer/transfer size is 4096 bytes. No maximum size exists²; the buffer size may very well be a few Mbytes.

- The number of bytes returned by method **Read** always is a multiple of 4. It is not guaranteed that the data aligns to Transport-Packet boundaries, even if the buffer size is a multiple of the packet size. The processing software should always start with a synchronisation stage.

2.10. Auxiliary ADC Data – Input

Some cards provide optional access to auxiliary streaming data on some of the input channels. This auxiliary streaming data can be requested in parallel to the transport stream data by specifying the DTAPI_SUBCH_ADC as additional sub-channel parameter in the read related calls.

For the DTA-2135 the 10bit ADC samples use 2 bytes.

```
// PRE: Dvc and TsIn attached
// and channel tuned to valid signal
char Buf[BUFSIZE];

// Start buffering ADC Samples
TsIn.SetRxControl(DTAPI_RXCTRL_RCV,
                  DTAPI_SUBCH_ADC);

// Main loop
while (!StopCondition()) {
    TsIn.ReadSubCh(Buf, BUFSIZE,
                  DTAPI_SUBCH_ADC);

    // Note: nSamples = BUFSIZE / 2;
    ProcessAdcData(Buf, BUFSIZE);
}
```

Figure 8. Processing ADC samples.

² However, to avoid swapping, the buffer size should not exceed the amount of physical RAM available to a process.

2.11. Streaming Data – Output

Streaming data to an output is somewhat more involved than inputting data. The core of an elementary output-streaming program is shown in Figure 9. The code assumes:

- **DtDevice** object **Dvc** and **DtOutputChannel** object **TsOut** have been attached to the hardware;
 - Transmit mode has been initialised;
 - **GetTsData(Buf, Max)** is the user-supplied function that writes maximally **Max** new Transport-Stream data bytes in **Buf**, and returns the number of bytes written.
- Note:** The hardware uses 32-bit words internally. Therefore, number of bytes must be a multiple of four at all times.

The first part of the code builds an initial load in the Transmit FIFO before actual transmission begins. Hereto transmission control is set to **Hold**, which enables DMA but keeps transmission disabled.

```
// PRE: Dvc and TsOut attached
TsOut.SetTxControl(DTAPI_TXCTRL_HOLD);

// Build initial load in Transmit FIFO
char Buf[BUFSIZE];
int Load=0;
int NumBytes = GetTsData(Buf, BUFSIZE);
while (Load<INILOAD && NumBytes!=0) {
    TsOut.Write(Buf, NumBytes);
    Load += NumBytes;
    NumBytes = GetTsData(Buf, BUFSIZE);
}
TsOut.SetTxControl(DTAPI_TXCTRL_SEND);

// Main loop
while (NumBytes != 0) {
    TsOut.Write(Buf, NumBytes);
    NumBytes = GetTsData(Buf, BUFSIZE);
}
```

Figure 9. Streaming data to a Transport-Stream output.

When the Transmit FIFO contains its initial load, actual transmission is started by setting transmission control to **Send**. The main loop then supplies additional data to the Transmit FIFO until the data source is exhausted.

The following factors should be considered to achieve optimal results:

- The buffer size (constant **BUFSIZE**) should not be too small. Every data transfer to the Transmit FIFO incurs non-negligible overhead: a system call invokes the device driver, which for PCI cards builds a DMA scatter/gather list for every transfer. A reasonable minimum buffer/transfer size is 4096 bytes. No maximum size exists³; the buffer size may very well be a few Mbytes.
- The initial Transmit-FIFO load (**INILOAD**) should not be too small either, to prevent an early Transmit-FIFO underflow in the main loop. A value close to the maximum FIFO size is recommended.
Warning. The initial load cannot be larger than the Transmit-FIFO size: when the Transmit FIFO is full, DMA will stall and the application “hangs.”
- As far as **DTAPI** is concerned, the **GetTsData** function may return Transport-Stream data aligned at arbitrary 4-byte boundaries. However, for many data-generating algorithms, alignment on packet boundaries will be a natural choice. In such applications it is convenient and efficient to set the buffer size to a multiple of the packet size.

2.12. Using VPD

Vital Product Data (VPD) is product identification information stored in an EEPROM onboard of DekTec Devices. The read-only part of VPD is loaded in the manufacturing process. The read/write part is used for licensing purposes and for storing customer-specific product information.

VPD is initialized as a collection of *items*, each identified by a *keyword*. Most keywords are 2-character strings (e.g. “PD” for Production Date), with the exception of the VPD ID String, which is identified by “VPDID”.

Three member functions of **DtDevice** are defined to manipulate VPD:

- **VpdRead** – Read VPD-item, given a keyword.

³ However, to avoid swapping, the buffer size should not exceed the amount of physical RAM available to a process.

- **VpdWrite** – Write VPD-item, given a keyword and item string. If the item existed, the item string is overwritten, unless the VPD item is read-only.
- **VpdDelete** – Delete VPD item. Read-only VPD item cannot be deleted.

2.13. Self-Test Support

The integrity of memory on DekTec Devices, as used for Input- and Transmit FIFOs, can be tested in *Loop-Back Mode*. In this mode, the FIFO is detached from the card’s input or output and directly accessible to software. This enables construction of a simple memory test: a known pattern is written to the FIFO, after some time the data is read back and compared to the original.

DTAPI-support for self-test is straightforward. Loop-back mode can be entered with method **SetLoopBackMode**. Data can be written to the Receive FIFO with **WriteLoopBackData**. Data can be read from the Transmit FIFO with **ReadLoopBackData**.

2.14. Complete Example

Figure 10 shows the code of a simple but fully functional command-line stream player that is capable of playing out a file. The file name and bit rate at which to play out the file can be specified as command-line arguments.

The example exploits good-old “**stdio**” functions for reading file data. By using a relatively large buffer, performance is more than adequate.

Obviously, this example is just a first step towards a production-quality streamer application. With respect to **DTAPI**, one obvious improvement would be to check the return code for every **DTAPI**-call, and add corresponding error-handling code.


```

// Command-line program TsOut
// Streams Transport-Stream file out of DTA-100

#define BUFSIZE (1<<16)
#define INILOAD (7*(1<<20))    // 7MB initial load

#include "DTAPI.h"
#include <stdio.h>

int main(int argc, char* argv[])
{
    if (argc != 3) { printf("Usage: TsOut bitrate tsfile\nQuitting...\n"); return -1; }
    // Open Transport-Stream file
    FILE* fp = fopen(argv[2], "rb");
    if (fp == NULL) {
        printf("Can't open '%s' for read\nQuitting...\n", argv[2]);
        return -2;
    }
    // Attach device and output channel objects to hardware
    DtDevice Dvc;
    if (Dvc.AttachToType(100) != DTAPI_OK) {
        printf("No DTA-100 in system. Quitting...\n");
        return -3;
    }
    DtOutpChannel TsOut;
    if (TsOut.AttachToPort(&Dvc, 1) != DTAPI_OK) {
        printf("Can't attach output channel.\nQuitting...\n");
        return -4;        // Detach is executed from DtDevice destructor
    }
    // Initialise bit rate and packet mode
    TsOut.SetTsRateBps(DTAPI_TXCLOCK_INTERNAL, atoi(argv[1]));
    TsOut.SetTxMode(DTAPI_TXMODE_188, 1);

    // Build initial load in Transmit FIFO
    TsOut.SetTxControl(DTAPI_TXCTRL_HOLD);
    char Buf[BUFSIZE];
    int Load = 0;
    int NumBytes = fread(Buf, 1, BUFSIZE, fp);
    while (Load<INILOAD && NumBytes!=0) {
        TsOut.Write(Buf, NumBytes);
        Load += NumBytes;
        NumBytes = fread(Buf, 1, BUFSIZE, fp);
    }
    // Start transmission
    TsOut.SetTxControl(DTAPI_TXCTRL_SEND);
    // Main loop
    while (NumBytes != 0) {
        TsOut.Write(Buf, NumBytes);
        NumBytes = fread(Buf, 1, BUFSIZE, fp);
    }
    TsOut.Detach (DTAPI_INSTANT_DETACH);
    return 0;        // Detach is executed from DtDevice destructor
}

```

Figure 10. Complete command-line application to stream a file with the DTA-100.

3. DTAPI Classes and Methods

3.1. Overview

Table 1. **DTAPI** – Global Functions

API Function	Description
<code>::DtapiCheckDeviceDriverVersion</code>	Check compatibility between Dta1xx and DTAPI
<code>::DtapiGetDeviceDriverVersion</code>	Get version of DTA1xx device driver
<code>::DtapiGetVersion</code>	Get version of DTAPI library
<code>::DtapiDeviceScan</code>	Scan for DTAPI devices
<code>::DtapiDtDevice2String</code>	Create descriptive string for a device
<code>::DtapiDtHwFuncDesc2String</code>	Create descriptive string for a hardware function
<code>::DtapiHwFuncScan</code>	Scan for DTAPI hardware functions
<code>::DtapiInitDtTslpParsFromIpString</code>	Initialise IP addresses in a DtTslpPars structure
<code>::DtapiPciScan (obsolete)</code>	Scan DTAPI hardware functions on PCI bus
<code>::DtapiModPars2SymRate</code>	Compute symbol rate from TS rate and modulation parameters
<code>::DtapiModPars2TsRate</code>	Compute TS rate from modulation parameters
<code>::DtapiResult2Str</code>	Convert DTAPI_RESULT value to a string

Table 2. **DTAPI** – **DtDevice** Functions

API Function	Description
<code>AttachToIpAddr</code>	Attach DtDevice object to hardware, based on IP address
<code>AttachToSerial</code>	Attach DtDevice object to hardware, based on serial number
<code>AttachToSlot</code>	Attach DtDevice object to hardware, based on slot number
<code>AttachToType</code>	Attach DtDevice object to hardware, based on type number
<code>Detach</code>	Detach DtDevice object from

	hardware
<code>GetDeviceDriverVersion</code>	Get version of device driver for this DtDevice
<code>GetDisplayName</code>	Get current name on LCD status display
<code>GetIoConfig</code>	Get current I/O configuration
<code>GetRefClkFreq</code>	Get frequency of onboard reference clock
<code>GetUsbSpeed</code>	Get USB bus speed
<code>GetFirmwareVersion</code>	Get version of the firmware loaded on the device
<code>I2Cread</code>	Read data from the I2C bus
<code>I2Cwrite</code>	Write data to the I2C bus
<code>LedControl</code>	Control of general-status LED
<code>SetDisplayName</code>	Set name on LCD status display
<code>SetIoConfig</code>	Set I/O configuration
<code>VpdDelete</code>	Delete Vital-Product-Data item
<code>VpdRead</code>	Read Vital-Product-Data item
<code>VpdWrite</code>	Write Vital-Product-Data item

Table 3. **DTAPI** – **DtInpChannel** Functions

API Function	Description
<code>Attach (obsolete)</code>	Attach channel object to hardware
<code>AttachToPort</code>	Attach channel object to hardware
<code>ClearFifo</code>	Clear receive FIFO
<code>ClearFlags</code>	Clear latched status flag(s)
<code>Detach</code>	Detach channel object from hardware function
<code>Equalise</code>	Control DVB-ASI cable equalisation (DTA-120 Rev 1 only).
<code>GetDemodControl</code>	Get current (de)modulation parameters
<code>GetDemodStatus</code>	Get current demodulator/tuner status
<code>GetFifoLoad</code>	Get current load of the Receive FIFO

GetFlags	Get status flags
GetRxControl	Get receive-control state
GetStatistics	Get statistics information
GetStatus	Get status information
GetMaxFifoSize	Get the size of the Receive FIFO
GetReceive-ByteCount	Get a sample from the free running 32-bit received number of bytes counter
GetTargetId	Get target-adaptor identifier (DTA-122 only)
GetTsRateBps	Get an estimate of the input Transport-Stream rate
GetTuner-Frequency	Get current tuner frequency
LedControl	Control of input-status LED
PolarityControl	Control polarity-detection circuitry of a DVB-ASI input
Read	Read input data
ReadDirect	Read input data without DMA
ReadFrame	Read one complete SDI frame from the input channel
Reset	Reset input channel
SetAdcSampleRate	Set sample-rate for ADC
SetAntPower	Enable power to active antennas
SetDemod-Control	Set (de)modulation parameters
SetIpPars	Setup IP parameters
SetLoopBack-Mode	Set loop-back mode
SetPower	Control power for target adaptor (DTA-122)
SetRxControl	Set receive-control state
SetRxMode	Set receive mode
SetTuner-Frequency	Set the tuner frequency
TuneChannel	Tune to specific channel
WriteLoopBack Data	Write data to Receive FIFO via loop-back channel

Table 4. **DTAPI** – **DtOutpChannel1** Functions

API Function	Description
Attach (obsolete)	Attach channel to hardware
AttachToPort	Attach channel to hardware
ClearFifo	Clear transmit FIFO
ClearFlags	Clear latched status flag(s)
Detach	Detach channel object from hardware function
GetExtClkFreq	Get external-clock frequency (DTA-102 only)
GetFailsafe-Config	Get current failsafe configuration
GetFailsafeAlive	Get status of failsafe relais
GetFifoLoad	Get current load of the Transmit FIFO
GetFlags	Get status flags
GetMaxFifoSize	Get maximum size of output FIFO
GetModControl	Get modulation parameters
GetOutputLevel	Get current output level in dBm
GetRfControl	Get RF parameters
GetTargetId	Get target-adaptor identifier (DTA-102 only)
GetTransmitByteCount	Get a sample from the free running 32-bit transmitted number of bytes counter
GetTsRateBps	Get Transport-Stream rate
GetTxControl	Get transmit-control state (Idle/Hold/Send)
GetTxMode	Get transmit mode (packet size, packet-stuffing mode)
ReadLoopBack-Data	Read data from loop-back channel
Reset	Reset output channel
SetFailsafeAlive	Toggle failsafe watchdog timer
SetFailsafeConfig	Set failsafe configuration parameters
SetFifoSize	Set size of Transmit FIFO
SetFifoSizeMax	Set size of Transmit FIFO to its maximum value
SetFifoSize	Set size of Transmit FIFO to a specified value
SetIpPars	Set IP parameters

SetLoopBack-Mode	Set loop-back mode
SetModControl	Set modulation parameters
SetOutputLevel	Set output level in dBm
SetPower	Control power for target adapter (DTA-102)
SetRfControl	Set RF parameters
SetRfMode	Set special modes for devices with on-board RF up-converter
SetSNR	Set noise generation mode and SNR
SetTsRateBps	Set Transport-Stream rate
SetTxControl	Set transmit-control state (Idle/Hold/Send)
SetTxMode	Set transmit mode (packet size, ...) and packet stuffing on/off
Write	Write data to the output channel
WriteDirect	Write data directly to the output channel without using DMA

Table 5. **DTAPI** – **DtLoop** Functions

API Function	Description
AttachToInput	Attach to an DtInpChannel object
AttachToOutput	Attach to an DtOutpChannel object
Detach	Detach from the associated DtInpChannel and DtOutpChannel objects
DetachFromInput	Detach from the associated DtInpChannel object
DetachFrom-Output	Detach from the associated DtOutpChannel object
IsStarted	Has looping started
SetSuffingMode	Set the NULL-packet stuffing parameters
Start	Start/Stop looping from input-to-output

Table 6. **DTAPI** – **DtSdi** Functions

API Function	Description
ConvertFrame	Convert between different SDI data formats

Data Structures

Struct DtConstelPoint

Structure describing a constellation point.

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

Members

m_X

The X-coordinate of the described constellation point.

m_Y

The Y-coordinate of the described constellation point.

Struct DtDeviceDesc

Structure describing a DekTec device.

```

Struct DtDeviceDesc {
    int    m_Category;           // Device category
    __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_DeviceId;          // Device ID
    int    m_VendorId;          // Vendor ID
    int    m_SubsystemId;       // Subsystem
    int    m_SubVendorId;       // Subsystem Vendor ID
    int    m_NumHwFuncs;        // #Hardware funtions hosted by device
    int    m_FirmwareVersion;   // Firmware version
    int    m_FirmwareVariant;   // Firmware variant
    int    m_NumDtInpChan;      // Number of input channels
    int    m_NumDtOutpChan;     // Number of output channels
    int    m_NumPorts;          // Number of physical ports
    int    m_Ip[4];             // IP address
};

```

Members

m_Category

Code indicating the device category.

Value	Meaning
DTAPI_CAT_PCI	PCI Bus device
DTAPI_CAT_USB	USB device
DTAPI_CAT_IP	IP connected device

m_Serial

The serial number that uniquely identifies a DekTec device.

m_PciBusNumber, m_SlotNumber

If the device belongs to category **DTAPI_CAT_PCI**, these integers identify the PCI bus and slot number in which the PCI card is installed. If the device belongs to another category, the values of these members are undefined.

m_UsbAddress

If the device belongs to category **DTAPI_CAT_USB**, this number identifies the USB address of the device. If the device belongs to another category, the value of this member is undefined.

m_TypeNumber

This integer corresponds to the number in the device's type string, e.g. 100 for the DTA-100.

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 configuration-space registers.

m_NumHwFuncs

Number of hardware functions hosted by the device.

m_FirmwareVersion

Version number of the firmware loaded on the device.

m_FirmwareVariant

Variant of the firmware loaded on the device. Some devices may support multiple variants of the firmware each with different functionality.

m_NumDtInpChan

Number of input channels available on the device.
IP ports count as 1 input channel and 1 output channel.

m_NumDtOutpChan

Number of output channels available on the device.
IP ports count as 1 input channel and 1 output channel.

m_NumPorts

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

m_Ip

If the device belongs to category **DTAPI_CAT_IP**, this member identifies the IP address of the device. If the device belongs to another category, the value of this member is undefined.

Struct DtHwFuncDesc

Structure describing a hardware function.

```

Struct DtHwFuncDesc {
    DtDeviceDesc m_DvcDesc;    // Device descriptor
    int m_ChanType;            // Channel type (OR-able)
    int m_StreamType;          // Stream type
    int m_Flags;                // Capability flags (OR-able)
    int m_IndexOnDvc;          // Relative index for this function
    int m_Port;                // Physical port number
    int m_Ip[4];               // IP address
    int m_MacAddr[6];          // MAC address
};

```

Members

m_DvcDesc

Structure describing the device that hosts this hardware function.

m_ChanType

This member variable identifies the channel type of the hardware function. Channel types may be OR-ed together, for TS-over-IP channels both **DTAPI_CHAN_INPUT** and **DTAPI_CHAN_OUTPUT** are set.

Value	Channel Object	Meaning
DTAPI_CHAN_INPUT	DtInpChannel	Input channel
DTAPI_CHAN_OUTPUT	DtOutpChannel	Output channel
DTAPI_CHAN_DBLBUF	DtOutpChannel	Double-buffered mode
DTAPI_CHAN_LOOPTHR	DtOutpChannel	Loop-through mode

m_StreamType

This member variable describes the type of stream type supported by the hardware function.

Value	Meaning
DTAPI_ASI_SDI	DVB-ASI and/or SDI Member variable <i>m_Flags</i> indicates whether the TS interface is capable of ASI, SDI or both
DTAPI_TS_MOD	Modulated signal Member variable <i>m_Flags</i> provides details about the modulation standards that are supported
DTAPI_TS_OVER_IP	TS-over-IP
DTAPI_TS_SPI	DVB-SPI

m_Flags

Extra flags that provide further information about the hardware function.

If *m_StreamType* is **DTAPI_ASI_SDI**, the following OR-able flags are supported:

Value	Meaning
DTAPI_CAP_ASI	Interface can carry DVB-ASI
DTAPI_CAP_BIDIR	Port is bi-directional
DTAPI_CAP_DBLBUF	Port can act as a doubly-buffered copy of another port
DTAPI_CAP_FAILSAFE	Output that can fail-over from an input, for high-availability
DTAPI_CAP_LOOPTHR	Port can act as a loop-through copy of another port
DTAPI_CAP_SDI	Interface can carry SDI (Serial Digital Interface)
DTAPI_CAP_SDITIME	Support for time-stamping of SDI frames
DTAPI_CAP_TRPMODE	Input supports transparent-packet mode

If *m_StreamType* is **DTAPI_TS_MOD**, the following OR-able flags specify the type(s) of modulation that are supported:

Value	Meaning
<i>General capabilities</i>	
DTAPI_CAP_DIGIQ	Port has a digital IQ output
DTAPI_CAP_IF	Port has an IF output
DTAPI_CAP_SHARED	Port supports antenna-sharing mode
DTAPI_CAP_DIVERSITY	Port supports DVB-T diversity mode
<i>Modulation capabilities</i>	
DTAPI_CAP_ATSC	ATSC (VSB)
DTAPI_CAP_CM	Channel Modelling
DTAPI_CAP_DTMB	DTMB
DTAPI_CAP_DVBS	DVB-S
DTAPI_CAP_DVBS2	DVB-S.2
DTAPI_CAP_DVBT	DVB-T, includes DVB-H
DTAPI_CAP_ISDBT	ISDB-T
DTAPI_CAP_QAM_A	QAM, ITU-T J.83 Annex A (DVB-C)
DTAPI_CAP_QAM_B	QAM, ITU-T J.83 Annex B (US)
DTAPI_CAP_QAM_C	QAM, ITU-T J.83 Annex C (Japan)

Upconverter / downconverter capabilities	
DTAPI_CAP_LBAND	L-Band 950 – 2150 MHz
DTAPI_CAP_UHF	UHF Band 400 – 862 MHz
DTAPI_CAP_VHF	VHF Band 47 – 470 MHz
DTAPI_CAP_ADJLVL	Port supports an adjustable output level
DTAPI_CAP_IF_ADC	Port supports an onboard ADC

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 (the top-most port) has port #4;
- Doubly-buffered outputs like on the DTA-140 count as a single port.

Please refer to Section 5 for an overview of port numbers per DekTec device.

m_Ip

IP address of the hardware function. This field is only valid for functions for which *m_StreamType* is **DTAPI_TS_OVER_IP**.

m_Mac

MAC address of the hardware function. This field is only valid for functions for which *m_StreamType* is **DTAPI_TS_OVER_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 (capability **DTAPI_CAP_BIDIR** is set) is either **DTAPI_TS_INPUT** or **DTAPI_TS_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 DtIsdbtLayerPars

Structure describing ISDB-T modulation for one hierarchical layer. This structure is used in class `DtIsdbtPars`, in an array of three structs 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. The sum of *m_NumSegment* must be 13.

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 **BTYPE_TV** for 1-segment modulation.

Struct DtapiHwFunc (Obsolete)

Structure that describes a hardware function on a PCI card.

```

Struct DtapiHwFunc {
    int    m_nPciBusNumber;        // PCI-bus number
    int    m_nSlotNumber;         // PCI-slot number
    int    m_nTypeNumber;         // PCI-card type number
    int    m_nDeviceId;           // Device ID of PCI card
    int    m_nVendorId;           // Vendor ID of PCI card
    int    m_nSubsystemId;        // Subsystem ID of PCI card
    int    m_nSubVendorId;        // Subsystem Vendor ID of PCI card
    int    m_nHwFuncType;         // Hardware-function type
    int    m_nHwFuncFlags;        // Hardware-function flags
    int    m_nIndexOnCard;        // Index of hardware function
};

```

Members

m_nPciBusNumber, m_nSlotNumber

Integers identifying the PCI-bus and the slot number in which the PCI card hosting the hardware function is plugged.

m_nTypeNumber

This integer corresponds to the number in the PCI card's type string, e.g. 100 for the DTA-100.

m_nDeviceId, m_nVendorId, m_nSubsystemId, m_nSubVendorId

Device ID, Vendor ID, Subsystem ID and Subsystem Vendor ID. Identification information of the PCI card read from its PCI configuration-space registers.

m_nHwFuncType

Identifies the hardware function according to the table below.

Value	Meaning
DTAPI_TS_INPUT	Transport-Stream input.
DTAPI_TS_OUTPUT	Transport-Stream output.

m_nHwFuncFlags

Hardware-function dependent flags that provide further (static) information about the hardware function. Not used (yet) for any of the current DekTec boards.

m_nIndexOnCard

This number identifies a specific hardware function when the PCI card contains multiple instantiations of the same hardware function. If the function occurs only once, *m_nIndexOnCard* = 0. If the card contains a function twice, the indices are 0 and 1; etc.

Remarks

This structure was targeted at PCI card. It has been superseded by **DtHwFuncDesc**, which supports the more generic notion of a (digital-video) "device".

Struct DtRawIpHeader

Structure placed in front of all IP Packets when **DTAPI_RXMODE_IPRAW** mode is used (refer to **DtInpChannel::SetRxMode** on page 118).

```
Struct DtRawIpHeader {  
    unsigned short m_Tag;           // 0x44A0h = 'D'160  
    unsigned short m_Length;       // IP Packet Length  
    unsigned int m_TimeStamp;      // IP Packet arrival/transmit time  
};
```

Members

m_Tag

Marks the beginning of a **DtRawIpHeader** structure. The value of this field is fixed to: 0x44A0.

m_Length

Indicates the number of bytes (i.e. size of the IP packet) following directly after the **DtRawIpHeader** structure.

m_TimeStamp

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

Struct DtTslpPars

Structure for storing parameters related to the transmission of Transport Streams over IP.

The parameters in this structure are used when starting transmission of a Transport Stream over IP to a unicast or multicast destination (**Tx**), and when starting reception of a Transport Stream (**Rx**). This structure is also used for reading back parameters.

```

Struct DtTslpPars {
    u_char  m_Ip[4];           // IP address
    u_short m_Port;           // Port number
    u_char  m_SrcFltIp[4];    // Source filter: IP address
    u_short m_SrcFltPort;     // Source filter: port number
    int      m_MulticastTtl;    // TTL for multicast Tx
    int      m_NumTpPerIp;     // #TPs per IP packet
    int      m_Protocol;       // Protocol: UDP/RTP
    int      m_DiffServ;       // Differentiated Services
    int      m_FecMode;        // Error correction mode
    int      m_FecNumRows;     // 'D' = #rows in FEC matrix
    int      m_FecNumCols;     // 'L' = #columns in FEC matrix
    int      m_Flags;          // Optional controls/status flags
};

```

Members (Rx)

m_Ip

Main IP Address. IP address from which too receive IP packets. If the IP address is in the multi-cast range, the DTAPI automatically joins and drops membership of the multicast group.

m_Port

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

m_SrcFltIp

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

m_SrcFltPort

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

m_MulticastTtl

Not used.

m_NumTpPerTp

Not used.

Read back: Number of Transport Packets (TPs) stored in one IP packet in the incoming stream.

m_Protocol

Protocol expected for encapsulation of Transport Packets.

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

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

m_DiffServ
Not used.

m_FecMode
Error-correction mode.

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

m_FecNumRows, *m_FecNumCols*
Not used.

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

Members (Tx)

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

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

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

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

m_Protocol
Protocol expected for encapsulation of Transport Packets.

Value	Meaning
DTAPI_PROTO_UDP	UDP
DTAPI_PROTO_RTP	RTP

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

m_FecMode

Error-correction mode.

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

m_FecNumRows, m_FecNumCols

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

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

m_Flags

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

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);
```

Parameters**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 driver(s) needs to be upgraded.
DTAPI_E_NO_DEVICE	Device-driver version cannot be queried because no DekTec device is installed in the system.

Remarks

::DtapiGetDeviceDriverVersion

Get device-driver version information. Two overloaded variants of this function are provided. The first variant is compatible with DTAPI v1.3.x.x and returns the version of the PCI-card driver (*Dta1xx*). The second variant includes a parameter to specify the device-driver category, either PCI (device driver *Dta1xx*) or USB (device driver *Dtu2xx*).

```
DTAPI_RESULT  ::DtapiGetDeviceDriverVersion (
    [out] int&  DriverVersionMajor,
    [out] int&  DriverVersionMinor,
    [out] int&  DriverVersionBugFix,
    [out] int&  DriverVersionBuild
);
DTAPI_RESULT  ::DtapiGetDeviceDriverVersion (
    [in] int  DvcCategory,          // Device-driver category (PCI/USB)
    [out] int&  DriverVersionMajor,
    [out] int&  DriverVersionMinor,
    [out] int&  DriverVersionBugFix,
    [out] int&  DriverVersionBuild
);
```

Parameters

DvcCategory

Parameter specifying the device category:

Value	Meaning
DTAPI_CAT_PCI	PCI-bus device; Query version of <i>Dta1xx</i> device driver.
DTAPI_CAT_USB	USB device; Query version of <i>Dtu1xx</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 is installed in the system.

Remarks

::DtapiGetVersion

Get version of the **DTAPI** library. Two overloaded variants are defined. The first function prototype, which is defined for backward compatibility, returns the major and minor version number. The second variant also provides bug fix and build number.

```
DTAPI_RESULT ::DtapiGetVersion (
    [out] int& LibVersionMajor,
    [out] int& LibVersionMinor
);

DTAPI_RESULT ::DtapiGetVersion (
    [out] int& LibVersionMajor,
    [out] int& LibVersionMinor,
    [out] int& LibVersionBugFix,
    [out] int& LibVersionBuild
);
```

Parameters

LibVersionMajor

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

LibVersionMinor

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

LibVersionBugFix

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

LibVersionBuild

Build number.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Version numbers have been retrieved successfully.

Remarks

This function always succeeds.

The device drivers have their own version number(s), independent from the **DTAPI** library version.

::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   InclIpDvcs=false    // Include DekTec IP devices
);
```

Parameters

NumEntries

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

NumEntriesResult

Output parameter that receives the number of devices found and described in *pDvcDesc*. The value of this parameter can be greater than *NumEntries* (when **DtapiDeviceScan** returns **DTAPI_E_BUF_TOO_SMALL**).

DvcDescArr

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

InclIpDvcs

Include scan for DekTec IP devices in the network.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Scan has completed successfully and the <i>pDvcDesc</i> array was large enough to contain all device descriptions.
DTAPI_E_BUF_TOO_SMALL	The number of device-description entries in <i>pDvcDesc</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.

In case *InclIpDvcs* is set to **true**, the **DtapiDeviceScan** also scans the network for DekTec IP devices and therefore it takes some 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 **DtDvcDesc** entries, allocate a new array with the number of entries returned in *NumEntriesResult*, and call **DtapiDeviceScan** again.

::DtapiDtDeviceDesc2String

Creates a descriptive string, e.g. "DTA-100 in Slot 5" for a device.

```
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
);
```

Parameters

pDvcDesc

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

StringType

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

Value	Example	Meaning
TYPE_NMB	"DTA-100"	Device type number
TYPE_AND_LOC	"DTA-102 in Slot 5"	Device type number and location

pString

Pointer to the buffer that receives the descriptive string

StringLength

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

Results

DTAPI_RESULT	Meaning
DTAPI_OK	Successfully created a string
DTAPI_E_INVALID_BUF	An invalid buffer pointer is supplied for <i>pDvcDesc</i> or <i>pString</i>

::DtapiDtHwFuncDesc2String

Creates a descriptive string, e.g. "DTA-100 in Slot 5" or "DVB-ASI", for a hardware function based on a DtHwFuncDesc structure.

```
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
);
```

Parameters

pHwFunc

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

StringType

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

Value	Example	Meaning
TYPE_NMB	"DTA-100"	Device type number
TYPE_AND_PORT	"DTA-124 port 1"	Device type number and port number
TYPE_AND_LOC	"DTA-102 in Slot 5"	Device type number and location
ITF_TYPE	"DVB-ASI"	Physical interface type
ITF_TYPE_SHORT	"ASI"	Physical interface type – short descriptive string

pString

Pointer to the buffer that receives the descriptive string

StringLength

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

Results

DTAPI_RESULT	Meaning
DTAPI_OK	Successfully created a string
DTAPI_E_INVALID_BUF	An invalid buffer pointer is supplied for <i>pHwFunc</i> or <i>pString</i>

::DtapiDvbT2ValidateParams

Validate DVB-T2 modulation parameters.

```
DTAPI_RESULT ::DtapiDvbT2ValidateParams (
    [in] int  ParXtra0,          // Extra modulation parameter #0
    [in] int  ParXtra1,          // Extra modulation parameter #1
    [in] int  ParXtra2          // Extra modulation parameter #2
);
```

Parameters

ParXtra0, ParXtra1, ParXtra2

Extra modulation parameters. See `DtOutputChannel::SetModcontrol`, modulation mode DVB-T2.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Parameters are valid
DTAPI_E_FRAME_INTERVAL	Frame interval must divide number of T2 frames
DTAPI_E_INVALID_BWT_EXT	Invalid bandwidth extension
DTAPI_E_INVALID_FFTMODE	Invalid FFT mode
DTAPI_E_INVALID_GUARD	Invalid guard interval
DTAPI_E_INVALID_NUMDTSYM	Invalid number of data symbols
DTAPI_E_INVALID_NUMT2FRM	Invalid number of T2 frames
DTAPI_E_INVALID_TIME_IL	Invalid time interleaver length
DTAPI_E_NUM_PLP	Too many PLPs (i.e. L1 data too large)
DTAPI_E_PLP_NUM_BLOCKS	Invalid number of PLP blocks (not enough bandwidth)

Remarks

::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    InclIpDvcs=false    // Include functions on IP devices
    devices
);
```

Parameters

NumEntries

Specifies the size, in number of **DtHwFuncDesc** entries, of the caller-supplied *pHwFuncs* array. Specifying zero is not allowed.

NumEntriesResult

Output parameter that receives the number of hardware functions found and described in *pHwFuncs*. The value of this parameter 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. Do not supply a NULL pointer.

InclIpDvcs

Include scan for functions hosted by DekTec IP devices in the network.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Scan has completed successfully and the <i>pHwFuncs</i> array was large enough to contain all function descriptions.
DTAPI_E_BUF_TOO_SMALL	The number of function-description entries in <i>pHwFuncs</i> is too small. The number of entries required is returned in <i>NumEntriesResult</i> .

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 host multiple hardware functions.

In case *InclIpDvcs* is set to **true**, the **DtapiHwFuncScan** also scans the network for DekTec IP devices and the result includes the hardware functions hosted DekTec IP devices. In this case the **DtapiHwFuncScan** takes some 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 cur-

rent array of **DtHwFuncDesc** entries, allocate a new array with the number of entries returned in *NumEntriesResult*, and call **DtapiHwFuncScan** 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

Initialises the `m_DstIp` and `m_SrcIp` members of a `DtTsIpPars` structure.

```
DTAPI_RESULT ::DtapiInitDtTsIpParsFromIpString (
    [out] DtTsIpPars&   TsIpPars,          // TsIpPars structure to initialise
    [in] const char*    pDstIp,            // String with destination IP
    [in] const char*    pSrcIp            // String with source IP
);

DTAPI_RESULT ::DtapiInitDtTsIpParsFromIpString (
    [out] DtTsIpPars&   TsIpPars,          // TsIpPars structure to initialise
    [in] const wchar_t* pDstIp,            // String with destination IP
    [in] const wchar_t* pSrcIp            // String with source IP
);
```

Parameters

TsIpPars

Ts-IP-Parameter structure to initialise.

pDstIp

Pointer to a string that holds the IP address (e.g. "127.0.0.1") to be used as destination IP address (`m_DstIp`). 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 (`m_SrcIp`). 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 initialises the `m_DstIp` and `m_SrcIp` members, it will leave the other members untouched.

::DtapiPciScan (Obsolete)

Scan hardware functions available on the DTA-xxx cards in the current system.

```
DTAPI_RESULT ::DtapiPciScan (
    [in] int    NumFuncEntries,           // #Function entries in pHwFuncs
    [out] int& NumFuncEntriesResult,     // #Functions found or #entries required
    [in] DtapiHwFunc* pHwFuncs         // Hardware-function descriptions
);
```

Parameters

NumFuncEntries

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

NumFuncEntriesResult

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

pHwFuncs

Points to a caller-supplied array of **DtapiHwFunc** entries to receive descriptions of the hardware functions.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Scan has completed successfully and the <i>pHwFuncs</i> array was large enough to contain all function descriptions.
DTAPI_E_BUF_TOO_SMALL	The number of function-description entries in <i>pHwFuncs</i> is too small. The number of entries required is returned in <i>NumFuncEntriesResult</i> .

Remarks

This function still exists for backward compatibility. Use **DtDeviceScan** in new software.

DtapiPciScan scans the PCI bus in the current system and lists the hardware functions available on the detected DTA-xxx PCI cards. Each card may contain multiple hardware functions.

This function may have to be called two times. The first time, *NumFuncEntries* 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 **DtapiHwFunc** entries, allocate a new array with the number of entries returned in *NumFuncEntriesResult*, and call **DtapiPciScan** again.

::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
);
```

Parameters

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 160 for more details about these parameters.

TsRate

Transport-Stream rate in bps.

Result

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

Remarks

::DtapiModPars2TsRate

Compute Transport-Stream rate from modulation parameters.

```
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
);
```

Parameters

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 160 for more details about these parameters.

SymRate

Symbol rate in baud. This parameter 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 <code>DtOutpChannel::SetModControl</code>

Remarks

::DtapiResult2Str

Convert DTAPI_RESULT value to a string.

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

Parameters

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.

DtDevice**DtDevice::AttachToIpAddr**

Attach device object to the device hardware, based on the IP address of the device.

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

Parameters

Ip

IP address of the DekTec device to which the device object is to be attached.

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	The device with the IP address could not be found

Remarks

AttachToIpAddr is non-intrusive. No initialisation 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
);
```

Parameters

Serial

Serial number of the DekTec device to which the device object is to be attached.

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	The device with the specified serial number could not be found
DTAPI_E_DRIVER_INCOMP	Version of device driver is incompatible with the DTAPI version, device driver needs to be upgraded

Remarks

AttachToSerial is non-intrusive. No initialisation 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
);
```

Parameters

PciBusNumber, SlotNumber

PCI-bus number and slot number of the DekTec device to which the device object is to be attached.

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 initialisation 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,          // Which device to look for
    [in] int   DeviceNo=0          // Relative device number
);
```

Parameters

TypeNumber

Integer value representing the type number of the device to which the device object is to be attached. The integer corresponds to the number in the hardware's type string, e.g. 100 for the DTA-100 or 225 for DTU-225.

DeviceNo

If the system contains multiple devices of the same type, this 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_INTERNAL	Internal DTAPI error
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 initialisation actions are performed.

DtDevice::Detach

Detach device object from device hardware.

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

Parameters

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

DtDevice::GetDescriptor

Get device descriptor.

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

Parameters

DvcDesc

Output parameter that receives the device descriptor.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Version numbers have been retrieved successfully
DTAPI_E_NOT_ATTACHED	Device object is not attached to device hardware

Remarks

DtDevice::GetDeviceDriverVersion

Get device-driver version information. Two overloaded variants of this function are provided. The first function prototype, which is defined for backward compatibility, returns the major and minor version number. The second version also provides bug fix and build number.

```
DTAPI_RESULT DtDevice::GetDeviceDriverVersion (
    [out] int&   DriverVersionMajor,
    [out] int&   DriverVersionMinor
);

DTAPI_RESULT DtDevice::GetDeviceDriverVersion (
    [out] int&   DriverVersionMajor,
    [out] int&   DriverVersionMinor,
    [out] int&   DriverVersionBugFix,
    [out] int&   DriverVersionBuild
);
```

Parameters

DriverVersionMajor

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.

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 is fixed, without functional enhancements.

DriverVersionBuild

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 the device.

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

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

Parameters

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  
);
```

Parameters

FirmwareVersion

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

Result

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

Remarks

DtDevice::GetIoConfig

Get the channel type of a physical port.

```
DTAPI_RESULT DtDevice::GetIoConfig (
    [in] int Port,           // Physical port
    [out] int& IoConfig      // DTAPI_IOCONFIG_XXX
);

DTAPI_RESULT DtDevice::GetIoConfig (
    [in] int Port,           // Physical port
    [out] int& IoConfig      // DTAPI_IOCONFIG_XXX
    [out] int& ParXtra       // Optional extra IO config option
);
```

Parameters

Port

Physical port number.

IoConfig

Port configuration, see `DtDevice::SetIoConfig`

If the port doesn't support I/O configuration then *Config* is set to `DTAPI_IOCONFIG_NOTSUP`.

ParXtra

Optional extra port configuration parameter, see `DtDevice::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

Remarks

DtDevice::GetVcxoState

Get the state of the onboard Vcxo.

```
DTAPI_RESULT DtDevice::GetVcxoState (
    [out] bool&   Enable
    [out] int&    Lock
    [out] int&    VcxoClkFreqHz
);
```

Parameters

Enable

Indicates 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::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
);
```

Parameters

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 \geq 4)
DTA-122	27.0MHz (FW Version \geq 4)
DTA-124	40.5MHz (FW Version 0) / 54MHz (FW Version \geq 1)
DTA-140	40.5MHz (FW Version \geq 1)
DTA-145	54.0MHz

DTA-160	54.0MHz
DTA-545	40.5Mhz (FW Version 0) / 54MHz (FW Version \geq 1)
DTA-2135	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 onboard reference-clock counter via the **DtDevice::GetRefClkFreq** method.

DtDevice::GetRefClkFreq

Get the frequency of the onboard reference clock.

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

Parameters

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 onboard reference-clock counter is used for assigning arrival time-stamps 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 time-stamps.

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
);
```

Parameters

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 bus⁴. 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.

⁴ USB-2 buses can operate in full-speed mode for backward compatibility reasons (support for USB-1 devices)

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
);
```

Parameters

NumEntries

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

NumEntriesResult

Output parameter that receives the number of hardware functions found and described in *pHwFuncs*. The value of this parameter 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.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Scan has completed successfully and the <i>pHwFuncs</i> array was large enough to contain all function descriptions.
DTAPI_E_BUF_TOO_SMALL	The number of function-description entries in <i>pHwFuncs</i> is too small. The number of entries required is returned in <i>NumEntriesResult</i> .

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.

DtDevice::I2Cread

Read data from the I2C bus.

```
DTAPI_RESULT DtDevice::I2Cread (
    [in] int    DvcAddr,           // I2C device address
    [out] char* pBuffer,          // Buffer for receiving the data
    [in] int    NumBytesToRead,   // #Bytes to read
);
```

Parameters

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	Device 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 onboard I2C bus

Remarks

The I2Cread method is intended for direct low-level access to the onboard I2C resources.

DtDevice::I2Cwrite

Write data to the I2C bus.

```
DTAPI_RESULT DtDevice::I2Cwrite (
    [in] int    DvcAddr,          // I2C device address
    [in] char*  pBuffer,         // Buffer with bytes to write
    [in] int    NumBytesToWrite, // #Bytes to write
);
```

Parameters

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	Device 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 onboard I2C bus.

Remarks

The I2C Write method is intended for direct low-level access to the onboard I2C resources.

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
);
```

Parameters

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 the device.

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

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

Parameters

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:

- Set the direction of a bi-directional ASI/SDI port;
- Set the antenna configuration of a receiver port;
- Select whether an ASI/SDI output is an independent output, a buffered copy of another port or a buffered loop-through of another port;
- Select whether this port has to operate as an SDI genlock input.

```
DTAPI_RESULT DtDevice::SetIoConfig (
    [in] int    Port,           // Physical port
    [in] int    IoConfig       // DTAPI_IOCONFIG_XXX
    [in] int    ParXtra=-1     // Optional extra I/O config parameter
);
```

Parameters

Port

Physical port number.

IoConfig

Port configuration:

Value	Meaning
INPUT CONFIGURATIONS	
DTAPI_IOCONFIG_DIVERSITY	Configure port to operate in diversity mode
DTAPI_IOCONFIG_INPUT	Configure port as an ASI/SDI input
DTAPI_IOCONFIG_GENLOCK	Configure port as an SDI genlock input
DTAPI_IOCONFIG_SHARED	Configure port to use a shared antenna input

OUTPUT CONFIGURATIONS	
DTAPI_IOCONFIG_OUTPUT	Configure port as an ASI/SDI output DTA-105, port #2: Configure port as an independent output
DTAPI_IOCONFIG_GENLOCKED	Configure port as a genlocked SDI output. If configured as genlocked SDI output, the port will lock the SDI output timing to the genlock input. When an application fails to present video data to the port in time, the port will insert black frames to keep the SDI timing synchronised.
DTAPI_IOCONFIG_FAILSAFE	Configure port as a failsafe port. DTA-145/2145, port #2: If failsafe timeout expires the signal on port #1 will be switched (by a relais) to port #2
DTAPI_IOCONFIG_DBLBUF (Deprecated)	Configure port as a doubly-buffered copy of another port
DTAPI_IOCONFIG_LOOPTHR (Deprecated)	Configure port as loop-through copy of another port
DTAPI_IOCONFIG_PORTCOPY	Configure port as output transmitting a copy of the configured peer port. The peer port may be an input or an output port.

ParXtra

Extra I/O configuration option.

Value	Meaning
WHEN USED FOR INPUT PORTS	
DTAPI_IOCONFIG_INPUT	Not used. Set <i>ParXtra</i> to -1
DTAPI_IOCONFIG_GENLOCK	Specifies expected SDI mode: DTA1XX_GENLOCK_SDI625 for 625-line SDI; DTA1XX_GENLOCK_SDI525 for 525-line SDI.
DTAPI_IOCONFIG_DIVERSITY	Port number to use as diversity 'buddy'. DTA-2135: Port 2 can act as diversity buddy for port 1.
DTAPI_IOCONFIG_SHARED	Specifies from which port number the antenna signal should be shared

WHEN USED FOR OUTPUT PORTS	
DTAPI_IOCONFIG_FAILSAFE DTAPI_IOCONFIG_GENLOCKED DTAPI_IOCONFIG_OUTPUT	Not used. Set <i>ParXtra</i> to -1
DTAPI_IOCONFIG_DBLBUF (Deprecated)	Specifies the source port number
DTAPI_IOCONFIG_LOOPTHR (Deprecated)	Specifies the source port number
DTAPI_IOCONFIG_PORTCOPY	Specifies the source port number from which the signal on this port is “copied”.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Channel type has been set successfully
DTAPI_E_ATTACHED	Cannot change I/O configuration when a channel object is attached to this port
DTAPI_E_DEV_DRIVER	Driver was not able to set the specified IO-configuration
DTAPI_E_INVALID_ARG	Invalid value of <i>ParExtra</i> option (e.g. specified ‘copy source’ does not exists)
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 specified port doesn’t permit I/O configuration

Remarks

The I/O configuration of a port can only be set if the device is attached to a **DtDevice** object. The channel may not be attached to a channel object.

DTAPI_IOCONFIG_GENLOCK cannot be set or changed when one or more ports of the device are in use (DTAPI_E_ATTACHED).

The DTAPI function **SetTxMode** will return **DTAPI_E_INVALID_MODE** when the I/O configuration of an SDI port is set to **DTAPI_IOCONFIG_GENLOCKED** and the requested SDI mode (number of lines) conflicts with the SDI genlock mode.

When an SDI port is opened as an input, the DTAPI function **SetRxMode** will return an error when the requested receive mode (number of lines) conflicts with the SDI genlock mode.

The new I/O configuration is persisted in the registry. The current I/O configuration of a port can be read back with **DtDevice::GetIoConfig**.

Setting a port to double-buffered mode will fail if the port specified in *ParXtra* is not an output. Similarly setting a port to loop-through mode will fail if the port specified in *ParXtra* is not an input.

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*  pKeyword // Keyword identifying VPD item, e.g. "Y0"  
);
```

Parameters

pKeyword

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::VpdRead (
    [in] const char* pKeyword, // Keyword identifying VPD item, e.g. "SN"
    [out] char* pVpdItem      // String read from EEPROM
);
```

Parameters

pKeyword

Null-terminated character string identifying the VPD item to be read. The keyword must consist of either two characters, or it should be the special string "VPDID".

The table below lists standard keywords supported by DekTec devices. Next to these standard keywords, custom VPD keywords can be created with **VpdWrite**.

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

pVpdItem

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

Result

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

Remarks

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

DtDevice::VpdWrite

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

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

Parameters

pKeyword

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)

```
class DtInpChannel;
```

Derived Classes*SdiInpChannel*

Class representing an input channel for receiving SDI.
MPEG-2 TS-specific methods applied to this class will fail.

TsInpChannel

Class representing an input channel for receiving an MPEG-2 Transport Stream.
SDI-specific methods applied to this class will fail.

DtInpChannel::Attach (obsolete)

This function is still supported for backward compatibility reasons. For new code, please use `DtInpChannel::AttachToPort`.

Attach the input-channel object to a hardware function hosted by a device.

```
DTAPI_RESULT DtInpChannel::Attach (  
    [in] DtDevice*  pDtDvc,           // Device object  
    [in] int    InpIndex=0,           // Relative index  
    [in] bool    ProbeOnly=false      // Just check whether channel is in use  
);
```

Parameters

pDtDvc

Pointer to the device object that represents a DekTec device. The device object must have been attached to the device hardware using method `DtDevice::AttachToSerial`, `DtDevice::AttachToSlot` or `DtDevice::AttachToType`.

InpIndex

If the device hosts multiple hardware input functions, this index specifies to which hardware function, and therefore to which physical input, the channel object is to be attached. The first hardware input function has an input-index value of 0, the next function 1, and so on.

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 hardware function
DTAPI_OK_FAILSAFE	Channel object has been attached successfully to the hardware function. However the buddy output port has been configured in failsafe mode, this means that if the failsafe watchdog timer is not reset in time, by the application using the output, we might loose the input signal when the onboard relais switches to failsafe mode. NOTE: this is not an error code; this result value is intended to make the user aware of failsafe mode
DTAPI_E_ATTACHED	Channel object is already attached to a hardware function
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 the hardware function
DTAPI_E_NO_DT_INPUT	No inputs on this device
DTAPI_E_NO_SUCH_INPUT	<i>InpIndex</i> refers to a non-existing hardware function
DTAPI_E_OUT_OF_MEM	TS-over-IP: Receive FIFO cannot be allocated

Remarks

On the DTA-160, this method cannot be used to attach to the TS-over-IP Ethernet port. Please use `DtInpChannel::AttachToPort` instead.

Next to establishing the link between input-channel object and hardware function, **Attach** also performs the following initialisation actions:

- The contents of the Receive FIFO are cleared
- Receive-control state is reset to `DTAPI_RXCTRL_IDLE`
- Loop-back mode is reset to `DTAPI_NOLOOPBACK`
- All *latched* status flags are cleared (refer to `ClearFlags`)

DtInpChannel::AttachToPort

Attach the input-channel object to a specific physical port.

```
DTAPI_RESULT DtInpChannel::AttachToPort (
    [in] DtDevice*  pDtDvc,           // Device object
    [in] int  Port,           // Port number
    [in] bool  ProbeOnly=false       // Just check whether channel is in use
);
```

Parameters

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, on which the top-most Ethernet port is port #4. Please refer to Section 4 for an overview of port numbers.

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. However the buddy output port has been configured in failsafe mode, this means that if the failsafe watchdog timer is not reset in time, by the application using the output, we might loose the input signal when the onboard relais switches to failsafe mode. NOTE: this is not an error code; this result value is intended to make the user 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_INPUT	<i>Port</i> is not an input
DTAPI_E_NO_SUCH_PORT	<i>Port</i> refers to a non-existing port number
DTAPI_E_OUT_OF_MEM	TS-over-IP: Receive FIFO cannot be allocated

Remarks

AttachToPort performs the same initialisation actions as **DtInpChannel::Attach**.

DtInpChannel::ClearFifo

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

```
DTAPI_RESULT DtInpChannel::ClearFifo (
    [in] int  SubCh          // Optional parameter to select sub-channel
);
```

Parameters

SubCh

This parameter specifies which sub-channel to clear. The default value is **DTAPI_SUBCH_MAIN**. Please refer to **DtInpChannel::ReadSubCh** for list of possible sub-channel values.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Receive FIFO has been cleared
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_SUBCH	Invalid sub-channel specified
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)**.

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

The effects of **ClearFifo(DTAPI_SUBCH_ADC)** on the ADC Subchannel are equivalent to **Reset(DTAPI_ADC_FIFO_RESET)**.

Calling **ClearFifo(DTAPI_SUBCH_ADC)** will clear the receive-FIFO-overflow flag for the ADC Fifo (**DTAPI_ADC_FIFO_OVF**) and set the receive-control state for the ADC Fifo to **DTAPI_RXCTRL_IDLE**.

DtInpChannel::ClearFlags

Clear *latched* status flag(s).

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

Parameters

Latched

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

Value	Meaning
DTAPI_RX_FIFO_OVF	See GetFlags
DTAPI_RX_SYNC_ERR	" "
DTAPI_RX_RATE_OVF	" "
DTAPI_RX_TARGET_ERR	" "
DTAPI_ADC_FIFO_OVF	" "

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

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

The latched status flags are automatically reset after the DTAPI-calls: **DtDevice::AttachToSerial**, **DtDevice::AttachToSlot**, **DtDevice::AttachToType** 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
);
```

Parameters

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 the receive-control state 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::GetConstellationPoints

Get a set of constellation points for receiver devices with RF-measurement capabilities, like the DTU-234, DTU-235 and DTA-2135.

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

Parameters

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.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	A valid set of constellation points has been returned
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_INVALID_BUF	The <i>pPoints</i> pointer is invalid

Remarks

DtInpChannel::Equalise

Obsolete.

Turn DVB-ASI cable equalizer on or off (DTA-120 Rev 1 only).

```
DTAPI_RESULT DtInpChannel::Equalise (  
    [in] int EqualiserSetting    // Equaliser on/off  
);
```

Parameters

EqualiserSetting

This parameter specifies the equalization setting, according to the following table.

Value	Meaning
DTAPI_EQUALISER_OFF	No equalization; for short cable lengths (<10m)
DTAPI_EQUALISER_ON	Provide 100m of coaxial cable equalisation

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Equaliser setting has been successfully applied
DTAPI_E_INVALID_MODE	The specified equalisation setting is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

The DTA-120 Rev 2, DTA-124, DTA-140 and DTU-225 contain auto-equalisation circuitry. For these devices, **Equalise** returns **DTAPI_E_INVALID_MODE**.

DtInpChannel::GetDemodControl

Get modulation-control parameters for Transport-Stream input channels with a built-in demodulator.

```
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
);
```

Parameters

ModType

Output parameter that is set to the modulation type. See `SetDemodControl` for a list of applicable values.

ParXtra0, ParXtra1, ParXtra2

Extra modulation parameters. See `SetDemodControl` for a list of applicable values.

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 device does not include a demodulator

Remarks

DtInpChannel::GetDemodStatus

Gets the demodulator Receiver- and FEC lock status, and Modulation Error Rate (MER).

```
DTAPI_RESULT DtInpChannel::GetDemodStatus (
    [out] int&  FLock,           // FEC lock
    [out] int&  RLock,           // Receiver lock
    [out] int&  MER              // Estimated MER in units of 0.1dB
);
```

Parameters

FLock

FEC lock status.

Value	Meaning
DTAPI_DEMOD_FECLOCK_FAIL	The demodulator failed to obtain FEC lock
DTAPI_DEMOD_FECLOCK_OK	The receiver obtained FEC lock

RLock

Receiver lock status.

Value	Meaning
DTAPI_DEMOD_RCVLOCK_FAIL	The demodulator failed to obtain receiver lock
DTAPI_DEMOD_RCVLOCK_OK	The demodulator obtained receiver lock

MER

Estimated modulation error ratio (MER) in units of 0.1dB (i.e. 301 → 301 × 0.1 = 30.1dB).

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 device does not include a demodulator

Remarks

DtInpChannel::GetFecErrorCounters

Get FEC related error counters receiver devices with RF-measurement capabilities, like the DTU-234.

```
DTAPI_RESULT DtInpChannel::GetFecErrorCounters (
    [out] int& NumUncorrectedErrors,    // Number of uncorrected errors
    [out] int& NumCorrectedErrors,      // Number of corrected errors
    [out] int& NumBurstErrors,          // Number of burst errors
);
```

Parameters

NumUncorrectedErrors

Number of uncorrected errors (in bits) after FEC.

NumCorrectedErrors

Number of errors corrected (in bits) using FEC.

NumBurstErrors

Number of burst errors detected.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Error counter have been retrieved successfully
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The input channel does not support the FEC error counters

Remarks

The error counters will be cleared after each call to this method i.e. the values returned by this method indicate the number of errors since the method was last called.

DtInpChannel::GetFifoLoad

Get the current load of the input-channel's Receive FIFO.

```
DTAPI_RESULT DtInpChannel::GetFifoLoad (
    [out] int& FifoLoad          // Load of Receive FIFO
    [in]  int  SubCh            // Optional parameter to select sub-channel
);
```

Parameters

FifoLoad

Number of bytes in the Receive FIFO.

SubCh (optional)

This parameter specifies for which sub-channel to get the load. The default value is **DTAPI_SUBCH_MAIN**. Please refer to **DtInpChannel::ReadSubCh** for list of possible sub-channel values.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	FIFO load has been retrieved successfully
DTAPI_E_INVALID_SUBCH	Invalid sub-channel specified
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

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 status flags for the input channel.

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

Parameters

Flags

Output parameter that is set to the current values 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, *Status* is set to zero.

Value	Meaning
DTAPI_RX_FIFO_OVF	A Receive-FIFO overflow condition has occurred. The data in the Receive FIFO could not be transferred fast enough to a system buffer.
DTAPI_ADC_FIFO_OVF	An ADC FIFO overflow condition has occurred. The data in the ADC FIFO could not be transferred fast enough to a system buffer
DTAPI_RX_SYNC_ERR	A synchronisation error has occurred in the packet-synchronisation logic of the input channel. Note: A synchronisation error cannot occur in packet mode DTAPI_RXMODE_RAW.
DTAPI_RX_RATE_OVF	Data is entering the system faster than the input channel can process the data (applies to DTA-122 only). When the input rate on the DTA-122 remains below 150 Mbit/s, this error cannot occur.
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 parameter that is set to the latched values 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**, **DtDevice::AttachToSerial**, **DtDevice::AttachToSlot**, **DtDevice::AttachToType** 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

DtInpChannel::GetMaxFifoSize

Get the maximum size of the Receive FIFO on the device.

```
DTAPI_RESULT DtInpChannel::GetMaxFifoSize (
    [out] int& MaxFifoSize    // Maximum size of FIFO in bytes
    [in]  int  SubCh         // Optional parameter to select sub-channel
);
```

Parameters

MaxFifoSize

Maximum size of the receive FIFO in number of bytes.

SubCh

This parameter specifies for which sub-channel to get the maximum size. The default value is **DTAPI_SUBCH_MAIN**. Please refer to **DtInpChannel::ReadSubCh** for list of possible sub-channel values.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Maximum size of FIFO has been read successfully
DTAPI_E_INVALID_SUBCH	Invalid sub-channel specified
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

The DTA-120, DTA-122, DTA-124, DTA-140, DTU-225 and DTU-234 support a maximum Receive-FIFO size of 8.388.608 bytes. Later revisions of these devices support at least this size, but potentially the maximum FIFO size may become larger.

At the present the FIFO size for an input channel cannot be modified. However future DekTec devices might support the setting the maximum Receive-FIFO size.

DtInpChannel::GetReceiveByteCount

This function is not available yet in the current version of the DTAPI.

Get a sample from the free running 32-bit received-number-of-bytes counter.

```
DTAPI_RESULT DtInpChannel::GetReceiveByteCount (
    [out] int& ByteCount    // Sample of #received-bytes counter
);
```

Parameters

ByteCount

Sample of the 32-bit received number of bytes counter. For SDI 10-bit words are counted.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	A valid sample of the counter was returned
DTAPI_E_NOT_SUPPORTED	The received number of bytes counter is not supported by the current input channel
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This method is supported on the following devices:

Device Type Number	Firmware Version
DTA-120	V4 or higher
DTA-122	V4 or higher
DTA-124	V0 or higher
DTA-140	V0 or higher
DTU-225	V2 or higher

DtInpChannel::GetRfLevel

Get the current RF signal level for the specified bandwidth.

```
DTAPI_RESULT DtInpChannel::GetRfLevel (
    [out] int&   LeveldBmV,      // Signal level in units of 0.1 dBmV
    [in] int    Bandwidth,      // Measurement bandwidth
);
```

Parameters

LeveldBmV

The input signal level expressed in 0.1dBmV units (e.g. 95 → 95×0.1 = 9.5dBmV).

Bandwidth

The bandwidth over which the level should be measured.

Value	Meaning
DTAPI_RFLVL_CHANNEL	Measure the signal level across the complete channel bandwidth
DTAPI_RFLVL_NARROWBAND	Measure the signal level across a narrow bandwidth (see also remarks section)

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The RF level has been retrieved successfully
DTAPI_E_NOT_SUPPORTED	The channel does not support measuring of the signal level
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

The bandwidth used for the narrow-band level measurement depends on the RF receiver device used. The table below lists the used bandwidth per device type.

Device	Narrow Bandwidth
DTU-234	280kHz ⁵
DTU-235	280kHz ⁵
DTA-2135	Not Applicable

⁵ The narrow-band level measurement will be taken 2.695MHz lower than the centre frequency of the current channel. In case of the DTU-234 this allows for the measurement of the ATSC (VSB) pilot level.

DtInpChannel::GetRxControl

Get the current value of the channel's receive-control state.

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

Parameters

RxControl

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

Refer to **SetRxControl** for a description of these states.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Receive-control state 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

Transport Streams

In receive modes **DTAPI_RXMODE_ST188**, **DTAPI_RXMODE_STMP2** and **DTAPI_RXMODE_ST204**, receive-control state 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::GetStatistics

Get statistics information from the input channel.

```
DTAPI_RESULT DtInpChannel::GetStatistics (
    [out] int& ViolCount          // Code-violation counter (ASI inputs only)
);

DTAPI_RESULT DtInpChannel::GetStatistics (
    [in] int Type,
    [out] int& Statistic,
    [in] bool UseCache=false
);

DTAPI_RESULT DtInpChannel::GetStatistics (
    [in] int Type,
    [out] double& Statistic,
    [in] bool UseCache=false
);
```

Parameters

ViolCount

Total number of DVB-ASI code violations since power-up of a DVB-ASI input channel (DTA-120, DTA-124, DTA-140, DTU-225). 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 int value ($2^{31}-1$) has been reached, the counter wraps around to the largest negative int value (-2^{31}).

Note: 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.

Type

Specifies the type of statistic to get:

Statistic Type	Return Type	Description	Supported
DTAPI_STAT_ASI_VIOLCNT	int	Total number of DVB-ASI code violations since power-up of the DVB-ASI input channel	All cards with an ASI input
DTAPI_STAT_BADPCKCNT	int	Count of uncorrected packets since last call. NOTE: returns 0 if receiver not locked.	DTU-235, DTA-2135
DTAPI_STAT_BER_PREVIT	double	Pre-Viterbi Bit error rate.	DTU-235, DTA-2135
DTAPI_STAT_BER_POSTVIT DTAPI_STAT_BER_PRERS	double	Post-Viterbi bit error rate.	DTU-235, DTA-2135
DTAPI_STAT_SNR	Int	Signal-to-Noise ratio of random carrier in dB in units of 0.1dB.	DTU-235, DTA-2135
	double	Signal-to-Noise ratio of random carrier in dB.	
DTAPI_STAT_MER	int	Estimated MER in units of 0.1dB	DTU-235, DTA-2135
	double	Estimated MER in units of dB	

Statistic

The integer or double value of the requested statistic

UseCache

An optional flag to indicate the function may return cached values. Some statistics are delivered in groups by the board to the driver. Setting this flag to **true** may increase performance when consecutive statistics are requested. When the flag is omitted or set to **false**, the driver is forced to issue a separate request for the requested statistic. This will result in the most up-to-date version of the requested statistic, but this may slow down consecutive statistic requests.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Statistics information has been read successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORT	Requested statistic is not supported by the hardware

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**.

```
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)
);
```

Parameters

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 parameter indicates whether a receive clock of sufficient frequency is detected at the SPI input;

For DVB-ASI and SDI input channels, this output parameter 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 parameter 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 parameter 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 parameter 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 parameter 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 parameter *PolarityControl* in the call to `PolarityControl`.

Value	Meaning
<code>DTAPI_ASIINV_NORMAL</code>	Polarity of DVB-ASI input signal is normal (not inverted)
<code>DTAPI_ASIINV_INVERT</code>	Polarity of DVB-ASI signal is inverted
<code>DTAPI_NOT_SUPPORTED</code>	Device does not support this parameter (DTA-122)

Result

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

Remarks

If no input signal is applied, output parameter *PacketSize* is set to `DTAPI_PCKSIZE_INV`.

DtInpChannel::GetTargetId

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

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

Parameters

Present

Output parameter 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 parameter 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

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-adaptor detection circuitry.

DtInpChannel::GetTsRateBps

Get an estimate of the input Transport-Stream rate.

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

Parameters

TsRate

Estimate 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

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 tuner frequency from input devices with an onboard tuner.

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

Parameters

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::LedControl

Take direct control of input-status LED, or let hardware drive the LED.

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

Parameters

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

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**).

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 used at the same time, then **DtDevice::LedControl** takes precedence over **DtInpChannel::LedControl**.

DtInpChannel::PolarityControl

Control the automatic polarity-detection circuitry of a DVB-ASI Transport-Stream input channel.

```
DTAPI_RESULT DtInpChannel::PolarityControl (
    [in] int  PolarityControl // Polarity-control setting
);
```

Parameters

PolarityControl

This parameter controls inversion of the DVB-ASI signal.

Value	Meaning
DTAPI_POLARITY_AUTO	Automatically detect and correct polarity of DVB-ASI signal
DTAPI_POLARITY_NORMAL	'Normal' operation: do not invert 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

Remarks

The DVB-ASI signal is sensitive to signal polarity. Without corrective measures, an inverted DVB-ASI signal – which may be caused by e.g. 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.

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 (must be multiple of 4!)
);

DTAPI_RESULT DtInpChannel::Read (
    [in] char*  pBuffer,          // Buffer to store data
    [in] int    NumBytesToRead    // #Bytes to be read (must be multiple of 4!)
    [in] int    Timeout           // Optional Max. Time to wait for data
);
```

Parameters

pBuffer

Pointer to the buffer into which the data bytes from the input channel will be written.
The pointer must be aligned to a 4-byte address boundary.

NumBytesToRead

Transfer size: Number of bytes to be read from the input channel.
The value of this parameter must be positive and a multiple of four.

Timeout (optional)

Transfer timeout: specifies the maximum time (in ms) too wait for the requested amount of data. This method will fail if the data cannot be read within the specified period.
The value of this parameter must larger than 0 or -1 to specify an infinite timeout. The default value is -1.

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_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
DTAPI_E_INVALID_SUBCH	Invalid sub-channel selected
DTAPI_E_INVALID_TIMEOUT	Invalid timeout period specified

Remarks

Read returns when *NumBytesToRead* bytes have been transferred into the buffer. The transfer size may be greater than the initial number of bytes available in the Receive FIFO. The thread executing **Read** sleeps until sufficient data has entered the Receive FIFO to complete the transfer.

If the transfer size is greater than the initial Receive-FIFO load, and either the input signal disappears or Receive Control is `DTAPI_RXCTRL_IDLE`, then the Read call may sleep for an indefinite period of time (the thread 'hangs'). To avoid this situation, it is recommended to use the overloaded version of `Read` and specify a timeout other than infinite (-1) or alternatively first check the FIFO load and then read an amount of data less than or equal to the FIFO load.

For PCI-card devices, the DTAPI applies DMA to directly transfer the data from PCI card to the buffer in user space. Hereto, the device driver locks the physical pages of the buffer in memory. A scatter/gather list is created and the DMA transfer is initiated.

For USB-devices, the data has to pass the USB device-driver stack. DMA may, or may not, be applied.

DtInpChannel::ReadDirect

This function is for diagnostics purposes only, please use method `Read` for 'ordinary' reading
 PCI cards only: Read data bytes from the input channel using "direct" read-cycles over the PCI bus.

```
DTAPI_RESULT DtInpChannel::ReadDirect (
    [out] char*  pBuffer,          // Buffer to store data
    [in] int    NumBytesToRead,    // Number of bytes to be read
    [out] int&   NumBytesRead      // Number of bytes read
);
```

Parameters

pBuffer

Pointer to the buffer to which the data from the input channel will be written.
 The pointer must be aligned to a 32-bit word boundary.

NumBytesToRead

Size of the read buffer, and as such the maximum number of bytes read from the input channel. The buffer size must be positive and a multiple of four.

NumBytesRead

Number of bytes actually read. *NumBytesRead* is always a multiple of four.

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 four
DTAPI_E_FIFO_EMPTY	The operation timed out because the Receive FIFO is empty
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

Method `ReadDirect` is very similar to `Read`. For PCI cards (DTA-1XX), `ReadDirect` is considerably slower than `Read` because the processor reads the data bytes directly via the PCI bus, instead of using DMA. `ReadDirect` should only be used for special purposes, e.g. for PCI-card validation.

The data buffer can be any buffer in user space that is aligned to a 4-byte boundary.

`ReadDirect` returns when the read buffer is full or when the Receive FIFO is empty.

Unlike `Read`, this function cannot block forever. If the read operation takes too long, error code `DTAPI_E_FIFO_EMPTY` is returned.

For USB devices, `ReadDirect` is equivalent to `Read`.

DtInpChannel::ReadFrame

Reads one 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                // Max. Time to wait for a frame
);
```

Parameters

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 parameter this parameter indicates the size of the frame buffer. The frame buffer should be large enough to receive a complete frame and is must 32-bit aligned. As an output parameter this parameter indicates the number of bytes returned in the frame buffer. NOTE: 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) too wait for a complete frame. This method will fail if a frame cannot be returned within the specified period.

The value of this parameter must larger than 0 or -1 to specify an infinite timeout. The default value is -1 (i.e. 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 to small for receiving a complete frame
DTAPI_E_NOT_SDI_MODE	The channel is not in SDI mode (see SetRxMode page 118)
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
DTAPI_E_INVALID_TIMEOUT	Invalid timeout period specified

Remarks

If an infinite timeout has been specified, this method will block until a complete frame has been received from the hardware.

DtInpChannel::ReadSubCh

Read data bytes from a specific input sub-channel.

```
DTAPI_RESULT DtInpChannel::ReadSubCh (
    [in] char*   pBuffer,           // Buffer to store data
    [in] int     NumBytesToRead     // #Bytes to be read (must be multiple of 4!)
    [in] int     SubCh              // Subchannel to read from
);
```

Parameters

pBuffer

Pointer to the buffer into which the data bytes from the input channel will be written.
The pointer must be aligned to a 4-byte address boundary.

NumBytesToRead

Transfer size: Number of bytes to be read from the input channel.
The value of this parameter must be positive and a multiple of four.

SubCh

This parameter the sub-channel from which to read.

Value	Meaning
DTAPI_SUBCH_MAIN	Main data channel
DTAPI_SUBCH_ADC	ADC sub-channel NOTE: Only supported on DTA-2135

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_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
DTAPI_E_INVALID_SUBCH	Invalid sub-channel selected

Remarks

DtInpChannel::Reset

Reset input channel.

```
DTAPI_RESULT DtInpChannel::Reset (
    [in] int ResetMode
);
```

Parameters

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"> • Data transfers are halted instantaneously • All data pending in the Receive FIFO is discarded • Receive-control state is reset to DTAPI_RXCTRL_IDLE • Receive-FIFO overflow flag is cleared
DTAPI_ADC_FIFO_RESET	Reset (clear) the ADC FIFO: <ul style="list-style-type: none"> • Data transfers are halted instantaneously • All data pending in the ADC Receive FIFO is discarded • ADC Receive-control state is reset to DTAPI_RXCTRL_IDLE • ADC Receive-FIFO overflow flag is cleared
DTAPI_FULL_RESET	Full input-channel reset: <ul style="list-style-type: none"> • All actions for DTAPI_FIFO_RESET, plus: • Synchronisation-error flag (DTAPI_RX_SYNC_ERR) is cleared • State machines in the device hardware are reset

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Input channel has been reset
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_MODE	The value specified for <i>ResetMode</i> is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

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`)
- ADC sample rate (refer to `DtInpChannel::AdcSetSampleRate`)

DtInpChannel::SetAdcSampleRate

Set the channel's ADC sample rate. The ADC sample-rate determines the rate samples are taken from the down converted RF signal.

```
DTAPI_RESULT DtInpChannel::SetAdcSampleRate (
    [in] int SampleRate // ADC Samplerate in Hz
);
```

Parameters

RxMode

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 ⁶
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

- Only the first channel of the DTA-2135 provides access the down converted RF signal
- The immediate frequency (IF) 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 the sampling theory on details how to recover the signal.

⁶ The exact frequency is $27 * 13 / 17 = 20.647059$ Mhz

DtInpChannel::SetAntPower

Turn power to external antenna on/off.

```
DTAPI_RESULT DtInpChannel::SetAntPower (
    [in] int  AntPower          // Antenna Power state
);
```

Parameters

AntPower

Power state according to the table below.

Value	Meaning
DTAPI_POWER_OFF	No power is applied. Connected 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 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 the connector

Remarks

- After **Attach** and after **Reset**, antenna power is turned off.
- When the DTA-2135 operates in diversity mode, the antenna power for both antenna connectors is turned on or off simultaneously, i.e. both connected antenna's are powered from the DTA-2135 antenna connectors or both connected antenna's must be self powered.

DtInpChannel::SetDemodControl

Set modulation-control parameters for input channels with a built-in demodulator.

```
DTAPI_RESULT DtOutpChannel::SetDemodControl (
    [in] int  ModType,           // Modulation type
    [in] int  ParXtra0,         // Extra parameter #0
    [in] int  ParXtra1,         // Extra parameter #1
    [in] int  ParXtra2         // Extra parameter #2
);
```

Parameters

ModType

Modulation type.

Device	ModType	Meaning
DTU-234	DTAPI_MOD_QAM64	64-QAM modulation
	DTAPI_MOD_QAM256	256-QAM modulation
	DTAPI_MOD_VSB8	8-VSB modulation
	DTAPI_MOD_VSB16	16-VSB modulation
DTU-235, DTA-2135	DTAPI_MOD_DVBT	DVB-T modulation

ParXtra0

Extra modulation parameter #0

For VSB mode this parameter is not used and should be set to 0.

ParXtra0 – DTU-234 in QAM mode

Device	ParXtra0	Meaning
DTU-234	DTAPI_MOD_J83_B	J.83 annex B ("American QAM") Channel filter roll-off factor: 12% (64-QAM) or 18% (256-QAM)

ParXtra1

Extra modulation parameter #1

Device	ParXtra0	Meaning
DTU-235, DTA-2135	DTAPI_MOD_DVBT_6MHZ	6 MHz
	DTAPI_MOD_DVBT_7MHZ	7 MHz
	DTAPI_MOD_DVBT_8MHZ	8 MHz

ParXtra2

Extra modulation parameter #2

At the moment this parameter is not used and should be set to 0.

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_J83ANNEX	Invalid value for J.83 annex in <i>ParXtra0</i>
DTAPI_E_INVALID_MODE	Modulation type is incompatible with modulator
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

DtInpChannel::SetIpPars

Set parameters for the reception of a Transport Stream over IP and start listening.

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

Parameters

SetIpPars

New parameter set to be applied. Please refer to the **DtTsIpPars** page for a description of the parameters.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	TS-over-IP parameters have been applied successfully
DTAPI_E_IN_USE	Parameters cannot be changed because the channel is busy. The receive-control state should be switched back to idle first
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

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 the 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 first dropped and a new multicast group is joined if required.

DtInpChannel::SetLoopBackMode

This function is for diagnostics purposes only

Set the input-channel's loop-back mode.

```
DTAPI_RESULT DtInpChannel::SetLoopBackMode (
    [in] int Mode           // Loop-back mode
);
```

Parameters

Mode

Loop-back mode according to the table below.

Value	Meaning
DTAPI_NO_LOOPBACK	Normal operation, no loop-back of data
DTAPI_LOOPBACK_MODE	Loop-back mode. The physical input circuitry is disconnected from the Receive FIFO. Data can be written directly into the Receive FIFO using WriteLoopBackData .

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Loop-back state has been successfully changed
DTAPI_E_INVALID_MODE	The value specified for loop-back mode is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

- After **Attach** and after **Reset**, loop-back mode is turned off.
- DTA-2135 Only. When this function is called on an input-channel with an Receive and an ADC FIFO, both FIFO's are set to loop-back mode.

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
);
```

Parameters

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 the channel's receive-control state.

```
DTAPI_RESULT DtInpChannel::SetRxControl (
    [in] int    RxControl    // Receive-control state
    [in] int    SubCh       // Optional parameter to select sub-channel
);
```

Parameters

RxControl

New receive-control state 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.

SubCh

This parameter specifies for which sub-channel to set the receive-control state. The default value is **DTAPI_SUBCH_MAIN**. Please refer to **DtInpChannel::ReadSubCh** for list of possible sub-channel values.

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_INVALID_SUBCH	Invalid sub-channel specified
DTAPI_E_NO_IP_PARS	For TS-over-IP channels: receive-control state cannot be set to DTAPI_RXCTRL_RCV because TS-over-IP parameters have not been specified
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

If the receive-control state is set to **DTAPI_RXCTRL_RCV**, but the application does not read any data from the Receive FIFO, then the Receive FIFO will quickly overflow.

Receive-control state is initialised to **DTAPI_RXCTRL_IDLE** by calling **DtDevice::AttachToSlot**, **DtDevice::AttachToType**, **DtDevice::AttachtoSerial**, **DtInpChannel::Reset** or **DtInpChannel::ClearFifo**.

DtInpChannel::SetRxMode

Set the channel's receive mode. The receive mode determines the (hardware) processing that is applied to the incoming stream.

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

Parameters

RxMode

Receive mode according to the table below.

Value	Meaning
DTAPI_RXMODE_ST188	188-byte mode Always store 188-byte packets in the Receive FIFO. When the input contains 204-byte packets, the 16 trailing bytes are dropped. Input data without 188- or 204-byte packet structure is dropped.
DTAPI_RXMODE_ST204	204-byte mode Always store 204-byte packets in the Receive FIFO. When the input contains 188-byte packets, 16 zero bytes are appended. Input data without 188- or 204-byte packet structure is dropped.
DTAPI_RXMODE_STMP2	MPEG-2 mode Store 188- or 204-byte packets in the Receive FIFO without modification. Input data without 188- or 204-byte packet structure is dropped.
DTAPI_RXMODE_STRAW	Raw mode No notion of packets. All incoming valid data bytes are stored in the Receive FIFO. NOTE: This mode is incompatible with DTAPI_POLARITY_AUTO! Please refer to the Remarks section.
DTAPI_RXMODE_STTRP	Transparent mode All incoming data bytes are stored in the Receive FIFO. The data is aligned to packet boundaries if valid packets are detected. This format includes a trailer that contains information about the detected packet size, sync status and valid data bytes within the packet. Refer to section 4.4 for details about the format of transparent-mode-packets.

The following modes are valid for SDI capable channels only.

Value	Meaning
<code>DTAPI_RXMODE_SDI_FULL</code>	<i>Full SDI mode</i> Store all SDI data (i.e. complete frames).
<code>DTAPI_RXMODE_SDI_ACTIVE_VIDEO</code>	<i>Active Video SDI mode</i> Store only the active video part of each SDI frame NOTE: This mode should only be used in combination with Huffman compression (i.e. with <code>DTAPI_RXMODE_SDI_HUFFMAN</code> flag)! When using this mode with Huffman compression disabled the format of the data, provided by the input channel, will be undefined.

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

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

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

Value	Meaning
<code>DTAPI_RX_TIMESTAMP</code>	<i>Time-stamped mode</i> Insert a 32-bit time stamp before each packet. The time stamp is a sample of the system clock counter on the device.
<code>DTAPI_RXMODE_SDI</code>	<i>SDI mode</i> Indicates the input channel should operate in SDI mode. If not used the channel operates in ASI mode. NOTE: this flag is already OR-ed into the SDI related receive-mode mention in the table above.
<code>DTAPI_RXMODE_SDI_10B</code>	<i>10-bit SDI samples</i> Indicates 10-bit SDI samples should be provided. If this flag is omitted the SDI data will be delivered as 8-bit samples.
<code>DTAPI_RXMODE_SDI_HUFFMAN</code>	<i>Huffman compression</i> Indicated the SDI frame should be compressed using a Huffman compression algorithm.

The `DTAPI_RX_TIMESTAMP` flag cannot be specified in raw mode (`DTAPI_RXMODE_IPRAW`) or any of the SDI modes.

Please refer to section 4 for more details about the SDI data formats.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Receive mode has been changed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_MODE	The specified receive mode is invalid or incompatible with the input channel
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	Current device is not supported by this function

Remarks

In receive mode **DTAPI_RXMODE_STRAW** ("raw" mode), the input channel does not care about the packet structure of the incoming data: All data bytes are stored in the input buffer. For DVB-ASI input channels, raw mode can only work reliably if *polarity control* is set to **DTAPI_POLARITY_NORMAL** or **DTAPI_POLARITY_INVERT**. If polarity control is set to **DTAPI_POLARITY_AUTO**, disaster may be the result: Automatic polarity detection assumes that the input has a valid packet structure. If such a structure cannot be found, the device tries again with the input signal inverted. In raw mode, such inversion may occur periodically and severely corrupt the input data!

For the DTA-122 DVB-SPI Input Adapter, 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.

Time stamps are stored in Little Endian format: the first byte contains the least significant 8 bits, the fourth byte the most significant 8 bits. On 32-bit platforms that use the Little-Endian convention (a.o. Intel IA-32), the time stamp can be read by code like:

```
unsigned int TimeStamp = *(unsigned int*) PtrInCharBuffer;
```

DtInpChannel::SetTunerFrequency

Set tuner frequency for input devices with an onboard tuner.

```
DTAPI_RESULT DtOutpChannel::SetTunerFrequency (
    [out] int& FreqHz           // Frequency in hertz
);
```

Parameters

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	-
DTA-2135	50,000,000 - 860,000,000 Hz	-

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The tuner frequency has been set successfully
DTAPI_E_INVALID_RATE	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::TuneChannel

(DTU-234 only)

Sets the tuner to the frequency of the specified channel.

```
DTAPI_RESULT DtOutpChannel::TuneChannel (
    [in] int  Band,           // Band
    [in] int  Channel,       // Channel number
    [out] int& FreqHz        // Frequency of the channel in hertz
);
```

Parameters

Band

Frequency Band.

Value	Meaning
DTAPI_BAND_BROADCAST_ONAIR	Broadcast / On-Air
DTAPI_BAND_FCC_CABLE	FCC / Cable
DTAPI_BAND_IRC	IRC
DTAPI_BAND_HRC	HRC

Channel

Channel number.

FreqHz

Frequency of the channel in hertz.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The channel has been set successfully
DTAPI_E_INVALID_ARG	Invalid frequency band
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::WriteLoopBackData

This function is for diagnostics purposes only

Write data bytes to the channel's Receive FIFO via the loop-back channel. The loop-back mode must be **DTAPI_LOOPBACK_MODE**.

```
DTAPI_RESULT DtInpChannel::WriteLoopBackData (
    [in] char*   pBuffer,           // Data written to Receive FIFO
    [in] int     NumBytesToWrite, // Number of bytes to be written
    [in] int     SubCh             // Optional parameter to select sub-channel
);
```

Parameters

pBuffer

Pointer to the buffer that will supply the data written to the Receive FIFO. The size of the buffer must be at least *NumBytesToWrite*.

NumBytesToWrite

Number of bytes to be written to the Receive FIFO. *NumBytesToWrite* must be positive.

SubCh

This parameter specifies to which sub-channel should be written. The default value is **DTAPI_SUBCH_MAIN**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Write operation has been completed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_SIZE	The specified transfer size is negative
DTAPI_E_INVALID_SUBCH	Invalid sub-channel specified
DTAPI_E_NO_LOOPBACK	Channel is not in loop-back mode
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This method can be used for writing a Receive-FIFO memory test.

Data can only be written to the Receive FIFO if the loop-back mode is **DTAPI_LOOPBACK_MODE**. This method does not check for Receive-FIFO overflow.

DtIsdbtPars**DtIsdbtPars**

Class describing parameters for ISDB-T modulation.

```
class DtIsdbtPars {
public:
    bool m_DoMux;           // Hierarchical multiplexing yes/no
    bool m_FilledOut;       // Members have been given a value
    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_ParXtra0;         // Extra parameters
    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_BType

Broadcast type.

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

m_Mode

Transmission mode.

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

m_Guard

Guard-interval length.

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

m_PartialRx

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

m_Emergency

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

m_IipPid

PID value used for multiplexing the IIP packet.

m_LayerPars

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

m_Pid2Layer

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

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

m_LayerOther

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

m_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_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.

m_SampleRate

Sample rate used by hardware to clock out I and Q samples.

Value	Meaning	Number of segments
0	512/63 MHz	1, 3 or 13
1	64/63 MHz	1-segment only (not supported by the hardware yet)
2	128/63 MHz	3-segment only (not supported by the hardware yet)

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
);
```

Parameters

ResultCode

Value	Meaning
DTAPI_ISDBT_OK	ISDB-T parameters are valid
DTAPI_ISDBT_E_NSEGM	Number of segments is not equal to 13
DTAPI_ISDBT_E_PARTIAL	'Partial Reception' is selected but number of segments in layer A is not 1

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 ();
```

Parameters

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  
);
```

Parameters

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

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;
```

Derived Classes*SdiOutpChannel*

Class representing an output channel for transmitting SDI.
MPEG-2 TS-specific methods applied to this class will fail.

TsOutpChannel

Class representing an output channel for transmitting an MPEG-2 Transport Stream.
SDI-specific methods applied to this class will fail.

DtOutpChannel::Attach

This function is still supported for backward compatibility reasons. For new code, please use `DtOutpChannel::AttachToPort`.

Attach the output-channel object to a hardware function hosted by a particular device. Two overloaded variants are defined.

- The first variant attaches to a hardware function of a `DtDevice` object and should be used in new software.
- The second – obsolete – function prototype, which is defined for backward compatibility with DTAPI version 1.3.x.xxx, attaches the output-channel object to a hardware function of a PCI Card object.

```
DTAPI_RESULT DtOutpChannel::Attach (
    [in] DtDevice*  pDtDevice,      // Object representing device
    [in] int  OutpIndex=0,          // Output index
    [in] bool  ProbeOnly=false     // Just check whether channel is in use
);

DTAPI_RESULT DtOutpChannel::Attach (      // Obsolete
    [in] PciCard*  pPciCard,        // Object representing PCI card
    [in] int  OutpIndex=0           // Output index on PCI card
);
```

Parameters

pDtDvc

Pointer to object that represents a DekTec device. The `DtDevice` object must be attached to the device hardware.

OutpIndex

If the device has multiple hardware output functions, this number specifies to which hardware function, and therefore to which physical output, the channel object is attached. The *OutpIndex* of the first hardware output function is 0.

ProbeOnly

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

pPciCard

Pointer to object that represents a PCI-card. The `PciCard` object must be attached to the device hardware.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Channel object has been attached successfully to the hardware function
DTAPI_OK_FAILSAFE	Channel object has been attached successfully to the hardware function. However output port has been configured in failsafe mode, this means the application should call the SetFailsafeAlive method on a regular basis to prevent the failsafe relais to enter fails safe mode. NOTE: this is not an error code; this result value is intended to make the application aware of failsafe mode
DTAPI_E_ATTACHED	The channel object is already attached a hardware function
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_IN_USE	The output is already in use by another channel object
DTAPI_E_NO_DT_OUTPUT	The device does not contain hardware output functions
DTAPI_E_NO_SUCH_OUTPUT	<i>OutpIndex</i> refers to a non-existing output
DTAPI_E_DEVICE DTAPI_E_PCICARD	The DtDevice pointer is not valid or the DtDevice object is not attached to the device hardware

Remarks

On the DTA-160, this method cannot be used to attach to the TS-over-IP Ethernet port. Please use **DtOutpChannel::AttachToPort** instead.

Next to establishing the link with the hardware, **Attach** also performs the following initialisation:

- The contents of the Transmit FIFO are cleared.
- The size of the Transmit FIFO is set to the maximum value.
- Transmit-control state is reset to **DTAPI_TXCTRL_IDLE**.
- Loop-back mode is reset to "no loop-back".
- All *latched* status flags are cleared (refer to **ClearFlags**).
- Enable LVDS output signals (DTA-102 only).

For Transport-Stream outputs, **Attach** does not reset the Transport-Stream rate to zero. This enables switching between applications that use the same output, without causing a dip in the Transport-Stream.

DtOutpChannel::AttachToPort

Attach the output-channel object to a specific physical port.

```
DTAPI_RESULT DtOutpChannel::AttachToPort (
    [in] DtDevice*  pDtDvc,           // Device object
    [in] int  Port,           // Port number
    [in] bool  ProbeOnly=false      // Just check whether channel is in use
);
```

Parameters

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, on which the top-most Ethernet port is port #4. Please refer to Section 4 for an overview of port numbers.

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. However output port has been configured in failsafe mode, this means the application should call the SetFailsafeAlive method on a regular basis to prevent the failsafe relais to enter fails safe mode. NOTE: this is not an error code; this result value 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	<i>Port</i> refers to a non-existing port number
DTAPI_E_OUT_OF_MEM	TS-over-IP: Receive FIFO cannot be allocated

Remarks

AttachToPort performs the same initialisation actions as **DtOutpChannel::Attach**.

DtOutpChannel::ClearFifo

Clear contents of the Transmit FIFO and set transmit-control state to **DTAPI_TXCTRL_IDLE**. Furthermore, clear the output channel's status flags: the transmit-FIFO-underflow flag (**DTAPI_TX_FIFO_UFL**) and the transmit-synchronisation-error flag (**DTAPI_TX_SYNC_ERR**).

```
DTAPI_RESULT DtOutpChannel::ClearFifo (  
    void  
);
```

Parameters

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)**.

ClearFifo does not reset the last part of the hardware pipeline, so that packets are not truncated (unless transmit mode is **DTAPI_TXMODE_RAW**.)

Method **clearFifo** is available on all DekTec output cards, except on DTA-100 firmware versions prior to V3, and on DTA-102 firmware versions prior to V5. On these elder PCI cards, **clearFifo** has the same effect as **Reset(DTAPI_FULL_RESET)**.

DtOutpChannel::ClearFlags

Clear *latched* status flag(s).

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

Parameters

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_FIFO_UFL	See GetFlags
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 calling **AttachToSlot**, **AttachToType** and after **Reset**. A call to **ClearFifo** clears **DTAPI_TX_FIFO_UFL** and **DTAPI_TX_SYNC_ERR**.

DtOutputChannel::Detach

Detach output channel object from a hardware function. Free resources allocated for the output channel, such as DMA buffers.

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

Parameters

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_TRISTATE_OUTPUT	Put all output signals in tri-state (DTA-102 only).
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.
DTAPI_ZERO_OUTPUT	Set the Transport-Stream rate to zero. On the DTA-100/140, only stuffing symbols are generated. On the DTA-102, all output signals are reset to constant '0'.

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

If the client does not specify DTAPI_ZERO_OUTPUT and/or DTAPI_TRISTATE_OUTPUT, and packet stuffing is turned on, the output channel keeps transmitting null packets after detaching.

If the hardware does not support tri-stating of the output, specifying DTAPI_TRISTATE_OUTPUT acts as a no-op. No error result is generated.

In certain circumstances **Detach** may “hang”, e.g. if DTAPI_WAIT_UNTIL_SENT is specified while the FIFO still contains data and the transmit-control state is DTAPI_TXCTRL_IDLE.

DtOutpChannel::GetExtClkFreq

Get an estimate of the frequency of the external clock (DTA-102 only).

```
DTAPI_RESULT DtOutpChannel::GetTsRateBpsExtClk (  
    [out] int& ExtClkFreq    // Measurement of external-clock frequency  
);
```

Parameters

TsRate

Output parameter 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 rate.

DtOutpChannel::GetFailsafeAlive

Get current alive (relais) status.

```
DTAPI_RESULT DtOutpChannel::GetFailsafeAlive (  
[out] bool& Alive  
);
```

Parameters

Alive

Indicates the current alive (relais) status.

If false the failsafe timeout has expired before the user called the **SetFailsafeAlive** method, the relais will be in failsafe mode (i.e. output = input). If true the failsafe time has not expired yet and the relais is not in failsafe mode (i.e. input =input and output=output).

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 as failsafe output
DTAPI_E_NOT_SUPPORTED	The channel is not failsafe capable
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

DtOutpChannel::GetFailsafeConfig

Get failsafe configuration.

```
DTAPI_RESULT DtOutpChannel::GetFailsafeConfig (  
    [out] bool& Enable,           // Failsafe configuration Enabled/Disabled  
    [out] int& Timeout           // Watchdog timeout (in ms)  
);
```

Parameters

Enable

Failsafe configuration has been enabled or disabled (see also **SetFailsafeConfig**).

Timeout

Current failsafe timeout period (in ms)

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 as failsafe output
DTAPI_E_NOT_SUPPORTED	The channel is not failsafe capable
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
);
```

Parameters

FifoLoad

Number of bytes in the Transmit 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

The value retrieved with this method call *approximates* the load of the Transmit FIFO. The actual number of bytes buffered on the card may be up to 2180 bytes higher. *FifoLoad* cannot be less than the Transmit-FIFO load.

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

DtOutpChannel::GetFifoSize

Get the current size of the channel's Transmit FIFO.

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

Parameters

FifoSize

Size of the Transmit FIFO in number of bytes.

Result

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

Remarks

DtOutpChannel::GetFlags

Get status flags of the output channel.

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

Parameters

Status

Output parameter that is set to a value describing 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_DMA_PENDING DTAPI_TX_PENDING	A data transfer is pending. No latched version of this flag is available.
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	An overflow has been detected in hierarchical multiplexing for ISDB-T
DTAPI_TX_READBACK_ERR	The hardware has detected that an output pin is forced to an erroneous signal level, e.g. because of a short-circuit (DTA-102 only).
DTAPI_TX_SYNC_ERR	A Transmit-FIFO synchronisation-error has occurred. The size of one or more packets in the output mode does not match the Transmit Mode. This status flag is not used in Transmit Mode DTAPI_TXMODE_RAW.
DTAPI_TX_TARGET_ERR	The target adapter signals a fault (DTA-102 only).
DTAPI_RX_LINK_ERR	The communication link with the device is broken (DTE-31xx devices only)
DTAPI_RX_DATA_ERR	Data is lost during transfer to the device (DTE-31xx devices only)

Latched

Output parameter 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 cleared explicitly by **ClearFlag**, or implicitly by one of the following DTAPI-calls: **ClearFifo**, **AttachToSlot**, **AttachToType** or **Reset**.

The following flag is not latched: **DTAPI_TX_PENDING**; the corresponding bit position is always set to '0'.

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

The **DTAPI_TX_PENDING** flag can be used to poll whether a Data transfer is complete. However, it is discouraged to create a polling loop that waits until DMA is done. Instead, use **DmaWaitForDone** to suspend the thread until DMA is done.

DtOutpChannel::GetMaxFifoSize

Get the maximum size of the Transmit FIFO on the device.

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

Parameters

MaxFifoSize

Maximum size of the Transmit FIFO in number of bytes.

Result

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

Remarks

The actual size of the Transmit-FIFO is adjustable in software with **SetFifoSize**.

The DTA-100, DTA-102 and DTA-140 support a maximum Transmit-FIFO size of 8.388.608 bytes. Later revisions of these boards support at least this size, but potentially the maximum FIFO size may become larger.

DtOutpChannel::GetModControl

Get modulation-control parameters for Transport-Stream output channels with a built-in modulator.

```
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
);
```

Parameters

ModType

Output parameter that is set to 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.

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 device does not include 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
```

Parameters

LeveldBm

Output level expressed in units of 0.1 dBm (e.g. -30 → $-30 \times 0.1 = -3\text{dBm}$).

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 controllable output level

Remarks

DtOutpChannel::GetRfControl

Get up-converter parameters for devices with on-board RF up-converter.

```
DTAPI_RESULT DtOutpChannel::GetRfControl (
    [in] __int64& RfRate,      // RF frequency in Hz
    [out] int& LockStatus      // Lock status of RF PLL
);
```

Parameters

RfRate

Output parameters that is set to the current carrier frequency as programmed into the RF up-converter, expressed Hertz.

The RF frequency returned in *RfRate* may be different from the frequency programmed with **SetRfControl** because of rounding to the RF step size (see **SetRfControl** for a list of step sizes).

LockStatus

Output parameter that is set to the status of the RF up-converter PLL.

Value	Meaning
DTAPI_RFPLL_LOCK	The RF PLL is in lock. This is the normal state in which the up-converter is generating a stable carrier frequency.
DTAPI_RFPLL_NO_LOCK	The RF PLL is out of lock. This status may occur temporarily when the RF rate is changed to another value. The no-lock status may occur permanently when the RF rate has been programmed (outside DTAPI) to a value that is incompatible with the output channel.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The up-converter 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 up-converter

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
);
```

Parameters

Present

Output parameter 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 parameter that is set to an integer value that uniquely identifies the target adapter. Refer to the DTA-102 documentation 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	Data 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::GetTransmitByteCount

This function is not available yet in the current version of the DTAPI

Get a sample from the free running 32-bit transmitted-number-of-bytes counter.

```
DTAPI_RESULT DtOutpChannel::GetTransmitByteCount (
    [out] int& ByteCount    // Sample of #transmitted-bytes counter
);
```

Parameters

ByteCount

Sample of the 32-bit transmitted-number-of-bytes counter.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	A valid sample of the counter was returned
DTAPI_E_NOT_SUPPORTED	The counter is not supported by the current output channel
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

This method is supported on the following devices:

Device Type Number	Firmware Version
DTA-100	V7 or higher
DTA-102	V9 or higher
DTA-107(S2)	V1 or higher
DTA-140	V3 or higher
DTU-205	V1 or higher

DtOutpChannel::GetTsRateBps

Get the channel's current clock-generator mode and Transport-Stream rate.

```
DTAPI_RESULT DtOutpChannel::GetTsRateBps (
    [out] int& ClockGenMode,    // Internal or external clock
    [out] int& TsRate           // Transport-Stream rate in bps
);

DTAPI_RESULT DtOutpChannel::GetTsRateBps (
    [out] int& TsRate           // Transport-Stream rate in bps
);
```

Parameters

ClockGenMode

Output parameter that is set to the current clock-generator mode.

Value	Meaning
DTAPI_TXCLOCK_INTERNAL	Internal clock generator
DTAPI_TXCLOCK_EXTERNAL	External clock input (DTA-102 only)

TsRate

Output parameter that is set to the current Transport-Stream rate expressed in bits per second. If an external clock generator is used, *TsRate* is set to a measurement of the Transport-Stream rate corresponding to the signal on the external clock input. For an accurate estimate, the external clock signal must be present and stable for also at least one second.

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

The same remarks regarding Transport-Stream rate vs. transmit-clock rate apply as listed for **SetTsRateBps**.

DtOutpChannel::GetTxControl

Get the channel's current transmit-control state.

```
DTAPI_RESULT DtOutpChannel::GetTxControl (
    [out] int& TxControl    // Transmit-control state
);
```

Parameters

TxControl

Refer to **SetTxControl** for a description of transmit-control states.

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 channel's current transmit mode.

```
DTAPI_RESULT DtOutpChannel::GetTxMode (
    [out] int& TxMode,           // Transmit mode
    [out] int& StuffMode        // Null-packet stuffing on/off
);
```

Parameters

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::ReadLoopBackData

This function is for diagnostics purposes only

Read data bytes from the channel's Transmit FIFO via the loop-back channel. The loop-back mode must be **DTAPI_LOOPBACK_MODE**.

```
DTAPI_RESULT DtOutpChannel::ReadLoopBackData (
    [out] char*  pBuffer,          // Data read back from Transmit FIFO
    [in] int    NumBytesToRead    // Number of bytes to be read
);
```

Parameters

pBuffer

Pointer to the buffer that will receive the data read from the Transmit FIFO. The size of the buffer must be at least *NumBytesToRead*. The buffer pointer must be aligned to a 32-bit word boundary.

NumBytesToRead

Number of bytes to be read from the Transmit FIFO. *NumBytesToRead* must be positive and a multiple of four.

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 four
DTAPI_E_NO_LOOPBACK	Channel is not in loop-back mode
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_UNDERFLOW	Transmit FIFO contains less than <i>NumBytesToRead</i> bytes

Remarks

This method can be used for writing a Transmit-FIFO memory test.

Data from the Transmit FIFO can only be read back if the loop-back mode is **DTAPI_LOOPBACK_MODE**.

DtOutpChannel::Reset

Reset output channel.

```
DTAPI_RESULT DtOutpChannel::Reset (
    [in] int ResetMode
);
```

Parameters

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"> Data transfers and packet transmission are halted instantaneously. All data pending in the Transmit FIFO is discarded. Transmit-control state is reset to DTAPI_TXCTRL_IDLE. Transmit-FIFO underflow flag is cleared.
DTAPI_FULL_RESET	Full reset: <ul style="list-style-type: none"> All actions for DTAPI_FIFO_RESET, plus: Transport-Stream rate is reset to zero (except for ISDB-T and OFDM modulation on DTA-110T)

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Output channel has been reset
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_MODE	The specified value for <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::SetFailsafeAlive

Reset the failsafe watchdog timer.

```
DTAPI_RESULT DtOutpChannel::SetFailsafeAlive ();
```

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 as failsafe output
DTAPI_E_NOT_SUPPORTED	The channel is not failsafe capable
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

Failing to call this method within the time-out set with **SetFailsafeConfig** will result in the onboard relays to switch to failsafe mode (i.e. output = input).

DtOutpChannel::SetFailsafeConfig

Set failsafe configuration.

```
DTAPI_RESULT DtOutpChannel::SetFailsafeConfig (
    [in] bool    Enable,          // Enable/Disable failsafe configuration
    [in] int     Timeout=0       // Watchdog timeout (in ms)
);
```

Parameters

Enable

Enables/disables the failsafe configuration.

When set to false, failsafe configuration is disabled and the relais will be permanently set to failsafe mode (i.e. output=input). Setting this parameter to true indicates the user application is ready to start the failsafe configuration. When enabled the user application should call the **SetFailsafeAlive** method once within the failsafe timeout period.

Timeout

Specifies the failsafe timeout period (in ms).

The timeout value must a multiple of 20 ms, if the value is not a multiple of 20ms it will be rounded downwards the closest multiple of 20ms. Setting the *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 as failsafe output
DTAPI_E_NOT_SUPPORTED	The channel is not failsafe capable
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

DtOutpChannel::SetFifoSize

Set the size the Transmit FIFO on the device to a specified value (**SetFifoSize**) or to the maximum value supported by the channel (**SetFifoSizeMax**).

```
DTAPI_RESULT DtOutpChannel::SetFifoSize (
    [in] int  FifoSize          // Size of FIFO in bytes
);
DTAPI_RESULT DtOutpChannel::SetFifoSizeMax ( void );
```

Parameters

FifoSize

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 (DTA-100, DTA-102 and DTA-140: 8.388.608 bytes).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Size of the Transmit FIFO has been set successfully
DTAPI_E_IN_USE	The FIFO size cannot be changed because transmission-control state is DTAPI_TXCTRL_HOLD or DTAPI_TXCTRL_SEND
DTAPI_E_INVALID_SIZE	The specified FIFO size is negative, zero, not a multiple of 16 or greater than the maximum size
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

The size of the Transmit FIFO determines the amount of packet data that is buffered on the device. It also determines the delay between transferring data to the device (with **DmaStartTransfer**) and transmission of that data.

A large value for *FifoSize* creates headroom for *jitter* in the supply of data. Jitter is caused by other processes that also need processor cycles, or e.g. by a disk that is seeking a new disk-head position. For applications in which the output data does not depend on any input – e.g. in a disk streamer application – it is recommended to set *FifoSize* to the largest possible value.

A small value for *FifoSize* enables a short end-to-end delay, which is suitable for data-only applications in which an occasional underflow of the Transmit FIFO may occur, e.g. IP gateway. For such applications, *FifoSize* may be set, for example, to twice the size of the DMA buffer. In the extreme the Transmit-FIFO size may even be set to 16 (the minimum), but then occasional null-packet stuffing is almost guaranteed.

For real-time applications that cannot tolerate buffer underflow – e.g. in a real-time encoder – a careful assessment must be made to find a suitable buffer size, involving both real-time analysis and a bit of experimentation.

DtOutpChannel::SetIpPars

Set parameters for the transmission of a Transport Stream over IP.

```
DTAPI_RESULT DtOutpChannel::SetIpPars (
    [in] DtTsIpPars* pTsIpPars    // TS-over-IP parameters
);
```

Parameters

SetIpPars

New parameter set to be applied. Please refer to the **DtTsIpPars** page for a description of the parameters.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	TS-over-IP parameters have been applied successfully
DTAPI_E_IN_USE	Parameters cannot be changed because the channel is busy. The transmit-control state should be switched back to idle first
DTAPI_E_INVALID_ARG	The value of one of the TS-over-IP parameters is invalid
DTAPI_E_NO_LINK	IP parameters cannot be applied because the link is down
DTAPI_E_DST_MAC_ADDR	IP parameters cannot be applied because MAC address of destination cannot be determined. Most likely the destination address currently is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

SetIpPars should be called at least once after attaching to the hardware but before setting the transmit-control state to **DTAPI_TXCTRL_HOLD** or **DTAPI_TXCTRL_SEND**.

After the initial call to **SetIpPars**, parameters can be changed again, but only when the transmit-control state is **DTAPI_TXCTRL_IDLE**.

DtOutpChannel::SetLoopBackMode

This function is for diagnostics purposes only.

Set the channel's loop-back mode.

```
DTAPI_RESULT DtOutpChannel::SetLoopBackMode (
    [in] int Mode           // New loop-back mode
);
```

Parameters

Mode

New loop-back mode according to the table below.

Value	Meaning
DTAPI_NO_LOOPBACK	Normal operation. No loop-back of data.
DTAPI_LOOPBACK_MODE	Loop-back mode. The physical output circuitry is disconnected from the Transmit FIFO. The data in the Transmit FIFO can be read using ReadLoopBackData .

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Loop-back state has been changed successfully
DTAPI_E_INVALID_MODE	The specified loop-back mode value is invalid
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

After **Attach** and after **Reset**, loop-back is turned off.

DtOutpChannel::SetModControl

Set modulation-control parameters for modulator channels. There are two overloads:

SetModControl(int, int, int, int)

To be used for all modulation modes except ISDB-T with hierarchical multiplexing.

SetModControl(DtIsdbtPars&)

This second overload is specifically intended for ISDB-T with hierarchical multiplexing. The layer definition and mapping is defined in **DtIsdbtPars**.

For ISDB-T without hierarchical multiplexing in the DTAPI both overloads can be used. In this case the input of the modulator shall already be multiplexed and consist of 204-byte TMCC encoded packets.

```
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
);

DTAPI_RESULT DtOutpChannel::SetModControl (
    [in] DtIsdbtPars& DtIsdbtPars // ISDB-T modulation parameters
                                   // for hierarchical multiplexing
);
```

Parameters

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.

DtIsdbtPars

ISDB-T modulation parameters for hierarchical multiplexing; see description of **struct DtIsdbtPars**.

Detailed Parameter Descriptions

Page	Modulation Type
161	Overview
162	ADTB-T
164	ATSC
165	DTMB
167	DVB-S

Page	Modulation Type
168	DVB-S.2
169	DVB-T / DVB-H
174	ISDB-T
176	QAM

Modulation Types

ModType

Modulation type:

L-Band

ModType	Meaning	Available on
DTAPI_MOD_DVBS_BPSK	DVB-S, BPSK	DTA-107
DTAPI_MOD_DVBS_QPSK	DVB-S, QPSK	DTA-107
DTAPI_MOD_DVBS2_8PSK	DVB-S.2, 8-PSK	DTA-107, DTA-107S2
DTAPI_MOD_DVBS2_QPSK	DVB-S.2, QPSK	DTA-107, DTA-107S2

VHF* / UHF

ModType	Meaning	Available on
DTAPI_MOD_ADTBT	ADTB-T	DTA-110T, DTA-115 (DTMB option must be installed)
DTAPI_MOD_ATSC	ATSC VSB	DTA-110T Fw > 4 DTA-115 Fw > 0
DTAPI_MOD_DMBTH	DMB-T/H	DTA-110T, DTA-115 (DTMB option must be installed)
DTAPI_MOD_DVBT	DVB-T / DVB-H	DTA-110T, DTA-115
DTAPI_MOD_QAM16	16-QAM	ITU-T J.83 Annex A/C: DTA-110, DTA-110T, DTA-115
DTAPI_MOD_QAM32	32-QAM	
DTAPI_MOD_QAM64	64-QAM	ITU-T J.83 Annex B: DTA-110, DTA-110T Fw > 3 DTA-115 Fw > 0
DTAPI_MOD_QAM128	128-QAM	
DTAPI_MOD_QAM256	256-QAM	

* VHF available on DTA-115 only

Modulation Mode : ADTB-T

ParXtra0

Extra modulation parameter #1 is the OR of values for the following fields: Bandwidth, Constellation, FEC Code Rate, Frame Header Mode, Interleaver Mode, Pilots and Use Frame Numbering.

Bandwidth

Value	Meaning
DTAPI_MOD_DTMB_5MHZ	5 MHz
DTAPI_MOD_DTMB_6MHZ	6 MHz
DTAPI_MOD_DTMB_7MHZ	7 MHz
DTAPI_MOD_DTMB_8MHZ	8 MHz
DTAPI_MOD_DTMB_BW_MSK	AND mask

Constellation

Value	Meaning
DTAPI_MOD_DTMB_QAM4NR	4-QAM-NR; can only be used with FEC code rate 0.8
DTAPI_MOD_DTMB_QAM4	4-QAM
DTAPI_MOD_DTMB_QAM16	16-QAM
DTAPI_MOD_DTMB_QAM32	32-QAM; can only be used with FEC code rate 0.8
DTAPI_MOD_DTMB_QAM64	64-QAM
DTAPI_MOD_DTMB_CO_MSK	AND mask

FEC Code Rate

Value	Meaning
DTAPI_MOD_DTMB_0_4	FEC code rate 0.4: FEC(7488, 3008)
DTAPI_MOD_DTMB_0_6	FEC code rate 0.6: FEC(7488, 4512)
DTAPI_MOD_DTMB_0_8	FEC code rate 0.8: FEC(7488, 6016)
DTAPI_MOD_DTMB_RATE_MSK	AND mask

Frame Header Mode

Value	Meaning
DTAPI_MOD_DTMB_PN420	PN420: Frame header 1 (420 symbols 55.6 μ s)
DTAPI_MOD_DTMB_PN595	PN595: Frame header 2 (595 symbols 78.7 μ s)
DTAPI_MOD_DTMB_PN945	PN945: Frame header 3 (945 symbols 125 μ 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	Available on	
DTAPI_MOD_ATSC	ATSC	DTA-110T	Fw > 4
		DTA-115	Fw > 0

ParXtra0

Extra modulation parameter #0 specifies the VSB constellation.

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

ParXtra1

This parameter specifies the number of taps of each phase of the root-raised cosine filter that is used to shape the spectrum of the output signal. The number of taps can have any value between 2 and 256 (the implementation is optimized for powers of 2). Specifying more taps improves the spectrum, but increases processor overhead.

The recommend number of taps is 64 taps; If insufficient CPU power is available, 32 taps produces acceptable results, too.

ParXtra2

Not used in ATSC modulation.

Modulation Mode : DTMB

ParXtra0

Extra modulation parameter #1 is the OR of values for the following fields: Bandwidth, Constellation, FEC Code Rate, Frame Header Mode, Interleaver Mode and Use Frame Numbering.

Bandwidth

Value	Meaning
DTAPI_MOD_DTMB_5MHZ	5 MHz
DTAPI_MOD_DTMB_6MHZ	6 MHz
DTAPI_MOD_DTMB_7MHZ	7 MHz
DTAPI_MOD_DTMB_8MHZ	8 MHz
DTAPI_MOD_DTMB_BW_MSK	AND mask

Constellation

Value	Meaning
DTAPI_MOD_DTMB_QAM4NR	4-QAM-NR; can only be used with FEC code rate 0.8
DTAPI_MOD_DTMB_QAM4	4-QAM
DTAPI_MOD_DTMB_QAM16	16-QAM
DTAPI_MOD_DTMB_QAM32	32-QAM; can only be used with FEC code rate 0.8
DTAPI_MOD_DTMB_QAM64	64-QAM
DTAPI_MOD_DTMB_CO_MSK	AND mask

FEC Code Rate

Value	Meaning
DTAPI_MOD_DTMB_0_4	FEC code rate 0.4: FEC(7488, 3008)
DTAPI_MOD_DTMB_0_6	FEC code rate 0.6: FEC(7488, 4512)
DTAPI_MOD_DTMB_0_8	FEC code rate 0.8: FEC(7488, 6016)
DTAPI_MOD_DTMB_RATE_MSK	AND mask

Frame Header Mode

Value	Meaning
DTAPI_MOD_DTMB_PN420	PN420: Frame header 1 (420 symbols 55.6 μ s)
DTAPI_MOD_DTMB_PN945	PN945: Frame header 3 (945 symbols 125 μ 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

ParXtra2

Physical layer scrambling initialization sequence "n", aka "Gold code".

Modulation Mode : DVB-T/DVB-H*ParXtra0*

Extra modulation parameter #0 is the code rate.

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

ParXtra1

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

Bandwidth

Value	Meaning
DTAPI_MOD_DVBT_5MHZ	5 MHz
DTAPI_MOD_DVBT_6MHZ	6 MHz
DTAPI_MOD_DVBT_7MHZ	7 MHz
DTAPI_MOD_DVBT_8MHZ	8 MHz
DTAPI_MOD_DVBT_BW_MSK	AND mask

Constellation

Value	Meaning
DTAPI_MOD_DVBT_QPSK	QPSK
DTAPI_MOD_DVBT_QAM16	16-QAM
DTAPI_MOD_DVBT_QAM64	64-QAM
DTAPI_MOD_DVBT_CO_MSK	AND mask

Guard Interval

Value	Meaning
DTAPI_MOD_DVBT_G_1_32	1/32
DTAPI_MOD_DVBT_G_1_16	1/16
DTAPI_MOD_DVBT_G_1_8	1/8
DTAPI_MOD_DVBT_G_1_4	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. Note: 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 : DVB-T2*ParXtra0*

Extra modulation parameter #0 is the OR of values for the following fields: Bandwidth, FFT Mode, Guard Interval, Pilot Pattern, L1 Signalling Constellation, PLPO Constellation.

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
DTAPI_DVBT2_BW_MSK	AND mask for bandwidth

Bandwidth Extension (aka Carrier Extension Mode)

Value	Meaning
DTAPI_DVBT2_BWTEXT	1-bit AND mask for bandwidth-extension field

FFT Mode

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
DTAPI_DVBT2_FFT_MSK	AND mask for FFT mode

Guard Interval

Value	Meaning
DTAPI_DVBT2_GI_1_32	1/32
DTAPI_DVBT2_GI_1_16	1/16
DTAPI_DVBT2_GI_1_8	1/8
DTAPI_DVBT2_GI_1_4	1/4
DTAPI_DVBT2_GI_1_128	1/128
DTAPI_DVBT2_GI_19_128	19/128
DTAPI_DVBT2_GI_19_256	19/256
DTAPI_DVBT2_GI_MSK	AND mask for guard interval

Pilot Pattern

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
DTAPI_DVBT2_PP_MSK	AND mask for pilot pattern

L1 Signalling Constellation

Value	Meaning
DTAPI_DVBT2_L1_BPSK	BPSK
DTAPI_DVBT2_L1_QPSK	QPSK
DTAPI_DVBT2_L1_QAM16	16-QAM
DTAPI_DVBT2_L1_QAM64	64-QAM
DTAPI_DVBT2_L1_CO_MSK	AND mask for L1 signalling constellation

PLP0 Constellation

Value	Meaning
DTAPI_DVBT2_PLP0_QPSK	QPSK
DTAPI_DVBT2_PLP0_QAM16	16-QAM
DTAPI_DVBT2_PLP0_QAM64	64-QAM
DTAPI_DVBT2_PLP0_QAM256	256-QAM
DTAPI_DVBT2_PLP0_CO_MSK	AND mask for PLP0 constellation

ParXtral

Extra modulation parameter #1 is the OR of values for the following fields: Number of T2 Frames per Superframe, Number of Data Symbols per T2-Frame.

Number of T2 Frames per Superframe

Value	Meaning
DTAPI_DVBT2_NUMT2FRM_MSK	8-bit AND mask for number of frames per superframe field. The valid range is 1..255.
DTAPI_DVBT2_NUMT2FRM_SHFT	

Number of T2 Data Symbols per T2 Frame

Value	Meaning
DTAPI_DVBT2_NUMDTSYM_MSK	12-bit AND mask for number of data symbols per T2 frame
DTAPI_DVBT2_NUMDTSYM_SHFT	

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 center 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&)` is 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 **BTTYPE_TV** for 1-segment modulation.

Modulation Mode : QAM

ModType

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

ModType	Meaning	Available on
DTAPI_MOD_QAM16	16-QAM	ITU-T J.83 Annex A/C: DTA-110, DTA-110T, DTA-115
DTAPI_MOD_QAM32	32-QAM	
DTAPI_MOD_QAM64	64-QAM	ITU-T J.83 Annex B: DTA-110, DTA-110T Fw > 3 DTA-115 Fw > 0
DTAPI_MOD_QAM128	128-QAM	
DTAPI_MOD_QAM256	256-QAM	

ParXtra0

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

ITU-T J.83 Annex	Meaning	Available on
DTAPI_MOD_J83_A	J.83 annex A (DVB-C)	DTA-110, DTA-110T, DTA-115
DTAPI_MOD_J83_B	J.83 annex B ("American QAM")	DTA-110, DTA-110T Fw > 3 DTA-115 Fw > 0
DTAPI_MOD_J83_C	J.83 annex C ("Japanese QAM")	DTA-110, DTA-110T, DTA-115

ParXtra1

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

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

ParXtra2

Not used.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The modulation parameters have been set successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_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 current level (in dBm) for outputs with an adjustable output level.

```
DTAPI_RESULT DtOutpChannel::SetOutputLevel (
    [in] int   LeveldBm);          // Level in units of 0.1dBm
```

Parameters

LeveldBm

Output level expressed in units of 0.1 dBm (e.g. -30 → -30×0.1 = -3 dBm). The table below specifies the valid range and the step size with which the output level can be specified.

LeveldBm is rounded to the nearest level compatible with the supported step size.

Device	Valid Range		Step Size
DTA-107(S2)	DVB-S(2)	-47.0 .. -27.0 dBm	0.1 dB
DTA-115(-ISDB)	QAM	-35.0 .. 0.0 dBm	0.5 dB
	OFDM, ISDB-T	-38.0 .. -3.0 dBm	

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
);
```

Parameters

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 a power connection for target adapters (DTA-100, DTA-140)

Remarks

After **Attach** and after **Reset**, power is turned off.

DtOutpChannel::SetRfControl

Set up-converter parameters for devices with on-board RF up-converter.

```
DTAPI_RESULT DtOutpChannel::SetRfControl (
    [in] __int64  RfRate      // RF frequency in Hz
);
```

Parameters

RfRate

New carrier frequency for RF up-converter, specified in Hertz. The table below specifies the valid range and the step size with which the RF rate can be specified. *RfRate* is rounded to the nearest RF frequency compatible with the frequency resolution.

Device	Valid Range	Step Size
DTA-107(S2)	950,000,000 – 2,150,000,000 Hz	200,000 Hz
DTA-110(T)(-ISDB)	400,000,000 – 862,000,000 Hz	250,000 Hz
DTA-115(-ISDB)	47,000,000 – 862,000,000 Hz	100,000 Hz

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 up-converter
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	The device does not have an RF up-converter

Remarks

Changing the RF frequency takes some time to let the PLL settle at the new frequency. The lock status of the PLL may be checked with `GetRfControl`.

DtOutpChannel::SetRfMode

Set special modes for devices with on-board RF up-converter.

```
DTAPI_RESULT DtOutpChannel::SetRfMode (
    [in] int   RfMode           // Special up-converter mode
);
```

Parameters

RfMode

New RF up-converter 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	Enable spectral inversion

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The new mode has been set successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_MODE	The specified up-converter 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.

```
DTAPI_RESULT DtOutpChannel::SetSNR (
    [in] int  Mode,           // Noise generation mode
    [in] int  SNR            // desired SNR
);
```

Parameters

Mode

Noise generation mode too use.

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 0.1 dB (e.g. 250 = 250×0.1 = 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::SetTsRateBps

Set the channel's *Transport-Stream* rate. Please note that the Transport-Stream rate is always based on 188-byte packets, as defined in the MPEG-2 Systems specification. Therefore, the Transport-Stream rate will be different from the transmit clock for 204-byte transmit modes.

```
DTAPI_RESULT DtOutpChannel::SetTsRateBps (
    [in] int    ClockGenMode,    // Internal or external clock
    [in] int    TsRate          // Transport-Stream rate in bps
);

DTAPI_RESULT DtOutpChannel::SetTsRateBps (
    [in] int    TsRate          // Transport-Stream rate in bps
);
```

Parameters

ClockGenMode

Clock generator mode.

Value	Meaning
DTAPI_TXCLOCK_INTERNAL	Use internal clock generator
DTAPI_TXCLOCK_EXTERNAL	Use external clock input (DTA-102 only)

TsRate

New Transport-Stream rate specified in bits per second. This parameter is ignored if *ClockGenMode* is **DTAPI_CLOCK_EXTERNAL**.

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_CGMODE	The specified clock-generator mode is invalid or incompatible with the attached hardware function
DTAPI_E_INVALID_RATE	The specified transport-rate is invalid (negative) or incompatible (too high) with the attached hardware function
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function
DTAPI_E_NOT_SUPPORTED	Setting the TS rate is not allowed (ISDB-T)

Remarks

In modes **DTAPI_TXMODE_204** and **DTAPI_TXMODE_ADD16** the transmit-clock rate is set to 204/188 times the specified Transport-Stream clock.

The Transport-Stream rate is usually set in the initialisation phase just after **AttachToSlot**, **AttachToType** or **Reset**. It is recommended to first set the transmit mode with **SetTxMode** before setting the bit rate so that all packets, including the initial ones, are sent in the same format.

SetTsRateBps may also be used while packets are being transmitted. The DTA/U series of devices imposes 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.

DtOutpChannel::SetTxControl

Set the channel's transmit-control state.

```
DTAPI_RESULT DtOutpChannel::SetTxControl (
    [in] int TxControl // Transmit-control state
);
```

Parameters

TxControl

New transmit-control state according to the table below.

Value	Meaning
DTAPI_TXCTRL_IDLE	Execution of data transfers is disabled. No packets from the Transmit FIFO are transmitted. However, the output is stuffed with null packets, except if transmit <i>mode</i> is DTAPI_TXMODE_RAW, in which case no packets are transmitted at all.
DTAPI_TXCTRL_HOLD	Data transfers are enabled. Data can be transferred until the Transmit FIFO is fully loaded. Packet transmission is disabled, in the same way as in state DTAPI_TXCTRL_IDLE.
DTAPI_TXCTRL_SEND	Normal operation. Both data transfers and packet transmission are enabled.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Transmit-control state 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_MODE	The specified transmit-control state is invalid or incompatible with the attached hardware function
DTAPI_E_NO_IPPARS	For TS-over-IP channels: cannot set transmission state because IP parameters have not been specified yet
DTAPI_E_NO_TSRATE	For modulators: TS rate has not been set but is required for starting modulation
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

Setting the transmit-control state can be used for controlled start-up and shutdown of the data-streaming process. In state `DTAPI_TXCTRL_HOLD`, the Transmit FIFO can be loaded with packets (using DMA transfers), while no packets are transmitted yet. Then, if enough *credit* has been built up in the Transmit FIFO, state `DTAPI_TXCTRL_SEND` is entered. This procedure prevents accidental Transmit-FIFO underflow in the start-up phase.

For TS-over-IP channels, state `DTAPI_TXCTRL_HOLD` or `DTAPI_TXCTRL_SEND` can only be entered if the IP transmission parameters have been specified using `SetIpPars`.

After `AttachToSlot`, `AttachToType` and after `Reset` the transmit-control state is initialised to `DTAPI_TXCTRL_IDLE`.

DtOutpChannel::SetTxMode

Set the channel's transmit mode.

```
DTAPI_RESULT DtOutpChannel::SetTxMode (
    [in] int    TxMode,           // Transmit mode
    [in] int    StuffMode        // Null-packet stuffing on/off
                                   // SDI: Black-frame stuffing on/off
);
```

Parameters

TxMode

New transmit mode according to the table below.

Value	Meaning
DTAPI_TXMODE_188	<i>188-byte mode</i> Packets in the Transmit FIFO are assumed to be 188-bytes long. Packets are transmitted without modification.
DTAPI_TXMODE_192	<i>192-byte mode (DTA-102 only)</i> 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	<i>204-byte mode</i> Packets in the Transmit FIFO are assumed to be 204-bytes long. Packets are transmitted without modification.
DTAPI_TXMODE_ADD16	<i>Add-16 bytes mode</i> Packets in the Transmit FIFO are assumed to be 188-bytes long. The device adds 16 placeholder bytes (0) to every packet.
DTAPI_TXMODE_RAW	<i>Raw mode</i> No assumptions are made on packet structure. Bytes in the buffer are transmitted unmodified. Null-packet stuffing cannot be applied. This mode is not allowed for TS-over-IP channels.

The following modes are valid for SDI-capable channels only.

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

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

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

For DVB-ASI output channels, *TxMode* can be OR-ed with **DTAPI_TXMODE_BURST**, which indicates that packets should be sent in bursts. If this flag is not specified, transmission of packet data is "continuous" (linear over time).

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

Value	Meaning
DTAPI_TXMODE_SDI	<i>SDI mode</i> Indicates that the output channel shall operate in SDI mode. If this flag is not used the channel operates in ASI mode. This flag is already OR-ed into the SDI-related transmit modes mentioned in the table above.
DTAPI_TXMODE_SDI_10B	<i>10-bit SDI samples</i> Indicates that SDI data in the Transmit FIFO is assumed to consist of 10-bit SDI samples. If this flag is omitted, the 8-bit samples SDI samples are assumed.
DTAPI_TXMODE_SDI_525	<i>525-line mode</i> Indicates that the SDI data in the Transmit FIFO is assumed to be 525-lines SDI.
DTAPI_TXMODE_SDI_625	<i>625-line mode</i> Indicates that the SDI data in the Transmit FIFO is assumed to be 625-lines SDI.
DTAPI_TXMODE_SDI_HUFFMAN	<i>Huffman compression</i> Indicated that the SDI frame in the Transmit FIFO is assumed to be Huffman compressed.

For genlock-capable cards the output channel can be configured to operate in genlock mode (see **SetIOConfig**). When the genlock mode and the requested transmit mode conflict, **DTAPI_E_INVALID_MODE** error will be returned.

Please refer to section 4 for more details about the SDI data formats.

StuffMode

This parameter controls the behaviour of the output when there is no packet data available for transmission from the Transmit FIFO.

For channels in ASI transmission mode:

If *StuffMode* is '1' (On), the output is stuffed with null packets. The size of inserted null packets is matched to *TxMode*. Packet Stuffing is not supported if *TxMode* is **DTAPI_TXMODE_RAW**, because both packet size and packet boundaries are unknown (**DTAPI_E_MODE** error is returned.)

If *StuffMode* is '0' (Off), null-packet stuffing is not applied. If the Transmit FIFO underflows:

- For DVB-ASI outputs (DTA-100, DTA-140), the output is stuffed with K28.5 characters;
- For DVB-SPI outputs (DTA-102), DVALID is de-asserted.

For channels in SDI transmission mode (e.g. **DTAPI_TXMODE_SDI_FULL**) this parameter is ignored for non-genlock capable channels.

When the card is SDI genlock capable but the output channel is not configured to operate in genlock mode this parameter can be used to manually enable *black-frame stuffing*. When black-frame stuffing is enabled, the SDI output is stuffed with black frames when there is no packet data available in the Transmit FIFO.

Note that this parameter is ignored when the TX Channel is in SDI genlock mode, since the driver will automatically enable black-frame stuffing in this case.

Result

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

Remarks**Changing the transmit mode may change the transmit-clock rate!**

E.g. 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.

SetTxMode may also be used while packets are being transmitted; no hardware constraints apply. However, an uncontrolled transition effect in the Transport Stream may occur. It is recommended to stop transmission and clear the Transmit FIFO with **clearFifo** before changing transmit mode.

DtOutpChannel::SetTxPolarity

Set polarity of DVB-ASI output signal.

```
DTAPI_RESULT DtOutpChannel::SetTxPolarity (
    [in] int    TxPolarity        // Polarity (normal or inverted)
);
```

Parameters

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	Polarity has been changed successfully
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_INVALID_MODE	The specified polarity mode is invalid or incompatible with the output channel
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 channel

Remarks

Use this method to test if your ASI receiver is available of receiving both normal and inverted ASI signals.

NOTE: many ASI receivers do not support an inverted ASI signal.

DtOutpChannel::Write

Write data bytes to the output channel.

```
DTAPI_RESULT DtOutpChannel::Write (
    [in] char*   pBuffer,           // Pointer to data to be written to hardware
    [in] int     NumBytesToWrite // Number of bytes to be written to hardware
);
```

Parameters

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.

NumBytesToWrite

Number of bytes to be written to the output channel. The buffer size must be positive and a multiple of four.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Write operation has been completed successfully
DTAPI_E_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 four
DTAPI_E_IDLE	For TS-over-IP channels: cannot write data because transmission-control state is DTAPI_TXCTRL_IDLE
DTAPI_E_NO_TSRATE	For TS-over-IP channels: cannot write data because Transport-Stream rate has not been specified, or is too low
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

The data buffer can be any buffer in user space that is aligned to a 4-byte boundary. The device driver locks the physical pages of the buffer in memory. If the pages were swapped to disk, they are read back into memory. Then for PCI Cards a scatter/gather list is created and the DMA transfer is initiated.

Write returns when all data has been transferred to the Transmit FIFO. Note that data is only transferred when the transmit-control state is **DTAPI_TXCTRL_HOLD** or **DTAPI_TXCTRL_SEND** (see [SetTxControl](#)), and if sufficient space is available in the Transmit FIFO.

The thread executing **Write** sleeps if one of the following conditions is true:

- Transmit-control state is **DTAPI_TXCTRL_IDLE** (except for TS-over-IP channels; in this case, **Write** returns **DTAPI_E_IDLE**).
- Transmit-control state is **DTAPI_TXCTRL_HOLD**, and the number of bytes to be written is greater than the size of the Transmit FIFO.
- Transmit-control state is **DTAPI_TXCTRL_SEND**, and the transmit rate is zero.

The call will be completed if the blocking condition is removed (from another thread).

The old name of this function is **WriteUsingDma**. This latter function name is still defined for backward compatibility.

DtOutpChannel::WriteDirect

This function is for diagnostics purposes only, please use method Write for 'ordinary' writing. Write data bytes to the output channel using "direct" write-cycles to the PCI bus.

```
DTAPI_RESULT DtOutpChannel::WriteDirect (
    [in] char*   pBuffer,           // Pointer to data to be written to hardware
    [in] int     NumBytesToWrite // Number of bytes to be written to hardware
);
```

Parameters

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.

NumBytesToWrite

Number of bytes to be written to the output channel. The buffer size must be positive and a multiple of four.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Write operation has been completed successfully
DTAPI_E_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 four
DTAPI_E_FIFO_FULL	The operation timed out because the Transmit FIFO is full
DTAPI_E_NOT_ATTACHED	Channel object is not attached to a hardware function

Remarks

The data buffer can be any buffer in user space that is aligned to a 4-byte boundary.

For USB Devices **WriteDirect** has exactly the same function as **Write**, but for PCI cards **WriteDirect** is considerably slower because the processor writes the data bytes directly to the PCI bus instead of using DMA. **WriteDirect** should only be used for special purposes, e.g. for PCI-card validation.

Unlike **Write**, this function cannot block. If the write operation takes too long, error code **DTAPI_E_FIFO_FULL** is returned.

TsInpChannel

TsInpChannel

Class representing an input channel for receiving an MPEG-2 Transport Stream.

```
class TsInpChannel : public DtInpChannel {  
    :  
    :  
    :  
};
```

Description

Class **TsInpChannel** is a specialisation of the generic **DtInpChannel** class. **TsInpChannel** supports the Transport-Stream functions of **DtInpChannel**. Methods dedicated to SDI will fail.

TsOutpChannel**TsOutpChannel**

Class representing an output channel for transmitting an MPEG-2 Transport Stream.

```
class TsOutpChannel : public DtOutpChannel {  
    :           :  
};
```

Description

Class **TsOutpChannel** is a specialisation of the generic **DtOutpChannel** class. **TsOutpChannel** supports the Transport-Stream functions of **DtOutpChannel**. Methods dedicated to SDI will fail.

DtLoop**DtLoop**

DtLoop class represents an object that provides functionality to loop MPEG-2 transport-streams from a DtInpChannel to a DtOutpChannel object.

```
class DtLoop;
```

DtLoop::AttachToInput

Attach a `DtInpChannel` object to the loop object.

```
DTAPI_RESULT DtLoop::AttachToInput (
    [in] DtInpChannel*  pDtInp          // Input channel to attach
);
```

Parameters

pDtInp

Pointer to the `DtInpChannel` object to be used as input by the `DtLoop` object. The `DtInpChannel` object, passed here, must stay in scope until the object is detached from the `DtLoop` object.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Loop object has been attached successfully to input channel
DTAPI_E_ATTACHED	Loop object is already attached to a other input channel
DTAPI_E_INVALID_ARG	Invalid <i>pDtInp</i> pointer (i.e. NULL-pointer)
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	The <i>pDtInp</i> object has not been attached to any hardware input function

Remarks

After attaching a `DtInpChannel` object to a `DtLoop` object the `DtInpChannel` object may still be used to apply certain settings like setting the IP-over-TS parameters (`DtInpChannel::SetTsIpPars`) or getting the Ts-Rate (`DtInpChannel::GetTsRateBps`). However note that certain methods on the `DtInpChannel` object will have limited functionality or may not be called while the object is attached to a `DtLoop` object. For example the `DtInpChannel::Read` method may not be called while attached to a `DtLoop` object.

DtLoop::AttachToOutput

Attach a `DtOutputChannel` object to the loop object.

```
DTAPI_RESULT DtLoop::AttachToOutput (
    [in] DtOutputChannel*  pDtOutp          // Output channel to attach
);
```

Parameters

pDtOutp

Pointer to the `DtOutputChannel` object to be used as input by the `DtLoop` object. The `DtOutputChannel` object, passed here, should stay in scope until the object is detached from the `DtLoop` object.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Loop object has been attached successfully to input channel
DTAPI_E_ATTACHED	Loop object is already attached to a other input channel
DTAPI_E_INVALID_ARG	Invalid <i>pDtOutp</i> pointer (i.e. NULL-pointer)
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver
DTAPI_E_NOT_ATTACHED	The <i>pDtOutp</i> object has not been attached to any hardware input function

Remarks

After attaching a `DtOutputChannel` object to a `DtLoop` object the `DtOutputChannel` object may still be used to apply certain settings like setting the IP-over-TS parameters (`DtOutputChannel::SetTsIpPars`). However note that certain methods on the `DtOutputChannel` object will have limited functionality or may not be called while the object is attached to a `DtLoop` object. For example the `DtOutputChannel::Write` method may not be called while attached to a `DtLoop` object.

DtLoop::Detach

Detaches the loop object from the associated `DtInpChannel` and `DtOutpChannel` objects.

```
DTAPI_RESULT DtLoop::Detach();
```

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Loop object has been detached successfully
DTAPI_E_NOT_ATTACHED	Loop object was not attached to any input or output
DTAPI_E_NOT_IDLE	Cannot detach while the loop object is started

Remarks

DtLoop::DetachFromInput

Detaches the loop object from the associated **DtInpChannel** object

```
DTAPI_RESULT DtLoop::DetachFromInput ();
```

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Loop object has been detached successfully
DTAPI_E_NOT_ATTACHED	Loop object was not attached to any input
DTAPI_E_NOT_IDLE	Cannot detach while the loop object is started

Remarks

DtLoop::DetachFromOutput

Detaches the loop object from the associated `DtOutputChannel` object

```
DTAPI_RESULT DtLoop::DetachFromOutput ();
```

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Loop object has been detached successfully
DTAPI_E_NOT_ATTACHED	Loop object was not attached to any output
DTAPI_E_NOT_IDLE	Cannot detach while the loop object is started

Remarks

DtLoop::IsStarted

Returns true if looping from input-to-output has started.

```
bool DtLoop::IsStarted ();
```

Result

Result	Meaning
true	Looping has started
false	Looping has not started

Remarks

DtLoop::SetStuffingMode

Set NULL-packet stuffing mode parameters.

```
DTAPI_RESULT DtLoop::SetStuffingMode (
    [in] int    Mode,           // Stuffing mode
    [in] int    TsRate         // Output rate
);
```

Parameters

Mode

New stuffing mode according to the table below.

Value	Meaning
DTAPI_STUFFMODE_OFF	Stuffing mode should be disabled
DTAPI_STUFFMODE_ON	Stuffing mode should be enabled

TsRate

Specifies the desired output Transport-Stream rate. The **DtLoop** object will automatically add/remove NULL-packets to/from the received transport-stream to reach the rate specified here.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Stuffing mode has been set successfully
DTAPI_E_INVALID_MODE	The specified stuffing mode is invalid
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver

DtLoop::Start

Start/Stop looping from input-to-output.

```
DTAPI_RESULT DtLoop::Start (  
    [in] bool    Start=true        // Start looping  
);
```

Parameters

Start

Set to true too start looping. Set to false stop looping.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Looping has started successfully
DTAPI_E_NOT_ATTACHED	Loop-controller object has not been attached to an input and/or output
DTAPI_E_DEV_DRIVER	Unclassified failure in device driver

Remarks

Before calling this method the loop-controller object must have been attached to an input and output through successful calls to `DtLoop::AttachToInput` and `DtLoop::AttachToOutput`

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
);
```

Parameters

pInFrame

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

InFrameSize

As an input parameter this parameter 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 parameter this parameter 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 parameter this parameter 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 parameter this parameter 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

Please refer to section 4 for more details about the different SDI data formats.

4. Definition of data formats

This section provides details about the different data formats used by the DTAPI input and output channels.

4.1. 10-bit SDI format

In 10-bit SDI format, all 10 bits of the SDI samples are stored. The first sample that is stored is the EAV code of the first line of a frame. The first line of a frame is considered to be the first line in which the Field bit in the EAV code is '0', indicating the first field: line 1 in 625-line mode or line 4 in 525-line mode. The first sample of a frame is always stored on a 32-bit boundary. Data stuffing of three bytes is needed in 525-line video mode, since the number of bytes in such a 10-bit frame is not a multiple of four.

Syntax	#bits	Mnemonic
sdi_10bit_stream() {		
if (timestamp_flag) {		
timestamp[7..0]	8	uimsbf
timestamp[15..8]	8	uimsbf
timestamp[23..16]	8	uimsbf
timestamp[31..24]	8	uimsbf
}		
do {		
for (line=1; line <= num_lines; line++) {		
sync_code /* '1111 1111 11' */	10	bsrtlb
sync_code /* '0000 0000 00' */	10	bsrtlb
sync_code /* '0000 0000 00' */	10	bsrtlb
eav_code (line)	10	bsrtlb
for (samp=1; samp<=hsyncs_per_line; samp++) {		
sample_data	10	bsrtlb
}		
sync_code /* '1111 1111 11' */	10	bsrtlb
sync_code /* '0000 0000 00' */	10	bsrtlb
sync_code /* '0000 0000 00' */	10	bsrtlb
sav_code (line)	10	bsrtlb
for (samp=1; samp<=samps_per_line; samp++) {		
sample_data	10	bsrtlb
}		
}		
if (sdi_std==Mode525) {		
for (i=0; i<3; i++) {		
stuffing_byte /* '0000 0000' */	8	bslbf
}		
}		
}		

timestamp

The *timestamp* is a 32-bit field that indicates the value of the reference clock at the moment the first SDI sample of the payload enters the input channel.

sync_code

Synchronisation byte as defined in BT-656 specification.

eav_code

End of Active Video code as defined in BT-656 specification. This code depends on the line number.

sav_code

Start of Active Video code as defined in BT-656 specification. This code depends on the line number.

sample_data

The 10-bit SDI samples.

stuffing_byte

Byte that is produced at the end of a 525-line mode frame only, with the purpose of aligning the first sample of the next frame to a 32-bit boundary

4.2. 8-bit SDI format

In 8-bit SDI format, only the most significant eight bits of each SDI sample are stored. The first byte is the EAV code of the first line of a frame. The first line of a frame is considered to be the first line in which the Field bit in the EAV code is '0', indicating the first field: line 1 in 625-line mode or line 4 in 525 line mode. The first sample of a frame is always stored on a 32-bit boundary. No data stuffing is needed since in all video modes the number of bytes in an 8-bit frame is divisible by four.

Syntax	#bits	Mnemonic
<code>sdi_8bit_stream() {</code>		
<code>if (timestamp_flag) {</code>		
<code>timestamp[7..0]</code>	8	uimsbf
<code>timestamp[15..8]</code>	8	uimsbf
<code>timestamp[23..16]</code>	8	uimsbf
<code>timestamp[31..24]</code>	8	uimsbf
<code>}</code>		
<code>do {</code>		
<code>for (line=1; line <= num_lines; line++) {</code>		
<code>sync_code /* '1111 1111' */</code>	8	bslbf
<code>sync_code /* '0000 0000' */</code>	8	bslbf
<code>sync_code /* '0000 0000' */</code>	8	bslbf
<code>eav_code(line)</code>	8	bslbf
<code>for (samp=1; samp<=hsyncs_per_line; samp++) {</code>		
<code>sample_byte</code>	8	bslbf
<code>}</code>		
<code>sync_code /* '1111 1111' */</code>	8	bslbf
<code>sync_code /* '0000 0000' */</code>	8	bslbf
<code>sync_code /* '0000 0000' */</code>	8	bslbf
<code>sav_code(line)</code>	8	bslbf
<code>for (samp=1; samp<=samps_per_line; samp++) {</code>		
<code>sample_byte</code>	8	bslbf
<code>}</code>		
<code>}</code>		
<code>}</code>		
<code>}</code>		

timestamp

The *timestamp* is a 32-bit field that indicates the value of the reference clock at the moment the first SDI sample of the payload enters the input channel.

sync_code

Synchronisation byte as defined in BT-656 specification.

eav_code

End of Active Video code as defined in BT-656 specification. This code depends on the line number.

sav_code

Start of Active Video code as defined in BT-656 specification. This code depends on the line number.

sample_byte

The SDI video data with the two least significant bits removed.

4.3. Huffman-Compressed SDI format

DekTec's SDI capable input and output channels support a custom Huffman encoding scheme for compressing of SDI frames. Using the compressed format can be useful to reduce the size of recorded SDI files. The using compression can also be used too reduce PCI or USB bandwidth requirements, leaving more bandwidth for additional traffic.

The table below provides the syntax of a compressed SDI frame.

Syntax	#bits	Mnemonic
<pre> sdi_compressed_stream_with_blanking () { if (timestamp_flag) { timestamp[7..0] timestamp[15..8] timestamp[23..16] timestamp[31..24] } do { sync_word /* '11 1111 1111 1111 1111' */ for (line=1; line <= num_lines; line++) { skip_samples(4); /* skip EAV */ prev_data = blanking_level for (samp=1; samp<=hsyncs_per_line; samp++) { huffman(sample_data - prev_data) prev_data = sample_data } skip_samples(4); /* skip SAV */ prev_data = blanking_level for (samp=1; samp<=samps_per_line; samp++) { huffman(sample_data - prev_data) prev_data = sample_data } } if (alignment()!=32) { stuffing_data /* '0' */ } } } </pre>	<p>8</p> <p>8</p> <p>8</p> <p>8</p> <p>18</p> <p>2-16</p> <p>2-16</p> <p>2-30</p>	<p>uimsbf</p> <p>uimsbf</p> <p>uimsbf</p> <p>uimsbf</p> <p>bsrtlb</p> <p>bsrtlb</p> <p>bsrtlb</p>

timestamp

The *timestamp* is a 32-bit field that indicates the value of the reference clock at the moment the first SDI sample of the payload enters the input channel.

sync_word

The synchronisation code.

sample_data

The SDI video data.

prev_data

The previous sample of the SDI video data of the same type (Cb, Y, or Cr) as the current sample.

stuffing_data

Data that is produced at the end of a frame only, with the purpose of aligning the **sync_word** of the next frame to a 32-bit boundary

The table below provides the syntax of a compressed frame with only the active video part.

Syntax	#bits	Mnemonic
<pre> sdi_compressed_stream_with_blanking () { if (timestamp_flag) { timestamp[7..0] timestamp[15..8] timestamp[23..16] timestamp[31..24] } do { sync_word /* '11 1111 1111 1111 1111' */ for (line=1; line <= num_lines; line++) { skip_samples(4); /* skip EAV */ prev_data = blanking_level for (samp=1; samp<=hsyncs_per_line; samp++) { huffman(sample_data - prev_data) prev_data = sample_data } skip_samples(4); /* skip SAV */ prev_data = blanking_level for (samp=1; samp<=samps_per_line; samp++) { huffman(sample_data - prev_data) prev_data = sample_data } } if (alignment()!=32) { stuffing_data /* '0' */ } } } </pre>	<p>8</p> <p>8</p> <p>8</p> <p>8</p> <p>18</p> <p>2-16</p> <p>2-16</p> <p>2-30</p>	<p>uimsbf</p> <p>uimsbf</p> <p>uimsbf</p> <p>uimsbf</p> <p>bsrtlb</p> <p>bsrtlb</p> <p>bsrtlb</p>

timestamp

The *timestamp* is a 32-bit field that indicates the value of the reference clock at the moment the first SDI sample of the payload enters the input channel.

sync_word

The synchronisation code.

sample_data

The SDI video data.

prev_data

The previous sample of the SDI video data of the same type (Cb, Y, or Cr) as the current sample.

stuffing_data

Data that is produced at the end of a frame only, with the purpose of aligning the **sync_word** of the next frame to a 32-bit boundary

To convert between the compressed Huffman format and one of the uncompressed SDI formats the **DtSdi::ConvertFrame** function (see page 207) can be used.

4.4. Transparent Mode Packets

Transparent mode is a combination of a tagged packet-oriented mode and raw mode. The tag stores for each packet a sequence-count number (required with PID filtering) and optionally a time stamp. The combination with raw mode enables detection of sync errors in parallel with performing timing analysis.

Transparent mode packets are generated by an input channel if the `DTAPI_RXMODE_STTRP` receive-mode is used (see `DtInpChannel::SetRxMode` page 118).

Syntax	#bits	Mnemonic
<code>transparent_packet() {</code>		
<code>if (timestamp_flag) {</code>		
<code>timestamp[7..0]</code>	8	uimbsf
<code>timestamp[15..8]</code>	8	uimbsf
<code>timestamp[23..16]</code>	8	uimbsf
<code>timestamp[31..24]</code>	8	uimbsf
<code>}</code>		
<code>for (i=0; i<204; i++) {</code>		
<code>payload_byte</code>	8	bslbf
<code>}</code>		
<code>sync_nibble /* '0101' */</code>	4	bslbf
<code>packet_sync</code>	1	bslbf
<code>reserved</code>	3	bslbf
<code>valid_count</code>	8	uimbsf
<code>sequence_count[7..0]</code>	8	uimbsf
<code>sequence_count[15..8]</code>	8	uimbsf
<code>}</code>		

timestamp

The *timestamp* is a 32-bit field that indicates the value of the reference clock at the moment the first byte of the payload enters the input channel.

payload_byte

The packet's payload. The number of valid bytes in the payload is indicated by the *valid_count* field. When *packet_sync* is '1' (packet synchronisation is achieved) the first payload byte will usually be 47h, but not necessarily! This is because an incidental error in the sync byte will not cause loss of synchronisation.

sync_nibble

The *sync_nibble* is a fixed 4-bit field whose value is '0101' (5).

packet_sync

When set to '1' this flag indicates that packet synchronisation has been achieved.

reserved

These bits are reserved for future use.

valid_count

This field indicates the number of valid payload bytes. If the *packet_sync* flag is set this field would indicate 188 or 204. If the *packet_sync* flag is not set the value can be anything between 1 and 204.

NOTE: if the number of bytes is less than 204 the value of remaining bytes is undefined.

sequence_count

The *sequence_count* is a 16-bit field that contains the original sequence number of the packet in the Transport Stream. The value of the sequence counter is only valid if *packet_sync* is '1'. Without PID filtering, *sequence_count* will be incremented with each received packet. When PID filtering is used, *sequence_count* can be used to retrieve the number of packets that has been skipped.

5. Ports and Hardware Functions per Device

Table 7. DekTec Devices – Ports and Hardware Functions

Device	Port	Index	ChanType	StreamType	Capabilities	Remark
DTA-100	1	0	OUTPUT	ASI_SDI	CAP_ASI	Doubly buffered
DTA-102	1	0	OUTPUT	TS_SPI		
DTA-105	1	0	OUTPUT	ASI_SDI	CAP_ASI	
	2	1	OUTPUT	ASI_SDI	CAP_ASI CAP_DBLBUF	
DTA-107	1	0	OUTPUT	TS_MOD	CAP_DVBS CAP_LBAND	Doubly buffered
DTA-107S2	1	0	OUTPUT	TS_MOD	CAP_DVBS CAP_DVBS2 CAP_LBAND	Doubly buffered
DTA-110	1	0	OUTPUT	TS_MOD	CAP_QAM CAP_UHF	Doubly buffered
DTA-110T	1	0	OUTPUT	TS_MOD	CAP_ATSC CAP_DVBT CAP_QAM CAP_UHF	CAP_DTMB is added when DTMB license is present CAP_ISDBT is added when ISDB-T license is present
DTA-112	1	0	INPUT	ASI_SDI	CAP_ASI CAP_BIDIR CAP_TRPMODE	Programmable input/output
	1	0	OUTPUT			
	2	0/1	OUTPUT	TS_MOD	CAP_QAM CAP_VHF CAP_UHF CAP_ADJLVL	
DTA-115	1	0	INPUT	ASI_SDI	CAP_ASI CAP_BIDIR	Programmable input/output
	1	0	OUTPUT			
	2	0/1	OUTPUT	TS_MOD	CAP_ATSC CAP_DVBT CAP_QAM CAP_VHF CAP_UHF CAP_ADJLVL	CAP_DTMB is added when DTMB license is present CAP_ISDBT is added when ISDB-T license is present
DTA-116	1	0	INPUT	ASI_SDI	CAP_ASI CAP_BIDIR	Programmable input/output
	1	0	OUTPUT			
	2	0/1	OUTPUT	TS_MOD	CAP_ATSC CAP_DVBT CAP_QAM CAP_VHF CAP_UHF CAP_IF CAP_DIGIQ	CAP_DTMB is added when DTMB license is present CAP_ISDBT is added when ISDB-T license is present

DTA-117	1	0	INPUT	ASI_SDI	CAP_ASI CAP_BIDIR	Programmable input/output
	1	0	OUTPUT			
	2	0/1	OUTPUT	TS_MOD	CAP_ATSC CAP_DVBT CAP_QAM CAP_VHF CAP_UHF CAP_IF CAP_DIGIQ	CAP_DTMB is added when DTMB license is present CAP_ISDBT is added when ISDB-T license is present
DTA-120	1	0	INPUT	ASI_SDI	CAP_ASI	
DTA-122	1	0	INPUT	TS_SPI		
DTA-124	1..4	0..3	INPUT	ASI_SDI	CAP_ASI CAP_SDI CAP_SDITIME	SDI time-stamping capability is supported in firmware version 2 or higher
DTA-140	1	0	INPUT	ASI_SDI	CAP_ASI	
	2	0	OUTPUT	ASI_SDI	CAP_ASI	Doubly buffered
DTA-145	1	0	INPUT	ASI_SDI	CAP_ASI CAP_BIDIR CAP_SDI CAP_SDITIME CAP_TRPMODE CAP_GENREF	Programmable input/output Transparent-packet-mode and SDI time-stamping capabilities are supported in firmware version 2 or higher
		0	OUTPUT			
	2	0/1	OUTPUT	ASI_SDI	CAP_ASI CAP_SDI CAP_DBLBUF CAP_LOOPTHR CAP_FAILSAFE CAP_GENLOCKED	Double-buffered and loop-through capabilities are available in firmware version 2 or higher
DTA-160	1..3	0..2	INPUT	ASI_SDI	CAP_ASI CAP_BIDIR CAP_SDI CAP_SDITIME CAP_TRPMODE CAP_DBLBUF CAP_LOOPTHR	Programmable input/output Transparent-packet, SDI time-stamping, double-buffered and loop-through mode are supported in firmware version 2 or higher
		0..2	OUTPUT			
	4	0	INPUT OUTPUT	TS_OVER_IP		Virtually unlimited number of parallel streams

DTA-2135	1	0	INPUT	TS_DEMOD	CAP_DVBT CAP_TRPMODE CAP_IF_ADC CAP_DIVERSITY DTAPI_CAP_UHF DTAPI_CAP_VHF	Programmable as normal receiver or as diversity receiver. In normal mode this channel provides additional access to the sampled down-converted IF.
	2	0	INPUT	TS_DEMOD	CAP_DVBT CAP_TRPMODE CAP_SHARED DTAPI_CAP_UHF DTAPI_CAP_VHF	Programmable as additional independent receiver or as an additional receiver connected to the antenna of channel 1. This port is unavailable when port 1 is configured in diversity mode.
DTA-2144	1..4	0..3	INPUT	ASI_SDI	CAP_ASI CAP_BIDIR CAP_SDI CAP_SDITIME CAP_TRPMODE CAP_GENREF CAP_GENLOCKED	Programmable input/output Transparent-packet-mode and SDI time-stamping capabilities supported
			OUTPUT			
DTA-2145	1	0	INPUT	ASI_SDI	CAP_ASI CAP_BIDIR CAP_SDI CAP_SDITIME CAP_TRPMODE CAP_GENREF	Programmable input/output Transparent-packet-mode and SDI time-stamping capabilities are supported in firmware version 2 or higher
		0	OUTPUT			
	2	0/1	OUTPUT	ASI_SDI	CAP_ASI CAP_SDI CAP_DBLBUF CAP_LOOPTHR CAP_FAILSAFE CAP_GENLOCKED	Double-buffered and loop-through capabilities are available in firmware version 2 or higher
DTE-3100	1	0	OUTPUT	ASI_SDI	CAP_ASI	Doubly buffered
DTE-3120	1	0	INPUT	ASI_SDI	CAP_ASI	