

# DTAPI

| **Advanced Demodulator API**

**REFERENCE**

July 2021



## Table of Contents

<b>Structures</b> .....	<b>3</b>	struct DtStreamSelPars .....	21
struct DtAtscStreamSelPars .....	4	struct DtT2MiStreamSelPars .....	24
struct DtAtsc3StreamSelPars .....	5	struct DtTransFuncPars .....	25
struct DtComplexFloat .....	6	<b>Callback Functions</b> .....	<b>26</b>
struct DtConstelPars .....	7	DtOutputRateChangedFunc .....	26
struct DtDabEtiStreamSelPars .....	8	DtReadIqFunc .....	27
struct DtDabStreamSelPars .....	9	DtWriteMeasFunc .....	28
struct DtDabFicStreamSelPars .....	10	DtWriteStreamFunc .....	29
struct DtDvbC2StreamSelPars .....	11	DtWriteStreamWithTimeFunc .....	30
struct DtDvbTStreamSelPars .....	12	<b>DtAdvDemod</b> .....	<b>31</b>
struct DtDvbT2StreamSelPars .....	13	DtAdvDemod .....	31
struct DtImpRespPars .....	14	DtAdvDemod::AttachVirtual .....	32
struct DtIsdbtStreamSelPars .....	15	DtAdvDemod::CloseStream .....	33
struct DtMeasurement .....	16	DtAdvDemod::GetStreamSelection .....	34
struct DtMerPars .....	18	DtAdvDemod::GetTsRateBps .....	35
struct DtQamStreamSelPars .....	19	DtAdvDemod::OpenStream .....	36
struct DtSpectrumPars .....	20	DtAdvDemod::RegisterCallback .....	37

Copyright © 2021 by DekTec Digital Video B.V.

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

## **Structures**

## struct DtAtscStreamSelPars

This structure specifies the selection parameters for a ATSC transport stream.

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

### Members

**DtAtscStreamSelPars** structure has no members

### Remark

No additional parameters are required to select a ATSC transport stream.

## struct DtAtsc3StreamSelPars

This structure specifies the selection parameters for an ATSC 3.0 stream. It is used in structure **DtStreamSelPars** to select a specific stream.

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

### Members

*m\_PlpId*

Unique identification of the data PLP within the ATSC 3.0 stream. The valid range is 0 ... 63 and **DTAPI\_ATSC3\_PLP\_ID\_AUTO**. The latter value specifies automatic selection of the first data PLP.

## struct DtComplexFloat

Structure describing a complex floating-point number.

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

### Members

*m\_Re*  
The real part of the complex floating-point number.

*m\_Im*  
The imaginary part of the complex floating-point number.

## struct DtConstelPars

This structure specifies the parameters for a stream of constellation points. It is used in structure **DtStreamSelPars**.

```
struct DtConstelPars
{
    int    m_Period;           // Minimum time between callbacks in ms
    int    m_ConstellationType; // Constellation per PLP (0) or carrier (1)
    int    m_Index;           // PLP-ID or index of the carrier
    int    m_MaxNumPoints;     // Maximum number of constellation points
};
```

### Members

#### *m\_Period*

The minimum time period in milliseconds between two calls of the constellation point callback function. This time period must be the same for all active constellation point streams.

#### *m\_ConstellationType*

Specifies whether constellation points apply to a single carrier or to a PLP.

Value	Meaning
0	Constellation points for the selected PLP or Preamble
1	Constellation points for the selected carrier

#### *m\_Index*

If *m\_ConstellationType* equals '1' it specifies the index of the carrier.

If *m\_ConstellationType* equals '0' it specifies the PLP-ID or Preamble.

Value	Meaning
DTAPI_ATSC3_CONSTEL_L1_BASIC	ATSC 3.0 L1-Basic constellation
DTAPI_ATSC3_CONSTEL_L1_DETAIL	ATSC 3.0 L1-Detail constellation

#### *m\_MaxNumPoints*

Maximum number of constellation points that will be passed through the constellation point callback function.

## struct DtDabEtiStreamSelPars

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

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

### Members

**DtDabEtiStreamSelPars** structure has no members

### Remark

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



## struct DtDabStreamSelPars

This structure specifies the selection parameters for a DAB sub-channel. It is used in structure **DtStreamSelPars** to select a specific stream.

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

### Members

*m\_BitrateKbps*

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

*m\_ErrProtLevel*

The valid range for UEP profile is: 1 ... 4. The valid range for EEP profile is: 1 ... 5.

*m\_ErrProtMode*

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

*m\_ErrProtOption*

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

*m\_StartAddress*

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

*m\_ExtractionMode*

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

## struct DtDabFicStreamSelPars

This structure specifies the selection parameters for a DAB Fast Information Channel (FIC). It is used in structure **DtStreamSelPars** to select a specific stream. The parameters are not used for selection but are passed in the `WriteStreamFunc()` callback function to indicate the parameters of the Fast Information Block that is passed.

```
struct DtDabFicStreamSelPars
{
    int    m_CifIndex;           // Index of the CIF in the DAB frame
    int    m_FibIndex;          // Index of this FIB
};
```

### Members

*m\_CifIndex*

Index of the Common Interleaved Frame (CIF) in the DAB frame to which this Fast Information Block (FIB) is associated

*m\_FibIndex*

Index of this Fast Information Block (FIB) in the group of FIBs that are associated to the same Common Interleaved Frame (CIF).

## struct DtDvbC2StreamSelPars

This structure specifies the selection parameters for a DVB-C2 stream. It is used in structure **DtStreamSelPars** to select a specific stream.

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

### Members

*m\_DSliceId*

Unique identification of the data slice within the DVB-C2 stream. Valid values are: 0 ... 255 and **DTAPI\_DVBC2\_DSLICE\_ID\_AUTO**. The latter value specifies automatic selection of the data slice. In this case the first data slice is selected.

*m\_PlpId*

Unique identification of the data PLP within the DVB-C2 stream. The valid range is 0 ... 255 and **DTAPI\_DVBC2\_PLP\_ID\_AUTO**. The latter value specifies automatic selection of the first data PLP.

*m\_CommonPlpId*

Unique identification of the common PLP within the DVB-C2 stream. It will be combined with the selected data PLP. The valid values are: 0 ... 255, **DTAPI\_DVBC2\_PLP\_ID\_NONE** and **DTAPI\_DVBC2\_PLP\_ID\_AUTO**.

The value **DTAPI\_DVBC2\_PLP\_ID\_NONE** indicates that no common PLP is selected. The value **DTAPI\_DVBC2\_PLP\_ID\_AUTO** indicates automatic selection of the common PLP.

## struct DtDvbTStreamSelPars

This structure specifies the selection parameters for a DVB-T transport stream. It is used in structure `DtStreamSelPars`.

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

### Members

### Remarks

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

## struct DtDvbT2StreamSelPars

This structure specifies the selection parameters for a DVB-T2 stream. It is used in structure **DtStreamSelPars** to select a specific stream.

```
struct DtDvbT2StreamSelPars
{
    int    m_PlpId;                // Data PLP ID
    int    m_CommonPlpId;          // Common PLP ID
};
```

### Members

*m\_PlpId*

Unique identification of the data PLP within the DVB-T2 stream. The valid range is 0 ... 255 and **DTAPI\_DVBT2\_PLP\_ID\_AUTO**. The value **DTAPI\_DVBT2\_PLP\_ID\_AUTO** specifies automatic selection of the PLP. In this case the first PLP is selected.

*m\_CommonPlpId*

Unique identification of the common PLP within the DVB-T2 stream. It will be combined with the selected data physical layer pipe. The valid values for *m\_CommonPlpId* are: 0 ... 255, **DTAPI\_DVBT2\_PLP\_ID\_NONE** and **DTAPI\_DVBT2\_PLP\_ID\_AUTO**.

The value **DTAPI\_DVBT2\_PLP\_ID\_NONE** specifies that no common PLP is used. The value **DTAPI\_DVBT2\_PLP\_ID\_AUTO** specifies automatic selection of the common PLP.

### Remarks

Multiple streams (PLPs) from a DVB-T2 stream can be selected simultaneously. However, selecting one or more PLPs cannot be combined with the selection of a T2-MI stream.

## struct DtImpRespPars

This structure specifies the parameters for an impulse-response stream. It is used in structure `DtStreamSelPars`.

```
struct DtImpRespPars
{
    int    m_Period;           // Minimum time between callbacks in ms
    int    m_Channel;         // Channel used for MISO
};
```

### Members

*m\_Period*

The minimum time period in milliseconds between two calls of the impulse-response callback function. This time period must be the same for all selected impulse-response and transfer-function streams.

*m\_Channel*

Specifies the MISO channel: '0' for TX1 and '1' for TX2. TX1 should be selected in case of a SISO input signal.

## struct DtIsdbtStreamSelPars

This structure specifies the selection parameters for an ISDB-T stream. It is used in structure **DtStreamSelPars** to select the ISDB-T stream.

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

### Members

**DtIsdbtStreamSelPars** structure has no members

### Remark

All layers are selected and output the same stream

## struct DtMeasurement

Structure describing a set of measurement values. It used to pass measurements from the advanced demodulator to the user application through user-supplied **DtWriteMeasFunc** callback.

```
struct DtMeasurement
{
    DtStreamType  m_MeasurementType;    // Type of measurement values
    __int64  m_TimeStamp;                // Timestamp in number of samples
    int  m_NumValues;                    // Number of measurement values
    DtComplexFloat*  m_pMeasurement;    // Measurement values
};
```

### Members

*m\_MeasurementType*

Type of measurement data.

Value	Meaning
STREAM_CONSTEL	Constellation points
STREAM_IMPRESP	Impulse response
STREAM_MER	MER
STREAM_SPECTRUM	Spectrum
STREAM_TF_ABS	Transfer function – Absolute value
STREAM_TF_PHASE	Transfer function – Phase
STREAM_TF_GROUPDELAY	Transfer function – Group delay
STREAM_TXID_IMPRESP	Impulse response of Tx-ID

*m\_TimeStamp*

Timestamp of measurement data. It is expressed in the number of I/Q samples processed since demodulator start up.

*m\_NumValues*

The number of measurement values in *m\_pMeasurement*.



### *m\_pMeasurement*

Pointer to a buffer of **DtComplexFloat** elements with length *m\_NumValues*. The meaning of the **DtComplexFloat** elements depends on the type of the measurement data. The buffer is allocated and released by the advanced demodulator.

Stream Type	DtComplexFloat.m_Re	DtComplexFloat.m_Im
<b>STREAM_CONSTEL</b>	X-coordinate	Y-coordinate
<b>STREAM_IMPRESP</b>	Delay ( $\mu$ s)	Relative power (db)
<b>STREAM_MER</b>	Frequency (MHz)	MER (db)
<b>STREAM_SPECTRUM</b>	Frequency (MHz)	Relative power (db)
<b>STREAM_TF_ABS</b>	Frequency (MHz)	Relative power (db)
<b>STREAM_TF_PHASE</b>	Frequency (MHz)	Phase (degrees)
<b>STREAM_TF_GROUPDELAY</b>	Frequency (MHz)	Group delay ( $\mu$ s)
<b>STREAM_TXID_IMPRESP</b>	Delay ( $\mu$ s)	Relative power (db)

## struct DtMerPars

This structure specifies the parameters for a MER stream. It is used in structure **DtStreamSelPars**.

```
struct DtMerPars
{
    int    m_Period;           // Minimum time between callbacks in ms
};
```

### Members

*m\_Period*

The minimum time period in milliseconds between two calls of the MER callback function.

*m\_ValueBool, m\_ValueDouble, m\_ValueInt, m\_pValue*

The value of the *DtPar.m\_ValueType* determines which parameter is used.

## struct DtQamStreamSelPars

This structure specifies the selection parameters for a QAM transport stream.

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

### Members

**DtQamStreamSelPars** structure has no members

### Remark

No additional parameters are required to select a QAM transport stream.

## struct DtSpectrumPars

This structure specifies the parameters for a spectrum stream. It is used in structure **DtStreamSelPars**.

```
struct DtSpectrumPars
{
    int    m_Period;           // Minimum time between callbacks in ms
    int    m_FftLength;       // FFT length, must be a power of tw
    int    m_AverageLength;   // Number of FFT blocks used for averaging
};
```

### Members

*m\_Period*

The minimum time period in milliseconds between two calls of the spectrum callback function.

*m\_FftLength*

FFT length, must be equal or greater than 32 and a power of 2.

*m\_AverageLength*

The number of FFT blocks on which the average is performed.

### Remarks

The processing time for creating the spectrum plot data is proportional to *m\_FftLength* \* *m\_AverageLength*.

## struct DtStreamSelPars

This structure is used in `DtAdvDemod::OpenStream()` to specify the parameters for the stream to be opened. It is also used as parameter in the callback function to identify the associated stream.

```
struct DtStreamSelPars
{
    intptr_t    m_Id;                // Unique stream identifier
    DtStreamType m_StreamType;        // Stream type
    union {
        // Selection parameters for demodulated streams
        DtAtsc3StreamSelPars m_Atsc3;    // ATSC 3.0 stream
        DtDabStreamSelPars   m_Dab;      // DAB stream
        DtDabEtiStreamSelPars m_DabEti;  // DAB Ensemble Transport Interface
        DtDabFicStreamSelPars m_DabFic;  // DAB Fast Information Channel
        DtDvbC2StreamSelPars  m_DvbC2;   // DVB-C2 stream
        DtDvbTStreamSelPars   m_DvbT;    // DVB-T stream
        DtDvbT2StreamSelPars  m_DvbT2;   // DVB-T2 stream
        DtIsdbtStreamSelPars  m_Isdbt;   // ISDB-T stream
        DtT2MiStreamSelPars   m_T2Mi;    // T2-MI stream

        // Parameters for streams of measurement values
        DtConstelPars         m_Constel;  // Constellation points
        DtImpRespSelPars      m_ImpResp;   // Impulse response
        DtMerPars             m_Mer;      // MER
        DtSpectrumPars        m_Spectrum; // Spectrum
        DtTransFuncPars       m_TransFunc; // Transfer function
    } u;
};
```

### Members

`m_Id`

Uniquely identifies the stream. The user can use any integer value or pointer, as long as the value is unique for each stream.

`m_StreamType`

Classifies the type of the stream.

Value	Meaning
<b>Stream types – Demodulated data</b>	
<b>STREAM_ATSC</b>	ATSC stream (transport stream packets)
<b>STREAM_ATSC3</b>	ATSC 3.0 stream (ATSC Link layer Protocol (ALP) packets)
<b>STREAM_ATSC3_BBFRAME</b>	ATSC 3.0 stream Base Band Frames (BB-frames)
<b>STREAM_DAB</b>	DAB stream
<b>STREAM_DABETI</b>	DAB Ensemble Transport Interface stream
<b>STREAM_DABFIC</b>	DAB Fast Information Channel stream
<b>STREAM_DVBC2</b>	DVB-C2 stream (transport stream packets)
<b>STREAM_DVBC2_BBFRAME</b>	DVB-C2 stream (BB-frames)
<b>STREAM_DVBC2_GSE</b>	DVB-C2 stream (GSE-packets)

<b>STREAM_DVBT</b>	DVB-T stream (transport stream packets)
<b>STREAM_DVBT2</b>	DVB-T2 stream (transport stream packets)
<b>STREAM_DVBT2_BBFRAME</b>	DVB-T2 stream (BB-frames)
<b>STREAM_DVBT2_GSE</b>	DVB-T2 stream (GSE-packets)
<b>STREAM_ISDBT</b>	ISDB-T stream (transport stream packets)
<b>STREAM_QAM</b>	QAM stream (QAM-A/B/C transport stream packets)
<b>STREAM_T2MI</b>	T2-MI stream (transport stream packets)
<b>Stream types – Measurement values</b>	
<b>STREAM_CONSTEL</b>	Constellation points
<b>STREAM_IMPRESP</b>	Impulse response
<b>STREAM_MER</b>	MER
<b>STREAM_SPECTRUM</b>	Spectrum
<b>STREAM_TF_ABS</b>	Transfer function absolute
<b>STREAM_TF_PHASE</b>	Transfer function phase
<b>STREAM_TF_GROUPDELAY</b>	Transfer function group delay
<b>STREAM_TXID_IMPRESP</b>	Impulse response of ATSC 3.0 Tx-ID

*u.m\_Atsc*

Structure used if *m\_Type* equals **STREAM\_ATSC**. See **DtAtscStreamSelPars** for the members.

*u.m\_Atsc3*

Structure used if *m\_Type* equals **STREAM\_ATSC3**. See **DtAtsc3StreamSelPars** for the members.

*u.m\_Constel*

Structure used if *m\_Type* equals **STREAM\_CONSTEL**. See **DtConstelPars** for the members.

*u.m\_Dab*

Structure used if *m\_Type* equals **STREAM\_DAB**. See **DtDabStreamSelPars** for the members.

*u.m\_DabEti*

Structure used if *m\_Type* equals **STREAM\_DABETI**. See **DtDabEtiStreamSelPars** for the members.

*u.m\_DabFic*

Structure used if *m\_Type* equals **STREAM\_DABFIC**. See **DtDabFicStreamSelPars** for the members.

*u.m\_DvbC2*

Structure used if *m\_Type* equals **STREAM\_DVBC2**. See **DtDvbC2StreamSelPars** for the members.

*u.m\_DvbT*

Structure used if *m\_Type* equals **STREAM\_DVBT**. See **DtDvbTStreamSelPars** for the members.

*u.m\_DvbT2*

Structure used if *m\_Type* equals **STREAM\_DVB\_T2**. See **DtDvbT2StreamSelPars** for the members.

*u.m\_ImpResp*

Structure used if *m\_Type* equals **STREAM\_IMP\_RESP**. See **DtImpRespPars** for the members.

*u.m\_Isdbt*

Structure used if *m\_Type* equals **STREAM\_ISDBT**. See **DtIsdbtStreamSelPars** for the members.

*u.m\_Mer*

Structure used if *m\_Type* equals **STREAM\_MER**. See **DtMerPars** for the members.

*u.m\_Qam*

Structure used if *m\_Type* equals **STREAM\_QAM**. See **DtQamStreamSelPars** for the members.

*u.m\_Spectrum*

Structure used if *m\_Type* equals **STREAM\_SPECTRUM**. See **DtSpectrumPars** for the members.

*u.m\_T2Mi*

Structure used if case *m\_Type* equals **STREAM\_T2MI**. See **DtT2MiSelPars** for the members.

*u.m\_TransFunc*

Structure used if *m\_Type* equals **STREAM\_TF\_ABS**, **STREAM\_TF\_PHASE** or **STREAM\_TF\_GROUPDELAY**. See **DtTransFuncPars** for the members.

## struct DtT2MiStreamSelPars

This structure specifies the selection parameters for a T2-MI transport stream containing a complete DVB-T2 stream. It is used in structure **DtStreamSelPars** to specify a selected stream.

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

### Members

*m\_T2MiOutPid*

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

*m\_T2MiTsRate*

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

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

### Remarks

If you select T2-MI transport stream, DTAPI (re)constructs a DVB-T2MI stream.

The idea behind this is that you can record a complete multi-PLP DVB-T2 stream as a T2MI-file. And you can playout the complete multi-PLP DVB-T2 stream afterwards. No timestamps are included.

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



## struct DtTransFuncPars

This structure specifies the parameters for a transfer-function stream. It is used in structure `DtStreamSelPars`.

```
struct DtTransFuncPars
{
    int    m_Period;           // Minimum time between callbacks in ms
    int    m_Channel;         // Channel used for MISO
};
```

### Members

*m\_Period*

The minimum time period in milliseconds between two calls of the transfer function callback function. This time period must be the same for all selected impulse-response and transfer-function streams.

*m\_Channel*

Specifies the MISO channel; '0' for TX1 and '1' for TX2. TX1 should be selected in case of SISO input signal.

## Callback Functions

### DtOutputRateChangedFunc

Prototype of a callback function to be supplied by the user with **DtAdvDemod::RegisterCallback**. The advanced demodulator invokes this callback function when the bitrate of the stream has changed.

```
void DtOutputRateChangedFunc(  
    [in] void* pOpaque,           // Opaque pointer  
    [in] DtStreamSelPars& StreamSel, // Stream selection parameters  
    [in] int Bitrate              // New bitrate in bps  
);
```

#### Parameters

*pOpaque*

The opaque pointer that was specified in **DtAdvDemod::RegisterCallback()**.

*StreamSel*

The stream selection parameters that were passed in **DtAdvDemod::OpenStream()**. This parameter can be used to identify the stream when multiple data streams are generated simultaneously.

*Bitrate*

New bitrate of the selected stream in bits per second.

#### Remarks

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

## DtReadIqFunc

Prototype of a callback function to be supplied by the user if he wants *virtual* I/Q input, this is supplying I/Q samples from a source other than a receiver device (e.g. from file). The advanced demodulator calls this function to obtain new I/Q samples.

```
void DtReadIqFunc(  
    [in] void* pOpaque,           // Opaque pointer  
    [in] unsigned char* pIqBuf,  // Buffer to write I/Q samples to  
    [in] int IqBufSize,          // Size of I/Q sample buffer in bytes  
    [out] int& IqLength           // Number of bytes written in pIqBuf  
);
```

### Parameters

*pOpaque*

The opaque pointer that was specified in `DtAdvDemod::AttachVirtual()`.

*pIqBuf*

Pointer to a buffer – allocated by the advanced demodulator – into which the user can write his I/Q samples. The I/Q samples shall be signed 16-bit integer in I, Q order.

*IqBufSize*

Size of *pIqBuf* in number of bytes. This is the maximum number of bytes that may be written into *pIqBuf*.

*IqLength*

Output argument that is to be set to the actual number of bytes written in *pIqBuf*.

## DtWriteMeasFunc

Prototype of a callback function to be supplied by the user with **DtAdvDemod::RegisterCallback**. The advanced demodulator calls this function when new measurement values are available. The user can process the measurements any way he likes, e.g. plot in a GUI or write to a file. If multiple streams of measurement values are used, they share a single callback function. The user must demultiplex the different measurement values using the stream selection parameters.

```
void DtWriteMeasFunc(  
    [in] void* pOpaque,           // Opaque pointer  
    [in] DtStreamSelPars& StreamSel, // Stream selection parameters  
    [in] DtMeasurement* pMeasurement // Buffer with measurement values  
);
```

### Parameters

*pOpaque*

The opaque pointer that was specified in **DtAdvDemod::RegisterCallback()**.

*StreamSel*

The stream selection parameters that were passed in **DtAdvDemod::OpenStream()**. This parameter can be used to identify the measurement values when multiple streams of measurement values are generated simultaneously.

*pMeasurement*

Pointer to a data buffer containing the measurement values.

### Remarks

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

## DtWriteStreamFunc

Prototype of a callback function to be supplied by the user with `DtAdvDemod::RegisterCallback`. The advanced demodulator calls this function when new demodulated stream data is available. The user can process the data any way he likes, e.g. analyse the stream in real time, write to a file, etc.

```
void DtWriteStreamFunc(  
    [in] void* pOpaque,           // Opaque pointer  
    [in] DtStreamSelPars& StreamSel, // Stream selection parameters  
    [in] const unsigned char* pData, // Demodulated data  
    [in] int Length               // Size in bytes of demodulated data  
);
```

### Parameters

*pOpaque*

The opaque pointer that was specified in `DtAdvDemod::RegisterCallback()`.

*StreamSel*

The corresponding stream selection parameters, passed in `DtAdvDemod::OpenStream()`.

*StreamSel*

The stream selection parameters that were passed in `DtAdvDemod::OpenStream()`. This parameter can be used to identify the stream when multiple data streams are generated simultaneously.

*pData*

Pointer to a buffer containing the demodulated data.

*Length*

Number of bytes available in the demodulated data buffer.

### Remarks

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

## DtWriteStreamWithTimeFunc

Prototype of a callback function to be supplied by the user with **DtAdvDemod::RegisterCallback**. The advanced demodulator calls this function when new demodulated stream data is available. The user can process the data any way he likes, e.g. analyse the stream in real time, write to a file, etc.

```
void DtWriteStreamWithTimeFunc (
    [in] void* pOpaque,           // Opaque pointer
    [in] DtStreamSelPars& StreamSel, // Stream selection parameters
    [in] const unsigned char* pData, // Demodulated data
    [in] int Length,              // Size in bytes of demodulated data
    [in] __int64 Timestamp        // 64-bit timestamp (54MHz clock)
);
```

### Parameters

*pOpaque*

The opaque pointer that was specified in **DtAdvDemod::RegisterCallback()**.

*StreamSel*

The corresponding stream selection parameters, passed in **DtAdvDemod::OpenStream()**.

*StreamSel*

The stream selection parameters that were passed in **DtAdvDemod::OpenStream()**. This parameter can be used to identify the stream when multiple data streams are generated simultaneously.

*pData*

Pointer to a buffer containing the demodulated data.

*Length*

Number of bytes available in the demodulated data buffer.

*m\_Timestamp*

A 64-bit timestamp, based on 54-MHz clock counts, indicating the arrival time of the data.

### Remarks

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

The timestamps are available when using one of the following stream types: **STREAM\_ATSC**, **STREAM\_ATSC3**, **STREAM\_DVBC2**, **STREAM\_DVBT**, **STREAM\_DVBT2**, **STREAM\_ISDBT** and **STREAM\_QAM**.

## **DtAdvDemod**

### **DtAdvDemod**

Class representing an advanced demodulator. **DtAdvDemod** can be considered a specialized input channel.

```
class DtAdvDemod;
```

Class **DtAdvDemod** is closely related to **DtInpChannel**. The following common methods are documented in the **DTAPI** documentation.

```
DtAdvDemod::AttachToPort ()  
DtAdvDemod::ClearFlags ()  
DtAdvDemod::Detach ()  
DtAdvDemod::GetDemodControl ()  
DtAdvDemod::GetDescriptor ()  
DtAdvDemod::GetFlags ()  
DtAdvDemod::GetIoConfig ()  
DtAdvDemod::GetPars ()  
DtAdvDemod::GetRxControl ()  
DtAdvDemod::GetStatistics ()  
DtAdvDemod::GetSupportedPars ()  
DtAdvDemod::GetTunerFrequency ()  
DtAdvDemod::LedControl ()  
DtAdvDemod::Reset ()  
DtAdvDemod::SetAntPower ()  
DtAdvDemod::SetDemodControl ()  
DtAdvDemod::SetIoConfig ()  
DtAdvDemod::SetPars ()  
DtAdvDemod::SetRxControl ()  
DtAdvDemod::SetTunerFrequency ()  
DtAdvDemod::Tune ()
```

## DtAdvDemod::AttachVirtual

Set up a *virtual* I/Q input channel that lets the user supply I/Q samples through a callback function (*pReadIqFunc*), instead of DTAPI reading the data from a physical receiver device. A DekTec device has to be specified (*pDtDvc*), but this device is used only for checking the **RX\_ADV** license.

```
DTAPI_RESULT DtAdvDemod::AttachVirtual(
    [in] DtDevice*  pDtDvc,           // DekTec device with RX_ADV license
    [in] DtReadIqFunc* pReadIqFunc, // Callback for supplying I/Q samples
    [in] void*      pOpaque          // Opaque pointer passed to callback
);
```

### Parameters

*pDtDvc*

DekTec device containing the **RX\_ADV** license. The **DtDevice** object must be attached to the device hardware. The device is used only for checking licenses.

*pReadIqFunc*

Pointer to the user-provided callback function that will supply I/Q samples to the advanced demodulator.

*pOpaque*

Opaque pointer that is passed to the callback function.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	The virtual I/Q input channel has been set up successfully
DTAPI_E_ATTACHED	The advanced demodulator is already attached
DTAPI_E_DEVICE	The <b>DtDevice</b> pointer is not valid or the <b>DtDevice</b> object is not attached to the device hardware

### Remarks

Virtual I/Q input channels enable usage of the advanced demodulator functions with I/Q samples from file, from non-DekTec I/Q sampling hardware or from computed I/Q samples.



## DtAdvDemod::CloseStream

Closes a stream.

```
DTAPI_RESULT DtAdvDemod::CloseStream(  
    [in] intptr_t  Id          // Identifier associated with the open stream  
);
```

### Parameters

*Id*

Identifies the stream that is to be closed. This is the value of the identifier that was specified in the stream selection parameters (`DtStreamSelPars.m_Id`).

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Stream is closed successfully
DTAPI_E_INVALID_MODE	Demodulator is not active
DTAPI_E_NOT_ATTACHED	Advanced demodulator object is not attached
DTAPI_E_NOT_FOUND	Stream is not open

## DtAdvDemod::GetStreamSelection

Returns all open streams.

```
DTAPI_RESULT DtAdvDemod::GetStreamSelection(  
    [out] std::vector<StreamSelPars> & StreamSelList // Open streams  
);
```

### Parameters

*StreamSelList*

A vector containing all open streams.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Open streams are returned successfully
DTAPI_E_INVALID_MODE	Demodulator is not active
DTAPI_E_NOT_ATTACHED	Advanced demodulator object is not attached

## DtAdvDemod::GetTsRateBps

Get the transport-stream rate of the stream with a given ID.

```
DTAPI_RESULT DtAdvDemod::GetTsRateBps(  
    [in] intptr_t Id          // Identifies the selected stream  
    [out] int& TsRate         // Transport-stream rate in bits per second  
);
```

### Parameters

*Id*

Identifies the selected stream. This is the value of the identifier that was specified in the stream selection parameters (`DtStreamSelPars.m_Id`).

*TsRate*

The transport stream rate, expressed in bits per second.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Transport-stream rate has been read successfully
DTAPI_E_NOT_ATTACHED	Advanced demodulator object is not attached
DTAPI_E_NOT_FOUND	Stream is not open
DTAPI_E_INVALID_MODE	Demodulator is not active

## DtAdvDemod::OpenStream

Opens the specified stream.

```
DTAPI_RESULT DtAdvDemod::OpenStream(  
    [in] DtStreamSelPars  StreamSel    // Stream selection  
);
```

### Parameters

*StreamSel*

Specifies a stream to open.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Stream is opened successfully
DTAPI_E_IN_USE	Stream identification ( <i>StreamSel.m_Id</i> ) is not unique
DTAPI_E_INVALID_ARG	Invalid stream selection parameter
DTAPI_E_INVALID_MODE	Demodulator is not active or demodulator does not match with stream selection (e.g. select a DVB-T stream while DVB-C2 demodulation is active)
DTAPI_E_NOT_ATTACHED	Advanced demodulator object is not attached

## DtAdvDemod::RegisterCallback

Register a callback function for handling demodulator data.

```
// Overload #1 -
// To be used for registering write measurement data callback function
DTAPI_RESULT DtAdvDemod::RegisterCallback(
    [in] DtWriteMeasFunc* pCallback,    // Callback function
    [in] void* pOpaque                // Opaque pointer for the callback
);
// Overload #2 -
// To be used for registering output bitrate changed callback function
DTAPI_RESULT DtAdvDemod::RegisterCallback(
    [in] DtOutputRateChangedFunc * pCallback, // Callback function
    [in] void* pOpaque                // Opaque pointer for the callback
);
// Overload #3 - To be used for registering stream-data callback function
DTAPI_RESULT DtAdvDemod::RegisterCallback(
    [in] DtWriteStreamFunc* pCallback,    // Callback function
    [in] void* pOpaque                // Opaque pointer for the callback
);
// Overload #4 - To be used for registering stream-data callback function
// with timestamps
DTAPI_RESULT DtAdvDemod::RegisterCallback(
    [in] DtWriteStreamWithTimeFunc* pCallback, // Callback function
    [in] void* pOpaque                // Opaque pointer for the callback
);
```

### Parameters

*pCallback*

Pointer to a callback function for handling the measurement-data, bitrate and stream-data. Use **NULL** to unregister the callback.

*pOpaque*

Opaque pointer that is passed to the callback function.

### Result

DTAPI_RESULT	Meaning
DTAPI_OK	Callback has been registered or unregistered successfully
DTAPI_E_NOT_ATTACHED	Advanced demodulator object is not attached