DTAPI – Matrix API

- ❑ Access full frames on line basis
- ❑ Audio de-embedding/embedding
- ❑ Easy access to ancillary data packets

FEATURES

- Extends DTAPI with classes for HD-SDI adapters supporting the frame buffer model
- Available for C++
- Same API classes and methods can be used on Windows and Linux

APPLICATION

- Logo-insertion
- Video editing
- Recording/streaming of HD-SDI

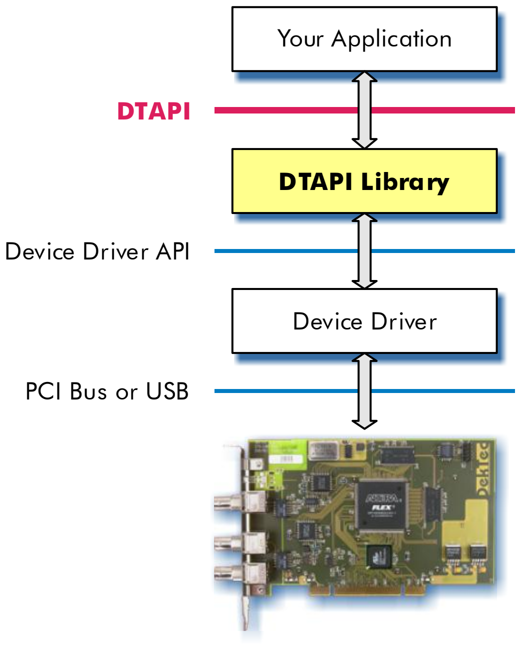


Table of Contents

DTAPI Revision History

Version	Date	Change Description
V4.14.2.157	2011.12.14	• First DTAPI with support for frame buffer model • General changes see *"C++ API for DekTec Devices"*

1. General Description

The DTAPI is the API that enables application programs to access the functions of DekTec devices in a uniform way. The basic concepts and object model of DTAPI are specified in "DTAPI – C++ *API for DekTec Devices*".

1.1. References

- DTAPI – *C++ API for DekTec Devices, DekTec Digital Video B.V., 2011.*
- *SMPTE-259*
- *SMPTE-274*
- *SMPTE-296*

2. Using the Matrix API

This section discusses the usage of the Matrix functions in the DTAPI library. Code snippets are provided to illustrate key methods.

2.1. Complete Example

3. DTAPI Methods for Matrix API

3.1. Overview

Table 1. DTAPI – Global Functions

API Function	Description
DtapiGetVidStdInfo	Get properties for a specific video standard

Table 2. DTAPI – `DtDevice` Functions

API Function	Description
DetectVidStd	Get detected video standard
GetGenlockState	Get current genlock state

Struct DtVidStdInfo

This structure describes a video standard (i.e. defines its properties).

```
struct DtVidStdInfo
{
  int   m_VidStd;              // Video standard
  bool   m_IsHd;               // HD (=true) or SD (=false)
  bool   m_IsInterlaced;       // Interlaced (=true) or progressive
                               // (=false)
  int   m_NumLines;            // Number of lines per frame
  int   m_Fps;                 // Framerate
  int   m_FrameNumSym;         // # of symbols per frame
  int   m_LineNumSym;          // # of symbols per line
  int   m_LineNumSymHanc;      // # of hanc symbols per line
  int   m_LineNumSymVanc;      // # of vanc symbols per line
  int   m_LineNumSymEav;       // # of EAV symbols per line
  int   m_LineNumSymSav;       // # of SAV symbols per line
  int   m_Field1StartLine;     // First line of field 1
  int   m_Field1EndLine;       // Last line of field 1
  int   m_Field1VidStartLine;  // First video line of field 1
  int   m_Field1VidEndLine;    // Last video line of field 1
  int   m_Field2StartLine;     // First line of field 2
  int   m_Field2EndLine;       // Last line of field 2
  int   m_Field2VidStartLine;  // First video line of field 2
  int   m_Field2VidEndLine;    // Last video line of field 2
};
```

Members

m_VidStd
Video standard described. See **::DtapiGetVidStdInfo** for a list of all possible standards.

m_IsHd
Indicates whether the standard has a HD (**true**) or SD (**false**) format.

m_IsInterlaced
Indicates whether the standard is interlaced (**true**) or progressive (**false**). For interlaced formats the field 2 (even field) members should be ignored.

m_NumLines
Number of SDI lines per frame

m_Fps
The frame rate

m_FrameNumSym
Total number of symbols in a frame

m_LineNumSym
Number of symbols per line

m_LineNumSymHanc
Number of HANC symbols per line (for HD, SUM of both streams)

m_LineNumSymVanc
Number of VANC symbols per line (for HD, SUM of both streams)

m_LineNumSymEav
Number of EAV symbols per line (for HD, SUM of both streams)

m_LineNumSymSav
Number of SAV symbols per line (for HD, SUM of both streams)

m_Field1StartLine
First line of field 1 (odd). NOTE: this is a 1 based index.

m_Field1EndLine
Last line of field 1 (odd). NOTE: this is a 1 based index.

m_Field1VidStartLine
First line of the active video section in field 1 (odd). NOTE: this is a 1 based index.

m_Field1VidEndLine
Last line of the active video section in field 1 (odd). NOTE: this is a 1 based index.

m_Field2StartLine
First line of field 2 (odd). NOTE: this is a 1 based index.

m_Field2EndLine
Last line of field 2 (odd). NOTE: this is a 1 based index.

m_Field2VidStartLine
First line of the active video section in field 2 (odd). NOTE: this is a 1 based index.

m_Field2VidEndLine
Last line of the active video section in field 2 (odd). NOTE: this is a 1 based index.

Struct DtBufferInfo

Structure describing the status of a frame buffer.

```
struct DtBufferInfo
{
   int   m_VidStd;               // Current video standard
   int   m_NumColums;            // Depth of buffer (in # frames/columns)
   __int64   m_NumReceived;      // # of received frames
   __int64   m_NumNumTransmitted; // # of frames transmitted
   __int64   m_NumDuplicated;    // # of duplicated frames
};
```

Members

m_VidStd
Video standard current set for the frame buffer.

m_NumColumns
Depth of the frame buffer in # frames/columns

m_NumReceived
Total # of frames received

m_NumTransmitted
Total # of frames transmitted

m_NumDuplicated
Total # of duplicated frames

Struct DtFrameInfo

Structure describing a frame in a frame buffer.

```
Struct DtFrameInfo
{
  int  m_VidStd;                 // Video standard
  __int64  m_Timestamp;          // Arrival timestamp
  __int64  m_FrameNumber;        // Seq # of frame
  __int64  m_Rp188;              // Extracted RP188 timestamp
};
```

Members

m_VidStd
Video standard

m_Timestamp
64-bit timestamp with the arrival time of the frame. The timestamp is derived from the free running reference counter clock on the board (see also **DtDevice::GetRefClkCnt**)

m_FrameNumber
64-bit sequence number assigned to the frame

m_Rp188
Extracted RP188 timestamp.

::DtapiGetVidStdInfo

Returns the properties for the specified video standard.

```
DTAPI_RESULT ::DtapiGetVidStdInfo(
  [in] int   VidStd           // Video standard
 [out] DtVidStdInfo  Info,    // Returns the properties
);
```

Parameters

VidStd

Video standard

Value	SMPTE	Resolution	FPS	Remark
DTAPI_VIDSTD_UNKNOWN	-	-	-	Unknown video standard
DTAPI_VIDSTD_SD_525_I59_94	SMPTE-259	720x480	29.97	Interlaced
DTAPI_VIDSTD_SD_625_I50	SMPTE-259	720x576	25.0	Interlaced
DTAPI_VIDSTD_HD_720_P23_98	SMPTE-296	1280x720	23.98	Progressive
DTAPI_VIDSTD_HD_720_P24	SMPTE-296	1280x720	24.0	Progressive
DTAPI_VIDSTD_HD_720_P25	SMPTE-296	1280x720	25.0	Progressive
DTAPI_VIDSTD_HD_720_P29_97	SMPTE-296	1280x720	29.97	Progressive
DTAPI_VIDSTD_HD_720_P30	SMPTE-296	1280x720	30.0	Progressive
DTAPI_VIDSTD_HD_720_P50	SMPTE-296	1280x720	50.0	Progressive
DTAPI_VIDSTD_HD_720_P59_94	SMPTE-296	1280x720	59.94	Progressive
DTAPI_VIDSTD_HD_720_P60	SMPTE-296	1280x720	60.0	Progressive
DTAPI_VIDSTD_HD_1080_P23_98	SMPTE-274	1920x1080	23.98	Progressive
DTAPI_VIDSTD_HD_1080_P24	SMPTE-274	1920x1080	24.0	Progressive
DTAPI_VIDSTD_HD_1080_P25	SMPTE-274	1920x1080	25.0	Progressive
DTAPI_VIDSTD_HD_1080_P30	SMPTE-274	1920x1080	30.0	Progressive
DTAPI_VIDSTD_HD_1080_P29_97	SMPTE-274	1920x1080	29.97	Progressive
DTAPI_VIDSTD_HD_1080_I50	SMPTE-274	1920x1080	25.0	Interlaced
DTAPI_VIDSTD_HD_1080_I59_94	SMPTE-274	1920x1080	29.97	Interlaced
DTAPI_VIDSTD_HD_1080_I60	SMPTE-274	1920x1080	30.0	Interlaced
DTAPI_VIDSTD_3G_1080_P50	SMPTE-274	1920x1080	50.0	Progressive
DTAPI_VIDSTD_3G_1080_P59_94	SMPTE-274	1920x1080	59.94	Progressive
DTAPI_VIDSTD_3G_1080_P60	SMPTE-274	1920x1080	60.0	Progressive

Info

This parameter receives the properties of the video standard.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Video properties have been returned
DTAPI_E_INVALID_VIDSTD	Invalid/unknown video standard was specified

::DtapiRxMode2VidStd

Helper function to convert a receive mode (i.e. **DTAPI_RXMODE_XXX**) to the corresponding a video standard, for use with **DtFrameBuffer::SetVidStd** and **DtMatrixr::SetVidStd**.

```
int ::DtapiRxMode2VidStd(
  [in] int   RxMode);          // receive mode
```

Parameters

RxMode

DTAPI_RXMODE_XXX value to be converted to a corresponding **DTAPI_VIDSTD_XXX**.

Result

Remarks

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

::DtapiTxMode2VidStd

Helper function to convert a transmit mode (i.e. **DTAPI_TXMODE_XXX**) to the corresponding a video standard, for use with **DtFrameBuffer::SetVidStd** and **DtMatrixr::SetVidStd**.

```
int ::DtapiTxMode2VidStd(
  [in] int   TxMode);          // transmit mode
```

Parameters

TxMode

DTAPI_TXMODE_XXX value to be converted to a corresponding **DTAPI_VIDSTD_XXX**.

Result

Remarks

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

::DtapiVidStd2RxMode

Helper function to convert a video standard to the corresponding receive mode (i.e. **DTAPI_RXMODE_XXX**), which can be used with **DtInpChannel::SetRxMode**.

```
int ::DtapiVidStd2RxMode(
  [in] int   VidStd);          // Video standard
```

Parameters

VidStd

DTAPI_VIDSTD_XXX value to be converted to a corresponding **DTAPI_RXMODE_XXX**.

Result

Remarks

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

::DtapiVidStd2TxMode

Helper function to convert a video standard to the corresponding transmit mode (i.e. **DTAPI_TXMODE_XXX**), which can be used with **DtOutpChannel::SetTxMode**.

```
int ::DtapiVidStd2TxMode(
  [in] int  VidStd);         // Video standard
```

Parameters

VidStd

DTAPI_VIDSTD_XXX value to be converted to a corresponding **DTAPI_TXMODE_XXX**.

Result

Remarks

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

AncPacket

Object representing an ancillary data packet.

```
class AncPacket {
      int  m_Did;              // Data identifier
      int  m_SdidOrDbn;        // Secondary data id / Data block number
      int  m_Dc;               // Data count
      int  m_Cs;               // Checksum
      unsigned short*  m_pUdw;   // User data words
};
```

Public members

m_Did
Data identifier for ancillary data packet

m_SdidOrDbn
Data block number or secondary data identifier depending on whether it is Type 1 or Type 2 packet

m_Dc
Data count (i.e. number of user words in the packet)

m_Cs
Checksum

m_pUdw
Pointer to buffer holding the user data words. Create/initialise this buffer using the **AncPacket::Create** method and destroy it using the **AncPacket::Destroy** method.

Remarks

None

AncPacket::Create

Allocates a buffer for the user data and optionally initialises the buffer from a supplied buffer with user data.

```
void AncPacket::Create (
  [in] int   NumWords          // Size of buffer to create
);
// OVERLOAD: just create from supplied buffer
void AncPacket::Create (
  [in] unsigned short*  pUserWords,  // init from this buffer
  [in] int   NumWords          // # of words to copy
);
```

Parameters

NumWords

Size (in # of words) of buffer to allocate.

pUserWords

Pointer to a buffer with data that should be copied to the **AncPacket** object.
NOTE: *m_Dc* will be initialised to *NumWords* in this case.

Remarks

None

AncPacket::Destroy

Destroys (frees) the allocated user data word buffer.

```
void AncPacket::Destroy ();
```

Remarks

None

AncPacket::Size

Returns the size of the user buffer (i.e. the maximum number of user words that can be stored in **`AncPacket::m_pUdw`**).

```
int AncPacket::Size () const;
```

Remarks

None

AncPacket::Type

Returns the type of packet (Type 1 or Type 2).

```
int AncPacket::Type () const;
```

Remarks

None

DtDevice::DetectVidStd

Detects the video standard currently applied to a specified physical input port.

```
DTAPI_RESULT DtDevice::DetectVidStd (
  [in] int   Port,             // Physical port number
 [out] int&  VidStd            // Detected video standard
);
```

Parameters

Port
Physical port number

VidStd
Returns the detected video standard. Refer to **::DtapiGetVidStdInfo** for a description of the possible values.

Result

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

Remarks

None

DtDevice::GetGenlockState

Detects the video standard currently applied to a specified physical input port.

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

Parameters

State

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

Value	Meaning
DTAPI_GENL_NO_REF	No reference input signal is detected on the input of the video clock generator
DTAPI_GENL_LOCKING	A valid reference input signal is detected on the input and internal PLLs are lock-ing to it
DTAPI_GENL_LOCKED	Full clock-lock has been achieved

RefVidStd

Returns the video standard set (with SetIoConfig) for the reference source. Refer to **::DtapiGetVidStdInfo** for a description of the possible values.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Genlock state has been returned
DTAPI_E_NOT_SUPPORTED	This function is not supported by the current hardware
DTAPI_E_DEV_DRIVER	Unexpected driver error

Remarks

None

DtFrameBuffer::AncAddAudio

Function for adding audio samples to the ancillary data area of the specified frame.

```
DTAPI_RESULT DtFrameBuffer::AncAddAudio (
  [in] __int64   Frame,        // Frame number
  [in] unsigned char*  pBuf, // Buffer with audio samples
 [i/o] int&   BufSize,         // Size of buffer
  [in] int   Format,           // Audio format
  [in] int   Channels,         // Valid channels
  [in] int   AudioGroup        // Audio group the samples should be added to
);
```

Parameters

Frame
Frame number of the SDI frame the audio should be added too.

pBuf
Buffer with the audio samples

BufSize
Size (in bytes) of the supplied buffer with audio samples. This parameter returns the number of bytes actually added from the buffer (can be less than the size of the buffer if max number of audio samples have been added to the frame).

Format

Specifies the format of the audio samples.

Value	Meaning
DTAPI_SDI_AUDIO_PCM16	16-bit PCM audio samples
DTAPI_SDI_AUDIO_PCM32	32-bit PCM audio samples (not supported at the moment)

Channels

Specifies the audio channels included in the buffer (can be OR-ed together).

Value	Meaning
DTAPI_SDI_AUDIO_CHAN1	Channel 1 is included
DTAPI_SDI_AUDIO_CHAN2	Channel 2 is included
DTAPI_SDI_AUDIO_CHAN3	Channel 3 is included
DTAPI_SDI_AUDIO_CHAN4	Channel 4 is included
DTAPI_SDI_AUDIO_CH_PAIR1	Channel pair 1 is included (= **DTAPI_SDI_AUDIO_CHAN1** | **DTAPI_SDI_AUDIO_CHAN2**)
DTAPI_SDI_AUDIO_CH_PAIR2	Channel pair 2 is included (= **DTAPI_SDI_AUDIO_CHAN3** | **DTAPI_SDI_AUDIO_CHAN4**)

AudioGroup

Specifies the audio group the samples should be added to.

Value	Meaning
DTAPI_SDI_AUDIO_GROUP1	Add samples to audio group 1
DTAPI_SDI_AUDIO_GROUP2	Add samples to audio group 2
DTAPI_SDI_AUDIO_GROUP3	Add samples to audio group 3
DTAPI_SDI_AUDIO_GROUP4	Add samples to audio group 4

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Audio samples have been added to the frame
DTAPI_E_NOT_STARTED	Cannot add audio while the **DtFrameBuffer** object is idle
DTAPI_E_NOT_ATTACHED	Cannot add audio as long as the **DtFrameBuffer** object is not attached to an output
DTAPI_E_INVALID_FORMAT	The specified format is invalid/not supported
DTAPI_E_VALID_CHANNEL	An unknown audio channel has been specified
DTAPI_E_INVALID_GROUP	An unknown audio group has been specified
DTAPI_E_BUF_TOO_SMALL	Buffer does not contain enough audio samples to fill the audio group. The min. number of bytes required is returned in the *BufSize* parameter.

Remarks

The audio samples will not be actually written to the frame buffer until the **`DtFrameBuffer::AncCommit`** method is called; until this time the audio samples are cached internally in the DTAPI and other changes can be made the ancillary data space of the frame (e.g. adding audio for another audio group or adding/deleting ancillary data packet).

If multiple channels are specified in the *`Channels`* parameter, then the **`AncAddAudio`** function expects the audio samples for the channels to be interleaved in memory. I.e. when **`DTAPI_SDI_AUDIO_CH_PAIR1`** is specified, the function expects: sample ch1, sample ch2, sample ch1, sample ch2, etc.

DtFrameBuffer::AncAddPacket

Function for adding ancillary data packet to the specified ancillary data space of a specific frame.

```
DTAPI_RESULT DtFrameBuffer::AncAddPacket (
  [in] __int64   Frame,         // Frame to add packet to
  [in] AncPacket&  AncPkt,     // Packet to add
  [in] int   Line,              // Line the packet should be added too
  [in] int   HancVanc,          // Add to HANC or VANC space
  [in] int   Stream             // Add to chrominance or luminance stream
);
```

Parameters

Frame
Frame number of the SDI frame the ancillary data packet should be added too.

AncPkt
Packet too add.

Line
Specifies the line the packet should be added too.

HancVanc

Specifies the ancillary data space in which the packet should be inserted.

Value	Meaning
DTAPI_SDI_HANC	Add to Horizontal ANC space
DTAPI_SDI_VANC	Add to Vertical ANC space

Stream

For HD video standards this parameter specifies the stream in which the packet should be inserted. For SD video standard this parameter should be set to -1.

Value	Meaning
DTAPI_SDI_CHROM	Add to chrominance stream
DTAPI_SDI_LUM	Add to luminance stream

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Packet was added to insertion list
DTAPI_E_NOT_STARTED	Can only be called if the **DtFrameBuffer** object has been started
DTAPI_E_NOT_ATTACHED	**DtFramebuffer** object is not attached to an output
DTAPI_E_INVALID_ANC	Invalid ancillary data space was specified
DTAPI_E_INVALID_STREAM	Invalid stream was specified
DTAPI_E_INVALID_LINE	Invalid line was specified

Remarks

The ancillary data packet will not be actually written to the frame buffer until the **DtFrameBuffer::AncCommit** method is called; until this time the ancillary data packets is cached internally in the DTAPI and other changes can be made the ancillary data space of the frame (e.g. adding audio for another audio group or adding/deleting ancillary data packet).

DtFrameBuffer::AncClear

Clear all existing data from the specified space ancillary data space.

```
DTAPI_RESULT DtFrameBuffer::AncClear (
  [in] __int64   Frame,         // Frame to clear
  [in] int   HancVanc,          // Hanc or Vanc data space
  [in] int   Stream             // stream to clear (HD-only)
);
```

Parameters

Frame

Sequence number of the frame to clear

HancVanc

Specifies which ancillary data space to clear.

Value	Meaning
DTAPI_SDI_HANC	HANC data space
DTAPI_SDI_VANC	VANC data space

Stream

Specifies which stream to clear. NOTE: this is an HD-only parameter and for SD this parameter should be set to -1.

Value	Meaning
DTAPI_SDI_CHROM	Chrominance stream
DTAPI_SDI_LUM	Luminance stream

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Existing data has been marked for deletion and will be deleted when **DtFrameBuffer::AncCommit** is called
DTAPI_E_NOT_STARTED	Can only be called if the **DtFrameBuffer** object has been started
DTAPI_E_NOT_ATTACHED	**DtFramebuffer** object is not attached to both an input and output
DTAPI_E_INVALID_ANC	Specified an invalid ancillary data space
DTAPI_E_INVALID_STREAM	Specified an invalid stream (use -1 for SD)

Remarks

This function can only be used for **DtFrameBuffer** object which are part of a matrix and have both an input and output attached to it (i.e. editing scenario); it will fail in all other cases.

Upon calling this function ancillary data space will not actually be cleared yet, the actual clearing takes place when the **`DtFrameBuffer::AncCommit`** method is called (see also remarks for **`AncCommit`**).

DtFrameBuffer::AncCommit

Commit changes made to ancillary data spaces.

```
DTAPI_RESULT DtFrameBuffer::AncCommit (
  [in] __int64   Frame         // Seq # of frame
);
```

Parameters

Frame
The sequence number of the frame for which changes need to be committed

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Changes have been committed
DTAPI_E_NOT_STARTED	This method can only be called when the **DtFrameBuffer** object has been started
DTAPI_E_NOT_ATTACHED	No output attached to the **DtFrameBuffer** object
DTAPI_E_INTERNAL	Unexpected internal DTAPI error was encountered
DTAPI_E_INVALID_FRAME	No ancillary data changes have been made for the specified frame or the frame number is invalid

Remarks

Upon calling this method the following sequence of events will be executed:

- all existing packets marked for clearing (via **AncClear** or **AncDelPacket**) will be removed;
- audio added via **AncAddAudio** will be embedded in the HANC space;
- new ancillary data packets added via **AncAddPacket** will be inserted in ancillary data spaces

DtFrameBuffer::AncDelPacket

Deletes specific ancillary data packets from a range of SDI lines.

```
DTAPI_RESULT DtFrameBuffer::AncDelPacket (
  [in] __int64  Frame,        // Frame number
  [in] int  DID,              // Ancillary packet Data-ID
  [in] int  SDID,             // Ancillary packet Secondary Data-ID
  [in] int  StartLine,        // first line to scan
  [in] int  NumLines,         // # of lines to scan
  [in] int  HancVanc,         // delete from hanc or vanc
  [in] int  Stream,           // stream to delete packet from (HD-only)
  [in] int  Mode              // deletion mode
);
```

Parameters

Frame

Sequence number of frame to delete the packets from

DID

Ancillary Data-ID of the packets to delete

SDID

Secondary Data-ID of the packet to delete. If not used set this parameter to -1.

StartLine

First line to scan for the specified ancillary data packets. 1 denotes the first line.

NumLines

Number of lines to delete the specified packet from. Use -1 for all lines beginning with *StartLine*.

HancVanc

Specifies which ancillary data space to delete the packet(s) from.

Value	Meaning
DTAPI_SDI_HANC	HANC data space
DTAPI_SDI_VANC	VANC data space

Stream

Specifies which stream to delete the packet()s from . NOTE: this is an HD-only parameter and for SD this parameter should be set to -1.

Value	Meaning
DTAPI_SDI_CHROM	Chrominance stream
DTAPI_SDI_LUM	Luminance stream

Mode

Specifies the deletion mode.

Value	Meaning
`DTAPI_ANC_MARK`	Mark the ancillary data packet for deletion (i.e. leave it in the ancillary data space, but set the DID to 0xFF)
`DTAPI_ANC_DELETE`	Delete the packet from the ancillary data stream

Result

DTAPI_RESULT	Meaning
`DTAPI_OK`	Packet have been deleted
`DTAPI_E_NOT_STARTED`	Can only be called if the **`DtFrameBuffer`** object has been started
`DTAPI_E_NOT_ATTACHED`	**`DtFramebuffer`** object is not attached to both an input and output
`DTAPI_E_INVALID_ANC`	Specified an invalid ancillary data space
`DTAPI_E_INVALID_STREAM`	Specified an invalid stream (use -1 for SD)
`DTAPI_E_INVALID_MODE`	An invalid deletion-mode has been specified

Remarks

Call **`AncCommit`** to commit the changes made by this method (see also remarks section for **`AncCommit`**).

DtFrameBuffer::AncGetAudio

Extracts the audio data from a frame.

```
DTAPI_RESULT DtFrameBuffer::AncGetAudio (
  [in] __int64   Frame,        // Seq. # of frame
  [in] unsigned char*  pBuf,  // Buffer to receive audio samples
 [i/o] int&  BufSize,          // Size of pBuf (in bytes) / # bytes returned
  [in] int   Format,           // Format of audio smaples
 [i/o] int&  Channels,         // Audio channel to get / channels returned
  [in] int   AudioGroup        // Audio group to get
);
```

Parameters

Frame
Sequence number of the frame to get the audio from.

pBuf
Pointer to the buffer to receive the audio samples. This buffer needs to be large enough to accommodate the maximum number of audio samples in a frame.

BufSize
Size (in bytes) of the *pBuf*. As output parameter it returns the actual number of bytes returned.

Format
Specifies the format (e.g. 16-bit PCM) of the audio samples. See **AncAddAudio** for possible values.

Channels
As input parameter, this parameter specifies the audio channels to return. As output parameter, this parameter returns which channels have actually been returned. See **AncAddAudio** for possible values.

AudioGroup
Specifies which audio group should be returned. See **AncAddAudio** for possible values.

Result

DTAPI_RESULT	Meaning
`DTAPI_OK`	Available audio samples have been returned
`DTAPI_E_NOT_STARTED`	Cannot get audio while the **`DtFrameBuffer`** object is idle
`DTAPI_E_NOT_ATTACHED`	Cannot get audio as long as the **`DtFrameBuffer`** object is not attached to an input
`DTAPI_E_INVALID_FORMAT`	The specified format is invalid/not supported
`DTAPI_E_VALID_CHANNEL`	An unknown audio channel has been specified
`DTAPI_E_INVALID_GROUP`	An unknown audio group has been specified
`DTAPI_E_BUF_TOO_SMALL`	Buffer is too small does to receive the audio samples. The min. number of bytes required is returned in the *`BufSize`* parameter.

Remarks

None

DtFrameBuffer::AncGetPacket

Gets ancillary data packet(s) from the specified ancillary data space in the frame.

```
DTAPI_RESULT DtFrameBuffer::AncGetPacket (
  [in] __int64   Frame,        // Seq # of frame
  [in] int   DID,              // Data-ID of packet(s) to get
  [in] int   SDID,             // Secondary Data-ID of packet(s) to get
 [i/o] AncPacket*  pAncPktBuf,  // Array of ancillary data packets
 [i/o] int&  NumPackets,       // [in] max. # packets to get / [out] #
                               // packets returned
  [in] int   HancVanc          // Get packet(s) from HANC or VANC area
  [in] int   Stream            // Get packet(s) from chrominance or luminance
                               // stream (HD-only)
);
```

Parameters

Frame
Sequence number of frame to get the packet(s) from

DID
Ancillary Data-ID of the packet(s) to get

SDID
Secondary Data-ID of the packet(s) to get. If not relevant set this parameter to -1.

pAncPktBuf
Array of **AncPacket** objects to receive the requested ancillary data packets

NumPackets
Max number of packets to get. As output, this parameter returns the actual number of packets returned.

HancVanc
Specifies the ancillary data space to get the packets from.

Value	Meaning
DTAPI_SDI_HANC	Get from Horizontal ANC space
DTAPI_SDI_VANC	Get from Vertical ANC space

Stream
For HD video standards this parameter specifies the stream to get the packet(s) from. For SD video standard this parameter should be set to -1.

Value	Meaning
DTAPI_SDI_CHROM	Get from chrominance stream
DTAPI_SDI_LUM	Get from luminance stream

Result

DTAPI_RESULT	Meaning
`DTAPI_OK`	Available packets have been returned
`DTAPI_E_NOT_ATTACHED`	Cannot call this method while **`DtFrameBuffer`** object has not been attached to an input
`DTAPI_E_NOT_STARTED`	Cannot call this method while **`DtFrameBuffer`** object is idle
`DTAPI_E_INVALID_BUF`	*`pAncPacket`* is invalid (i.e. NULL pointer)
`DTAPI_E_INVALID_ANC`	Specified an invalid ancillary data space
`DTAPI_E_INVALID_STREAM`	Specified an invalid stream (use -1 for SD)
`DTAPI_E_BUF_TOO_SMALL`	Not enough entries in *`pAncPacket`* for all ancillary data packets requested. The *`NumPackets`* parameter returns the number of entries needed

Remarks

None

DtFrameBuffer::AncReadRaw

Read raw ancillary data into a memory buffer.

```
DTAPI_RESULT DtFrameBuffer::AncReadRaw (
  [in] __int64   Frame,        // Seq # of frame
  [in] unsigned char*  pBuf,  // Buffer to receive data
 [i/o] int&  BufSize,          // Size of buffer / # bytes returned
  [in] int  DataFormat,        // Data format
  [in] int  StartLine,         // First line to read
  [in] int  NumLines,          // # of lines to read
  [in] int  HancVanc           // HANC or VANC space
);
```

Parameters

Frame
Sequence number of frame to read.

pBuf
Pointer to the destination buffer to receive the ancillary data from requested lines.

BufSize
Size of destination buffer in number of bytes. Also used as output variable, to return the number of bytes written to the buffer.

DataFormat
Specifies the requested data format.

Value	Meaning
DTAPI_SDI_8BIT	8-bit words, with the MSB 8-bit of a 10-bit SDI symbol (i.e. 2-LSB bits have been discarded)
DTAPI_SDI_10BIT	10-bit SDI symbols concatenated in memory
DTAPI_SDI_16BIT	16-bit words with LSB 10-bit = SDI symbols and MSB 6-bit = '0'

StartLine
Defines the first line to read. 1 denotes the first line.

NumLines
Defines the number of lines to read. Set to -1 to get all lines beginning with the *StartLine*. As output, this parameter returns the number of lines actually read.

HancVanc
Specifies the ancillary data space to read.

Value	Meaning
DTAPI_SDI_HANC	Get from Horizontal ANC space
DTAPI_SDI_VANC	Get from Vertical ANC space

Result

DTAPI_RESULT	Meaning
`DTAPI_OK`	Data has been read successfully
`DTAPI_E_INVALID_VIDSTD`	No (valid) video standard has been set yet (make sure **`DtFrameBuffer::SetVidStd`** has been called)
`DTAPI_E_NOT_ATTACHED`	The **`DtFrameBuffer`** object is not attached to an input
`DTAPI_E_INVALID_BUF`	Buffer pointer is invalid (e.g. *`pBuf`*==NULL or not aligned on a 64-bit boundary)
`DTAPI_E_BUF_TOO_SMALL`	The supplied buffer is too small to receive the requested number of lines (+ optional stuffing). *`BufSize`* returns the minimum buffer size required.
`DTAPI_E_INVALID_FORMAT`	Specified format is invalid/not supported
`DTAPI_E_INVALID_LINE`	*`StartLine`* or *`NumLines`* is invalid (i.e. out of range).
`DTAPI_E_INVALID_ANC`	Specified ANC space (*`HancVanc`*) is invalid/not supported
`DTAPI_E_INTERNAL`	Unexpected internal error occurred

Remarks

Use this method to get raw content HANC or VANC part of a line(s). You will need to parse the returned data yourself to extract individual ancillary data packets.

This method uses DMA transfers to read the ancillary data from the card; since all DMA transfers are 64-bit aligned there may be 1..7 stuffing bytes added to the end of the buffer (the stuffing bytes are included in the count returned by the *`BufSize`* parameter).

NOTE: This method can only be called if the **`DtFrameBuffer`** object has been attached to input and a video standard has been set.

DtFrameBuffer::AncWriteRaw

Write raw ancillary data to the frame-buffer.

```
DTAPI_RESULT DtFrameBuffer::AncWriteRaw (
  [in] __int64   Frame,        // Seq # of frame
  [in] unsigned char*  pBuf,  // Buffer wita data to write
 [i/o] int&  BufSize,          // Size of buffer / # bytes returned
  [in] int  DataFormat,        // Data format
  [in] int  StartLine,         // First line to write
  [in] int  NumLines,          // # of lines to read
  [in] int  HancVanc,          // Write to HANC or VANC space
);
```

Parameters

Frame
Sequence number of frame to write too.

pBuf
Pointer to a buffer holding the data to write.

BufSize
Size of the buffer in number of bytes. Also used as output variable, to return the number of bytes actually read from *pBuf* and written to the frame buffer.

DataFormat
Specifies data format of the data in *pBuf*.

Value	Meaning
DTAPI_SDI_8BIT	8-bit words, with the MSB 8-bit of a 10-bit SDI symbol (i.e. 2-LSB bits have been discarded)
DTAPI_SDI_10BIT	10-bit SDI symbols concatenated in memory
DTAPI_SDI_16BIT	16-bit words with LSB 10-bit = SDI symbols and MSB 6-bit = ‘0’

StartLine
Defines the first line to write too. 1 denotes the first line.

NumLines
Defines the number of lines to write. Set to -1 to write to all lines beginning with the *StartLine*. As output, this parameter returns the number of lines actually written too.

HancVanc

Specifies the ancillary data space to target.

Value	Meaning
DTAPI_SDI_HANC	Write to Horizontal ANC space
DTAPI_SDI_VANC	Write to Vertical ANC space

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Data has been written to the frame-buffer
DTAPI_E_INVALID_VIDSTD	No (valid) video standard has been set yet (make sure **DtFrameBuffer::SetVidStd** has been called)
DTAPI_E_NOT_ATTACHED	The **DtFrameBuffer** object is not attached to an input
DTAPI_E_INVALID_BUF	Buffer pointer is invalid (e.g. *pBuf*==NULL or not aligned on a 64-bit boundary)
DTAPI_E_BUF_TOO_SMALL	The supplied buffer is too small; it does not contain enough data to make up the number of lines (+ optional stuffing). *BufSize* returns the minimum buffer size expected.
DTAPI_E_INVALID_FORMAT	Specified format is invalid/not supported
DTAPI_E_INVALID_LINE	*StartLine* or *NumLines* is invalid (i.e. out of range).
DTAPI_E_INVALID_ANC	Specified ANC space (*HancVanc*) is invalid/not supported
DTAPI_E_INTERNAL	Unexpected internal error occurred

Remarks

Use this method to write raw data to the ancillary data space section of a line.

This method can only write complete lines (that is the HANC/VANC part of a line) and therefore *pBuf* should contain at least *NumLines* worth of data. For the HANC data space each line should start with an EAV and end with a SAV; a VANC line should contain only the data immediate starting after the SAV.

DMA transfers are used to write the ancillary data to the card; since all DMA transfers are 64-bit aligned it may be necessary add between 1and 7 stuffing bytes after the end of the last line to write (the content of these stuffing bytes does not matter as they will be flushed by the hardware).

NOTE: This method can only be called if the **DtFrameBuffer** object has been attached to input and a video standard has been set.

DtFrameBuffer::AttachToInput

Attach the **DtFrameBuffer** object to a physical input port.

```
DTAPI_RESULT DtFrameBuffer::AttachToInput (
  [in] DtDevice*  pDtDvc,     // Device object
  [in] int  Port,             // Port number
);
```

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 of the input port the **DtFrameBuffer** object should attach to.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	**DtFrameBuffer** object has been attached successfully to the port
DTAPI_E_ATTACHED	The **DtFrameBuffer** object has already been attached to an input or to an output
DTAPI_E_STARTED	Cannot attach while the **DtFrameBuffer** object has is started
DTAPI_E_INVALID_VIDSTD	No or an invalid video standard has been set
DTAPI_E_OUT_OF_MEM	Not enough memory resources available
DTAPI_E_INTERNAL	Unexpected internal DTAPI error occurred

Remarks

Before attaching an input to the **DtFrameBuffer** object you need to first set the video standard (**DtFrameBuffer::SetVidStd**).

If a **DtFrameBuffer** object is embedded in a **DtSdiMatrix** object you can attach both an input and one or more outputs to the same **DtFrameBuffer** object. If the object is used stand-alone you can only attach an input if no output is attached to the object.

DtFrameBuffer::AttachToOutput

Attach the **DtFrameBuffer** object to a physical output port.

```
DTAPI_RESULT DtFrameBuffer::AttachToOutput (
  [in] DtDevice*  pDtDvc,     // Device object
  [in] int   Port,            // Port number
  [in] int   Delay,           // Tx-delay
);
```

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 of the output port the **DtFrameBuffer** object should attach to.

Delay
Tx-delay in number of frames. This value determines the transmission buffer size. A larger delay relaxes the real-time requirements of an application but increases the delay between the frame being created / received and the frame being visible on the output. Specifying -1 will set the maximum delay.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	**DtFrameBuffer** object has been attached successfully to the port
DTAPI_E_ATTACHED	The **DtFrameBuffer** object has already been attached to an to this port or to an input port
DTAPI_E_STARTED	Cannot attach while the **DtFrameBuffer** object has is started
DTAPI_E_INVALID_VIDSTD	No or an invalid video standard has been set
DTAPI_E_OUT_OF_MEM	Not enough memory resources available
DTAPI_E_INTERNAL	Unexpected internal DTAPI error occurred

Remarks

Before attaching an output to the **DtFrameBuffer** object you need to first set the video standard (**DtFrameBuffer::SetVidStd**) the object should use.

If a **DtFrameBuffer** object is embedded in a **DtSdiMatrix** object you can attach both an input and one or more outputs to the same **DtFrameBuffer** object. If the object is used stand-alone you can only attach an input if no output is attached to the object.

DtFrameBuffer::Detach

Detaches all associated input and outputs from the **`DtFrameBuffer`** object.

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

Result

DTAPI_RESULT	Meaning
`DTAPI_OK`	Detach was successful
`DTAPI_E_NOT_ATTACHED`	**`DtFrameBuffer`** object is not attached to any input or output
`DTAPI_E_STARTED`	Cannot detach while the **`DtFrameBuffer`** object has is started

Remarks

None

DtFrameBuffer::GetBufferInfo

Retrieve configuration and statistics information for the frame-buffer.

```
DTAPI_RESULT DtFrameBuffer::GetBufferInfo (
 [out] DtBufferInfo&   Info,   // Buffer info
);
```

Parameters

Info

This parameter receives the frame buffer information (see **DtBufferInfo** structure definition for more details).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Buffer information was returned successfully

Remarks

None

DtFrameBuffer::GetCurFrame

Get the sequence number of the frame that is currently being received or transmitted

```
DTAPI_RESULT DtFrameBuffer::GetCurFrame (
 [out] __int64&   CurFrame,    // Seq # of current tx/rx frame
);
```

Parameters

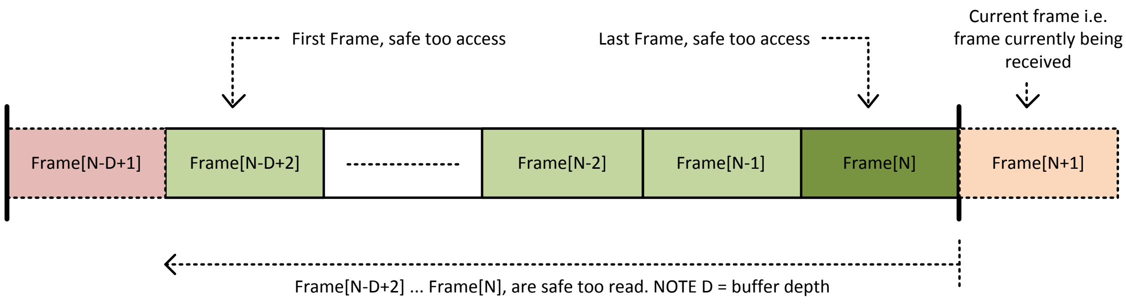
CurFrame
The sequence number of the frame currently being received or transmitted.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Current frame was returned
DTAPI_E_NOT_ATTACHED	**DtFrameBuffer** object must be attached to an input and/or output
DTAPI_E_EMBEDDED	**DtFrameBuffer** object is part of an **DtSdiMatrix** object and this method cannot be used; use **DtSdiMatrix::GetCurFrame** instead

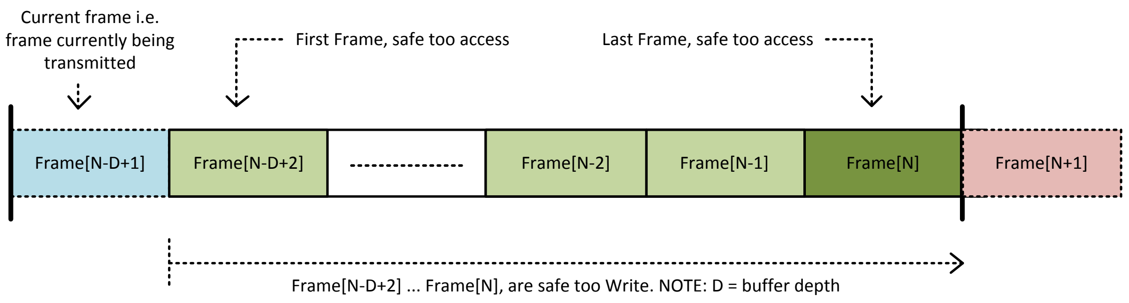
Remarks

In case the **DtFrameBuffer** object is operation in input mode (i.e. attached to an input) *CurFrame* indicates the frame that is currently being received, this means that it is safe to read frames numbers: *CurFrame* – (D – 1) ≤ Frame ≤ *CurFrame*-1, where D is the depth of the frame buffer (# columns in frame buffer.


First Frame, safe too access
Last Frame, safe too access
Current frame i.e. frame currently being received
Frame[N-D+1]
Frame[N-D+2]

Frame[N-2]
Frame[N-1]
Frame[N]
Frame[N+1]
Frame[N-D+2] ... Frame[N], are safe too read. NOTE D = buffer depth

In case of output mode *CurFrame* indicates the frame that is currently being transmitted (i.e. it is safe to write to the frames: *CurFrame*+1 ≤ Frame ≤ *CurFrame* + (D – 1)).


Current frame i.e. frame currently being transmitted
First Frame, safe too access
Last Frame, safe too access
Frame[N-D+1]
Frame[N-D+2]
Frame[N-2]
Frame[N-1]
Frame[N]
Frame[N+1]
Frame[N-D+2] ... Frame[N], are safe too Write. NOTE: D = buffer depth

NOTE: use **`DtFrameBuffer::GetBufInfo`** to determine the depth (#columns) of the frame-buffer.

DtFrameBuffer::GetFrameInfo

Retrieve information about a specific frame.

```
DTAPI_RESULT DtFrameBuffer::GetFrameInfo (
  [in] __int64   Frame,        // Seq # of frame to get info for
 [out] DtFrameInfo&  Info,    // Frame info object
);
```

Parameters

Frame
Frame number of the frame for which the information should be returned

Info
This parameter receives the frame information (see **DtFrameInfo** structure definition for more details).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Frame info was successfully retrieved

Remarks

None

DtFrameBuffer::ReadSdiLines

Read raw SDI lines into a memory buffer.

```
DTAPI_RESULT DtFrameBuffer::ReadSdiLines (
  [in] __int64   Frame,        // Seq # of frame to read
  [in] unsigned char*  pBuf, // Buffer to receive lines
 [i/o] int&   BufSize,         // [i] size of buffer / [o] # bytes returned
  [in] int   DataFormat,       // Desired data format
  [in] int   StartLine,        // First line to get
  [in] int&   NumLines         // # of lines to get
);

// OVERLOAD: read all lines (i.e. full frame)
DTAPI_RESULT DtFrameBuffer::ReadSdiLines (
  [in] __int64   Frame,        // Seq # of frame to read
  [in] unsigned char*  pBuf, // Buffer to receive lines
 [i/o] int&   BufSize,         // [i] size of buffer / [o] # bytes returned
  [in] int   DataFormat        // Desired data format
);
```

Parameters

Frame
Sequence number of frame to read.

pBuf
Pointer to the destination buffer to receive the requested lines.

BufSize
Size of destination buffer in number of bytes. Also used as output variable, to return the number of bytes written to the buffer.

DataFormat
Specifies the requested data format.

Value	Meaning
DTAPI_SDI_8BIT	8-bit words, with the MSB 8-bit of a 10-bit SDI symbol (i.e. 2-LSB bits have been discarded)
DTAPI_SDI_10BIT	10-bit SDI symbols concatenated in memory
DTAPI_SDI_16BIT	16-bit words with LSB 10-bit = SDI symbols and MSB 6-bit = '0'

StartLine
Defines the first line to read. 1 denotes the first line.

NumLines
Defines the number of lines to read. Set to -1 to get all lines beginning with the *StartLine*. As output, this parameter returns the number of lines actually read.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	The requested lines have been read
DTAPI_E_INVALID_VIDSTD	No (valid) video standard has been set yet (make sure **DtFrameBuffer::SetVidStd** has been called)
DTAPI_E_NOT_ATTACHED	The **DtFrameBuffer** object is not attached to an input
DTAPI_E_INVALID_BUF	Buffer pointer is invalid (e.g. *pBuf*==NULL or not aligned on a 64-bit boundary)
DTAPI_E_BUF_TOO_SMALL	The supplied buffer is too small to receive the requested number of lines (+ optional stuffing). *BufSize* returns the minimum buffer size required.
DTAPI_E_INVALID_FORMAT	Specified format is invalid/not supported
DTAPI_E_INVALID_LINE	*StartLine* or *NumLines* is invalid (i.e. out of range).
DTAPI_E_INTERNAL	Unexpected internal error occurred

Remarks

This method uses DMA transfers to read the SDI lines from the card; since all DMA transfers are 64-bit aligned there may be 1..7 stuffing bytes added to the end of the buffer (the stuffing bytes are included in the count returned by the *BufSize* parameter).

NOTE: This method can only be called if the **DtFrameBuffer** object has been attached to input and a video standard has been set.

DtFrameBuffer::ReadVideo

Read active video part of the specified lines into a memory buffer.

```
DTAPI_RESULT DtFrameBuffer::ReadVideo (
  [in] __int64   Frame,        // Seq # of frame to get
  [in] unsigned char*  pBuf,  // Buffer to receive video data
 [i/o] int&  BufSize,          // [i] size of buffer / [o] # bytes returned
  [in] int   Field,            // Field to get
  [in] int   Scaling,          // Desired scaling mode
  [in] int   DataFormat,       // Desired data format
  [in] int   StartLine,        // First line to get
  [in] int&  NumLines          // # of lines to get
);
```

Parameters

Frame
Sequence number of frame to read.

pBuf
Pointer to the destination buffer to receive the video lines.

BufSize
Size of destination buffer in number of bytes. Also used as output variable, to return the number of bytes written to the buffer.

Field
Specifies from which field the lines should be read.

Value	Meaning
DTAPI_SDI_FIELD1	Field 1 (=odd field or the only field for progressive)
DTAPI_SDI_FIELD2	Field 2 (=even field)

Scaling
Specifies whether the video should be scaled.

Value	Meaning
DTAPI_SCALING_OFF	Do not scale
DTAPI_SCALING_1_4	Scale video to 1/4th of its original size (i.e. half the vertical and horizontal size)
DTAPI_SCALING_1_16	Scale video to 1/16th of its original size (i.e. quarter the vertical and horizontal size)

NOTE: Scaling should only be used on the full field.

DataFormat

Specifies the requested data format.

Value	Meaning
DTAPI_SDI_8BIT	8-bit words, with the MSB 8-bit of a 10-bit SDI symbol (i.e. 2-LSB bits have been discarded)
DTAPI_SDI_10BIT	10-bit SDI symbols concatenated in memory
DTAPI_SDI_16BIT	16-bit words with LSB 10-bit = SDI symbols and MSB 6-bit = '0'

StartLine

Specifies the relative line, within the selected field, to read first. The value of 1 denotes the first line within the selected field.

NumLines

Specifies the number of lines to read. Set to -1 to get all lines beginning with the *StartLine*. As output, this parameter returns the number of lines actually read.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Requested lines have been retrieved
DTAPI_E_INVALID_VIDSTD	No (valid) video standard has been set yet (make sure **DtFrameBuffer::SetVidStd** has been called)
DTAPI_E_NOT_ATTACHED	The **DtFrameBuffer** object is not attached to an input
DTAPI_E_INVALID_BUF	Buffer pointer is invalid (e.g. *pBuf*==NULL or not aligned on a 64-bit boundary)
DTAPI_E_BUF_TOO_SMALL	The supplied buffer is too small to receive the requested number of lines (+ optional stuffing). *BufSize* returns the minimum buffer size required.
DTAPI_E_INVALID_FORMAT	Specified format is invalid/not supported
DTAPI_E_INVALID_LINE	*StartLine* or *NumLines* is invalid (i.e. out of range).
DTAPI_E_INTERNAL	Unexpected internal error occurred
DTAPI_E_INVALID_FIELD	Invalid/unsupported field specified. NOTE: for progressive frames there is no Field 2, so Field 1 is the only valid field.
DTAPI_E_INVALID_MODE	Invalid/unsupported scaling mode specified

Remarks

This method uses DMA transfers to read the SDI lines from the card; since all DMA transfers are 64-bit aligned there may be 1..7 stuffing bytes added to the end of the buffer (the stuffing bytes are included in the count returned by the *BufSize* parameter).

When retrieving scaled video the number of lines returned by the *NumLines* parameter denotes the number of un-scaled lines (i.e. for **DTAPI_SCALING_1_4** the number of scaled lines in *pBuf* is

NumLines/2 and for **DTAPI_SCALING_1_16** it is *NumLines*/4). Also note that scaling should only be used on the full field (i.e. *StartLine*=1 and *NumLines*=-1).

NOTE: This method can only be called if the **DtFrameBuffer** object has been attached to input and a video standard has been set.

DtFrameBuffer::SetVidStd

Set the video standard for the **DtFrameBuffer** object. The first action after creation of a **FrameBuffer** object is to set the video standard.

```
DTAPI_RESULT DtFrameBuffer::SetVidStd(
  [in] int   VidStd,           // Video Standard
);
```

Parameters

VidStd

Specifies the video standard of frames received/transmitted by the **DtFrameBuffer** object. Refer to **::DtapiGetVidStdInfo** for a description of the possible values.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Video standard has been set
DTAPI_E_STARTED	Cannot change the video while the **DtFrameBuffer** object is started
DTAPI_E_ATTACHED	Cannot be attached to an input and output (i.e. set video standard before attaching an input or output).
DTAPI_E_INVALID_VIDSTD	The specified video standard is invalid/not supported
DTAPI_E_OUT_OF_MEM	Not enough memory resources available

Remarks

When the **DtFrameBuffer** object is part of an SDI matrix please use the **DtSdiMatrix::SetVideoStd** method to set the standard.

DtFrameBuffer::Start

Start/stop receiving or transmitting frames.

```
DTAPI_RESULT DtFrameBuffer::Start (
  [in] bool   Start,          // true=start tx/rx; false=stop tx/rx
);
```

Parameters

Start

Set to true to begin receiving/transmitting frames and set to false to stop reception/transmission.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	**DtFrameBuffer** object has started/stopped
DTAPI_E_INVALID_VIDSTD	No (valid) video standard has been set yet (make sure **DtFrameBuffer::SetVidStd** has been called)
DTAPI_E_NOT_ATTACHED	**DtFrameBuffer** object is not attached to an input and/or output.

Remarks

NOTE: This method can only be called if the **DtFrameBuffer** object has been attached to input and a video standard has been set.

DtFrameBuffer::WaitFrame

Wait's for the next frame to be transmitted/received and returns the range of frames which are available/safe too access.

```
DTAPI_RESULT DtFrameBuffer::WaitFrame (
 [out] __int64&   FirstFrame, // First 'safe' frame
 [out] __int64&   LastFrame,  // Last 'safe' frame
);
// OVERLOAD: returns just the last 'safe' frame
DTAPI_RESULT DtFrameBuffer::WaitFrame (
 [out] __int64&   LastFrame,  // Last 'safe' frame
);
```

Parameters

FirstFrame
Sequence number of the first frame in the 'safe area'. The safe area is the range of frames, in the frame buffer, which are safe to read from or write to (i.e. the frames which are not currently being transmitted or received).

LastFrame
Sequence number of the last frame in the 'safe area'.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Wait was successful
DTAPI_E_NOT_ATTACHED	**DtFrameBuffer** object must be attached to an input and/or output
DTAPI_E_EMBEDDED	**DtFrameBuffer** object is part of an **DtSdiMatrix** object and this method cannot be used; use **DtSdiMatrix::WaitFrame** instead
DTAPI_E_DEV_DRIVER	Wait failed due to internal driver error
DTAPI_E_TIMEOUT	This function will wait a maximum of 100ms for a new frame after which it timeouts and returns this error.

Remarks

This function returns immediately after the hardware has transmitted (in case of output) or received (in case of input) a new frame. The safe area returned by this function is valid for one frame period (e.g. 40ms for 25fps).

When the **DtFrameBuffer** object is operation in output mode *FirstFrame* is the first of the 'safe area' to be transmitted, meaning that you will have the least amount of time to make sure that this frame is up to date.

NOTE: refer to description **DtFrameBuffer::GetCurFrame** of for more details about the 'safe area'.

In input mode the `LastFrame` is the most recently received frame and the `FirstFrame` is the eldest frame in the 'safe area'. As for output mode you will have the least amount of time to access the first frame as the frame buffer it is stored in is the first be overwritten.

DtFrameBuffer::WriteSdiLines

Write RAW SDI lines to the frame buffer.

```
DTAPI_RESULT DtFrameBuffer::WriteSdiLines (
  [in] __int64   Frame,        // Seq # of frame to write too
  [in] unsigned char*  pBuf, // Buffer with data to write
 [i/o] int&   BufSize,         // [i] size of buffer / [o] # of bytes written
  [in] int   DataFormat,       // Format of data in buffer
  [in] int   StartLine,        // First line to write too
  [in] int&   NumLines         // # of lines to write
);

// OVERLOAD: write all lines (i.e. full frame)
DTAPI_RESULT DtFrameBuffer::WriteSdiLines (
  [in] __int64   Frame,        // Seq # of frame to write too
  [in] unsigned char*  pBuf, // Buffer with data to write
 [i/o] int&   BufSize,         // [i] size of buffer / [o] # of bytes written
  [in] int   DataFormat,       // Format of data in buffer
);
```

Parameters

`Frame`
Sequence number of frame to write.

`pBuf`
Pointer to the source buffer to with the lines to write.

`BufSize`
Size of source buffer in number of bytes. Also used as output variable, to return the number of bytes read from the buffer.

`DataFormat`
Specifies the data format of the lines in the source buffer.

Value	Meaning
DTAPI_SDI_8BIT	8-bit words, with the MSB 8-bit of a 10-bit SDI symbol (i.e. 2-LSB bits have been discarded)
DTAPI_SDI_10BIT	10-bit SDI symbols concatenated in memory
DTAPI_SDI_16BIT	16-bit words with LSB 10-bit = SDI symbols and MSB 6-bit = '0'

`StartLine`
Defines the first line to write. 1 denotes the first line in the frame (i.e. first line of Field 1).

`NumLines`
Defines the number of lines to write. Set to -1 to write all lines beginning with the `StartLine`. As output, this parameter returns the number of lines actually written.

Result

DTAPI_RESULT	Meaning
`DTAPI_OK`	The lines have been written to the frame-buffer on the card
`DTAPI_E_INVALID_VIDSTD`	No (valid) video standard has been set yet (make sure **`DtFrameBuffer::SetVidStd`** has been called)
`DTAPI_E_NOT_ATTACHED`	The **`DtFrameBuffer`** object is not attached to an output
`DTAPI_E_INVALID_BUF`	Buffer pointer is invalid (e.g. *`pBuf`*==NULL or not aligned on a 64-bit boundary)
`DTAPI_E_BUF_TOO_SMALL`	The supplied buffer is too small; it does not contain enough data to make up the number of lines (+ optional stuffing). *`BufSize`* returns the minimum buffer size expected.
`DTAPI_E_INVALID_FORMAT`	Specified format is invalid/not supported
`DTAPI_E_INVALID_LINE`	*`StartLine`* or *`NumLines`* is invalid (i.e. out of range).
`DTAPI_E_INTERNAL`	Unexpected internal error occurred

Remarks

This method uses DMA transfers to write the SDI lines to the card; since all DMA transfers are 64-bit aligned it may be necessary add between 1and 7 stuffing bytes after the end of the last line to write (the content of the stuffing bytes does not matter as they will be flushed by the hardware).

NOTE: This method can only be called if the **`DtFrameBuffer`** object has been attached to output and a video standard has been set.

DtFrameBuffer::WriteVideo

Write the active video part of the specified lines to the frame buffer.

```
DTAPI_RESULT DtFrameBuffer::WriteVideo (
  [in] __int64   Frame,        // Seq # of frame to write too
  [in] unsigned char*  pBuf,   // Buffer with data to write
 [i/o] int&   BufSize,         // [i] size of buffer / [o] # bytes written
  [in] int   Field,            // Field to write to
  [in] int   DataFormat,       // Format of data in buffer
  [in] int   StartLine,        // First line to write to
 [i/o] int&   NumLines,        // # of lines to write
);
```

Parameters

Frame
Sequence number of frame to write too.

pBuf
Pointer to the source buffer to containing the video lines to be written to the frame buffer.

BufSize
Size of source buffer in number of bytes. Also used as output variable, to return the actual number of bytes read from the source buffer.

Field
Specifies to which field the lines should be written.

Value	Meaning
DTAPI_SDI_FIELD1	Field 1 (=odd field or the only field for progressive)
DTAPI_SDI_FIELD2	Field 2 (=even field)

DataFormat
Specifies the format of the video data in the source buffer.

Value	Meaning
DTAPI_SDI_8BIT	8-bit words, with the MSB 8-bit of a 10-bit SDI symbol (i.e. 2-LSB bits have been discarded)
DTAPI_SDI_10BIT	10-bit SDI symbols concatenated in memory
DTAPI_SDI_16BIT	16-bit words with LSB 10-bit = SDI symbols and MSB 6-bit = '0'

StartLine
Specifies the relative line, within the selected field, to write too first. The value of 1 denotes the first line within the selected field.

NumLines
Specifies the number of lines to write. Set to -1 to write to all lines beginning with the *StartLine*. As output, this parameter returns the number of lines actually written.

Result

DTAPI_RESULT	Meaning
`DTAPI_OK`	Specified lines have been written to the frame buffer
`DTAPI_E_INVALID_VIDSTD`	No (valid) video standard has been set yet (make sure **`DtFrameBuffer::SetVidStd`** has been called)
`DTAPI_E_NOT_ATTACHED`	The **`DtFrameBuffer`** object is not attached to an output
`DTAPI_E_INVALID_BUF`	Buffer pointer is invalid (e.g. *`pBuf`*==NULL or not aligned on a 64-bit boundary)
`DTAPI_E_BUF_TOO_SMALL`	The supplied buffer is too small; it does not contain enough data to make up the number of lines (+ optional stuffing). *`BufSize`* returns the minimum buffer size expected.
`DTAPI_E_INVALID_FORMAT`	Specified format is invalid/not supported
`DTAPI_E_INVALID_LINE`	*`StartLine`* or *`NumLines`* is invalid (i.e. out of range).
`DTAPI_E_INTERNAL`	Unexpected internal error occurred
`DTAPI_E_INVALID_FIELD`	Invalid/unsupported field specified. NOTE: for progressive frames there is no Field 2, so Field 1 is the only valid field.

Remarks

This method uses DMA transfers to write the SDI lines to the card; since all DMA transfers are 64-bit aligned it may be necessary add between 1and 7 stuffing bytes after the end of the last line to write (the content of the stuffing bytes does not matter as they will be flushed by the hardware).

DtSdiMatrix::Attach

Attach to the specified device.

```
DTAPI_RESULT DtSdiMatrix::Attach (
  [in] DtDevice*  pDvc,        // device to attach too
 [out] int&  MaxNumRows,       // max # of rows
);
```

Parameters

pDvc

Pointer to the device object to attach to.

MaxNumRows

Returns the maximum number of rows that are supported for this device.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_ATTACHED	**DtSdiMatrix** object is already attached
DTAPI_E_INVALID_ARG	*pDvc* pointer is NULL
DTAPI_E_NOT_ATTACHED	The **DtDevice** object pointed to by *pDvc* is not attached
DTAPI_E_NOT_SUPPORTED	Matrix functionality is not sup[ported for the supplied device

Remarks

None

DtSdiMatrix::Detach

Detach from the hardware.

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

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success

Remarks

None

DtMatrix::GetCurFrame

Get the sequence number of the frame that is currently being received or transmitted

```
DTAPI_RESULT DtMatrix::GetCurFrame (
 [out] __int64&  CurFrame,    // Seq # of current tx/rx frame
);
```

Parameters

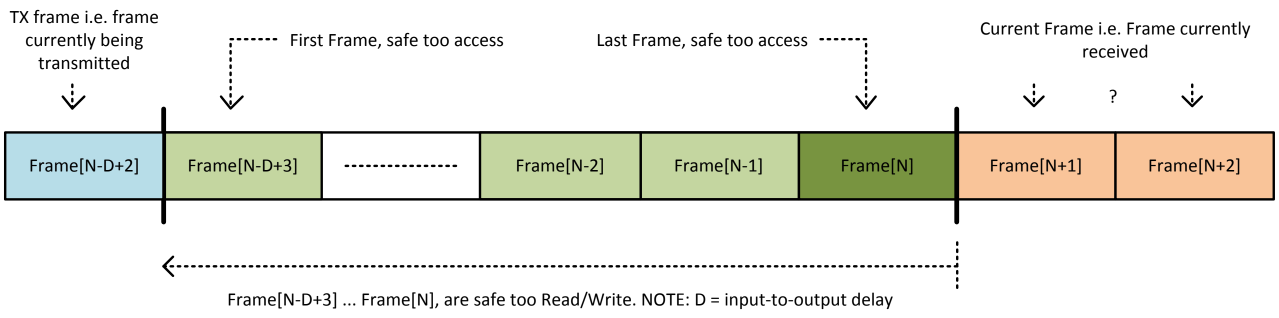
CurFrame
The sequence number of the frame currently being received.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Current frame was returned
DTAPI_E_NOT_ATTACHED	**DtMatrix** object must be attached

Remarks

A matrix consists out of one or more rows with one input and optionally also one or more outputs associated to each row. An output has an input-to-out associated with it and this delay determines how many frames delay there is available for an application to process frames. The figure below shows the relationship between the current frame, the TX frame (frame being transmitted) and the frames which are safe too access (see also **DtMatrix::WaitFrame**).


TX frame i.e. frame currently being transmitted
First Frame, safe too access
Last Frame, safe too access
Current Frame i.e. Frame currently received
?
Frame[N-D+2]
Frame[N-D+3]

Frame[N-2]
Frame[N-1]
Frame[N]
Frame[N+1]
Frame[N+2]
Frame[N-D+3] ... Frame[N], are safe too Read/Write. NOTE: D = input-to-output delay

DtSdiMatrix::GetMatrixInfo

Retrieve the configuration of the matrix.

```
DTAPI_RESULT DtSdiMatrix::GetMatrixInfo (
  [in] DtMatrixInfo&  Info,  // receives matrix info
);
```

Parameters

Info

This parameter receives the matrix information (see **DtMatrixInfo** structure definition for more details).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_INVAILD_VIDSTD	Cannot call this method until a video standard has been set (**DtSdiMatrix::SetVidStd**)

Remarks

None

DtSdiMatrix::Row

Returns the **`DtFrameBuffer`** object associated with a specific row in the matrix.

```
DtFrameBuffer& DtSdiMatrix::Row (
  [in] int   n,                  // index of row to get
);
```

Parameters

n

Zero-based index the row to get

Result

DTAPI_RESULT	Meaning
`DTAPI_OK`	

Remarks

None

DtSdiMatrix::SetVidStd

Set the video standard for the **DtSdiMatrix** object (i.e. all embedded **DtFrameBuffer** object). The first action after creation of a **DtSdiMatrix** object is to set the video standard.

```
DTAPI_RESULT DtSdiMatrix::SetVidStd(
  [in] int   VidStd,           // Video Standard
);
```

Parameters

VidStd

Specifies the video standard of frames received/transmitted by the **DtFrameBuffer** object. Refer to **::DtapiGetVidStdInfo** for a description of the possible values.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Video standard has been set
DTAPI_E_STARTED	Cannot change the video while the **DtFrameBuffer** object is started
DTAPI_E_ATTACHED	Cannot be attached to an input and output (i.e. set video standard before attaching an input or output).
DTAPI_E_INVALID_VIDSTD	The specified video standard is invalid/not supported
DTAPI_E_OUT_OF_MEM	Not enough memory resources available

Remarks

None

DtSdiMatrix::Start

Start/stop receiving and transmitting of frames.

```
DTAPI_RESULT DtSdiMatrix::Start (
  [in] bool   Start,           // true=start tx/rx; false=stop tx/rx
);
```

Parameters

Start
Set to true to start reception/transmission and set to false to stop reception/transmission.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_INVALID_VIDSTD	No (valid) video standard has been set yet (make sure **DtSdiMatrix::SetVidStd** has been called)

Remarks

None.

DtMatrix::WaitFrame

Wait's for the next frame to be received and returns the range of frames which are available/safe too access.

```
DTAPI_RESULT DtMatrix::WaitFrame (
 [out] __int64&   FirstFrame, // First 'safe' frame
 [out] __int64&   LastFrame,  // Last 'safe' frame
);

// OVERLOAD: returns just the last 'safe' frame
DTAPI_RESULT DtMatrix::WaitFrame (
 [out] __int64&   LastFrame,  // Last 'safe' frame
);
```

Parameters

FirstFrame
Sequence number of the first frame in the 'safe area'. The safe area is the range of frames, in the frame buffer, which are safe to read from or write to (i.e. the frames which are not currently being transmitted or received).

LastFrame
Sequence number of the last frame in the 'safe area'.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Wait was successful
DTAPI_E_NOT_ATTACHED	**DtMatrix** object must be attached
DTAPI_E_DEV_DRIVER	Wait failed due to internal driver error
DTAPI_E_TIMEOUT	This function will wait a maximum of 100ms for a new frame after which it timeouts and returns this error.

Remarks

This function returns immediately after the hardware has received a new frame. The safe area returned by this function is valid for one frame period (e.g. 40ms for 25fps). The 'safe area' is valid for all inputs and outputs that are part of the **DtMatrix** object i.e. the API guarantees that for all inputs *LastFrame* has been received and that all outputs are transmitting a frame prior to *FirstFrame*.