DekTec Matrix API

Matrix extensions for DTAPI

REFERENCE

February 2019

Table of Contents

Structures

Struct DtVidStdInfo

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

```
struct DtVidStdInfo
{
   int   m_VidStd;              // Video standard
   int   m_LinkStd;             // Link standard
   bool   m_IsHd;               // HD (=true) or SD (=false)
   bool   m_Is4k;               // 4K resolution (=true) or HD/SD (=false)
   int   m_VidWidth;            // Width in pixels
   int   m_VidHeight;           // Height in number of lines
   bool   m_IsInterlaced;       // Interlaced (=true) or progressive
                                // (=false)
   int   m_NumLines;            // Number of lines per frame
   double   m_Fps;              // Framerate
   double   m_Pps;              // Picture rate
   bool   m_IsFractional;       // Fractional (=true) or integer 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_LinkStd
Link 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_Is4k
Indicates whether the standard has a 4k (**true**) or HD/SD (**false**) resolution.

m_VidWidth
Indicates the width of the video area in pixels.

m_VidHeight
Indicates the height of the video area in number of lines.

m_IsInterlaced
Indicates whether the standard is interlaced (**true**) or progressive (**false**). For non-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_Pps
The picture rate. For interlaced standards the picture rate is double the frame rate, for progressive standards the picture rate is equal to the frame rate.

m_IsFractional
Indicates whether the standard has a fractional frame rate (**true**) or not (**false**).

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.

Remarks

For a multi-link standard all of the above members starting at *m_NumLines* describe the properties of a Frame in a single link (i.e. the number of lines indicates the number of lines in a frame from one link and not the sum of all lines in the combined 4k frame).

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_NumDropped;      // # of dropped 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_NumDropped
Total # of frames dropped

m_NumTransmitted
Total # of frames transmitted

m_NumDuplicated
Total # of duplicated frames

Struct DtMxSchedulingArgs

Structure describing the priority and affinity of a group of threads.

```
struct DtMxSchedulingArgs
{
   std::vector<int>  m_Affinity; // Thread affinity of the threads
   int  m_SchedPolicy;          // Scheduling policy (linux only)
   int  m_SchedPrio;            // Thread priority
};
```

Members

m_Affinity

List of CPU cores that threads in this group are allowed to run on. Leave empty to allow threads to run on all cores (default).

m_SchedPolicy

Scheduling policy for all threads in this group. This parameter is only supported in Linux builds. Allowed values are **SCHED_OTHER**, **SCHED_FIFO** and **SCHED_RR**. See *man sched_setscheduler* for details on those policies.

m_SchedPrio

Scheduling priority of threads in this group. Value depends on OS and **m_SchedPolicy**. On Windows builds the allowed values are constants prefixed with **THREAD_PRIORITY_**. See MSDN documentation on the **SetThreadPriority** function for details. For Linux with **SCHED_OTHER** (default), the valid range is from -20 to 19 (inclusive, lower values are higher priority). For the other scheduling policies, the valid values are 1 to 99 with higher values being a higher priority.

Struct DtMxThreadScheduling

Structure describing the scheduling parameters for all threads groups inside the Matrix API.

```
struct DtMxThreadScheduling
{
   DtMxSchedulingArgs   m_EventThreads;  // Parameters for event threads
   DtMxSchedulingArgs   m_WorkerThreads; // Parameters for worker threads
   DtMxSchedulingArgs   m_CallbackThreads;  // Paramters for callback
threads
};
```

Members

m_EventThreads

Scheduling settings for threads in the events group. Those threads have low CPU usage but need to react very quickly to various events. They shall have the highest priority of all threads in the system.
Default priority windows: **THREAD_PRIORITY_TIME_CRITICAL**
Default priority linux: **SCHED_FIFO** 80

m_WorkerThreads

The worker threads are CPU intensive but can tolerate a bit more scheduling latency than the event threads. In general those threads should be run at a fairly high priority to make sure no frames are dropped.
Default priority windows: **THREAD_PRIORITY_HIGHEST**
Default priority linux: **SCHED_FIFO** 20

m_CallbackThreads

The callback threads are the ones that run the callback functions. The priority of these threads depends on the kind of work the callback does. It should not be higher than the priority of the worker threads.
Default priority windows: **THREAD_PRIORITY_ABOVE_NORMAL**
Default priority linux: **SCHED_OTHER** -10

::DtapiGetVidStdInfo

Returns the properties for the specified video standard.

```
DTAPI_RESULT ::DtapiGetVidStdInfo(
  [in] int  VidStd           // Video standard
 [out] DtVidStdInfo&  Info,  // Returns the properties
);
// OVERLOAD: get properties for a combination of video and link standard
DTAPI_RESULT ::DtapiGetVidStdInfo(
  [in] int  VidStd           // Video standard
  [in] int  LinkStd          // Link standard
 [out] DtVidStdInfo&  Info,  // Returns the properties
);
```

Parameters

VidStd
Video standard

Value	SMPTE	Resolution	FPS	Remark
DTAPI_VIDSTD_UNKNOWN	-	-	-	Unknown video standard
DTAPI_VIDSTD_525I59_94	SMPTE-259	720x480	29.97	Interlaced
DTAPI_VIDSTD_625I50	SMPTE-259	720x576	25.0	Interlaced
DTAPI_VIDSTD_720P23_98	SMPTE-296	1280x720	23.98	Progressive
DTAPI_VIDSTD_720P24	SMPTE-296	1280x720	24.0	Progressive
DTAPI_VIDSTD_720P25	SMPTE-296	1280x720	25.0	Progressive
DTAPI_VIDSTD_720P29_97	SMPTE-296	1280x720	29.97	Progressive
DTAPI_VIDSTD_720P30	SMPTE-296	1280x720	30.0	Progressive
DTAPI_VIDSTD_720P50	SMPTE-296	1280x720	50.0	Progressive
DTAPI_VIDSTD_720P59_94	SMPTE-296	1280x720	59.94	Progressive
DTAPI_VIDSTD_720P60	SMPTE-296	1280x720	60.0	Progressive
DTAPI_VIDSTD_1080P23_98	SMPTE-274	1920x1080	23.98	Progressive
DTAPI_VIDSTD_1080P24	SMPTE-274	1920x1080	24.0	Progressive
DTAPI_VIDSTD_1080P25	SMPTE-274	1920x1080	25.0	Progressive
DTAPI_VIDSTD_1080P29_97	SMPTE-274	1920x1080	29.97	Progressive
DTAPI_VIDSTD_1080P30	SMPTE-274	1920x1080	30.0	Progressive
DTAPI_VIDSTD_1080PSF23_98	SMPTE-274	1920x1080	23.98	PsF
DTAPI_VIDSTD_1080PSF24	SMPTE-274	1920x1080	24.0	PsF
DTAPI_VIDSTD_1080PSF25	SMPTE-274	1920x1080	25.0	PsF
DTAPI_VIDSTD_1080PSF29_97	SMPTE-274	1920x1080	29.97	PsF
DTAPI_VIDSTD_1080PSF30	SMPTE-274	1920x1080	30.0	PsF
DTAPI_VIDSTD_1080I50	SMPTE-274	1920x1080	25.0	Interlaced
DTAPI_VIDSTD_1080I59_94	SMPTE-274	1920x1080	29.97	Interlaced
DTAPI_VIDSTD_1080I60	SMPTE-274	1920x1080	30.0	Interlaced
DTAPI_VIDSTD_1080P50	SMPTE-274	1920x1080	50.0	Progressive, 3G lvl A
DTAPI_VIDSTD_1080P50B	SMPTE-274	1920x1080	50.0	Progressive, 3G lvl B
DTAPI_VIDSTD_1080P59_94	SMPTE-274	1920x1080	59.94	Progressive, 3G lvl A
DTAPI_VIDSTD_1080P59_94B	SMPTE-274	1920x1080	59.94	Progressive, 3G lvl B
DTAPI_VIDSTD_1080P60	SMPTE-274	1920x1080	60.0	Progressive, 3G lvl A
DTAPI_VIDSTD_1080P60B	SMPTE-274	1920x1080	60.0	Progressive, 3G lvl B
DTAPI_VIDSTD_2160P23_98	SMPTE-2036	3840x2160	23.98	Progressive

DTAPI_VIDSTD_2160P24	SMPTE-2036	3840x2160	24.0	Progressive
DTAPI_VIDSTD_2160P25	SMPTE-2036	3840x2160	25.0	Progressive
DTAPI_VIDSTD_2160P29_97	SMPTE-2036	3840x2160	29.97	Progressive
DTAPI_VIDSTD_2160P30	SMPTE-2036	3840x2160	30.0	Progressive
DTAPI_VIDSTD_2160P50	SMPTE-2036	3840x2160	50.0	Progressive
DTAPI_VIDSTD_2160P50B	SMPTE-2036	3840x2160	50.0	Progressive, 4x3G lvl B
DTAPI_VIDSTD_2160P59_94	SMPTE-2036	3840x2160	59.74	Progressive
DTAPI_VIDSTD_2160P59_94B	SMPTE-2036	3840x2160	59.74	Progressive, 4x3G lvl B
DTAPI_VIDSTD_2160P60	SMPTE-2036	3840x2160	60.0	Progressive
DTAPI_VIDSTD_2160P60B	SMPTE-2036	3840x2160	60.0	Progressive, 4x3G lvl B

LinkStd

Link standard. NOTE: a value of -1 means no multi-link, single-6G, or single-12G standard is used.

Value	Meaning
DTAPI_VIDLNK_4K_SMPTE425	4k SMPTE-425 quad-3G link
DTAPI_VIDLNK_4K_SMPTE425B	4k SMPTE-425 quad-3G link, using quadrant mapping as described in Annex B of the specification
DTAPI_VIDLNK_4K_SMPTE2081	4k SMPTE-2081 single-6G link
DTAPI_VIDLNK_4K_SMPTE2082	4k SMPTE-2082 single-12G link

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
DTAPI_E_INVALID_LINKSTD	Invalid/unknown link standard was specified

::DtapiGetRequiredUsbBandwidth

Returns the properties for the specified video standard.

```
DTAPI_RESULT ::DtapiGetVidStdInfo(
  [in] int  VidStd           // Video standard
  [in] int  RxMode           // RxMode
 [out] long long&  Bw,       // Returns the required bandwidth
);
```

Parameters

VidStd

Video standard. See **DtapiGetVidStdInfo** for a list of valid video standards.

RxMode

The RxMode you intend to use. See **DtFrameBuffer::SetRxMode** for a list of valid RxModes.

Bw

The bandwidth that is required for the given combination of video standard and RxMode. The bandwidth is in bits/s and can directly be used as argument for SetIoConfig(**DTAPI_IOCONFIG_BW**).

::DtapiMxFrameStatus2Str

Convert **DtMxFrameStatus** value to a string.

```
const char* ::DtapiMxFrameStatus2Str(
  [in] DtMxFrameStatus  Status     // Status code to be converted
);
```

Function Arguments

Status

DtMxFrameStatus value to be converted to a string.

Result

Remarks

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

DtMxAncPacket

Object representing an ancillary data packet.

```
class DtMxAncPacket {
   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
   int   m_Line;                 // Line number where packet was found
};
```

Public members

m_Did
Data identifier for ancillary data packet (8-bit, parity not included).

m_SdidOrDbn
Data block number or secondary data identifier, depending on whether it is Type 1 or Type 2 packet (see **DtMxAncPacket::Type**). As *m_Did* only the 8 significant bits, the parity bits are not part of *m_SdidOrDbn*.

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.

m_Line
The line number in which this packet was found or should be inserted.

DtMxAncPacket::Type

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

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

Remarks

The type determines whether the *DtMxAncPacket::m_SdidOrDbnType* member should be interpreted as Data Block Number (=Type 1) or as Secondary Data ID (=Type 2).

DtMxAudioChannel

Object representing an audio channel.

```
class DtMxAudioChannel {
  int  m_Index;                 // Index of channel in underlaying AV format
  bool  m_Present;              // Channel was present in the input frame
  int  m_Service;               // Index of the service, in the
                                // DtMxAudioData::m_Services list, this
                                // channel is part of
  const DtMxAudioSampleType  m_Format;
                                // Format of audio samples: PCM or AES
  unsigned int*  m_pBuf;        // Buffer with audio samples
  const int  m_BufSizeSamples;
                                // Total size of the buffer (in #samples)
  int  m_NumValidSamples;       // Number of valid samples inside buffer
  DtMxAudioChannelStatus  m_Status;  // AES3 status word
  const int  m_NumSamplesHint;
                                // Suggested number of audio samples for
                                // current frame
  const DtMxAudioSampleType  m_Format;
                                // Format of audio samples: PCM32 or AES
};
```

Public members

m_Index
Zero based index of the channel. 0=channel 1, 1=channel 2, …, etc.

m_Present
Channel has been found in the input stream.

m_Service
Zero based index of the audio service the channel is part of. A value of -1 indicates the channel is not part of any service.

m_pBuf
Pointer to the buffer with audio samples.

m_BufSizeSamples
Size, in number of samples, of the audio buffer. Is a read-only field.

m_NumValidSamples
Number of valid audio samples in the audio buffer.

m_Status
The AES status word associated with the audio channel (see **DtMxAudioChannelStatus** for details).

m_NumSamplesHint
Read-only field, with suggested number of audio samples required for the current frame. The suggestion is based on where in the audio-frame-sequence the frame resides. The matrix process initialises this parameter before a frame is passed to the call-back.

For rows with an output the call-backs recommended behaviour is to fill the audio buffer with the number of samples suggested by this field. The exception to this recommended behaviour is when the call-back is audio-frame-sequence aware and manually controls the sequence.
For input-only rows the field can be ignored or used as a check to see if the number of actual received samples matches with expectation. In all cases treat this member as read-only field.

m_Format

Read-only field with data format (e.g. PCM or AES sub-frames) of the audio samples. See **DtMxAudioConfig::m_Format** description for possible values.

DtMxAudioChannelStatus

Object representing an audio channel status word.

```
class DtMxAudioChannelStatus {
  unsigned char  m_Data[24];  // Raw AES3 channel-status word data
  bool  m_Valid;              // True, if status word has been initialised
};
```

Public members

m_Data
: Buffer holding the 24 AES channel-status-word bytes.

m_Valid
: True, if the channel-status-word has been initialised (i.e. **m_Data** is valid).

DtMxAudioChannelStatus::GetSampleRate

Returns the audio sampling rate field from the AES channel-status-word.

```
DTAPI_RESULT DtMxAudioChannelStatus::GetSampleRate (
 [out] int&  SampleRate,     // Audio sample rate (in Hz)
);
```

Parameters

SampleRate

The audio sample rate (in Hz).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_NOT_INITIALIZED	Chanel status word has not been initialised

Remarks

None

DtMxAudioChannelStatus::SetSampleRate

Set the audio sampling rate field in the AES channel-status-word.

```
DTAPI_RESULT DtMxAudioChannelStatus::SetSampleRate (
  [in] int  SampleRate,      // Audio sample rate (in Hz)
);
```

Parameters

SampleRate

The audio sample rate (in Hz).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_NOT_IMPLEMENTED	Function has not been implemented in this DTAPI release

Remarks

None

DtMxAudioChannelStatus::GetPcmAudio

Returns whether the channel contains linear audio PCM data or something else.

```
DTAPI_RESULT DtMxAudioChannelStatus::GetPcmAudio (
 [out]bool&  IsPcm,          // True, for PCM
);
```

Parameters

IsPcm

If true, the audio channel carries linear PCM data. Otherwise, the channel carries 'generic' data.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_NOT_INITIALIZED	Chanel status word has not been initialised

Remarks

None

DtMxAudioChannelStatus::SetPcmAudio

Sets whether the channel carries linear audio PCM data or something else.

```
DTAPI_RESULT DtMxAudioChannelStatus::SetPcmAudio (
  [in] bool  IsPcm,          // True, for PCM and false, for data
);
```

Parameters

IsPcm

Set to true for linear PCM data and false for 'generic' data.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success

Remarks

None

DtMxAudioChannelStatus::GetPcmNumBits

Get the number of significant PCM and AUX data of bits.

```
DTAPI_RESULT DtMxAudioChannelStatus::GetPcmNumBits (
 [out] int&  NumBits,        // Number of bits used for PCM data
 [out] int&  NumAuxBits      // Number of bits used for AUX data
);
```

Parameters

NumBits

Number of significant bits used PCM data.

NumAuxBits

Number of bits used for the AUX data field.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_NOT_INITIALIZED	Chanel status word has not been initialised

Remarks

None

DtMxAudioChannelStatus::SetPcmNumBits

Sets the number of significant PCM and AUX data of bits.

```
DTAPI_RESULT DtMxAudioChannelStatus::GetPcmNumBits (
 [out] int&  NumBits,        // Number of bits used for PCM data
 [out] int&  NumAuxBits      // Number of bits used for AUX data
);
```

Parameters

NumBits

Number of significant bits used PCM data. Valid range: 16 - 24

NumAuxBits

Number of bits used for the AUX data field. Valid range: 0 - 4.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_INVALID_ARG	*NumBits* or *NumAuxBits* has an invalid value

Remarks

None

DtMxAudioConfig

Object describing an audio channel configuration.

```
class DtMxAudioConfig {
   int   m_Index;                 // Identifies the corresponding audio
                                  // channel
   bool   m_DeEmbed;              // If true, decode the input to data buffers
                                  // per stream
   DtMxOutputMode   m_OutputMode;  // Specifies output behaviour
   DtMxAudioSampleFormat   m_Format;  // Desired format for audio samples
};
```

Public Members

m_Index

Zero-based index identifying the audio channel number. I.e. 0 = 1st channel, 1 = 2nd channel, … , etc.

m_DeEmbed

Set to true, to de-embed received audio samples. For output-only rows this setting has no meaning.

m_OutputMode

Specifies the output behaviour for the audio channel.

Value	Meaning
DT_OUTPUT_MODE_ADD	Audio samples for this channel are embedded in the outgoing AV frames. If de-embedding is enabled the audio channel's data buffer will be initialised with the de-embedded audio samples from the incoming AV frames, but can be overwritten/replaced from within the call-back function. Otherwise (when de-embedding is disabled) the channel's data buffer will initially be empty.
DT_OUTPUT_MODE_COPY	Audio samples are one-to-one copied from the input to the output. The incoming audio samples are available, in the audio channel's data buffer, as read-only data only when de-embedding is enabled. For output-only rows this mode has the same effect as a DROP.
DT_OUTPUT_MODE_DROP	Audio samples will be dropped and not embedded into the outgoing AV frames, regardless of whether the audio channel was present or not on the incoming AV frames. If de-embedding is enabled the audio samples will be present as read-only data in the audio channel's data buffer.

NOTE: this setting has only effect on rows with an output.

m_Format

Desired audio sample format (e.g. PCM or AES sub-frames) for embedding/de-embedding.

Value	Meaning
DT_AUDIO_SAMPLE_PCM	32-bit PCM samples
DT_AUDIO_SAMPLE_AES3	32-bit AES sub-frames

DtMxAudioData

Represents the audio data associated with an AV frame.

```
class DtMxAudioData {
  DtFixedVector<DtMxAudioChannel>  m_Channels;  // Audio channels
  DtFixedVector<DtMxAudioService>  m_Services;  // Audio services
};
```

Public Members

m_Channels
List with audio channels.

m_Services
List with audio services.

Remarks

The two list have a fixed size and cannot be resized, the matrix framework will initialise the lists to the maximum number of audio channel/services it supports. Check the validity of an entry in these list before reading them and mark entries as valid or invalid after modifying an entry.

DtMxAudioData::GetAudio

Get the audio samples for the specified service and interleave them into one buffer

```
DTAPI_RESULT DtMxAudioData::GetAudio (
  [in] const DtMxAudioService&  Service,    // Service whose audio to get
  [in] unsigned char*  pSamples,   // Buffer to receive samples
 [i/o] int&  NumSamples,           // [in] size of buffer / [out] #samples
                                   // returned
  [in] int  SampleSize,            // Desired size of audio samples
);
```

Parameters

Service

Reference to service whose audio samples to get.

pSamples

Pointer to a buffer to receive the interleaved audio samples.

NumSamples

As input indicates the number of samples, with a size as indicated by *SampleSize*, the sample buffer can hold. As output returns the number of samples returned.

SampleSize

Desired size, in #bits, of the audio samples. Allowed values are: 16, 24 or 32.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_INVALID_SIZE	An invalid sample size was specified
DTAPI_E_INVALID_BUF	Sample buffer (*pSamples*) is invalid
DTAPI_E_BUF_TOO_SMALL	Sample buffer is too small to receive all samples. *NumSamples* returns the minimum number of samples the buffer should be able to hold.

Remarks

DtMxAudioData::InitChannelStatus

Initialise audio channel status words for all valid audio channels.

```
DTAPI_RESULT DtMxAudioData::InitChannelStatus();
// OVERLOAD: init channels status for a specific service only
DTAPI_RESULT DtMxAudioData::InitChannelStatus (
  [in] const DtMxAudioService&  Service,   // Service to initialise
);
```

Parameters

Service

Reference to a specific audio service who's audio channel status words must be initialised.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_INVALID_CHANNEL	An invalid audio channel index encountered in the service's audio channel list (**DtMxAudioService::m_Channels**)

Remarks

The channel status words will be initialised on basis of audio properties (e.g. service type, sample rate, etc) described by the **DtMxAudioService** object.

DtMxAudioService

Represents a description of an audio service.

```
class DtMxAudioService {
  bool  m_Valid;              // Service entry is valid
  DtMxAudioServiceType  m_ServiceType;  // Type of service
  std::vector<int>  m_Channels;  // Indices of associated audio channels
  int  m_PcmNumBits;          // Number of significant bits
  bool  m_ContainsData;       // True if data, false if PCM samples
  int  m_SampleRate;          // Sample rate of audio channels
  const int  m_SamplesInFrame;
                              // Number of samples in the current frame
  int  m_AudioFrameNumber;    // Current frame in the audio frame sequence
};
```

Public Members

m_Valid
Service description is valid.

m_ServiceType

Specifies the type of service (e.g. mono, stereo, etc).

Value	Meaning
DT_AUDIOSERVICE_UNKNOWN	Unknown/undefined service type
DT_AUDIOSERVICE_MONO	Service consists out of one or more unrelated mono channels
DT_AUDIOSERVICE_DUAL_MONO	Service consists out of one or more mono channel pairs
DT_AUDIOSERVICE_STEREO	Service consists out of one or more stereo channel pairs
DT_AUDIOSERVICE_5_1	Service consists out of one or more 5.1 audio channel groups

m_Channels
List with associated audio channels. The channels contain the buffers with the audio samples.

m_PcmNumBits
Number of valid significant bits in the audio samples. Only valid if the audio channel contains PCM samples (i.e. *m_ContainsData* = 'false').

m_ContainsData
True when the AES frames carry data and false if they carry linear PCM audio samples.

m_SampleRate
The audio sample rate (in Hz) for the all the channels in the service.

m_SamplesInFrame
Read-only field with total number of audio samples found/expected in the frame.

m_AudioFrameNumber
Sequence number within the audio-frame-sequence.

Remarks

When there is any audio to de-embed the `m_PcmNumBits` and `m_ContainsData` members are initialised by the matrix process with information extracted from the channel status word. The call-back must make sure these members are valid when it wants the matrix process to embed the audio service.

Similarly the audio frame number is initialised by matrix process before a frame is passed to the call-back. The call-back can overrule the audio frame number, in which case it is responsible for making sure it adds the correct number of audio samples for the point in the audio sequence.

DtMxAuxData

Represent a description of auxiliary data.

```
class DtMxAuxData {
   DtMxSdVideoIndex  m_VidIndex[2];     // Video index
   DtMxSdWss  m_Wss[2];         // Wide-screen signalling
   DtMxLine21  m_Line21[2];    // CEA-608 closed captioning
   std::vector<DtMxTeletextPacket>  m_Teletext; // Teletext data
};
```

Public Members

m_VidIndex
Video index data as can be present in SD-SDI signals (525I59.49 lines 14 and 277, 625I50 lines 19 and 332). First member belongs to the first video field, second member to the second field.

m_Wss
Wide screen signalling data as can be present in SD-SDI signals (525I59.49 lines 20 and 280, 625I50 lines 23 and 336). Currently only implemented for 625I50. First member belongs to the first video field, second member to the second field.

m_Line21
Raw closed captioning CEA-608 byte pairs. First member belongs to the first video field, second member to the second field.

m_Teletext
Raw teletext packet data.

DtMxAuxDataConfig

Auxiliary data configuration settings.

```
class DtMxAuxDataConfig {
  bool  m_DeEmbedAll;         // De-embed all aux-data. Defaults to false
  DtMxAuxDataType  m_DataType;
                              // Type of aux data configured
  union {
        DtMxAuxConfigSdi  m_Sdi;  // SDI auxiliary data configuration
  };
};
```

Public Members

m_DeEmbedAll

Set to true to force de-embedding of all supported auxiliary data objects.

m_DataType

Specifies which type of the auxiliary data configuration this object contains.

Value	Meaning
DT_AUXDATA_SDI	SDI auxiliary configuration data

m_Sdi

Configuration for SDI auxiliary data objects. See **DtMxAuxConfigSdi** for detailed description.

DtMxAuxObjConfig

Auxiliary data object configuration settings.

```
class DtMxAuxObjConfig {
  bool  m_DeEmbed;           // If true, data object must be de-embedded
  DtMxOutputMode  m_OutputMode;  // Specifies output behaviour
};
```

Public Members

m_DeEmbed

Set to true to de-embed the data objects. For output-only rows this setting has no meaning.

m_OutputMode

Specifies the output behaviour for the auxiliary data objects.

Value	Meaning
DT_OUTPUT_MODE_ADD	Auxiliary data objects are embedded in the outgoing AV frames. If de-embedding is enabled the data objects will be de-embedded from the incoming AV frames, but can be overwritten/replaced from within the call-back function. Otherwise (when de-embedding is disabled) the data objects will initially be empty.
DT_OUTPUT_MODE_COPY	Auxiliary data objects are one-to-one copied from the input to the output. The incoming objects are available in de-embedded form, as read-only data, only when de-embedding is enabled. For output-only rows this mode has the same effect as a DROP.
DT_OUTPUT_MODE_DROP	Auxiliary data objects will be dropped and not embedded into the outgoing AV frames, regardless of whether the object was present or not on the incoming AV frames. If de-embedding is enabled the object will be present, as read-only, data in their de-embedded form.

NOTE: this setting has only effect on rows with an output.

DtMxAuxConfigSdi

SDI auxiliary data object configuration settings.

```
class DtMxAuxConfigSdi {
  DtMxAuxObjConfig  m_AncPackets;
                    // Settings for ancillary data packets
  DtMxAuxObjConfig  m_VideoIndex;
                    // Settings for video index data
  DtMxAuxObjConfig  m_Wss;    // Settings for wide screen signalling
  DtMxAuxObjConfig  m_Line21; // Settings for CEA-608 CC
  DtMxAuxObjConfig  m_Teletext; // Settings for (PAL only) teletext
  VpidList          m_Vpid;   // VPID(s) to insert in output signal
};
```

Public members

m_AncPackets
Configuration settings for ancillary data packets. See **DtMxAuxObjConfig** for more details.

m_VideoIndex
Configuration settings for video index data. See **DtMxAuxObjConfig** for more details. Only supported for SD-SDI signals. Output is currently not supported.

m_Wss
Configuration settings for wide screen signalling packets closed captioning or insertion. See **DtMxAuxObjConfig** for more details. Only supported for SD-SDI signals.

m_Line21
Configuration settings for CEA-608 closed captioning or insertion. See **DtMxAuxObjConfig** for more details. Only supported for SD-SDI signals.

m_Teletext
Configuration settings for teletext capturing or insertion. See **DtMxAuxObjConfig** for more details. Only supported for 625I50 signals.

m_Vpid
Configuration settings for custom VPID insertion in output signal. Can be left empty for the default VPID or contain as many items as there are physical links. Each pair is used for a single physical link. The first number is inserted in the primary stream. The second number is only used for the secondary data stream in 3G level B signals.

DtMxData

Top-level data object holding all AV data made available to the user call-back.

```
class DtMxData {
   __int64  m_Frame;            // Current frame
   int  m_Phase;                // Current phase
   int  m_NumSkippedFrames;     // Number of skipped frames
   DtFixedVector<DtMxRowData>  m_Rows;  // Data per row
};
```

Public Members

m_Frame

Sequence number of the current frame. The current frame is the frame most recently received and/or the frame that is up next for transmission.

m_Phase

Current phase (0…NumPhases-1). In case the matrix process has been setup for multiphase processing, this member will indicate to the call-back which phase it is handling.

m_NumSkippedFrames

Error counter, will normally be 0. If due to a timeout the matrix process has to skip the call-back processing of a number of frames, this member indicates the number of frames that where skipped.

m_Rows

List with the AV data per configured matrix row (see **DtMxRowData** for details).

DtMxFrame

Data object holding the data for an AV frame.

```
class DtMxFrame {
  const DtMxRowConfig* const  m_Config;
                              // Pointer to the rows configuration
  DtMxFrameStatus  m_Status;  // Status of the frame (e.g. ok, error, etc)
  int  m_VidStd;              // Configured/received video standard
  DtMxFrame*  m_DroppedFrame;// Points to previously dropped frame
  bool  m_InpPhaseValid;      // True, if m_InpPhase is valid
  double  m_InpPhase;         // Phase relative to clock source
  bool  m_RawTimestampValid; // True, if m_RawTimestamp is valid
  __int64  m_RawTimestamp;    // Raw 64-bit timestamp (54Mhz clock), the
                              // time this frame was received
  bool  m_RawDataValid;       // True, if valid raw data is available
  DtMxRawData  m_RawData;     // Holds raw frame data
  bool  m_VideoValid;         // True, if valid video data is available
  DtMxVideoBuf  m_Video[2];   // Holds the frame's video data
  bool  m_AudioValid;         // True, if valid audio data is available
  DtMxAudioData  m_Audio;     // Holds the frame's audio data
  bool  m_AuxValid;           // True, if valid aux data is available
  DtMxAudioData  m_AuxData;   // Holds the frame's auxiliary data
};
```

Public Members

m_Config

Points to the row's configuration. See description **DtMxRowConfig** of for more details.

m_Status

Status of the frame.

Value	Meaning
DT_FRMSTATUS_OK	Frame data is valid and ready for processing
DT_FRMSTATUS_SKIPPED	Frame has been received, but the call-back was never called for it (i.e. skipped by matrix frame work). This will only be set for historic buffers, never for the current frame buffer.
DT_FRMSTATUS_DISABLED	Row has been disabled, frame buffers are not available
DT_FRMSTATUS_DUPLICATE	Frame data is duplicated from previous frame, because the input was too slow
DT_FRMSTATUS_LATE	Frame data was dropped because the input data arrived too late. Frame data is not available. Later frames will report NO_SIGNAL if the input signal is lost or will return to OK if the frame was dropped because the CPU was overloaded.
DT_FRMSTATUS_NO_SIGNAL	No signal at input port. Frame data not available.
DT_FRMSTATUS_WRONG_VIDSTD	Signal at the row's input does not match with the configured video standard. Frame data not available.
DT_FRMSTATUS_DEV_DISCONNECTED	Input (USB) device has been disconnected. Frame data not available
DT_FRMSTATUS_ERROR_INTERNAL	Unspecified internal API error. Frame data is not available

m_VidStd

The frame's video standard. See **DtapiGetVidStdInfo** for possible values.

m_DroppedFrame

For input-only and input/output row this member could point to the previous frame in case it was dropped by the matrix process (i.e. no call-back was called for this frame). Such a condition could occur in case the row's input is faster than the matrix clock source and the frame had to be dropped to maintain clock sync. By providing a pointer to the data of the dropped frame the call-back might be able to prevent audio hick-ups.
In case no there is no dropped frame this member will be **NULL**.

m_InpPhaseValid

True if **m_InpPhase** contains a valid value. Never true for output-only rows.

m_InpPhase

Phase of this input relative to the matrix's clock source. The matrix process will try to keep the phase between an early arrival time of no more than 1.25 frame period and a late arrival of no more than 0.05 frame period. A negative phase indicates the frame was received early, while a positive phase means the frame arrived too late. A frame arriving more than 1.25 frame-period early will be dropped and when the frame is more than 0.05 frame-period late the previous frame will be repeated. In a genlock-ed system one would expect the phase to be approximately zero at all times and no frame drops or repeats will be needed.

m_RawDataValid

True, when raw data is available and valid.

m_RawData
: If raw-data is enabled in the row configuration, this member holds the raw AV data. See **DtMxRawData** for a detailed description.

m_VideoValid
: True, when valid video data is available.

m_Video
: If video data is enabled in the row configuration, this member holds the video data for field 1 and field 2 (if interlaced). See **DtMxVideoBuf** for a detailed description.

m_AudioValid
: True, when valid audio data is available.

m_Audio
: If audio data is enabled in the row configuration, this member holds the audio data. See **DtMxAudioData** for a detailed description.

m_AuxDataValid
: True, when valid auxiliary data is available.

m_AuxData
: If auxiliary data is enabled in the row configuration, this member holds the aux data. See **DtMxAuxData** for a detailed description.

Remarks

When the frame's status is **DT_FRMSTATUS_DISABLED** all data should be considered as invalid, regardless of the individual data valid flags.

DtMxFrame::AncAddPacket

Adds an ancillary data packet to the specified ancillary data space.

```
DTAPI_RESULT DtMxFrame::AncAddPacket (
  [in] DtMxAncPacket&  AncPkt,     // Packet to add
  [in] int  HancVanc,         // Add to HANC or VANC space
  [in] int  Stream,           // Add to chrominance or luminance stream
  [in] int  Link=-1           // Link to add the packet too
);
```

Parameters

AncPkt

Packet to add.

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_0	Add to chrominance stream For 3G-level B: add to chrominance stream 1
DTAPI_SDI_LUM_0	Add to luminance stream For 3G-level B: add to luminance stream 1
DTAPI_SDI_CHROM_1	3G-level B only: add to chrominance stream 2
DTAPI_SDI_LUM_1	3G-level B only: add to luminance stream 2

Link

In case a multi-link output port (e.g. SMPTE 425-5 quad link) is used this member is the index (zero-based index) of the link to add the packet to. Set to -1 for a single link output configuration.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_CONFIG	Auxiliary data is disabled in the row's configuration
DTAPI_E_INVALID_ANC	Invalid ancillary data space was specified
DTAPI_E_INVALID_LINE	Invalid line was specified (**DtMxAncPacket::m_Line**)
DTAPI_E_INVALID_LINK	Invalid link index
DTAPI_E_INVALID_STREAM	Invalid stream was specified
DTAPI_E_INVALID_SIZE	Invalid packet size (**DtMxAncPacket::m_Dc**)

DtMxFrame::AncDelPacket

Deletes one or more ancillary data packets from the specified ancillary data space.

```
DTAPI_RESULT DtMxFrame::DelAddPacket (
  [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
  [in] int  Link=-1           // Link to delete the packet from
);
```

Parameters

Did

Ancillary Data-ID of the packets to delete. Valid values: 0…255.

Sdid

Secondary Data-ID of the packet to delete. Set to -1 for Type 1 packets and to a value between 0…255 for Type 2 packets.

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 packets from. NOTE: this is an HD-only parameter and for SD this parameter should be set to -1.

Value	Meaning
DTAPI_SDI_CHROM_0	Chrominance stream For 3G-level B: Chrominance stream 1
DTAPI_SDI_LUM_0	Luminance stream For 3G-level B: Luminance stream 1
DTAPI_SDI_CHROM_1	3G-level B only: Chrominance stream 2
DTAPI_SDI_LUM_1	3G-level B only: Luminance stream 2

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

Link

In case a multi-link input port (e.g. SMPTE 425-5 quad link) is used this member is the index (zero-based index) of the link to delete the packet from. Set to -1 for a single link input configuration.

Results

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_CONFIG	De-embedding of auxiliary data is disabled in the row's configuration settings
DTAPI_E_INVALID_ANC	Invalid ancillary data space was specified
DTAPI_E_INVALID_ARG	Invalid DID and/or SDID was specified
DTAPI_E_INVALID_LINE	Invalid start-line or number of lines was specified
DTAPI_E_INVALID_LINK	Invalid link index
DTAPI_E_INVALID_MODE	Invalid deletion mode was specified
DTAPI_E_INVALID_STREAM	Invalid stream was specified

DtMxFrame::AncGetPacket

Get ancillary data packets from the specified ancillary data space.

```
DTAPI_RESULT DtMxFrame::GetAddPacket (
  [in] int  Did,              // Ancillary packet Data-ID
  [in] int  Sdid,             // Ancillary packet Secondary Data-ID
 [i/o] DtMxAncPacket*  pPacket,  // Array of ancillary data packets
 [i/o] int&  NumPackets,     // [in] max. # packets to get
                             // [out] # packets actually returned
  [in] int  HancVanc,        // Get from HANC or VANC area
  [in] int  Stream,          // Get from chrom or lum stream (HD-only)
  [in] int  Link=-1          // Link to delete the packet from
  [in] int  StartLine=-1     // First line to scan
  [in] int  NumLines=-1,     // # of lines to scan
);
```

Parameters

Did

Ancillary Data-ID of the packets to get. Valid values: 1, 0…255.
NOTE: use -1 as a wildcard to get all DIDs

Sdid

Secondary Data-ID of the packet to get. Set to -1 for Type 1 packets and to a value between 1, 0…255 for Type 2 packets.
NOTE: use -1 as a wildcard to get all SDIDs

pPacket

Array of **DtMxAncPacket** objects to receive the requested ancillary data packets.
NOTE: set to **NULL** for the purpose of determining the number of packets available (i.e. available number packets will be return in *NumPackets*).

NumPackets

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

HancVanc

Specifies which ancillary data space to get the packet(s) from. NOTE: flags can be OR-ed together to get packets from both data spaces

Value	Meaning
DTAPI_SDI_HANC	HANC data space
DTAPI_SDI_VANC	VANC data space

Stream

Specifies which stream(s) to get the packets from. Multiple flags can be OR-ed together to get packets from different streams or set to -1 to get packets from all streams. NOTE: this is an HD-only parameter and for SD this parameter should be set to -1 by definition.

Value	Meaning
DTAPI_SDI_CHROM_0	Chrominance stream For 3G-level B: Chrominance stream 1
DTAPI_SDI_LUM_0	Luminance stream For 3G-level B: Luminance stream 1
DTAPI_SDI_CHROM_1	3G-level B only: Chrominance stream 2
DTAPI_SDI_LUM_1	3G-level B only: Luminance stream 2

Link

In case a multi-link input port (e.g. SMPTE 425-5 quad link) is used this member is the index (zero-based index) of the link to get the packets from. Set to -1 for a single link input configuration.

StartLine

First line to scan for the specified ancillary data packets. 1 denotes the first line and
NOTE: -1, indicates that all lines should be scanned

NumLines

Number of lines to get the specified packet(s) from. Use -1 for all lines beginning with *StartLine*.
NOTE: if *StartLine* is -1, *NumLines* should be set to -1 as well and all lines will be scanned

Results

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_BUF_TOO_SMALL	ANC packet buffer is too small to hold all available packets matching the specified DID/SDID pair. *pPacket* will be filled to the brim and *NumPackets* will return the total number of packets that are available
DTAPI_E_CONFIG	De-embedding of auxiliary data is disabled in the row's configuration settings
DTAPI_E_INVALID_ANC	Invalid ancillary data space was specified
DTAPI_E_INVALID_ARG	Invalid DID and/or SDID was specified
DTAPI_E_INVALID_LINE	Invalid start-line or number of lines was specified
DTAPI_E_INVALID_LINK	Invalid link index
DTAPI_E_INVALID_STREAM	Invalid stream was specified

DtMxLine21

Represents raw data included as waveform on line 21.

```
class DtMxLine21 {
  unsigned char  m_Data[2];   // Raw data including parity bit
  bool  m_Valid;              // True if m_Data contains valid data
};
```

Public Members

m_Data
Raw data including the parity bit as found on the waveform in line 21.

m_Valid
Indicates that a valid waveform was found on line 21 and successfully parsed.

DtMxPort::DtMxPort

Constructor for a **DtMxPort** object.

```
// 1. Constructor that doesn't link to a physical port yet
DtMxPort::DtMxPort();
// 2. Constructor that links to a single physical port. Video standard and
// link standard are not explicitly set and will be determined from
// IO-configuration.
DtMxPort::DtMxPort(
  [in] DtDevice*  pDvc,       // Device object
  [in] int  Port,             // Physical port number (1…#ports)
  [in] int  ClockPriority,    // Clock priority (≥0, default=0)
);
// 3. Constructor that initializes the object for a multi-link setup
DtMxPort::DtMxPort(
  [in] int  VidStd,           // Video standard
  [in] int  LinkStd,          // Link standard
);
```

Parameters

pDvc

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

Port

Physical port number. The port object is attached to this port. Valid values: 1…#ports

ClockPriority

Determines the ports priority for serving as the clock source within the matrix process. See **DtMxPort::AddPhysicalPort** for more details.

VideoStd

Video standard the port should be configured for. See description of **DtapiGetVidStdInfo** function for list of possible values.

LinkStd

Link setup the port should be configured to. See description of **DtapiGetVidStdInfo** function for list of possible values.

DtMxPort::AddPhysicalPort

Attach the port to a physical port on a DekTec device.

```
DTAPI_RESULT DtMxPort::AddPhysicalPort (
  [in] DtDevice*  pDvc,       // Device object
  [in] int  Port,             // Physical port number (1…#ports)
  [in] int  ClockPriority,    // Clock priority (≥0, default=0)
);
```

Parameters

pDvc

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

Port

Physical port number. The port object is attached to this port. Valid values: 1…#ports

ClockPriority

Determines the ports priority for serving as the clock source within the matrix process. When there are multiple possible ports the matrix process can use a clock source, the port with the highest priority will be used as clock source. Valid values: 0…**INT_MAX**

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_NOT_ATTACHED	*pDvc* has not been attached or the pointer is NULL

DtMxSdVideoIndex

Represent video index data.

```
class DtMxSdVideoIndex {
  unsigned char  m_Data[90]; // Raw video index data
  bool  m_Valid;             // True if m_Data contains valid data
};
```

Public Members

m_Data
Raw data including the parity bit as found on the waveform in line 21.

m_Valid
Indicates that a valid waveform was found on line 21 and successfully parsed.

DtMxSdVideoIndex::GetScanningSystem

Get the scanning system as indicated in the video index.

```
DTAPI_RESULT DtMxSdVideoIndex::GetScanningSystem(
 [out] ScanningSystem&  ScanSystem,   // Scanning system indicated
);
```

Parameters

ScanSystem

Reference that will be set to the scanning system indicated in the video index.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success

Remarks

The returned value will be the raw data, as such it might not have a corresponding value in the ScanningSystem enum.

DtMxSdVideoIndex::SetScanningSystem

Set the scanning system in the video index data.

```
DTAPI_RESULT DtMxSdVideoIndex::SetScanningSystem(
  [in] ScanningSystem  ScanSystem,// Scanning system
);
```

Parameters

ScanSystem
Value that will be set in the video index data.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_INVALID_ARG	ScanSystem was out of range

Remarks

DtMxSdVideoIndex::UpdateCrcs

Update all CRCs that are part of the video index data.

```
void DtMxSdVideoIndex::UpdateCrcs(
);
```

Parameters

Remarks

DtMxSdWss

Represent wide-screen signalling data as can be found in SD-SDI signals.

```
class DtMxSdWss {
  WssType  m_Type;            // Type of the data
  int  m_Data;                // Raw video index data
  bool  m_Valid;              // True if m_Data contains valid data
};
```

Public Members

m_Type

Type of WSS data found in input signal.

Value	Meaning
DtMxSdWss::WSS_PAL	Wide-screen signalling data was formatted as it would be in a PAL signal.
DtMxSdWss::WSS_NTSC	Wide-screen signalling data was formatted as it would be in an NTSC signal.

m_Data

Raw data (14 bits).

m_Valid

Indicates that *m_Data* contains valid data.

DtMxProcess::AddMatrixCbFunc

Register a new call-back function that will be called by the framework whenever a new frame is ready for processing.

```
DTAPI_RESULT DtMxProcess::AddMatrixCbFunc (
  [in] DtMxProcFrameFunc*  pFunc,  // Pointer to the callback function
  [in] void*  pContext,            // Opaque pointer passed to callback
);
```

Parameters

pFunc

Pointer to the user-provided call-back function that will be called when a new frame is ready for processing.

pContext

Opaque pointer that will be passed to the call-back function when it is called by the framework. Use for identifying a call-back when it is called and/or maintaining state information across multiple calls to a call-back.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_STARTED	Cannot add call-backs while process is running
DTAPI_E_INVALID_ARG	*pFunc* pointer is NULL
DTAPI_E_OUT_OF_RESOURCES	Maximum number (=32) of call-backs have already been registered

Remarks

None

DtMxProcess::AttachRowToInput

Attaches an input port to a row in the matrix.

```
DTAPI_RESULT DtMxProcess::AttachRowToInput (
  [in] int  Row,              // Index of the row
  [in] const DtMxPort&  Port,  // Input port to attach to the row
);
```

Parameters

Row

Index of the row to attach the input too. Valid values: 0...31

Port

The port to attach to the row.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_INVALID_ROW	*Row* index is invalid
DTAPI_E_NOT_ATTACHED	*Port* is not attached to any hardware
DTAPI_E_STARTED	Cannot attach a port, while process is running

DtMxProcess::AttachRowToOutput

Attaches an output port to a row in the matrix.

```
DTAPI_RESULT DtMxProcess::AttachRowToOutput (
  [in] int  Row,              // Index of the row
  [in] const DtMxPort&  Port,  // Output port to attach to the row
  [in] int  ExtraDelay,       // Extra output delay (in #frames)
);
```

Parameters

Row

Index of the row to attach the output too.

Port

The port to attach to the row.

ExtraDelay

Optional additional output delay, in number of frames, to be added to overall the end-to-end delay (see **DtMxProcess::SetEndToEndDelay**). Valid values: 0...10.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_INVALID_DELAY	*ExtraDelay* is invalid
DTAPI_E_INVALID_ROW	*Row* index is invalid
DTAPI_E_NOT_ATTACHED	*Port* is not attached to any hardware
DTAPI_E_STARTED	Cannot attach a port, while process is running

DtMxProcess::GetDefEndToEndDelay

Get the default end-to-end delay.

```
DTAPI_RESULT DtMxProcess::GetDefEndToEndDelay (
 [out] int&   Delay,          // Default end-to-end (in #frames)
 [out] double&   CbFrames,    // Time the call-back function has
);
```

Parameters

Delay

The default the end-to-end delay (in #frames), given the current matrix configuration. The default end-to-end delay is the safe minimum setting for the end-to-end delay. Alternatively the minimal end-to-end delay as returned by **GetMinEndToEndDelay** can be used to have the minimal delay, but at the cost of a less relaxed margin for scheduling delays.

CbFrames

An approximation of the time the user call-back function has relative to the time of a complete frame. For example, a value 1.5 means that the call-back has one-and-a-half frame period to process a frame.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_STARTED	Cannot get delay, while process is running

Remarks

See description of **DtMxProcess::SetEndToEndDelay** for more details about the end-to-end delay.

DtMxProcess::GetMinEndToEndDelay

Get the minimum end-to-end delay.

```
DTAPI_RESULT DtMxProcess::GetMinEndToEndDelay (
 [out] int&   Delay,          // Minimum end-to-end (in #frames)
 [out] double&   CbFrames,    // Time the call-back function has
);
```

Parameters

Delay

The minimum end-to-end delay (in #frames), given the current matrix configuration.

CbFrames

An approximation of the time the user call-back function has relative to the time of a complete frame.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_STARTED	Cannot get delay, while process is running

Remarks

See description of **DtMxProcess::SetEndToEndDelay** for more details about the end-to-end delay.

DtMxProcess::NewClockSample (NOT SUPPORTED)

THIS FUNCTION IS NOT SUPPORTED IN THE CURRENT RELEASE.

```
DTAPI_RESULT DtMxProcess::NewClockSample (
  [in] __int64  PcrOrFifoload,    //
  [in] int  RefClkCount,    //
);
```

DtMxProcess::PrintProfilingInfo

Print diagnostics regarding time spend in the call-backs threads and internal DMA and processing threads.

```
DTAPI_RESULT DtMxProcess::PrintProfilingInfo ();
```

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_STARTED	Cannot print profiling info, while process is running

Remarks

This method is intended for debugging purposes only.

DtMxProcess::Reset

Reset the matrix process.

```
DTAPI_RESULT DtMxProcess::Reset ();
```

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_STARTED	Cannot reset, while process is running

Remarks

A reset will restore the matrix process to its initial un-configured state. Upon a reset the following steps will be performed:

- Detach the rows from all attached ports.
- Reset the number of phases to its default value.
- Clear all row configurations, applied via **SetRowConfig**.
- Reset the end-to-end delay to its default value.
- Deregister all call-backs, registered via **AddMatrixCbFunc**.

DtMxProcess::SetClockControl (NOT SUPPORTED)

THIS FUNCTION IS NOT SUPPORTED IN THE CURRENT RELEASE.

```
DTAPI_RESULT DtMxProcess::SetClockControl (
  [in] DtMxClockMode  ClockMode,   //
  [in] DtDevice*  pDvc,       //
  [in] int  AvgFifoLoad,     //
);
```

DtMxProcess::SetEndToEndDelay

Set the end-to-end delay.

```
DTAPI_RESULT DtMxProcess::SetEndToEndDelay (
  [in] int  Delay,            // End-to-end in #frames
);
```

Parameters

Delay

End-to-end delay in number of frames. Must be equal or larger than the minimum end-to-end delay (see **DtMxProcess::GetMinEndToEndDelay**).

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_INVALID_DELAY	*Delay* is invalid
DTAPI_E_STARTED	Cannot change delay, while process is running

Remarks

DtMxProcess::SetNumPhases

Set the number of phases.

```
DTAPI_RESULT DtMxProcess::SetNumPhases (
  [in] int  NumPhases,       // Number of phases
);
```

Parameters

NumPhases

Number of phases uses in the call-back. Valid values: 1…8

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_INVALID_ARG	*NumPhases* is invalid
DTAPI_E_STARTED	Cannot change number phases, while process is running

Remarks

DtMxProcess::SetRowConfig

Apply a row configuration.

```
DTAPI_RESULT DtMxProcess::SetRowConfig (
  [in] int  Row,              // Index of the row
  [in] const DtMxRowConfig&  Config,  // Row configuration
);
```

Parameters

Row

Index of the row to configure. Valid values: 0...31

Config

Row configuration to be applied.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_NOT_ATTACHED	No input or outputs ports have been attached to the row
DTAPI_E_INVALID_ARG	*NumPhases* is invalid
DTAPI_E_INVALID_ROW	*Row* index is invalid
DTAPI_E_STARTED	Cannot change row configuration, while process is running

DtMxProcess::SetThreadScheduling

Apply a row configuration.

```
DTAPI_RESULT DtMxProcess::SetThreadScheduling(
  [in] const DtMxScheduling&  Scheduling,  // Scheduling settings
);
```

Parameters

Scheduling
Scheduling parameters. See **DtMxScheduling**.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_INVALID_ARG	Parameter is invalid
DTAPI_E_STARTED	Cannot change scheduling settings while process is running

DtMxProcess::SetVidBufFreeCb (NOT SUPPORTED)

THIS FUNCTION IS NOT SUPPORTED IN THE CURRENT RELEASE.

```
DTAPI_RESULT DtMxProcess::SetVidBufFreeCb (
  [in] DtMxVidBufFreeCallback*  pFunc, //
);
```

DtMxProcess::SetVidStd (NOT SUPPORTED)

Configure row for a specific video standard.

```
DTAPI_RESULT DtMxProcess::SetRowConfig (
  [in] int  Row,              // Index of the row
  [in] int  VidStd,           // Video standard to apply
);
```

Parameters

Row

Index of the row to configure.

VidStd

Video standard to be applied. See description of **DtapiGetVidStdInfo** function for list of possible values.

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_NOT_IMPLEMENTED	Function has not been implemented in this DTAPI release
DTAPI_E_INVALID_ROW	*Row* index is invalid
DTAPI_E_STARTED	Cannot change row configuration, while process is running

DtMxProcess::Start

Start the matrix process.

```
DTAPI_RESULT DtMxProcess::Start ();
```

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_CONFIG	Invalid set of configuration settings for one of the rows
DTAPI_E_FRAMERATE_MISMATCH	Not all input and/or output ports have a matching frame rate
DTAPI_E_NOT_ATTACHED	None of the matrix rows has been attached to in- and/or output port
DTAPI_E_INVALID_DELAY	The end-to-end delay is too low (i.e. delay< min. delay)
DTAPI_E_INVALID_VIDSTD	The input and output ports of one or more of the rows have not been set to the same video standard (in- and output video standard must match within a single row)
DTAPI_E_STARTED	Process already started

Remarks

DtMxProcess::Stop

Stop the matrix process.

```
DTAPI_RESULT DtMxProcess::Stop ();
```

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_NOT_STARTED	Process is not started

Remarks

DtMxRawConfig

Raw data configuration settings.

```
class DtMxRawConfig {
   DtMxRawDataType  m_Type;    // Type of raw data configured
   union {
         DtMxRawConfigSdi  m_Sdi;  // Raw SDI data configuration
   };
};
```

Public Members

m_Type

Specifies which type of the raw data configuration this object contains.

Value	Meaning
DT_RAWDATA_SDI	Raw SDI configuration data

m_Sdi

Configuration settings for raw SDI data. See **DtMxRawConfigSdi** for more details.

DtMxRawConfigSdi

Configuration settings for raw SDI data.

```
class DtMxRawConfigSdi {
  DtMxPixelFormat  m_PixelFormat;
                                  // Pixel format
  int  m_StartLine;               // First line to process
  int  m_NumLines;                // Number of lines to process
  int  m_LineAlignment;           // Desired line alignment
};
```

Public members

m_PixelFormat

Specifies the format of the pixels (i.e. packed o r planer and order of symbols) and symbol size.

Value	Meaning	#bits
DT_PXFMT_INVALID	Undefined pixel format	-
DT_PXFMT_UYVY422_10B	Packed 422, order of symbols is: Cb Y Cr Y. First symbol (Cb) is stored in LSB position of buffer.	10
DT_PXFMT_UYVY422_16B		16
DT_PXFMT_UYVY422_10B_NBO	Packed 422, order of symbols is: Cb Y Cr Y. The 8 MSB of the first symbol (Cb) are stored in the first byte. The 2nd byte contains the 2 LSB bits of the first symbol in bits 7..6 and the 6 MSB bits of the first Y symbol in bits 5..0. The 3rd byte contains the 4 LSB bits of the first Y symbol in bits 7..4 and 4 MSB bits of the Cr symbol in bits 3..0.	10
DT_PXFMT_YUYV422_10B	Packed 422, order of symbols is: Y Cb Y Cr. First symbol (Y) is stored in LSB position of buffer.	10
DT_PXFMT_YUYV422_16B		16

NOTE: in 16B formats above only the LSB 10-bits of each 16-bit word are significant, the MSB six are tied to zero.

m_StartLine

First line to process (1-based relative to start of the frame).

m_NumLines

Number of lines to process beginning at *m_StartLine*. A value of -1 indicates all lines between the start line and the end of the frame must be processed.

m_LineAlignment

Desired minimum byte alignment each line should have.
See **DtMxVideoConfig::m_LineAlignment** for more details.

DtMxRowConfig

Row configuration settings.

```
class DtMxRowConfig {
  bool   m_Enable;              // Global row enable/disable
  int    m_RowSize;             // Number of frame buffered in row
  void*  m_Opaq;                // Opaque user pointer
  bool   m_RawDataEnable;       // Enable raw data processing
  DtMxRawConfig  m_RawData;     // Configuration of raw data input/output
  bool   m_VideoEnable;         // Enable video data processing
  DtMxVideoConfig  m_Video;     // Configuration of video data input/output
  bool   m_AudioEnable;         // Enable audio data processing
  DtMxAudioConfig  m_DefAudio;  // Default audio configuration
  vector<DtMxAudioConfig>  m_Audio;  // Per channel configuration of
                                // audio data input/output
  bool   m_AuxDataEnable;       // Enable auxilliary data processing
  DtMxAudioConfig  m_AuxData; // Configuration of aux data input/output
};
```

Public Members

m_Enable
Global enable/disable for the row. True, enables the row and the remaining configuration settings will determine which data is processed. False, disables the row, meaning no data processing will be performed for this row regardless of the other configuration settings. Disabled rows are not supported in the current Matrix API release.

m_RowSize
Number of frames that will be kept in the row's frame buffer. Minimum value is 1 and the maximum is bounded only by available memory.

m_Opaq
Opaque user pointer, which can be used to link user defined data to the configuration for any purpose the user sees fit. The matrix API will not change or touch this value at all.

m_RawDataEnable
Set to true, to enable raw data processing. False disables raw data processing.
NOTE: is mutual exclusive with other processing options (see also Remarks).

m_RawData
Configuration settings for raw data input/output. See **`DtMxRawConfig`** for a detailed description.

m_VideoEnable
Set to true, to enable video data processing. False disables video data processing.

m_Video
Configuration settings for video data input/output. See **`DtMxVideoConfig`** for a detailed description.

m_AudioEnable
Set to true, to enable audio data processing. False disables audio data processing.

m_DefAudio
Default configuration settings for audio data input/output. These settings are used as defaults for all channels and can be overridden on a per-channel (see *m_Audio*). See **DtMxAudioConfig** for a detailed description.

m_Audio
Per channel configuration settings for audio data input/output. Overrules the default audio configuration (see *m_DefAudio*). See **DtMxAudioConfig** for a detailed description.

m_AuxDataEnable
Set to true, to enable auxiliary data processing. False disables auxiliary data processing.

m_AuxData
Configuration settings for auxiliary data input/output. See **DtMxAuxDataConfig** for a detailed description.

Remarks

A matrix row operates either in RAW mode or in "parsed data" mode. This means that raw data processing is mutually exclusive with video, audio and auxiliary data processing.

DtMxRowData

Data object holding all AV data for a specific row.

```
class DtMxRowData {
   DtMxFrame*   m_CurFrame;     // Pointer to the current frame
   std::vector<const DtMxFrame*>  m_Hist;  // List with previous frames
};
```

Public Members

m_CurFrame

Points to the current frame i.e. the frame to process. The call-back may read and write to this frame. See **DtMxFrame** description for details about the data a frame object holds.

m_Hist

Read-only list with previous frames (size is configured row size minus 1). Index 0 holds the previous frame, index 1 the frame before that etc. The frames in this list must be treated as read-only, so the call-back should not modify the data they hold.

DtMxVideoBuf

Video data buffer.

```
class DtMxVideoBuf {
  DtMxVideoPlaneBuf  m_Planes[3];
                              // Buffer per plane
  int  m_NumPlanes;           // Number of planes
  DtMxPixelFormat  m_PixelFormat;
                              // Pixel format
  int  m_Scaling;             // Scaling mode (OFF, 1/4, 1/16)
  int  m_Width;               // Width in pixels
  int  m_Height;              // Height in pixels
};
```

Public Members

m_Planes[3]
Buffer per plane. See **DtMxVideoPlaneBuf** for more details.

m_NumPlanes
Number of planes, directly depends on the pixel format chosen.

m_PixelFormat
Pixel format of the video. See **DtMxVideoConfig::m_PixelFormat** for details.

m_Scaling
Scaling applied to the video. See **DtMxVideoConfig::m_Scaling** for details.

m_Width
Width of the video image in number of pixels.

m_Height
Height of the video image in number of pixels.

Remarks

All fields above are managed (initialised) by the matrix process and should be treated as read-only fields in the call-back function.

DtMxVideoBuf::InitBuf

Initialises the video buffer with a specific video-pattern.

```
DTAPI_RESULT DtMxVideoBuf::InitBuf (
  [in] DtMxVidPattern  Pattern,    // Desired video pattern
);
```

Parameters

Pattern

Specifies the video pattern used for initialising the video buffer.

Value	Meaning
DT_VIDPAT_BLACK_FRAME	Fully black video pattern
DT_VIDPAT_BLUE_FRAME	Fully blue video pattern
DT_VIDPAT_GREEN_FRAME	Fully green video pattern
DT_VIDPAT_RED_FRAME	Fully red video pattern
DT_VIDPAT_WHITE_FRAME	Fully white video pattern

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Success
DTAPI_E_INVALID_ARG	Unknown video pattern specified
DTAPI_E_NOT_SUPPORTED	Initialisation is not supported for the video buffer's data or pixel format

DtMxVideoConfig

Video configuration settings.

```
class DtMxVideoConfig {
  int  m_StartLine1;          // First line to process (for field 1)
  int  m_NumLines1;           // Number of lines to process (for field 1)
  int  m_StartLine2;          // First line to process (for field 2)
  int  m_NumLines2;           // Number of lines to process (for field 2)
  int  m_Scaling;             // Scaling mode (OFF, 1/4, 1/16)
  int  m_LineAlignment;       // Desired line alignment
  int  m_BufAlignment;        // Desired buffer alignment
  DtMxPixelFormat  m_PixelFormat;
                              // Pixel format
  bool  m_UserBuffer;         // Use a user supplied video buffer
};
```

Public Members

m_StartLine1, m_StartLine2

First video line to process (1-based relative to start of the field). For progressive formats only *m_StartLine1* is used. For interlaced formats *m_StartLine2* indicates the first line in the 2nd field.

m_NumLines1, m_NumLines2

Number of video lines to process beginning at *m_StartLine#*. A value of -1 indicates all lines between the start line and the end of the field must be processed. For progressive formats only *m_NumLines1* is used. For interlaced formats *m_NumLines2* indicates the number of lines to process for the 2nd field.

m_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 (i.e. start-line=1 and #lines = -1).

m_LineAlignment

Desired minimum byte alignment each line should have. Valid values are 1, 2, 4, 8, … 512 (must be a power of 2). A special case is -1, which means no alignment (i.e. all symbols directly follow each other).
The matrix process will choose a stride of **N***m_LineAligment*. Usually **N** will be smallest value where the stride is greater or equal to the line length in number of bytes, but for performance reasons a larger stride may be used by the framework.
Typical values for the alignment are -1 (no alignment), 1 (align each line on a byte boundary) or 16 (optimal alignment for SSE2 instructions).

m_BufAlignment

Desired minimum byte alignment for each video buffer. Valid values are 1, 2, 4, 8, … 4096 (must be a power of 2). A special case is -1, which means no specific alignment is required.

m_PixelFormat

Specifies the format of the pixels (i.e. packed or planer and order of symbols) and symbol size.

Value	Meaning	#bits
DT_PXFMT_INVALID	Undefined pixel format	-
DT_PXFMT_UYVY422_8B	Packed 422, order of symbols is: Cb Y Cr Y. First symbol (Cb) is stored in LSB position of buffer.	8
DT_PXFMT_UYVY422_10B		10
DT_PXFMT_UYVY422_16B		16
DT_PXFMT_YUYV422_8B	Packed 422, order of symbols is: Y Cb Y Cr. First symbol (Y) is stored in LSB position of buffer.	8
DT_PXFMT_YUYV422_10B		10
DT_PXFMT_YUYV422_16B		16
DT_PXFMT_Y_8B	Single plane with luminance symbols only	8
DT_PXFMT_Y_16B		16
DT_PXFMT_YUV422P_8B	Planar 422, 3 planes: 1st=Y, 2nd=Cb and 3rd=Cr NOTE: Cb and Cr-planes, have same the height as the Y-plane, but only half the width of the Y-plane.	8
DT_PXFMT_YUV422P_16B		16
DT_PXFMT_YUV422P2_8B	Planar 422, 2 planes: 1st=Y, 2nd=CbCr (with Cb in LSB position of buffer)	8
DT_PXFMT_YUV422P2_16B		16
DT_PXFMT_YUV420P2_8B	Planer 420, 2 planes. 1st=Y, 2nd=CbCr (with Cb in LSB position of buffer). The CbCr buffer has half the height of the Y buffer	8
DT_PXFMT_BGR_8B	Packed 8-bit RGB data. LSB=B, then G, MSB=R	8
DT_PXFMT_V210	Packed 422, with three 10-bit symbols per 32-bit. Order of symbols is Y Cb Y Cr. First symbol is stored in LSB 10-bits of first 32-bit word.	10

NOTE: in 16B formats above only the LSB 10-bits of each 16-bit word are significant, the MSB six are tied to zero.

m_UserBuffer

Not supported in current release of the API. Must be set to false.

DtMxVideoPlaneBuf

Data buffer for a single video plane.

```
class DtMxVideoPlaneBuf {
  unsigned char*  m_pBuf;     // Pointer to buffer
  int  m_BufSize;             // Total size of buffer (in #bytes)
  int  m_Stride;              // Image stride each line (in #bytes)
  int  m_StartLine;           // First line in buffer
  int  m_NumLines;            // Number of video lines in buffer
};
```

Public m

embers

m_pBuf
Pointer to the buffer with the video symbols.

m_BufSize
Size of the buffer in number of bytes.

m_Stride
The number of bytes for one video line in the buffer. The stride includes extra padding bytes, required to align the starts of the video lines on a specific byte boundary. A value of -1 means no alignment was applied and all line directly follow each other without any form of padding.

m_StartLine
Line number (1-based relative to start of the field) of the first line in the buffer.

m_NumLines
Number of lines stored in the buffer.

AncPacket (DEPRECATED)

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
      int  m_Line;          // Line number wehre packet was found
};
```

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.

m_Line
The line number where this packet was found or should be inserted.

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

DtFrameBuffer (DEPRECATED)

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_PAIR 1	Channel pair 1 is included (= **DTAPI_SDI_AUDIO_CHAN1** | **DTAPI_SDI_AUDIO_CHAN2**)
DTAPI_SDI_AUDIO_CH_PAIR 2	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_INVALID_FRAME	The frame number is invalid
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  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.

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_FRAME	The frame number is invalid
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	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::AncDelAudio

Delete audio from a specific group from a frame.

```
DTAPI_RESULT DtFrameBuffer::AncDelAudio (
  [in] __int64  Frame,        // Seq. # of frame
  [in] int  AudioGroup        // Audio group to delete
  [in] int  Mode              // deletion mode
);
```

Parameters

Frame

Sequence number of the frame to delete the audio from.

AudioGroup

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

Mode

Specifies the deletion mode.

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

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Audio has 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_MODE	An invalid deletion-mode has been specified
DTAPI_E_INVALID_FRAME	The frame number is invalid

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
DTAPI_E_INVALID_FRAME	The frame number is invalid

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.
DTAPI_E_INVALID_FRAME	The frame number is invalid

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
DTAPI_E_INVALID_FRAME	The frame number is invalid

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
  [in] bool  EnableAncFilter // Internal use only
);
```

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. The value -1 is only valid when reading the HANC.

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
  [in] bool   EnableAncFilter // Internal use only
);
```

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. The value -1 is only valid when writing the HANC.

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 (**DtDevice::SetIoConfig(DTAPI_IOCONFIG_IOSTD,…)**) 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::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. This parameter is only relevant when using the a matrix configuration with at least one input and one output via the DtSdiMatrix class. If you're using a standalone channel, specify 0.

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 (**DtDevice::SetIoConfig(DTAPI_IOCONFIG_IOSTD,…)**) 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::DetectIoStd

Detects the video standard currently applied to the input port. See DtDevice::DetectIoStd() for the parameters and return values.

```
DTAPI_RESULT DtFrameBuffer::DetectIoStd (
 [out] int&   Value           // Detected video standard
 [out] int&   SubValue        // Detected video standard
);
```

Result

DTAPI_RESULT	Meaning
DTAPI_OK	Call succeeded
DTAPI_E_NOT_ATTACHED	**DtFrameBuffer** object must be attached to an input
DTAPI_E_DEV_DRIVER	Unexpected driver error

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

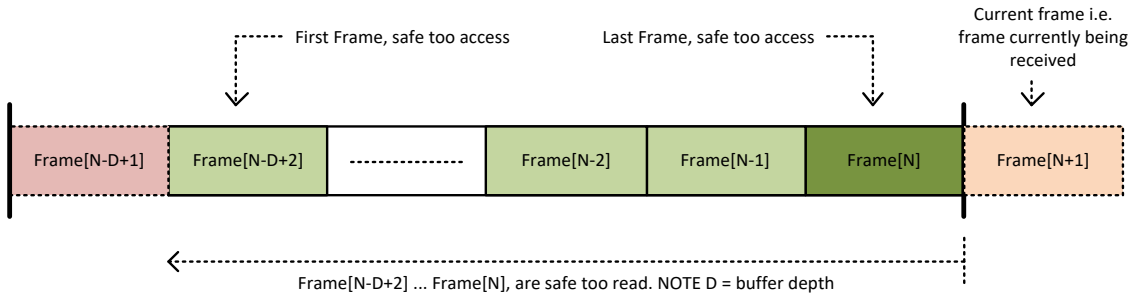
The sequence number of the frame currently being received or transmitted.

Result

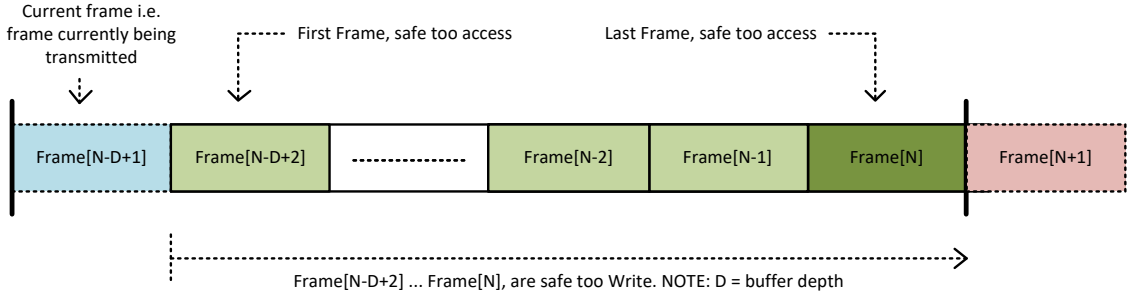
DTAPI_RESULT	Meaning
DTAPI_OK	Current frame was returned
DTAPI_E_NOT_ATTACHED	**DtFrameBuffer** object must be attached to an input and/or output
DTAPI_E_EMBEDDED	**DtFrameBuffer** object is part of an **DtSdiMatrix** object and this method cannot be used; use **DtSdiMatrix::GetCurFrame** instead

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.



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)).



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::SetRxMode

Change the way in which the driver/DTAPI read data from the DTU-351.

```
DTAPI_RESULT DtFrameBuffer::SetRxMode (
 [in] int RxMode,            // New RxMode
 [out] __int64&  FirstFrame, // First frame that will use the new mode
);
```

Parameters

RxMode

Sequence number of frame to read.

Value	ANC/Audio data available	ReadSdiLines() available	Video data available
DTAPI_RXMODE_ANC	Yes	No	No
DTAPI_RXMODE_RAW	Only via AncReadRaw()	Yes	Yes
DTAPI_RXMODE_FULL (default)	Yes	Yes	Yes
DTAPI_RXMODE_FULL8	Yes	No	Yes, but only 8bpp
DTAPI_RXMODE_FULL8_SCALED4	Yes	No	Yes, but only 8bpp and only scaled4 and scaled16
DTAPI_RXMODE_FULL8_SCALED16	Yes	No	Yes, but only 8bpp and scaled16
DTAPI_RXMODE_VIDEO	No	No	Yes
DTAPI_RXMODE_VIDEO8	No	No	Yes, but only 8bpp
DTAPI_RXMODE_VIDEO8_SCALED4	No	No	Yes, but only 8bpp and only scaled4 and scaled16
DTAPI_RXMODE_VIDEO8_SCALED16	No	No	Yes, but only 8bpp and scaled16

FirstFrame

The first frame that will be received with the new mode.

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
DTAPI_E_NOT_ATTACHED	**DtFrameBuffer** object is not attached to an input and/or output.
DTAPI_E_NOT_USB3	**DTU-351 is connected to a USB2 port**
DTAPI_E_INSUF_BW	**Either the driver failed to allocate the requested bandwidth or not enough bandwidth was requested.**

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::SetIoConfig

Configure all attached ports. See DtDevice::SetIoconfig for parameters and return values.

```
DTAPI_RESULT DtFrameBuffer::SetIoConfig (
  [in] int  Group,           // I/O configuration group
  [in] int  Value,           // I/O configuration value
  [in] int  SubValue=-1,     // I/O configuration subvalue
  [in] __int64  ParXtra0=-1, // Extra parameter #0
  [in] __int64  ParXtra1=-1  // Extra parameter #1
);
```

Remarks

If this function returns an error some ports may have the new value and some may not yet have been configured.

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.
DTAPI_E_MATRIX_HALTED	This function returns immediately with this code if it's called on a DTU-351 that is not in lock.

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 (DEPRECATED)

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 supported 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

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::SetIoConfig**)

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::SetIoConfig

Configure all attached ports. See DtDevice::SetIoconfig for parameters and return values.

```
DTAPI_RESULT DtSdiMatrix::SetIoConfig (
  [in] int  Group,           // I/O configuration group
  [in] int  Value,           // I/O configuration value
  [in] int  SubValue=-1,     // I/O configuration subvalue
  [in] __int64  ParXtra0=-1, // Extra parameter #0
  [in] __int64  ParXtra1=-1  // Extra parameter #1
);
```

Remarks

Only the video standard can be changed using this function, so Group must be DTAPI_IOCONFIG_IOSTD.

If this function returns an error some ports may have the new value and some may not yet have been configured.

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.

DtSdiMatrix::WaitFrame

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

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