MPEG Video Decoder (VdecMpeg) API

Note

The VdecMpeg is an implementation of the “Recommendation ITU-T H262, ISO/IEC 13818-2” standard. ◆

VdecMpeg API Overview

The VdecMpeg component is a software TSSA MPEG2 video decoder. It accepts MPEG1 and MPEG2 MP@ML video elementary streams. VdecMpeg detects and recovers from bit stream errors but it performs no error concealment. Presentation time stamps (if present) are attached to the outgoing video packets. Decoding time stamps (if present) are compared with an installed reference clock. The result of this comparison is then used by the decoder to determine when the decoding of a video frame must be skipped in order to maintain synchronization with other components. The skipping based on DTS comparison is only done for B-frames.

Normally, VdecMpeg requires 4 output frame buffers. A special “still” mode has been added which allows VdecMpeg to run with 1 output frame buffer. However, in this mode, VdecMpeg is capable of decoding only 1 I-frame before it must be stopped and restarted.

The user can request that user data be extracted from the incoming video stream and passed to a component which resides down stream from the video decoder.

Limitations

VdecMpeg does not run on the TM-1000. The VdecMpeg component uses instructions supported by the TM-1100 and later processors to reduce the processing load. This decoder relies on the TM-1*xxx* family VLD. It will therefore not run on TM-2*xxx* processors.

The decoder is not re-entrant, which means that only one decoder can be alive at any point in time.

VdecMpeg Inputs and Outputs

Overview

The input and outputs of the MPEG video decoder are depicted in Figure 1-1. The data input should be an MPEG video elementary stream, with optional timestamps. The two outputs are; (1) the decoded video stream and (2) a data stream that contains extracted user data. The latter is only sent along on user request. Via the control input, the component can be controlled.

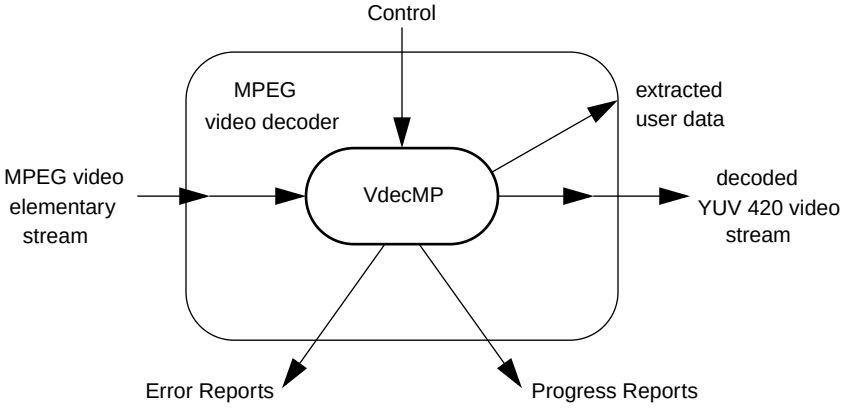

Figure 2-1 Overview of the Decoder

Inputs

VdecMpeg only operates in data streaming mode. Input packets are requested via the registered datain callback function. Input packets should be of the type tmAvPacket_t. Typically, VdecMpeg retains possession of two input packets. To avoid copying the incoming data, no internal buffering of the input stream is done. Therefore, to ensure efficient operation, the component immediately upstream from VdecMpeg should maintain a rate buffer for the incoming data.

Timestamps are passed in with data packets. The timestamps of packets with the avhValidTimestamp flag set, are used as PTS values, unless also the avhValidDts flag is set, in which case the timestamp is used as DTS value. DTS timestamps extracted from empty packets are associated with the next non-empty input packet. The PTS values are attached to the next decoded video frame and passed to the component immediately down stream from VdecMpeg along with the decoded video frame. If more than one PTS is received for a particular video frame, VdecMpeg always uses the last value received.

The capability format for the input descriptor is set to

```
tmAvFormat_t input_format = {
        sizeof(tmAvFormat_t),                /* size        */
        0,                                   /* hash        */
        0,                                   /* referenceCount */
        avdcVideo,                           /* dataClass   */
        vtfMPEG,                             /* dataType    */
        vmfMPEG1 | vmfMPEG2 | vmfNone,       /* dataSubtype */
        0                                    /* description */
};
```

Outputs

VdecMpeg has two outputs. One output contains the decoded video frames. The other contains user data which has been extracted from the incoming stream.

In the case of video output, each packet contains one entire video frame. In the case of interlaced frames, one tmAvPacket contains both fields. The top field is located at the location indicated by the tmAvPacket‘s data pointer. While the bottom field is located at data pointer + stride (the regular vdfInterlaced format).

PTS values for the video output are located in the timeStamp field of the tmAvPacket. If the incoming stream contains valid PTS values, the decoder will linearly extrapolate these PTS values such that every decoded video frame out of the decoder will have a PTS. The extrapolated PTS values are only used if the incoming video frame does not have a valid PTS.

The decoded video output has its capability format set to:

```
tmAvFormat_t videoFormat = {
        sizeof(tmAvFormat_t),/* size        */
        0,                   /* hash        */
        0,                   /* referenceCount */
        avdcVideo,           /* dataClass   */
        vtfYUV,              /* dataType    */
        vdfYUV420Planar,     /* dataSubtype */
        0                    /* description */
};
```

The user data output contains user data extracted from the Sequence, GOP and Picture layers of the bitstream. The user can dynamically enable or disable extraction of these user data streams via the command interface. However, the I/O descriptors must be initialized at instance setup. The packets contain un-interpreted data from the bitstream. If the data does not fit in one packet, an error conditions is signalled. Once such an error has occurred, the remaining user data is discarded and a full but incomplete user data packet is sent to the user data output. User data packets are sent out immediately if reordering is not enabled.

Otherwise, user data is sent to its output when the corresponding video frame is sent to the video output.

The data output has the following format:

```
tmAvFormat_t videoFormat = {
       sizeof(tmAvFormat_t),/* size        */
       0,                   /* hash        */
       0,                   /* referenceCount */
       avdcGeneric,         /* dataClass   */
       avdtGeneric,         /* dataType    */
       avdsGeneric,         /* dataSubtype */
       0                    /* description */
};
```

The output descriptor assignment is:

```
#define VDECMPEG_OUTPUT        0
#define VDECMPEG_DATA_OUTPUT   1
```

VdecMpeg Errors

VdecMpeg detects and recovers from a wide variety of bitstream errors. Errors are reported via the registered error callback function. Errors which are reported with the `tsaErrorFlagsFatal` set should result in termination of the instance. Bitstream errors are never fatal. It is assumed that the incoming data stream is a stream that the decoder should be able to decode.

Once an error has been reported, the default recovery mechanism is to seek to the next group of pictures and resume decoding at that point. Video frames in which an error was encountered part way through the decoding process are sent to the down stream component with buffersInUse set to 0. Note that this is the only time which it is acceptable to return buffers to the video decoder out of order. In all other instances, video frame buffers must be returned to the video decoder in the order in which they were sent.

VdecMpeg Progress

There are three progress reports produced by VdecMpeg. The decoder reports the decode of a frame (and frame type), the skip of a frame and the sequence information from which the application can determine what type of bitstream is decoded. The VdecMpeg component uses the tsaProgressFlagChangeFormat, which is handled by TSSA internally, to install a format on the Video output queue.

```
tmLibappErr_t
VdecMpegProgress(Int instId, UInt32 flags, ptsaProgressArgs_t args)
```

VdecMpeg Configuration

The following control modes can be set via calls to tmolVdecMpegInstanceConfig:

1. Enable extraction of user data. Sending this command will tell the video decoder to extract user data and send it to the data output. The argument to this command is a boolean which indicates whether or not to reorder the user data with the decoded video frames. A value of false causes the user data to be sent down stream immediately. Note that there are actually 3 separate commands. One each to enable sequence, GOP and picture user data independently.
2. Disable extraction of user data. Sending this command disables extraction of user data by the video decoder. Again, there are actually 3 separate commands. One each to enable sequence, GOP and picture user data independently.
3. Flush. Assumed to be called only when there are no more input packets, In this case a flush buffer is installed in the VLD, the last data is decoded and then the decoded frames, if any, are sent out.
4. Ignore DTS, in which case all incoming frames are decoded regardless of their decoding time stamp. This mode can be used to implement trick modes.
5. Resume decoding with taking DTS into account, the default operation mode when a clock is installed.

VdecMpeg API Data Structure Descriptions

This section describes the VdecMpeg component data structures.

tmolVdecMpegInstanceSetup_t, tmalVdecMpegInstanceSetup_t

```
typedef struct tmalVdecMpegInstance {
       ptsaDefaultInstanceSetup_t      defaultSetup;
       UInt32                          imageStride;
       UInt32                          numberOfOutputPackets;
       UInt32                          numberOfUserDataPackets;
} tmalVdecMpegInstanceSetup_t, *ptmalVdecMpegInstanceSetup_t;

typedef tmalVdecMpegInstanceSetup_t tmolVdecMpegInstanceSetup_t;
typedef ptmalVdecMpegInstanceSetup_t ptmolVdecMpegInstanceSetup_t;
```

Fields

`defaultSetup`	See TSSA documentation
`imageStride`	In case the display component that interprets the video output has a stride restriction, for instance the ICP. When set to 0, no stride restriction is assumed and the image width of the decoder picture is taken as stride.
`numberOfOutputPackets`	Either set to 1 or 4. When set to 1, only 1 I frame is decoded.
`numberOfUserDataPackets`	Total number of packets available for user data.

Description

Data structure passed to tmolVdecMpegInstanceSetup, tmalVdegMpegInstanceSetup to describe the input and output connections and other initial values.

tmolVdecMpegCapabilities_t, tmalVdecMpegCapabilities_t

```
typedef struct tmalVdecMpegCapabilities{
       ptsaDefaultCapabilities_t       defaultCaps;
} tmalVdecMpegCapabilities_t, *ptmalVdecMpegCapabilities_t;

typedef tmalVdecMpegCapabilities_t tmolVdecMpegCapabilities_t;
typedef ptmalVdecMpegCapabilities_t ptmolVdecMpegCapabilities_t;
```

Fields

`defaultCaps`	See TSSA documentation.

Description

For input and output descriptors see VdecMpeg Inputs and Outputs. The text section of VdecMpeg is about 100KB, the initialized data section is about 4KB, there is no bss requirement.

tmolVdecMpegErrorFlags_t

Err_base_VdecMpeg is set to 0x13070000.

```
typedef enum {
/* Fatal errors */
VDECMPEG_ERR_VLD_OPEN_FAILED            Err_base_VDecMpeg + 0x0001,
VDECMPEG_ERR_INVALID_PROCESSOR          Err_base_VdecMpeg + 0x0002,
VDECMPEG_ERR_NO_PICTURE_INFO_ALLOCATED  Err_base_VdecMpeg + 0x0003,

/* Non-fatal errors (action by decoder itself) */
VDECMPEG_ERR_RESERVED_EXT_STARTCODE_ID  Err_base_VdecMpeg + 0x0100,
VDECMPEG_ERR_UNEXPECTED_STARTCODE       Err_base_VdecMpeg + 0x0101,
VDECMPEG_ERR_ODD_FIELD_PICTURES         Err_base_VdecMpeg + 0x0102,
VDECMPEG_ERR_LAST_FRAME_NOT_COMPLETE    Err_base_VdecMpeg + 0x0103,
VDECMPEG_VLD_ERROR                      Err_base_VdecMpeg + 0x0104,
VDECMPEG_ERR_MBA_OVERFLOW               Err_base_VdecMpeg + 0x0105,
VDECMPEG_ERR_MBA_EXCEEDS_PICTURE_SIZE   Err_base_VdecMpeg + 0x0106,
VDECMPEG_ERR_DCT_COEFFS_EXCEED_64       Err_base_VdecMpeg + 0x0107,
VDECMPEG_ERR_INVALID_MOTION_TYPE        Err_base_VdecMpeg + 0x0108,

VDECMPEG_ERR_AV_BUFFERS_TOO_SMALL       Err_base_VdecMpeg + 0x0109,
VDECMPEG_ERR_ONLY_420_SUPPORTED         Err_base_VdecMpeg + 0x010A,
VDECMPEG_ERR_ONLY_MPML_SUPPORTED        Err_base_VdecMpeg + 0x010B,
VDECMPEG_ERR_SPATIAL_SCALABILITY_NOT_SUPPORTED
                                        Err_base_VdecMpeg + 0x010C,
VDECMPEG_ERR_TEMPORAL_SCALABILITY_NOT_SUPPORTED
                                        Err_base_VdecMpeg + 0x010D,
VDECMPEG_ERR_INTERNAL_ERROR             Err_base_VdecMpeg + 0x01FF
} tmalVdecMpegErrorFlags_t;
```

Fields

Fatal errors

`VDECMPEG_ERR_VLD_OPEN_FAILED`

The VLD open failed, the interrupt could not be allocated.

`VDECMPEG_ERR_INVALID_PROCESSOR`

The decoder is executed on a TM-1000 processor. For speed and compliance reasons, some special instruction supported by TM-1100 or later processors are required.

`VDECMPEG_ERR_NO_PICTURE_INFO_ALLOCATED`

A video packet was taken from the queue and it did not have a preallocted tmalVdecMpegPictureInfo_t installed in the userPointer.

`VDECMPEG_ERR_RESERVED_EXT_STARTCODE_ID`

An unknown extension start code was encountered. The extension startcode id is returned in the args.description field. Decoding resumes at the next start code.

`VDECMPEG_ERR_UNEXPECTED_STARTCODE`

A non-video startcode was encountered. The startcode is described in the args.description field. Decoding is restarted at the next gop.

`VDECMPEG_ERR_ODD_FIELD_PICTURES`

An odd number of field pictures has been encountered before the current frame picture.

`VDECMPEG_ERR_LAST_FRAME_NOT_COMPLETE`

An odd number of field pictures was decoded before a sequence end code was encountered.

`VDECMPEG_VLD_ERROR` The VLD has detected and illegal Huffman code.

`VDECMPEG_ERR_MBA_OVERFLOW` A macroblock address increment value has exceeded the maximum alllowable value (i.e. number of macroblocks per row). Only valid for MPEG2 sequences.

`VDECMPEG_ERR_MBA_EXCEEDS_PICTURE_SIZE`

The number of macrblocks decoded for the current picture has exceeded the picture size specified in the sequence header.

`VDECMPEG_ERR_DCT_COEFFS_EXCEED_64`

A block with more than 64 DCT coefficients has been encountered.

`VDECMPEG_ERR_INVALID_MOTION_TYPE`

The motion type for the current macroblock is illegal with respect to the current picture structure.

`VDECMPEG_ERR_AV_BUFFERS_TOO_SMALL`

The given YUV output buffers were to small to decode this bitstream. Decoding is restarted at the next gop. The user may want to stop the instance, insert bigger buffers and restart.

`VDECMPEG_ERR_ONLY_420_SUPPORTED`
: A chroma format value (see 13818-2) other than 1 has been encountered in the sequence extension. For MP@ML streams, the chroma format field can only be 1. Decoding is restarted at the next gop.

`VDECMPEG_ERR_ONLY_MPML_SUPPORTED`
: Only main profile, main level is supported, decoding isrestarted at the next gop.

`VDECMPEG_ERR_SPATIAL_SCALABILITY_NOT_SUPPORTED`
: A spatial scalable extension (picture or sequence) has been detected. No such extensions are allowed in MP@ML streams. Decoding is restarted at the next gop,

`VDECMPEG_ERR_TEMPORAL_SCALABILITY_NOT_SUPPORTED`
: A temporal scalable extension (picture or sequence) has been detected. No such extensions are allowed in MP@ML streams. Decoding is restarted at the next gop.

`VDECMPEG_ERR_INTERNAL_ERROR` Contact the vendor, an internal error has occurred.

Description

These error codes are passed as args.errorCode in the installed errorFunc. Only when explicitly mentioned the description field is set. Usually the args.description is set to Null.

tmalVdecMpegControlCommand_t

```
typedef enum {
    VDECMPEG_CMD_FREEZE        tsaCmdUserBase + 0,
    VDECMPEG_CMD_UNFREEZE      tsaCmdUserBase + 1,
    VDECMPEG_CMD_IGNORE_DTS    tsaCmdUserBase + 2,
    VDECMPEG_CMD_NORMAL_DTS    tsaCmdUserBase + 3,
    VDECMPEG_CMD_SEQ_UD_ON     tsaCmdUserBase + 4,
    VDECMPEG_CMD_SEQ_UD_OFF    tsaCmdUserBase + 5,
    VDECMPEG_CMD_GOP_UD_ON     tsaCmdUserBase + 6,
    VDECMPEG_CMD_GOP_UD_OFF    tsaCmdUserBase + 7,
    VDECMPEG_CMD_PIC_UD_ON     tsaCmdUserBase + 8,
    VDECMPEG_CMD_PIC_UD_OFF    tsaCmdUserBase + 9,
    VDECMPEG_CMD_FLUSH         tsaCmdUserBase + 10
    VDECMPEG_CMD_SKIP_BFRAMES  tsaCmdUserBase + 11
} tmalVdecMpegControlCommand_t;
```

Fields

`VDECMPEG_CMD_FREEZE`	CURRENTLY UNIMPLEMENTED. indicate that the output picture needs to be frozen. The decoder will decode I and P frames (if the number of buffers set by instance setup allows this), such that unfreeze is smooth and quick. When the decoder was frozen this command has no effect.
`VDECMPEG_CMD_UNFREEZE`	CURRENTLY UNIMPLEMENTED. unfreeze a frozen decoder. When the decoder is not frozen this command has no effect.
`VDECMPEG_CMD_IGNORE_DTS`	decode all incoming frames regardless of whether the DTS has expired.
`VDECMPEG_CMD_NORMAL_DTS`	Interpret the DTS, when the DTS has expired, do not decode the frame. This is the default operation mode when a valid clock is passed in the instance setup.
`VDECMPEG_CMD_SEQ_UD_ON`	Enable extraction of user data at the sequence level. A boolean cast of “parameter” is used to indicate whether the extracted user data should be reorderd with the outgoing video frames.
`VDECMPEG_CMD_SEQ_UD_OFF`	Disable extraction of user data at the sequence level.
`VDECMPEG_CMD_GOP_UD_ON`	Enable extraction of user data at the group of pictures level. A boolean cast of “parameter” is used to indicate whether the extracted user data should be reorderd with the outgoing video frames.

VDECMPEG_CMD_GOP_UD_OFF	Disable extraction of user data at the group of pictures level.
VDECMPEG_CMD_PIC_UD_ON	Enable extraction of user data at the picture level. A boolean cast of “parameter” is used to indicate whether the extracted user data should be reorderd with the outgoing video frames.
VDECMPEG_CMD_PIC_UD_OFF	Disable extraction of user data at the picture level.
VDECMPEG_CMD_FLUSH	Decode all data that has been passed to the decoder. Flush the decoded output pictures. It is assumed that there is no incoming data anymore.
VDECMPEG_CMD_SKIP_BFRAMES	Skip decoding of B-frames.

Description

These commands can be passed as ‘command’ in a ptsaControlArgs_t structure that is passed to tmolVdecMpegInstanceConfig. Unless otherwise indicated, ‘parameter’ of the ptsaControlArgs_t structure has no meaning.

tmalVdecMpegProgressFlags_t

```
typedef enum {
    VDECMPEG_NEW_SEQUENCE    = 0x0001,
    VDECMPEG_DECODED_A_FRAME = 0x0002,
    VDECMPEG_SKIPPED_A_FRAME = 0x0004,
    VDECMPEG_TIMEDIFF        = 0x0008
} tmalVdecMpegProgressFlags_t;
```

Fields

`VDECMPEG_NEW_SEQUENCE`	A new sequence header is encountered, see tmalVdecMpegSequenceDescription_t.
`VDECMPEG_DECODED_A_FRAME`	A frame was successfully decoded. The args.description field is set to I_TYPE, P_TYPE, or B_TYPE and indicates which frame has just been decoded. I_TYPE etc are defined in the tmalVdecMpeg.h include file.
`VDECMPEG_SKIPPED_A_FRAME`	A frame was skipped because the DTS was expired. The args.description field is set to B_TYPE since these are the only type of frames the decoder can safely skip.
`VDECMPEG_TIMEDIFF`	Reserved for future use.

Description

Used in progress reports, as `args.progressCode` in the `ptsaProgressArgs_t` structure.

tmalVdecMpegSequenceLevel_t

```
typedef enum {
    VDECMPEG_MPEG1_SEQ,
    VDECMPEG_MPEG2_SEQ
} tmalVdecMpegSequenceLevel_t;
```

Fields

`VDECMPEG_MPEG1_SEQ`	indication of mpeg level 1 sequence.
`VDECMPEG_MPEG2_SEQ`	indication of mpeg level 2 sequence.

Description

This data structure is used in `VDECMPEG_NEW_SEQUENCE` progress report. It is passed via the `tmalVdecMpegSequenceDescription_t` structure.

tmalVdecMpegSequenceDescription_t

```
typedef struct{
   UInt32                       size;
   tmalVdecMpegSequenceLevel_t  level;
   UInt32                       imageWidth;
   UInt32                       imageHeight;
   UInt32                       bitRateValue;
   UInt16                       aspectRatio;
   Bool                         progressiveSequence;
   Bool                         sequenceDisplayExtensionPresent;
} tmalVdecMpegSequenceDescription_t,
*ptmalVdecMpegSequenceDescription_t;
```

Fields

`size`	Used by TSSA, always the size of the structure.
`level`	Either `VDECMPEG_MPEG1_SEQ` or `VDECMPEG_MPEG2_SEQ`.
`imageWidth`	The width of the decoded fields as indicated in the sequence header.
`imageHeight`	The height of the decoded fields as indicated in the sequence header.
`bitRateValue`	The bit-rate as indicated in the sequence header.
`aspectRatio`	The aspect-ratio as indicated in the sequence header.
`progressiveSequence`	Whether this bitstream is progressive or interlaced.
`sequenceDisplayExtensionPresent`	Whether this sequence has display extension set.

Description

This data structure is passed by reference in the description field of the `ptsaProgressArgs_t` structure that is passed to the installed progress function. The `progressCode` is set to `VDECMPEG_NEW_SEQUENCE`.

tmalVdecMpegPictureInfo_t

```
typedef struct{
   UInt32       size;
   UInt32       dataFormat;
   Int16        displayHorizontalSize;
   Int16        displayVerticalSize;
   Int16        frameCentreHorizOffset[3];
   Int16        frameCentreVertOffset[3];
   UInt16       aspectRatio;
   ptmAvFormat  userData[MAX_UD_INDEX];
} tmalVdecMpegPictureInfo_t, *ptmalVdecMpegPictureInfo_t;
```

Fields

`size`	Used by TSSA, the size of this structure.
`dataFormat`	Data format, defined as follows:

```
dataFormat =
((picture_structure    & VO_DF_PS_MASK) << VO_DF_PS_SHIFT)        |
((chroma_format        & VO_DF_CF_MASK) << VO_DF_CF_SHIFT)        |
((matrix_coefficients  & VO_DF_COL_CONV_MASK)
                                        << VO_DF_COL_CONV_SHIFT)  |
((progressive_frame    & VO_DF_PROG_FR_MASK)
                                        << VO_DF_PROG_FR_SHIFT)   |
((top_field_first      & VO_DF_TFF_MASK)<< VO_DF_TFF_SHIFT)       |
((repeat_first_field   & VO_DF_RFF_MASK)<< VO_RFF_SHIFT)          |
((progressive_sequence & VO_DF_PROGSEQ_MASK)
                                        << VO_DF_PROGSEQ_SHIFT)   |
(((picture_rate -1)    & VO_DF_FRAME_RATE_MASK)
                                        << VO_DF_FRAME_RATE_SHIFT)|
((pict_type)           & VO_DF_PTYPE_SHIFT);
```

picture_structure

`TOP_FIELD`	`0x1`	Frame is encoded in the form of 2 fields and current field is the top field.
`BOTTOM_FIELD`	`0x2`	Frame is encoded in the form of 2 fields and current field is the bottom field.
`FRAME_PICTURE`	`0x3`	Both fields are encoded as one single frame. This is also the case for MPEG1 encoded streams.

chroma_format

This 2 bit integer indicates the chrominance format. For VdecMP, only `CHROMA420` is supported.

`CHROMA420 0x1`	4:2:0 format.
`CHROMA422 0x2`	4:2:2 format.
`CHROMA444 0x3`	4:4:4 format.

matrix_coefficients

This 8 bits integer describes the matrix coefficients used to perform RGB to YCrCb conversion. In the case there is no `sequence_display_extension`() in the bit stream, the matrix coefficients is determined by the recommendation ITU_R BT.709.

progressive_frame

When set to zero, it indicates that the 2 fields of the frame are interlaced fields. When set to 1, it indicates that the 2 fields of the frame are from the same time instant as one another.

progressive_sequence

When set to 1, the video sequence contains only progressive frame-pictures (for instance as in MPEG1), when set to 0, video sequence can contain both frame-picture and field-picture, and frame-pictures may be interlaced or progressive.

top_field_first

If `progressive_sequence == 0`, top_field_first set to 1 indicates that the top field of the reconstructed frame is the first field output by the decoding process. If `progressive_sequence == 1`, this field, combined with `repeat_first_field` indicates how many times the reconstructed frame is output by the decoding process.

repeat_first_field

This flag is applicable only in a frame picture. In case `progressive_frame == 1`, and `progressive_sequence == 0`, if set to 1, then the first field is displayed, then the other field, and then the first field is repeated.

pict_type

The picture coding type. Not used by any renderer.

All this bit stream information is packed into one 32-bit `dataFormat` register, as defined previously, using the following masks:

`VO_DF_PS_MASK`	`0x3` 2 bits mask.
`VO_DF_CF_MASK`	`0x3` 2 bits mask.
`VO_DF_COL_CONV_MASK`	`0x7` 3 bits mask.
`VO_DF_PROG_FR_MASK`	`0x1` 1 bit mask.
`VO_DF_TFF_MASK`	`0x1` 1 bit mask.
`VO_DF_RFF_MASK`	`0x1` 1 bit mask.
`VO_DF_PROGSEQ_MASK`	`0x1` 1 bit mask.
`VO_DF_LASTFRAME_MASK`	`0x1` 1 bit mask.
`VO_DF_FRAME_RATE_MASK`	`0xF` 4 bits mask.
`VO_DF_FRAME_SENT_MASK`	`0x1` 1 bit mask (for internal use).
`VO_DF_PTYPE_MASK`	`0x3` 2 bits mask.
`VO_DF_PS_SHIFT`	`0x0`
`VO_DF_CF_SHIFT`	`0x2`
`VO_DF_COL_CONV_SHIFT`	`0x4`
`VO_DF_PROG_FR_SHIFT`	`0x7`
`VO_DF_TFF_SHIFT`	`0x8`
`VO_DF_RFF_SHIFT`	`0x9`
`VO_DF_PROGSEQ_SHIFT`	`0xA`
`VO_DF_LASTFRAME_SHIFT`	`0xB`
`VO_DF_FRAME_RATE_SHIFT`	`0xC`
`VO_DF_FRAME_SENT_MASK`	`0x10` (for internal use)
`VO_DF_PTYPE_SHIFT`	`0x11`

Fields, *continued*

`displayHorizontalSize` `displayVerticalSize`	These two fields define a display rectangle considered as the intended display area. If it is smaller than the encoded frame size, then only a portion of the encoded frame is displayed.
`frameCentreHorizOffset` `frameCentreVertOffset`	These two fields indicate the position of the center of the display rectangle. If both are 0, the center of the display rectangle is located at the center of the decoded frame. Those 2 fields are in 1/16th sample units.
`aspectRatio`	This field gives the display aspect ratio: 3/4, 16/9 or 1/2.21.

userData	Contains three pointers to memory buffers where the user data extracted by the decoder will be stored.

Description

This data structure is passed via the `userPointer` field of the `tmAvHeader_t` of each video packet sent out. The `format.description` field has the `vdfMPEGExtension` flag set, which indicates to the renderer that the packet has an MPEG extension attached to it.

VdecMpeg API Function Descriptions

This section describes the VdecMpeg component functional interface.

tmolVdecMpegGetCapabilities, tmalVdecMpegGetCapabilities

```
extern tmLibappErr_t tmolVdecMpegGetCapabilities(
   ptmolVdecMpegCapabilities_t    *cap
);
extern tmLibappErr_t tmalVdecMpegGetCapabilities(
   ptmolVdecMpegCapabilities_t    *cap
);
```

Parameters

`cap`	pointer to the capabilities structure pointer.

Return Codes

`TMLIBAPP_OK`	on success.

Description

This function fills in the pointer of a static structure, `tmolVdecMpegCapabilities_t`, `tmalVdecMpegCapabilities_t`, maintained by the decoder, to describe the capabilities and requirements of this library.

tmolVdecMpegOpen, tmalVdecMpegOpen

```
extern tmLibappErr_t tmolVdecMpegOpen(
   Int   *instance
);
extern tmLibappErr_t tmalVdecMpegOpen(
   Int   *instance
);
```

Parameters

instance	Returned instance.

Return Codes

TMLIBAPP_ERR_MEMALLOC_FAILED	Memory allocation failed.
TMLIBAPP_ERR_MODULE_IN_USE	No more instances are available. Currently only one instance is supported, due to the amount of memory and processing power needed.
VDECMPEG_ERR_INVALID_PROCESSOR	Trying to run the decoder on a TM-1000. It needs a TM-1100 (or later) processor.
TMLIBAPP_OK	On success.

Or, in case of `tmolVdecMpegOpen`, any return code produced by `tsaDefaultOpen`.

Description

Opens an instance of the VdecMpeg component.

The VdecMpeg task is created with preemption. Usually the task should have low priority. The default stacksize is set to 10 kb.

tmolVdecMpegInstanceSetup, tmalVdegMpegInstanceSetup

```
extern tmLibappErr_t tmolVdecMpegInstanceSetup(
   Int                          instance,
   ptmolVdecMpegInstanceSetup_t setup
);
extern tmLibappErr_t tmalVdecMpegInstanceSetup(
   Int                          instance,
   ptmolVdecMpegInstanceSetup_t setup
);
```

Parameters

`instance`	Instance previously opened by `tmolVdecMpegOpen`, `tmalVdecMpegOpen`.
`setup`	Pointer to the demultiplexer's setup data structure, see `tmolVdecMpegInstanceSetup_t`, `tmalVdecMpegInstanceSetup_t`.

Return Codes

`TMLIBAPP_ERR_INVALID_INSTANCE`	When the instance is not a valid instance open with `tmolVdecMpegOpen, tmalVdecMpegOpen`, triggered via `tmAssert`.
`TMLIBAPP_ERR_NOT_OPEN`	When the instance is not opened with `tmolVdecMpegOpen, tmalVdecMpegOpen`, triggered via `tmAssert`.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	No memory could be allocated for the instance.
`TMLIBAPP_ERR_INVALID_SETUP`	When the numbers of output buffers is not 1 or 4, see Limitations on page 1-2
`TMLIBAPP_OK`	On success.

Or, case of `tmolVdecmpegInstanceSetup`, any error code returned by `tsaDefaultInstanceSetup`.

Description

The instance previously opened by `tmolVdecMpegOpen` is set up. Memory is allocated for the internally held buffers that are needed for decoding. `tmolVdecMpegInstanceSetup` should be called only once for each instance.

tmolVdecMpegGetInstanceSetup, tmalVdecMpegGetInstanceSetup

```
extern tmLibappErr_t tmolVdecMpegInstanceSetup(
   Int                           instance,
   ptmolVdecMpegInstanceSetup_t *setup
);
extern tmLibappErr_t tmalVdecMpegInstanceSetup(
   Int                           instance,
   ptmolVdecMpegInstanceSetup_t *setup
);
```

Parameters

`instance`	Instance previously opened by `tmolVdecMpegOpen, tmalVdecMpegOpen`.
`setup`	Pointer to a pointer to the VdecMpeg setup data structure, see `tmolVdecMpegInstanceSetup_t, tmalVdecMpegInstanceSetup_t`.

Return Codes

`TMLIBAPP_ERR_INVALID_INSTANCE`	When the instance is not a valid instance open with `tmolVdecMpegOpen, tmalVdecMpegOpen`, triggered via tmAssert.
`TMLIBAPP_ERR_NOT_OPEN`	When the instance is not opened with `tmolVdecMpegOpen, tmalVdecMpegOpen`, triggered via tmAssert.
`TMLIBAPP_OK`	On success.

Description

This function is used during initialization of the decoder. It returns the default settings for the decoder instance. The setup can then be further initialized by the application which normally is filling all the queues and the progress and error functions and then passed to `tmolVdecMpegInstanceSetup` or `tmalVdegMpegInstanceSetup`.

tmolVdecMpegStart, tmalVdecMpegStart

```
extern tmLibappErr_t tmolVdecMpegStart(
   Int   instance
);
extern tmLibappErr_t tmalVdecMpegStart(
   Int   instance
);
```

Parameters

`instance`	Instance previously opened by `tmolVdecMpegOpen` or `tmalVdecMpegOpen`.

Return Codes

`TMLIBAPP_ERR_INVALID_INSTANCE`	When the instance is not a valid instance open with `tmolVdecMpegOpen, tmalVdecMpegOpen`, triggered via `tmAssert`.
`TMLIBAPP_ERR_NOT_OPEN`	When the instance is not opened with `tmolVdecMpegOpen, tmalVdecMpegOpen`, triggered via `tmAssert`.
`TMLIBAPP_ERR_NOT_SETUP`	When the instance is not set up with `tmolVdecMpegInstanceSetup, tmalVdegMpegInstanceSetup`, triggered via `tmAssert`.
`TMLIBAPP_OK`	On success.

or, in case of `tmolVdecMpegStart`, any error code returned by `tsaDefaultStart`.

Description

The previously opened and set up instance of the decoder is started. It is expected that the empty queues of the video output contains empty video packets, with allocated `tmalVdecMpegPictureInfo_t` allocated and assigned to the `userPointer` of the packets. Then the decoder starts to wait for input data from the input queue.

tmolVdecMpegStop, tmalVdecMpegStop

```
extern tmLibappErr_t tmolVdecMpegStop(
   Int   instance
);
```

Parameters

`instance`	Instance previously opened by `tmolVdecMpegOpen, tmalVdecMpegOpen`.

Return Codes

`TMLIBAPP_ERR_INVALID_INSTANCE`	When the instance is not a valid instance open with `tmolVdecMpegOpen, tmalVdecMpegOpen`, triggered via `tmAssert`.
`TMLIBAPP_ERR_NOT_OPEN`	When the instance is not opened with `tmolVdecMpegOpen, tmalVdecMpegOpen`, triggered via `tmAssert`.
`TMLIBAPP_OK`	On success.

or, in case of `tmolVdecMpegStop`, any error code returned by `tsaDefaultStop`.

Description

After a call to Stop, the VdecMpeg instance can be restarted via a call to Start. Stop does not free the internally claimed memory.

tmolVdecMpegClose, tmalVdecMpegClose

```
extern tmLibappErr_t tmolVdecMpegClose(
   Int   instance
);
extern tmLibappErr_t tmalVdecMpegClose(
   Int   instance
);
```

Parameters

`instance`	Instance previously opened by tmolVdecMpegOpen, tmalVdecMpegOpen.

Return Codes

`TMLIBAPP_ERR_INVALID_INSTANCE`	When the instance is not a valid instance open with `tmolVdecMpegOpen, tmalVdecMpegOpen`, triggered via `tmAssert`.
`TMLIBAPP_ERR_NOT_STOPPED`	When the instance is not stopped before, triggered via `tmAssert`.
`TMLIBAPP_OK`	On success.

Or, in case of `tmolVdecMpegClose`, any return value from `tsaDefaultClose`.

Description

Closes a stopped VdecMpeg instance.

tmolVdecMpegInstanceConfig

```
extern UInt32 tmolVdecMpegInstanceConfig(
   Int                  instance,
   UInt32               flags,
   ptsaControlArgs_t    args
);
```

Parameters

`instance`	Instance previously opened by `tmolVdecMpegOpen`.
`flags`	Ignored
`args`	`args->command` is one of the command codes from `tmalVdecMpegControlCommand_t`. There are no other required fields to be set in args.

Return Codes

`TMLIBAPP_ERR_INVALID_INSTANCE`	When the instance is not a valid instance open with `tmolVdecMpegOpen`, triggered via `tmAssert`.
`TMLIBAPP_ERR_NOT_OPEN`	When the instance is not opened with `tmolVdecMpegOpen`, triggered via `tmAssert`.
`TMLIBAPP_ERR_NOT_SETUP`	When the instance is not set up with `tmolVdecMpegInstanceSetup`, triggered via `tmAssert`.
`TMLIBAPP_OK`	On success.

Description

See `tmalVdecMpegControlCommand_t` for possible control commands.

tmalVdecMpegInstanceConfig

```
extern UInt32 tmalVdecMpegInstanceConfig(
   Int                instance,
   ptsaControlArgs_t  args
);
```

Parameters

`instance`	Instance previously opened by tmalVdecMpegOpen.
`args`	`args->command` is one of the command codes from tmalVdecMpegControlCommand_t. There are no other required fields to be set in args.

Return Codes

`TMLIBAPP_ERR_INVALID_INSTANCE`	When the instance is not a valid instance open with `tmalVdecMpegInstanceConfig`, triggered via tmAssert.
`TMLIBAPP_ERR_NOT_OPEN`	When the instance is not opened with `tmalVdecMpegOpen`, triggered via `tmAssert`.
`TMLIBAPP_ERR_NOT_SETUP`	When the instance is not set up with `tmalVdecMpegInstanceSetup`, triggered via `tmAssert`.
`TMLIBAPP_OK`	On success.

Description

See `tmalVdecMpegControlCommand_t` for possible control commands. Control commands are handled on all blocking datain and dataout functions.