

Dirac video codec: A programmer's guide

[Table of Contents](#) | [Introduction](#) 

This document details the Dirac video codec from the perspective of a programmer, focussing on the public API. This document is WORK IN PROGRESS. This document was last updated on 21 November 2008.

Authors

Anuradha Suraparaju, BBC R&D, November 2008

Chris Bowley, BBC R&D, August 2004

1st draft by Scott R Ladd, May 2004



Dirac: A programmer's guide

[↩ Abstract](#)

Table of Contents

1. [Introduction](#)
2. [Overview of the codec](#)
 - ◆ [Video Coding principles](#)
 - ◆ [Codec Architecture](#)
 - ◆ [Picture sequence structure](#)
 - ◆ [Codec Library structure](#)
3. [Software Environment](#)
 - ◆ [GNU/Linux, UNIX, MACOS X, Cygwin, MingW](#)
 - ◆ [MS Windows, MSYS, VC++ 2008 Express Edition](#)
 - ◆ [MS Windows, Visual C++ 2008 Express Edition](#)
4. [Input Formats](#)
5. [Encoder API](#)
 - ◆ [API Overview](#)
 - ◆ [API Reference](#)
 - ◆ [Code Examples](#)
6. [Decoder API](#)
 - ◆ [API Overview](#)
 - ◆ [API Reference](#)
 - ◆ [Code Examples](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Abstract](#) | [Codec overview](#)

Introduction

This document is aimed at the users of the Dirac codec release version 1.0.2. The layout of this guide is as follows

Chapter 2, [Overview of the Codec](#), gives a brief overview of the Dirac coder. Little details are given of the underlying algorithms used in the Dirac codec and the reader is advised to look at [the Dirac algorithm](#) for further information.

Chapters 3 thru 6 are aimed at the user of the Dirac Codec Library .e.g users who wish to include Dirac encode/decode functionality in Media Players, video processing tools, etc.

Chapter 3, [Software Environment](#), describes how to download Dirac software and build the Dirac codec libraries.

Chapter 4, [Input Formats](#), describes the uncompressed input formats that the Dirac Codec library supports and the utilities available to convert from other formats to that required by the codec library.

Chapter 5, [Encoder API](#), describes the public API to the Dirac Encoder in detail. It also includes a simple working example as to how to use the Encoder API.

Chapter 6, [Decoder API](#), describes the public API to the Dirac Decoder in detail. It also includes a simple working example as to how to use the Decoder API.

It is beyond the scope of this document to fully describe the algorithms used in the Dirac codec. Details of the underlying algorithms used can be found in [the Dirac algorithm](#). Further educational material and tutorials on signal processing and data compression can be found at:

Digital signal processing guru

www.dspguru.com

The scientist and engineer's guide to digital signal processing

www.dspguide.com

University of Birmingham Dept. of electronics, electrical and computer engineering: data compression


www.eee.bham.ac.uk/woolleysi/links/datacomp.htm

ACM Siggraph education

www.siggraph.org/education



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Introduction](#) | [Software Environment](#) 

Codec overview

This section details the basic principles on which the Dirac codec is built. The structure of codec is also outlined. If you are familiar with video compression techniques you may wish to skip to the [Codec Library Structure](#) section below.

Video Coding principles

Video compression is the process of condensing a digital video signal into a smaller number of bits for practical storage and transmission of digital video. Compression involves a pair of systems, an encoder and a decoder. The encoder converts the source data into a compressed bitstream prior to transmission or storage and the decoder converts the compressed bitstream into uncompressed video data. The encoder/decoder pair is often described as a CODEC (enCOder/DECoder).

Data compression can be classified into two types - lossy compression and lossless compression. Lossless compression reduces the amount of information required to completely recreate the original data. Many types of data contains statistical redundancy that can be exploited to compress the data. However, lossless compression of video data gives only amount of compression.

Lossy compression cannot result in the recreation of the original data but may yield a much greater compression ratio (amount of original data to compressed data). Some of the original data is lost in the process which manifests itself as a perceived loss of quality in the compressed video sequence. The higher the compression ratio, in general, the more data is thrown away. The goal of video compression is to yield the maximum compression ratio whilst maintaining the highest level of video quality. Lossy video compression exploits characteristics in the video sequence which allow redundant information to be thrown away. This is split into two forms: *spatial* and *temporal* compression. Spatial compression removes redundancies within a single frame and employs the same principles as image compression. Temporal compression relies on the similarity between successive frames, removing redundancy by utilising motion prediction and compensation. Further compression is achieved by encoding the output of the spatial and temporal compression modules using the statistical redundancy methods of lossless compression. This is known as entropy coding.

[top](#)

Codec Architecture

Dirac is a general-purpose, high compression, lossy video codec. It consists of three main elements - the temporal compression module, the the spatial compression module and the entropy coder.

Spatial Compression

In Dirac, spatial compression is achieved using wavelet decomposition which represents the image in the frequency domain and exploits the eye's lack of sensitivity to noise in high frequencies. The wavelet decomposition converts the input data into the frequency domain in which they are respresented as transform coefficients. These coefficients are qauntised to remove insignificant values to provide a more compact

representation. The output of this module is a set of quantised transform coefficients.

Temporal compression

In Dirac, temporal compression is achieved by construction a prediction of the current video frame by exploiting the similarities between neighbouring video frames. Block-based motion estimation and compensation methods are used to predict a current frame from one or two of the previous or future frames. This prediction is then subtracted from the current frame to get a residual frame. The output of this modules comprises of the residual frame and the set of motion vectors describing how the motion was compensated. The residual frame is then spatially compressed using the technique described in the section above.

Entropy Coder

The inputs to this modules are the transform coefficients and motion vectors of the Spatial Compression module and Temporal Compression modules. Dirac uses arithmetic coding which removes statistical redundancy from the output data by representing common values with short bit codes.

[▲top](#)

Picture sequence structure

In video compression, some frames must be encoded without reference to any other frame in order to avoid cascading picture degradation and when there are no previously coded frames (and also to allow the picture sequence to be decoded from an arbitrary point). Such frames are known as *intra* frames. Frames which are compressed with reference to other previous and/or subsequent frames using motion prediction are known as *inter* frames. The ratio of inter frames to intra frames can be specified for each picture sequence and the choice may depend on application. In Dirac, any frame structure is permitted and due to the nature of motion prediction and compensation, the order of frames in encoded form is usually different to the original display order.

[▲top](#)

Codec Library structure

The current version of Dirac (1.0.x) comprises of two libraries, `libdirac_encoder` and `libdirac_decoder`. Source code related these two libraries is maintained in the following directories.

`libdirac_byteio`

Classes for IO. Used by both `libdirac_encoder` and `libdirac_decoder`.

`libdirac_common`

Classes for general-purpose functions including wavelet transformation and arithmetic encoding and decoding. Used by both `libdirac_encoder` and `libdirac_decoder` and additional Dirac tools.

`libdirac_encoder`

Classes for management of compression of sequences, frames and components. Used exclusively by the `libdirac_encoder`.

`libdirac_motionest`

Classes for management of motion estimation and mode decision. Used exclusively by the `libdirac_encoder`.

libdirac_decoder

Classes for management of decompression of sequences, frames and components. Used exclusively by the libdirac_decoder.

In addition to the encoder and decoder libraries, the codec has several utilities which are maintained in the following directories.

encoder

Utility to encode data. Uses libdirac_encoder.

decoder

Utility to decode data. Uses libdirac_decoder.

util/conversion

Conversion utilities to convert data to input format required by Dirac.

util/instrumentation

Diagnostics library and utility. The [Diagnostic Instrumentation](#) document has more information on how to use this utility.

unit_tests

Unit test based on the CPPUnit test framework.

tests

Test suite based on GNU Autotest and test data.

[▲top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Overview](#) | [Input file formats](#)

Software Environment

The core Dirac codec is implemented in platform-neutral ISO Standard C++. The current encoder and decoder run from the command prompt and do not require specific operating system elements. The Dirac source code is distributed as a compressed UNIX tarball, which can be downloaded from [here](#) or [here](#). This tarball can be extracted with utilities such as the Unix tar command or the Windows WinZip utility.

GNU/Linux, UNIX, MacOS X, Cygwin, MinGW

Under these environments, Dirac has been successfully compiled and tested with the GNU g++ compiler (version 3.4.x and 4.0.x, 4.1.x, 4.2.x, 4.3.x) and Intel's C++ compiler (icc version 8.0). Core Dirac libraries have no dependencies on any libraries beyond the standard C and C++ libraries that are included with the compiler. Unit tests in Dirac have been implemented using the CppUnit Test Framework v1.10. If this library is not available then unit tests are automatically skipped during the build.

The Dirac distribution includes a configure script generated using GNU Autotools. This script is used to create the files necessary to build the codec. To build the codec and utilities, the sequence of commands is

```
configure
make
make install
```

The last command must be executed with root privileges so that the libraries and programs will be copied to the appropriate directories. By default, the libraries are copied to /usr/local/lib, the header files to /usr/local/include/dirac and the utilities to /usr/local/bin.

The configure script accepts several command line arguments to customise the build. Use the following command to get a full list of the configure options available.

```
configure --help
```

Some of the more useful options are described below -

--prefix

Use the directory name specified as the installation location prefix instead of the default. For e.g.

```
configure --prefix=/opt/local
```

--enable-debug

Build with optimisation turned off and debug flags enabled

--enable-profile

Build with profiling flags enabled

--disable-shared

By default, both shared and static libraries are built. Use this option to only build the static libraries.

--disable-static

By default, both shared and static libraries are built. Use this option to only build the shared libraries

--disable-mmx

By default, MMX optimisations are enabled on x86 platforms. Use this option to disable MMX optimisations.

[top](#)

MS Windows 2000/XP, MSYS and Microsoft Visual C++ 2008

In this environment, Dirac has been successfully compiled and tested using MSYS version 1.0.10 and the no-cost Microsoft Visual C++ 2008 Express Edition.

Download and install the no-cost Microsoft C++ compiler from [here](#). Make sure that the PATH, INCLUDE and LIB environment variables are set correctly to point to the MS VC++ 2008 Express Edition.

Download and install MSYS (the MinGW Minimal SYSTEM), MSYS-1.0.10.exe, from [here](#). An MSYS icon will be available on the desktop. The MSYS environment inherits the PATH, LIB and INCLUDE environment variables so they need not be set again. Click on the MSYS icon to open an MSYS shell window and change directory to where the Dirac distribution was unpacked. The [command sequence](#) described in the previous section can now be used to build Dirac. Note that in this environment, the user can either build shared libraries or static libraries but not both. By default, shared libraries are built. Static libraries can be built by specifying the --disable-shared option on the configure command line.

[top](#)

MS Windows 2000/XP, Microsoft Visual C++ 2008 Express Edition

The MS VC++ 2008 Express edition solution and project files are in the win32/VisualStudio directory. Double-click on the solution file, dirac.sln, in this directory to bring up the IDE. The target 'Everything' builds the codec, libraries and utilities. Eight types of build are supported:

Debug - builds unoptimised encoder and decoder DLLs, encoder and decoder apps and utilities with debug symbols. The "C" public API is exported using the _declspec(dllexport) mechanism.

Release - builds optimised encoder and decoder DLLs, encoder and decoder apps and utilities. The "C" public API is exported using the _declspec(dllexport) mechanism.

Debug-mmx - builds unoptimised encoder and decoder DLLs, encoder and decoder apps and utilities with debug symbols and mmx optimisations enabled. The "C" public API is exported using the _declspec(dllexport) mechanism.

Release-mmx - builds optimised encoder and decoder DLLs, encoder and decoder apps and utilities with mmx optimisations enabled. The "C" public API is exported using the _declspec(dllexport) mechanism.

Static-Debug - builds unoptimised encoder and decoder static libraries, encoder and decoder apps with debug symbols.

Static-Release - builds optimised encoder and decoder static libraries, encoder and decoder apps.

Static-Debug-mmx - builds unoptimised encoder and decoder static libraries, encoder and decoder apps with debug symbols and mmx optimisations enabled.

Static-Release-mmx - builds optimised encoder and decoder static libraries, encoder and decoder apps with mmx optimisations enabled.

Static libraries are created in the win32/VisualStudio/build/lib/<build-type> directory.

Encoder and Decoder DLLS and import libraries, encoder and decoder apps are created in the win32/VisualStudio/build/bin/<build-type> directory.

Conversion utilities are created in the win32/VisualStudio/build/utls/conversion/<build-type> directory, where build-type is Debug or Release.

Instrumentation utility is created in the win32/VisualStudio/build/utls/instrumentation/<build-type> directory, where build-type is Debug or Release.

[top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Software Environment](#) | [Encoder API Contents](#)

Input Formats

At this time, Dirac supports planar YUV format streams. YUV is a popular way of representing colour images. It separates the brightness information (Y) from the colour information (U and V). The Y component is a weighted average of the three additive primaries - red, green and blue. The U component is the difference between the blue primary and the Y component. The V component is the colour difference between the red primary and Y component. All three colour primaries can be reconstructed from the Y, U and V components. Because the human eye is more sensitive to brightness than it is to colour, the chroma components generally are recorded with fewer samples than is the brightness data.

The supported YUV formats are:

Chroma Format	YUV format Description
format444	Planar 4:4:4 format. Full chroma resolution in both vertical and horizontal directions.
format422	Planar 4:2:2 format. Full vertical chroma resolution and 1/2 horizontal chroma resolution.
format420	Planar 4:2:0 format. 1/2 chroma resolution in both vertical and horizontal directions.

Dirac does not support any other input formats currently. The util/conversion directory has a number of utilities to and from Dirac formats to other formats like UYVY, RGB and BMP.

[top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Input Formats](#) | [Decoder API Contents](#)

Encoder API

Encoder API contents:

- [API Overview](#)
- [API Reference](#)
- [Code Examples](#)

[top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Back to Encoder API Contents](#) | [API Reference](#)

Encoder API Overview

Though Dirac is implemented in C++, the public API of the encode engine is implemented in "C". The idea behind this is to have a stable public interface even if the underlying C++ implementation changes radically.

The Encoder API can be divided into the following categories:

- Base Data Structures
- Setup/Teardown functions
- Encoding functions

Base Data Structures

There are several structures that hold information required by the encoder during the encoding process. These structures are defined in `libdirac_common/dirac_types.h` and `libdirac_encoder/dirac_encoder.h`.

<u>dirac_encoder_state_t</u>	State of the encoder
<u>dirac_encoder_presets_t</u>	Presets used to initialise the encoder context
<u>dirac_prefilter_t</u>	Prefilter types used in the Encoder
<u>dirac_mvprecision_t</u>	Motion vector precision used in the Encoder
<u>dirac_wlt_filter_t</u>	Wavelet filters used in the Encoder
<u>dirac_encparams_t</u>	Parameters specific to Dirac Encoder
<u>dirac_sourceparams_t</u>	Parameters common to the source material being encoded.
<u>dirac_encoder_context_t</u>	Encoder context used to initialise the sequence and encoder parameters.
<u>dirac_enc_data_t</u>	Buffer to hold encoded data. Allocated by the Encoder Library user
<u>dirac_enc_picstats_t</u>	Encoded picture statistics
<u>dirac_enc_seqstats_t</u>	Encoded Sequence Statistics
<u>dirac_mv_t</u>	Motion Vector Information for each block.
<u>dirac_mv_cost_t</u>	Costs associated with motion vectors. The Encoder uses this information to determine if the block of data is intra-coded or inter-coded.
<u>dirac_instr_t</u>	Structure that holds the instrumentation data for the encoded picture.
<u>dirac_picparams_t</u>	Encoded/Decoded picture parameters
<u>dirac_framebuf_t</u>	Uncompressed/Decoded frame
<u>dirac_encoder_t</u>	Encoder handle

[top](#)

Setup/Teardown Functions

<u>dirac_encoder_context_init</u>	Initialise the encoder context with a preset value. This is a convenience function that can be used to set up the encoding parameters. If used, it should be called before calling the Dirac Encoder initialisation function.
<u>dirac_encoder_init</u>	Initialises the Dirac encoder. This function must be called before the encoding functions are used. It returns a <code>dirac_encoder_t</code> structure that should be used in the subsequent calls to encoding and teardown functions.
<u>dirac_encoder_pts_offset</u>	This function queries the encoder for the reordering depth. This function must be called after <code>dirac_encoder_init</code> and before the encoding functions are used. It returns the number of pictures a realtime decoder must wait before outputting the first picture in display order upon success.
<u>dirac_encoder_close</u>	Frees up the Dirac Encoder resources. It must be called after the encoding process. The <code>dirac_encoder_t</code> structure returned by <code>dirac_encoder_init</code> is passed as a parameter to this function. After this function returns, the <code>dirac_encoder_t</code> structure is no longer valid and cannot be used for other encoding/teardown operations.

[▲top](#)

Encoding functions

<u>dirac_encoder_load</u>	This function loads a frame of uncompressed data into the encoder. The input parameters are a <code>dirac_encoder_t</code> structure returned by a previous call to <code>dirac_encoder_init</code> , a pointer to the buffer of uncompressed data, and the size of the buffer.
<u>dirac_encoder_output</u>	This function retrieves data from the Dirac Encoder. It takes a <code>dirac_encoder_t</code> structure returned by a previous call to <code>dirac_encoder_init</code> .
<u>dirac_encoder_end_sequence</u>	This function requests the encoder to end the sequence. It must be called at the end of encoding a sequence and just before calling <code>dirac_encoder_close</code> .

[▲top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Back to Encoder API Contents](#) | [API Overview](#) | [Code Examples](#)

Encoder API Reference

Base Data Structures

[dirac_encoder_state_t](#)
[dirac_encoder_presets_t](#)
[dirac_prefilter_t](#)
[dirac_mvprecision_t](#)
[dirac_wlt_filter_t](#)
[dirac_encparams_t](#)
[dirac_sourceparams_t](#)
[dirac_encoder_context_t](#)
[dirac_enc_data_t](#)
[dirac_enc_picstats_t](#)
[dirac_enc_seqstats_t](#)
[dirac_mv_t](#)
[dirac_mv_cost_t](#)
[dirac_instr_t](#)
[dirac_picparams_t](#)
[dirac_framebuf_t](#)
[dirac_encoder_t](#)

Setup/Teardown Functions

[dirac_encoder_context_init](#)
[dirac_encoder_init](#)
[dirac_encoder_pts_offset](#)
[dirac_encoder_close](#)

Encoding functions

[dirac_encoder_load](#)
[dirac_encoder_output](#)
[dirac_encoder_end_sequence](#)

[top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Back to Encoder API Contents](#) | [API Reference](#)

Encoder API Example

The following code describes a simple example on how to use the Dirac Encoder API. For a more detailed usage refer to the sample encode application, `encoder/dirac_encoder.cpp`, in the Dirac distribution. The following sample reads uncompressed Standard Definition format digital data (SD576I50) in Planar YUV 4:2:0 format from an input file, and writes the encoded bitstream to an output file.

```
#include <stdio.h>
#include <errno.h>

/* Include Dirac Encoder Header file */
#include <libdirac_encoder/dirac_encoder.h>

#define ENCBUF_SIZE (1024*1024)

extern int main (int argc, char **argv)
{
    FILE *in;
    FILE *out;
    dirac_encoder_context_t enc_ctx;
    dirac_encoder_t *encoder;
    unsigned char *unc_frame;
    int unc_frame_size;
    unsigned char enc_frame[ENCBUF_SIZE];
    int go = 1;
    dirac_encoder_state_t state;

    if (argc < 3)
    {
        printf ("Usage : %s input-file output-file\n", argv[0]);
        exit(0);
    }

    /* open input file */
    if ( (in = fopen (argv[1], "rb")) == NULL)
    {
        perror (argv[1]);
        exit(errno);
    }

    /* open output file */
    if ( (out = fopen (argv[2], "wb")) == NULL)
    {
        perror (argv[2]);
        exit(errno);
    }

    /* Initialise the encoder context with presets for SD576I50 */
    dirac_encoder_context_init (&enc_ctx, VIDEO_FORMAT_SD576I50);

    /* override some of the preset defaults */
    /* Set quality factor to 7.5 */
    enc_ctx.enc_params.qf = 7.5;

    /* set input video type to progressive video */
    enc_ctx.source_params.source_sampling = 0;
```

```

/* set flag to retrieve locally decoded frames from encoder */
enc_ctx.decode_flag = 1;

/* Initialise the encoder with the encoder context */
if ( (encoder = dirac_encoder_init(&enc_ctx, 0)) == NULL)
{
    printf ("Error initialising the encoder\n");
    exit(-1);
}

/* Calculate the size of an uncompressed frame */
unc_frame_size = (encoder->enc_ctx.src_params.height *
    encoder->enc_ctx.src_params.width) +
    2*(encoder->enc_ctx.src_params.chroma_height
        * encoder->enc_ctx.src_params.chroma_width);

/* Allocate memory for an uncompressed frame */
unc_frame = (unsigned char *)malloc(unc_frame_size);

/* Main loop */
do
{
    /*
    * Read one frame of uncompressed data into the buffer and
    * load it into the encoder
    */
    if ( fread( unc_frame, unf_frame_size, 1, in) == 1)
    {
        if (dirac_encoder_load( encoder, unc_frame, unc_frame_size ) < 0)
        {
            fprintf (stderr, "dirac_encoder_load failed. Unrecoverable error...\n");
            exit(-1);
        }
    }
    else
    {
        /* signal end of sequence to encoder */
        dirac_encoder_end_sequence( encoder );
    }

    do
    {
        /*
        * Frame loaded successfully, so now set up the encode
        * buffer in the encoder handler before calling the
        * encoding function.
        */
        encoder->enc_buf.buffer = enc_frame;
        encoder->enc_buf.size = ENCBUF_SIZE;

        /* Call the encoding function */
        state = dirac_encoder_output (encoder);

        /*
        * Depending on the return value of the encode function
        * take appropriate action
        */
        switch (state)
        {
            case ENC_STATE_AVAIL:
                /*
                * Encoded picture available in enc_buf. Write the
                * buffer to the output file
                */

```

```

        fwrite (encoder->enc_buf.buffer, encoder->enc_buf.size,
                1, out);
        /*
        * In addition to the encoded picture, the following metadata
        * is also available.
        * encoded picture stats in encoder->enc_pstats
        * encoded picture params in encoder->enc_pparams
        */
        break;

    case ENC_STATE_BUFFER:
        /*
        * Encoder needs more data to continue processing
        */
        break;

    case ENC_STATE_EOS:
        /*
        * End of sequence info available in enc_buf. Write the
        * buffer to the output file
        */
        fwrite (encoder->enc_buf.buffer, encoder->enc_buf.size,
                1, out);
        /*
        * In addition to the encoded picture, the following metadata
        * is also available.
        * encoded sequence stats in encoder->enc_seqstats
        */
        break;

    case ENC_STATE_INVALID:
    default:
        printf ("Irrecoverable error. quitting...");
        free (unc_frame);
        dirac_encoder_close(encoder);
        exit(-1);
        break;
    }
    if (encoder->decoded_frame_avail)
    {
        /*
        * Locally decoded frame available in encoder->dec_buf
        */
    }
    if (encoder->instr_data_avail)
    {
        /*
        * Instrumentation data available in encoder->instr
        */
    }
    } while (state == ENC_STATE_AVAIL);
} while (go);

/* Free the encoder resources */
dirac_encoder_close(encoder);

/* Free the uncompressed data buffer */
free (unc_frame);

fclose (in);
fclose (out);
}

```

[▲top](#)



Dirac video codec: A programmer's guide

[☰ Table of Contents](#) | [Back to Encoder API](#) | [Back to Decoder API](#)

Common Base Data Structures

The following structures are common to the Dirac Encoder and Decoder API

DIRAC API VERSION

Declared in libdirac_common/dirac_types.h

These macros are provided to determine the API version.

DIRAC_RESEARCH_MAJOR_VERSION

Major version corresponds to major version of the software.

```
#define DIRAC_RESEARCH_MAJOR_VERSION 1 /* 0..255 */
```

DIRAC_RESEARCH_MINOR_VERSION

Minor version corresponds to minor version of the software. This is bumped up by one whenever there are major feature changes to the software.

```
#define DIRAC_RESEARCH_MINOR_VERSION 0 /* 0..255 */
```

DIRAC_RESEARCH_PATCH_VERSION

Patch version corresponds to changes in the API. It should be bumped up by 1 for every committed change to the API

```
#define DIRAC_RESEARCH_PATCH_VERSION 2 /* 0..255 */
```

DIRAC_RESEARCH_VERSION_ATLEAST(x, y, z)

Check if the API version is atleast x.y.z before using an api function or variable. Sample usage

```
if DIRAC_RESEARCH_VERSION_ATLEAST(1, 0, 2) {  
    enc_params.combined_me = 1;  
}
```

dirac_chroma_t

Declared in libdirac_common/common_types.h

This enumerated type defines the chroma sampling formats that Dirac supports.

```
typedef enum {  
    format444,  
    format422,  
}
```

```
    format420,  
    formatNK  
} ChromaFormat;  
typedef ChromaFormat dirac_chroma_type_t;
```

Enumerated type values

format444
Planar YUV 4:4:4 format

format422
Planar YUV 4:2:2 format

format420
Planar YUV 4:2:0 format

formatNK
Unknown format

[top](#)

dirac_picture_type_t

Declared in libdirac_common/common_types.h
This enumerated type lists the encoded picture types in Dirac.

```
typedef enum  
{  
    INTRA_PICTURE,  
    INTER_PICTURE  
} PictureType;  
typedef PictureType dirac_picture_type_t;
```

Enumerated type values

INTRA_PICTURE
Intra picture i.e picture encoded without reference to any other picture

INTER_PICTURE
Inter picture, i.e. a picture encoded with reference to other pictures

[top](#)

dirac_reference_type_t

Declared in libdirac_common/common_types.h
This enumerated type lists the encoded reference types in Dirac.

```
typedef enum  
{  
    REFERENCE_PICTURE,  
    NON_REFERENCE_PICTURE,  
} ReferenceType;  
typedef ReferenceType dirac_reference_type_t;
```

Enumerated type values

REFERENCE_PICTURE

This picture is used as a reference picture for another picture

NON_REFERENCE_PICTURE

This picture is a non-reference picture i.e. it is not used as a reference picture for other pictures

[top](#)

dirac_prefilter_t

Declared in libdirac_common/common_types.h

This enumerated type lists the prefilter types in Dirac.

```
typedef enum
{
    NO_PF,
    DIAGLP,
    RECTLP,
    CWM
} PrefilterType;
typedef PrefilterType dirac_prefilter_t;
```

Enumerated type values

NO_PF

No prefiltering used on input picture

DIAGLP

Diagonal low pass filter applied to input picture before encoding

RECTLP

Rectangular low pass filter applied to input picture before encoding

CWM

Centre weighted median filter applied to input picture before encoding

[top](#)

dirac_wlt_filter_t

Declared in libdirac_common/common_types.h

This enumerated type lists the wavelet filter types used in Dirac.

```
typedef enum
{
    DD9_7,
    LEGALL5_3,
    DD13_7,
    HAAR0,
    HAAR1,
    FIDELITY,
    DAUB97,
    filterNK
} WltFilter;
typedef WltFilter dirac_wlt_filter_t;
```

Enumerated type values

DD9_7

Deslauriers-Dubuc (9,5)

LEGALL5_3

LeGall (5,3)

DD13_7

Deslauriers-Dubuc (13,5)

HAARO

Haar, no shift per level

HAARI

Haar, one shift per level

FIDELITY

Fidelity wavelet

DAUB97

Integer Approximation of Daubechies filter

filterNK

Unknown wavelet filter

[top](#)

dirac_rational_t

Declared in libdirac_common/dirac_types.h

This structure represents a rational number as a combination of two values - a numerator and a denominator.

```
typedef struct
{
    int numerator;
    int denominator;
} dirac_rational_t;
typedef dirac_rational_t dirac_frame_rate_t;
typedef dirac_rational_t dirac_pix_asr_t;
```

[top](#)

dirac_parseparams_t

Declared in libdirac_common/dirac_types.h

This structure defines the parse parameters in the Access Unit Header. An Access Unit is a point in the bytestream from where the decoder can start decoding.

```
typedef struct
{
    unsigned int major_ver;
    unsigned int minor_ver;
    unsigned int profile;
    unsigned int level;
} dirac_parseparams_t;
```

Structure Members

major_ver

Major version of the encoder used to encode the stream

minor_ver

Minor version of the encoder used to encode the stream

Profile

Profile of the encoder used to encode the stream.

Level

Level at which the encoder coded the stream.

[▲top](#)

dirac_clean_area_t

Declared in libdirac_common/dirac_types.h

This structure represents the actual viewing area in the picture. It covers an area less than or equal to the actual picture dimensions specified in the [Source parameters](#).

```
typedef struct
{
    unsigned int width;
    unsigned int height;
    unsigned int left_offset;
    unsigned int top_offset;
} dirac_clean_area_t;
```

[▲top](#)

dirac_signal_range_t

Declared in libdirac_common/dirac_types.h

This structure describes how the signal is coded and is purely metadata describing the source. It provides information about how a signal is to be restored for display.

```
typedef struct
{
    unsigned int luma_offset;
    unsigned int luma_excursion;
    unsigned int chroma_offset;
    unsigned int chroma_excursion;
} dirac_clean_area_t;
```

[▲top](#)

dirac_col primaries_t

Declared in libdirac_common/common_types.h

This enumerated type defines the colour primaries that Dirac supports.

```
typedef enum {
    CP_HDTV_COMP_INTERNET,
    CP_SDTV_525,
    CP_SDTV_625,
    CP_DCINEMA,
    CP_UNDEF
} CplourPrimaries;
typedef ColourPrimaries dirac_col_primaries_t;
```

Enumerated type values

CP_HDTV_COMP_INTERNET
HDTV, Computer and Internet (ITU709 & sRGB)

CP_SDTV_525
SMPTE C as used to NTSC

CP_SDTV_625
EBU Tech 3213 as used fo PAL

CP_DCINEMA
CIE XYZ for lossless coding

CP_UNDEF
Unknown colour primary

dirac_transfer_func_t

Declared in libdirac_common/common_types.h

This enumerated type defines the transfer function characteristics that Dirac supports.

```
typedef enum {
    TF_TV,
    TF_EXT_GAMUT,
    TF_LINEAR,
    TF_REVERSIBLE,
    TF_UNDEF
} TransferFunction;
typedef TransferFunction dirac_transfer_func_t;
```

Enumerated type values

TF_TV
Transfer functiion used by TV

TF_EXT_GAMUT
Extended colour gamut

TF_LINEAR
Linear characteristic

TF_REVERSIBLE

Digital Cinema

TF_UNDEF

Unknown Transfer function index

dirac_col_matrix_t

Declared in libdirac_common/dirac_types.h

This structure holds the colour matrix values Dirac supports.

```
typedef struct {
    float kr;
    float kb;
} dirac_col_matrix_t;
```

dirac_colour_spec_t

Declared in libdirac_common/dirac_types.h

This structure describes colour specifications of the source material supported by Dirac.

```
typedef struct
{
    dirac_col primaries_t col_primary;
    dirac_col_matrix_t col_matrix;
    dirac_transfer_func_t trans_func;
} dirac_colour_spec_t;
```

[▲top](#)

dirac_sourceparams_t

Declared in libdirac_common/dirac_types.h

This structure defines the parameters common to the video sequence to be encoded by the encoder or the sequence in the bitstream being decoded. Some Source parameters like width, height, chroma, source_sampling, topfieldfirst, frame_Rate, are necessary for encoding/decoding the video sequence. The rest of the parameters are not necessary to decode a sequence but are useful to the end-user application to correctly display the sequence.

```
typedef struct
{
    unsigned int width;
    unsigned int height;
    dirac_chroma_t chroma;
    unsigned int chroma_width;
    unsigned int chroma_height;
    unsigned int source_sampling;
```

Enumerated type values

```

int topfieldfirst;
dirac_frame_rate_t frame_rate;
dirac_pix_asr_t pix_asr;
dirac_clean_area_t clean_area;
dirac_signal_range_t signal_range;
dirac_colour_spec_t colour_spec;
} dirac_srcparams_t;

```

Structure Members

width

Number of pixels per line in the input/decoded frame

height

Number of lines in the input/decoded frame

chroma

Chroma sampling format. This field is of type dirac_chroma_t.

chroma_width

Number of pixels of chroma per line. Calculated from the values of *width* and *chroma*.

chroma_height

Number of lines of chroma in the input frame. Calculated from the values of *height* and *chroma*.

source_sampling

Determines whether the input/decoded sequence is interlaced or progressive. It takes one of two values -

0 - Progressive

1 - Interlaced

topfieldfirst

If the input/decoded sequence is interlaced, this field determines if the top field precedes the bottom field in time. It takes one of two values

0 - Bottom field precedes the top field in time

1 - Top field precedes the bottom field in time

frame_rate

The number of frame per second. This field is of type dirac_frame_rate_t.

pix_asr

The pixel aspect ratio. This field is of type dirac_pix_asr_t.

clean_area

The actual viewable area. This is less than or equal to the actual picture dimension. This field is of type dirac_clean_area_t.

signal_range

This field defined how the signal is coded within the bits specified in the video depth. This field is of type dirac_signal_range_t.

colour_spec

The colour specification of the source video. This field is of type dirac_colour_spec_t.

[▲top](#)

dirac_picparams_t

Declared in libdirac_common/dirac_types.h

This structure hold the parameters of an encoded/decoded picture.

dirac_sourceparams_t

```
typedef struct
{
    dirac_picture_type_t ptype;
    dirac_reference_type_t rtype;
    int pnum;
} dirac_picparams_t;
```

Structure Members

ptype

Type of encoded/decoded picture. This field is of type [dirac_picture_type_t](#).

rtype

Specified in the encoded/decoded picture is a reference picture for another picture. This field is of type [dirac_reference_type_t](#).

pnum

Picture number of encoded/decoded frame in display order.

[▲top](#)

dirac_framebuf_t

Declared in `libdirac_common/dirac_types.h`

This structure holds the uncompressed/decoded frame data in Y, U, V format.

```
typedef struct
{
    unsigned char *buf[3];
    void *id;
} dirac_framebuf_t;
```

Structure members

buf

An array of 3 buffers to hold the Y, U and V component of the uncompressed frame to be encoded or the decoded frame

id

User data. The encoder does not use this field. The decoder does not modify this field if it is set by the end user

[▲top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Back to Encoder API](#)

Encoder Base Data Structures

The following structures are used exclusively by the Dirac Encoder and API

dirac_encoder_state_t

Declared in libdirac_encoder/dirac_encoder.h

This enumerated type defines the state of the Dirac encoder.

```
typedef enum {
    ENC_STATE_INVALID = -1,
    ENC_STATE_BUFFER,
    ENC_STATE_AVAIL,
    ENC_STATE_EOS
} dirac_encoder_state_t;
```

Enumerated type values

ENC_STATE_INVALID

The Dirac encoder has encountered an irrecoverable error. Stop further processing

ENC_STATE_BUFFER

The Dirac Encoder needs further data input to continue processing

ENC_STATE_AVAIL

An encoded picture is available to the end user

ENC_STATE_EOS

End of sequence information is available to the end user

[top](#)

dirac_encoder_presets_t

Declared in libdirac_common/common_types.h

This enumerated type defines the presets used to initialise the Dirac encoder. The Dirac encoder needs several parameters to make encoding decisions. To make things simpler for the end user, a set of presets have been defined which initialise the [source parameters](#) and Dirac specific [encoder parameters](#). These presets are passed to the [dirac_encoder_context_init](#) function to initialise a [dirac_encoder_context_t](#) structure. The context is then passed to the [dirac_encoder_init](#) to initialise the encoder. Refer to the Appendix D of the [Dirac Bytestream Definition](#) Document to find out the encoder parameter defaults associated with the presets. The users can override the defaults in the encoder context before initialising the Dirac Encoder if they wish to do so.

```
typedef enum {
    VIDEO_FORMAT_CUSTOM=0,
```

```

VIDEO_FORMAT_QSIF525,
VIDEO_FORMAT_QCIF,
VIDEO_FORMAT_SIF525,
VIDEO_FORMAT_CIF,
VIDEO_FORMAT_4SIF525,
VIDEO_FORMAT_4CIF,
VIDEO_FORMAT_SD_480I60,
VIDEO_FORMAT_SD_576I50,
VIDEO_FORMAT_720P60,
VIDEO_FORMAT_720P50,
VIDEO_FORMAT_1080I60,
VIDEO_FORMAT_1080I50,
VIDEO_FORMAT_1080P60,
VIDEO_FORMAT_1080P50,
VIDEO_FORMAT_DIGI_CINEMA_2K24,
VIDEO_FORMAT_DIGICINEMA_4K24,
VIDEO_FORMAT_UHDTV_4K60,
VIDEO_FORMAT_UHDTV_4K50,
VIDEO_FORMAT_UHDTV_8K60,
VIDEO_FORMAT_UHDTV_8K50,
VIDEO_FORMAT_UNDEFINED
} VideoFormat;
typedef VideoFormat dirac_encoder_presets_t

```

Enumerated type values

VIDEO_FORMAT_CUSTOM

Initialise using custom values specified by the user

VIDEO_FORMAT_QSIF525

Initialise using default values for the Quarter Size Intermediate Format

VIDEO_FORMAT_QCIF

Initialise using default values for the Quarter Common Intermediate Format

VIDEO_FORMAT_SIF525

Initialise using default values for the Standard Intermediate Format

VIDEO_FORMAT_CIF

Initialise using default values for the Common Intermediate Format

VIDEO_FORMAT_4SIF525

Initialise using default values for the Common Intermediate Format

VIDEO_FORMAT_4CIF

Initialise using default values for the PAL Format

VIDEO_FORMAT_SD_576I50

Initialise using default values for the Digital PAL format

VIDEO_FORMAT_SD_480I60

Initialise using default values for the Digital NTSC format

VIDEO_FORMAT_HD_720P60

Initialise using default values for the HD720 Progressive @ 60Hz format

VIDEO_FORMAT_HD_720P50

Initialise using default values for the HD720 Progressive @ 50Hz format

VIDEO_FORMAT_HD_1080I60

Initialise using default values for the HD1080 Interlaced @ 60Hz format

VIDEO_FORMAT_HD_1080I50

Initialise using default values for the HD1080 Interlaced @ 50Hz format

VIDEO_FORMAT_HD_1080P60

Initialise using default values for the HD1080 Progressive @ 60Hz format

VIDEO_FORMAT_HD_1080P50

Initialise using default values for the HD1080 Progressive @ 50Hz format

VIDEO_FORMAT_DIGI_CINEMA_2K24

Initialise using default values for the Digital Cinema 2K Format

VIDEO_FORMAT_DIGI_CINEMA_4K24

Initialise using default values for the Digital Cinema 4K Format

VIDEO_FORMAT_UHDTV_4K60

Initialise using default values for the Ultra High Definition 4K @ 60Hz Format

VIDEO_FORMAT_UHDTV_4K50

Initialise using default values for the Ultra High Definition 4K @ 50Hz Format

VIDEO_FORMAT_UHDTV_8K60

Initialise using default values for the Ultra High Definition 8K @ 60Hz Format

VIDEO_FORMAT_UHDTV_8K50

Initialise using default values for the Ultra High Definition 8K @ 50Hz Format

[top](#)

dirac_mvprecision_t

Declared in `libdirac_encoder/common_types.h`

This enumerated type defines the different motion vector precision values used in the Dirac software.

```
typedef enum {
    MV_PRECISION_PIXEL=0,
    MV_PRECISION_HALF_PIXEL,
    MV_PRECISION_QUARTER_PIXEL,
    MV_PRECISION_EIGHTH_PIXEL,
    MV_PRECISION_UNDEFINED
} MVPrecisionType;
typedef MVPrecisionType dirac_mvprecision_t;
```

Enumerated type values

MV_PRECISION_PIXEL

Pixel accurate motion vector precision

MV_PRECISION_HALF_PIXEL

Half pixel accurate motion vector precision

MV_PRECISION_QUARTER_PIXEL

Quarter pixel accurate motion vector precision

MV_PRECISION_EIGHTH_PIXEL

Eighth pixel accurate motion vector precision

MV_PRECISION_UNDEFINED

Motion vector precision not supported by Dirac software

[top](#)

dirac_encparams_t

Declared in libdirac_encoder/dirac_encoder.h

This structure defines the parameters specific to the Dirac encoder that are needed to encode a sequence.

```
typedef struct
{
    int lossless;
    float qf;
    int full_search;
    int combined_me;
    int x_range_me;
    int y_range_me;
    int L1_sep;
    int num_L1;
    float cpd;
    int xblen;
    int yblen;
    int xbsep;
    int ybsep;
    int video_format;
    dirac_wlt_filter_t intra_wlt_filter;
    dirac_wlt_filter_t inter_wlt_filter;
    unsigned int wlt_depth;
    unsigned int spatial_partition;
    dirac_prefilter_t prefilter;
    unsigned int prefilter_strength;
    unsigned int multi_quants;
    dirac_mvprecision_t mv_precision;
    int trate;
    unsigned int picture_coding_mode;
    int using_ac;
} dirac_encparams_t;
```

Structure Members

lossless

Lossless coding. If set to a non-zero value, the encoder performs lossless encoding. Overrides *qf*.

qf

Quality factor. This is a number from 0 to 10. The higher the number the better the quality. The encoder attempts to adapt the encoding process to produce constant quality across the sequence.

full_search

Use full search motion estimation. If set to a non-zero value, the encoder performs full search motion estimation at pixel level.

combined_me

Use a combination of all three components to do motion estimation. If set to a non-zero value, the encoder will use all three (Y, U and V) components to do motion estimation. By default it uses only the Y component to do motion estimation.

x_range_me

x-range for full-search motion estimation.

y_range_me

y-range for full-search motion estimation.

L1_sep

The separation between L1 pictures, i.e. the inter pictures that can be used as reference pictures.

num_L1

dirac_encparams_t

The number of L1 pictures before the next Intra picture (I_picture). Together with L1_sep it determines the GOP (group of pictures) size. Set this value to 0 to enable I_picture only encoding, i.e. all pictures are intra coded.

cpd Normalised view distance parameter in cycles per degree

xblen The width of the blocks used for motion compensation.

yblen The height of the blocks used for motion compensation.

xbsep The horizontal separation between blocks used for motion compensation. Must be less than xblen.

ybsep The vertical separation between blocks used for motion compensation. Must be less than yblen.

intra_wlt_filter Wavelet filter used for encoding Intra pictures. Default value is DD13_7. This field is of type dirac_wlt_filter_t.

inter_wlt_filter Wavelet filter used for encoding Inter pictures. Default value is DD13_7. This field is of type dirac_wlt_filter_t.

wlt_depth Wavelet transform depth for encoding. Default value is 4.

spatial_partition Spatial partitioning flag. If set to a non-zero value, the subband is divided into coeff blocks before entropy coding is performed.

prefilter Prefilter type. Type of prefiltering to apply to input picture before encoding.

prefilter_strength Prefilter strength. Valid range is 0-10.

multi_quants Multiple quantisers flag. If set to a non-zero value, and spatial partitioning is enabled, each coefficient block within the subband can potentially be assigned a different quantiser in entropy coding.

mv_precision Motion vector precision. This field is of type dirac_mvprecision_t. The default values is MV_PRECISION_QUARTE_PIXEL.

trate Target bit-rate in kbps. The default value is 0. If set to a non-zero value, the encoder will attempt to maintain the bit-rate constant at this value. The resulting bit-rate may be slightly different from this target

picture_coding_mode Picture coding mode. Determines how the input picture is coded. Supported range is 0-1. The default value is 0. If this field is 0 input video is coded as frames. If it is set to 1 the input video is coded as fields. Future versions of the software might support other modes.

using_ac Entropy coding flag. Default value is true. If set to true Arithmetic coding is used for entropy coding of coefficients. If set to false VLC is used.

[▲top](#)

dirac_encoder_context_t

Declared in libdirac_encoder/dirac_encoder.h

This structure defines the encoding context of the Dirac Encoder. The user should set up this structure and pass it to the [dirac_encoder_init](#) function to initialise the encoder. Since the encoder requires numerous parameters to be set, the Encoder API provides a utility function [dirac_encoder_context_init](#) to initialise this structure using [presets](#). The user can always override the defaults manually.

```
typedef struct
{
    dirac_sourceparams_t src_params;
    dirac_encparams_t enc_params;
    int instr_flag;
    int decode_flag;
} dirac_encoder_context_t;
```

Structure Members

src_params

This field is used to initialise the parameters common to all frames in the sequence to be encoded. It is of type [dirac_sourceparams_t](#). Some source parameters, like picture dimensions, sampling format, frame rate are used by the encoder to encode the sequence, The remaining source parameters are not used directly by the encoder. They are metadata used to describe the source material and are written as is to the encoded bytestream.

enc_params

This field is used to initialise the parameters, specific to the Dirac encoder, common to all frames in the sequence to be encoded. It is of type [dirac_encparams_t](#).

instr_flag

If set, the Encoder returns instrumentation data relating to motion estimation and mode decisions taken by the Encoder. This data can be used to visualise the inner workings of the encoder.

decode_flag

If set, the Encoder returns the locally decoded version of the encoded frame. This data can then be displayed to view the encoder performance subjectively.

[▲top](#)

dirac_enc_data_t

Declared in libdirac_encoder/dirac_encoder.h

This structure holds the encoded picture data. It must be initialised and managed by the user.

```
typedef struct
{
    unsigned char *buf;
    int size;
} dirac_enc_data_t;
```

Structure members

buf

A buffer to hold the encoded picture. It must be initialised and by the user before every call to the encoding functions. Upon successful completion of encoding a picture, the encoder copies the encoded output to this buffer.

size

The user should initialise this field to the size of the buffer, in bytes, used to receive the encoded data. If the encoded picture size is greater than the value of this field, the encoding functions will return an error. Upon successful encoding of a picture, the encoder sets this field to the actual size, in bytes, of the encoded picture.

[▲top](#)

dirac_enc_picstats_t

Declared in libdirac_encoder/dirac_encoder.h

This structure holds data about the picture encoded that might be of interest to the user, e.g. size of encoded picture, etc. The encoder manages this field. This field is set up by the encoder after the successfully encoding a picture.

```
typedef struct
{
    unsigned int mv_bits;
    unsigned int ycomp_bits;
    unsigned int ucomp_bits;
    unsigned int vcomp_bits;
    unsigned int pic_bits;
} dirac_enc_picstats_t;
```

Structure members

mv_bits

Number of bits used to encode motion estimation information. This field is set to 0 for intra coded pictures.

ycomp_bits

Number of bits used to encode the luma (Y) component of the input picture.

ucomp_bits

Number of bits used to encode the U chroma component of the input picture.

vcomp_bits

Number of bits used to encode the V chroma component of the input picture.

pic_bits

Number of bits used to encode the the entire picture.

[▲top](#)

dirac_enc_seqstats_t

Declared in libdirac_encoder/dirac_encoder.h

This structure holds data about the sequence encoded that might be of interest to the user, e.g. size of encoded sequence, bit rate of the encoded sequence, etc. The encoder manages this field. This field is set up by the encoder after the successfully encoding an entire sequence of frames.

Structure members

```
typedef struct
{
    int64_t mv_bits;
    int64_t seq_bits;
    int64_t ycomp_bits;
    int64_t ucomp_bits;
    int64_t vcomp_bits;
    int64_t bit_rate;
} dirac_enc_picstats_t;
```

Structure members

mv_bits

Number of bits used to encode motion estimation information for the entire sequence. This field is set to 0 if INTRA_PICTURE only encoding is used for the sequence.

seq_bits

Number of bits used to encode the entire sequence

ycomp_bits

Number of bits used to encode the Luma (Y) component of all the frames in the input sequence.

ucomp_bits

Number of bits used to encode the U chroma component of all the frames in the input sequence. This field is set to 0 for gray-scale (Yonly) inputs.

vcomp_bits

Number of bits used to encode the V chroma component of all the frames in the input sequence. This field is set to 0 for gray-scale (Yonly) inputs.

bit_rate

The number of bits per second required to transmit/store the sequence.

[top](#)

dirac_mv_t

Declared in libdirac_encoder/dirac_encoder.h

This structure defines the motion vectors for a block of data.

```
typedef struct
{
    int x;
    int y;
} dirac_mv_t;
```

Structure members

x

X component of the motion vector.

y

Y component of the motion vector.

[top](#)

dirac_mv_cost_t

Declared in libdirac_encoder/dirac_encoder.h

Costs associated with motion vectors. The Encoder uses this information to determine if the block of data is intra-coded or inter-coded.

```
typedef struct
{
    float SAD;
    float mvcost;
} dirac_mv_cost_t;
```

Structure members

SAD

Sum of absolute differences.

mvcost

The Lagrangian weighted motion vector cost.

[top](#)

dirac_instr_t

Declared in libdirac_encoder/dirac_encoder.h

This structure holds the instrumentation data returned by the encoder. The field is managed by the encoder. The encoder copies the motion estimation and mode decision information into this field after successfully encoding a picture of data. This information can be used by the end user to visualise the decisions taken by the encoder while encoding.

```
typedef struct
{
    dirac_picture_type_t ptype;
    dirac_reference_type_t rtype;
    int pnum;
    int num_refs;
    int refs[2];
    int xbsep;
    int ybsep;
    int sb_xlen;
    int sb_ylen;
    int mv_xlen;
    int mv_ylen;
    int *sb_split_mode;
    float *sb_costs;
    int *pred_mode;
    float *intra_costs;
    dirac_mv_cost_t *bipred_costs;
    short *dc_ycomp;
    short *dc_ucomp;
    short *dc_vcomp;
    dirac_mv_t *mv[2];
    dirac_mv_cost_t *pred_costs[2];
} dirac_instr_t;
```

Structure members

ptype	Encoded picture type i.e intra coded (INTRA_PICTURE) or inter coded (INTER_PICTURE). This field is of type <u>dirac_picture_type_t</u> .
rtype	Specifies if the picture is used as a reference for predicted pictures or not. This field is of type <u>dirac_reference_type_t</u> .
pnum	Picture number of the picture in display order. Since the coded order can be different from the display order, this number does not reflect the temporal position of the picture in the coded bitstream, but the temporal display order.
num_refs	Number of previously encoded pictures used to encoded this picture. It ranges from 0 to 2. This field is set to 0 for intra coded pictures.
refs	Array holding the picture numbers (in display order) of the reference pictures used to encode the current picture. This field is not set for intra coded pictures.
xbsep	Horizontal separation between blocks. This field is not set for intra coded pictures.
ybsep	Vertical separation between blocks. This field is not set for intra coded pictures.
sb_xblen	Width of the macro-block. This field is not set for intra coded pictures.
sb_yblen	Height of the macro-block. This field is not set for intra coded pictures.
mv_xblen	Motion vector array length in x direction. This field is not set for intra coded pictures.
mv_yblen	Motion vector array length in y direction. This field is not set for intra coded pictures.
sb_split_mode	Array holding macro-block split level decisions. The size of the array is sb_xlen*sb_ylen. This field is not set for intra coded pictures.
sb_costs	Array holding macro-block cost parameters. The size of the array is sb_xlen*sb_ylen. This field is not set for intra coded pictures.
pred_mode	Array holding block prediction mode. The size of the array is mv_xlen*mv_ylen. This field is not set for intra coded pictures.
intra_costs	Array holding the costs for predicting each block in intra mode. The size of the array is mv_xlen*mv_ylen. This field is not set for intra coded pictures.
bipred_costs	Array holding the costs for predicting each block bi-directionally. Each element of the array is of type <u>dirac_mv_cost_t</u> . The size of the array is mv_xlen*mv_ylen. This field is not set for intra coded pictures.
dc_ycomp	Array holding DC value for the Luma (Y) component. The size of the array is mv_xlen*mv_ylen. This field is not set for intra coded pictures.
dc_ucomp	Array holding DC value for the U Chroma Component. The size of the array is mv_xlen*mv_ylen. This field is not set for intra coded pictures.

dc_vcomp

Array holding DC value for the V Chroma component. The size of the array is mv_xlen*mv_ylen. This field is not set for intra coded pictures.

mv

Array holding motion vectors arrays of the current picture for each reference picture(s) from which it was predicted. Each element in this array is an array of elements of type [dirac_mv_t](#) and the size of each motion vector array is mv_xlen*mv_ylen. This field is not set for intra coded pictures.

pred_costs

Array holding prediction cost arrays of the current picture for each reference picture(s) from which it was predicted. Each element in this array is an array of elements of type [dirac_mv_cost_t](#) and the size of each prediction cost array is mv_xlen*mv_ylen. This field is not set for intra coded pictures.

[top](#)

dirac_encoder_t

Declared in libdirac_encoder/dirac_encoder.h

This structure defines encoder handle. It is used by all Dirac encoder functions. Before it can be used, it must be initialised by a call to [dirac_encoder_init](#). After use, the encoder handle must be freed with a call to [dirac_encoder_close](#).

```
typedef struct
{
    dirac_encoder_context_t enc_ctx;
    int encoded_picture_avail;
    dirac_enc_data_t enc_buf;
    dirac_picparams_t enc_pparams;
    dirac_enc_picstats_t enc_pstats;
    dirac_enc_seqstats_t enc_seqstats;
    int end_of_sequence;
    int decoded_frame_avail;
    dirac_framebuf_t dec_buf;
    dirac_picparams_t dec_pparams;
    dirac_instr_t instr;
    int instr_data_avail;
    const void *compressor;
} dirac_encoder_t;
```

Structure members

enc_ctx

Local copy of the [dirac_encoder_context_t](#) passed to the encoder initialisation function [dirac_encoder_init](#).

encoded_picture_avail

This flag is set when an encoded picture is available in the encoder buffer enc_buf.

enc_buf

Buffer to hold an encoded picture. This field is of type [dirac_enc_data_t](#). This buffer must be initialised by the user before calling the encode function [dirac_encoder_output](#).

enc_pparams

Parameters of the picture just encoded. The encoder sets this field after successfully encoding a picture. This field is of type [dirac_picparams_t](#). The end user must check to see if the encoded_picture_avail flag is set before using this field.

enc_pstats

dirac_encoder_t

Information related to the picture just encoded . The encoder manages this field. This field is of type [dirac_enc_picstats_t](#). The encoder sets this field after successfully encoding a picture.

enc_seqstats

Information related to the video sequence just encoded . The encoder manages this field. This field is of type [dirac_enc_seqstats_t](#). This field is set up by the encoder after encoding an entire video sequence.

end_of_sequence

This field is set by the encoder after encoding an entire video sequence.

decoded_frame_avail

This flag is set by the encoder if a locally decoded frame, in display order, is available in the buffer dec_buf.

dec_buf

Buffer to hold the locally decoded frame. This field is of type [dirac_framebuf_t](#). It is managed by the encoder. The end user must check to see if the decoded_frame_avail flag is set before using this field.

dec_pparams

Parameters of the locally decoded frame. This field is of type [dirac_picparams_t](#). It is set by the encoder. The end user must check to see if the decoded_frame_avail flag is set before using this field.

instr_data_avail

This flag is set by the encoder if instrumentation data is available in the buffer instr.

instr

Buffer to hold the instrumentation data associated with the picture just encoded. This field is of type [dirac_instr_t](#). It is managed by the encoder. The end user must check to see if the instr_data_avail flag is set before using this field.

compressor

Internal encoder field. The user must not use/modify this field.

[▲top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Back to Encoder API](#)

Encoder API Functions

The following functions are part of the Encoder API

dirac_encoder_context_init

Declared in `libdirac_encoder/dirac_encoder.h`

Encoding is a complex operation. The encoder needs to be initialised with several parameters to make encoding decisions. To make things simpler for the user, the encoder library defines a set of presets which initialise the Encoder parameters to suitable default values. The user can then override the preset defaults if necessary. This function sets up the default values in an `dirac_encoder_context_t` structure passed to it with the default values for the preset specified by the user. The user can then adjust the values in the context structure, if necessary, before using it to initialise the Encoder. This function initialises the `src_params` and `enc_params` fields of the context passed to it. The mapping of presets to values for each preset defined in `dirac_encoder_presets_t` can be found in Appendix E of the [Dirac Bytestream Specification](#).

```
void dirac_encoder_context_init (dirac_encoder_context_t *enc_ctx,  
                               dirac_encoder_presets_t preset);
```

Parameters

enc_ctx

A pointer to the `dirac_encoder_context_t` to be initialised.

preset

Preset value to use to initialise the encoder context. Supported preset values are defined in `dirac_encoder_presets_t`.

Return Values

None.

[top](#)

dirac_encoder_init

Declared in `libdirac_encoder/dirac_encoder.h`

This function initialises the Dirac Encoder. It must be called before the encoding or teardown functions are used. It returns a `dirac_encoder_t` structure that must be used in subsequent calls to other Encoder functions. The encoder makes a local copy of the context passed to it. So any changes to the encoder context after calling this function will not affect the the Encoder.

```
dirac_encoder_t *dirac_encoder_init (dirac_encoder_context_t *enc_ctx,  
                                     int verbose);
```

Parameters

enc_ctx

Pointer to dirac_encoder_context_t structure that has source and encoder parameters set up to initialise the Encoder.

verbose

If set, the Encoder will write verbose debug information to standard error. It is useful to set this flag to during the development phase for debugging purposes.

Return Values

A pointer to a dirac_encoder_t structure if successful.
NULL on failure

 [top](#)

dirac_encoder_pts_offset

Declared in libdirac_encoder/dirac_encoder.h

This function queries the encoder for the reordering depth. This function is available only from version 1.0.2 of the Dirac API. The end user application can use the DIRAC_API_VERSION macros to check the API version.

```
int dirac_encoder_pts_offset (dirac_encoder_t *encoder);
```

Parameters

encoder

The encoder handle, i.e. the pointer to the dirac_encoder_t returned by dirac_encoder_init.

Return Values

>=0

The number of pictures a realtime decoder must wait before outputting the first picture in display order.

-1

Function failed.

 [top](#)

dirac_encoder_load

Declared in libdirac_encoder/dirac_encoder.h

This function loads one frame of uncompressed data into the Encoder.

```
int dirac_encoder_load (dirac_encoder_t *encoder, unsigned char *uncdata,
                       int uncdsize);
```

Parameters

encoder

Encoder handle, i.e. the pointer to the [dirac_encoder_t](#) returned by [dirac_encoder_init](#).

uncdata

Buffer containing one full frame of uncompressed input.

uncdata_size

Size of the uncompressed data buffer, i.e. the size of a frame. The user should ensure that this value is set correctly according to the frame dimensions and chroma format specified in the source parameters of the context the encoder was initialised with. Otherwise, the encoder returns -1 to indicate the load operation failed.

Return Values

>=0 Success.

-1 Failure.

[top](#)

dirac_encoder_output

Declared in libdirac_encoder/dirac_encoder.h

This function encodes one frame of uncompressed data. The [state](#) of the encoder after the encode operation is returned. An encoded output frame might not be available after each encode operation. e.g. when not encoding in I_frame only mode. So the user should monitor the return value to determine if encoded data is available. Before calling this function, the user should set up the [enc_buf](#) field in the encoder handle passed to the function, so that encoded output can be written to the buffer.

In addition to encoded frames, this function also retrieves locally decoded frames and instrumentation data into buffers managed by the encoder. Locally decoded frames are retrieved only if the [decode_flag](#) is set in the [encoder context](#) when initialising the encoder. Similarly the instrumentation data is retrieved only if the [instr_flag](#) is set in the encoder context when initialising the encoder. The encoder sets the [decoded_frame_avail](#) flag and [instr_data_avail](#) flag in the encoder handle if locally decoded frames and instrumentation data are available. The user must confirm that these flags are set before attempting to use the [dec_buf](#) and [instr](#) buffers in the [encoder handle](#).

```
dirac_encoder_state_t dirac_encoder_output (dirac_encoder_t *encoder);
```

Parameters

encoder

Encoder handle, i.e. the pointer to the `dirac_encoder_t` returned by `dirac_encoder_init`. The `enc_buf` field in this structure must be set up so that the encoded frame can be copied into it.

E.g.

```
unsigned char video_buffer[1024*1024];
encoder->enc_buf.buffer = video_buffer;
encoder->enc_buf.size = 1024*1024;
state = dirac_encoder_output(encoder);
```

The encode operation will fail with a return value of `STATE_INVALID` if the buffer to hold the encoded data is not large enough for the encoded frame.

Return Values

ENC_STATE_BUFFER

The encoder needs more input to encode a frame of data. The user should load uncompressed data into the encoder using the `dirac_encoder_load` function.

ENC_STATE_AVAIL

Encoded picture available in `enc_buf` in the encoder handle. The encoded picture statistics are available in `enc_pstats` and picture parameters are available in `enc_pparams` fields of the encoder handle.

ENC_STATE_EOS

End of sequence information available in `enc_buf` in the encoder handle. The sequence statistics are now available in the `enc_seqstats` field of the encoder handle.

ENC_STATE_INVALID

The encoder encountered an irrecoverable error. Stop all further processing.

[▲top](#)

dirac_encoder_end_sequence

Declared in `libdirac_encoder/dirac_encoder.h`

This function signals the encoder to end the sequence.

```
void dirac_encoder_end_sequence (dirac_encoder_t *encoder);
```

Parameters

encoder

The encoder handle, i.e. the pointer to the `dirac_encoder_t` returned by `dirac_encoder_init`.

Return Values

None.

[▲top](#)

dirac_encoder_close

Declared in libdirac_encoder/dirac_encoder.h

This function frees the resources held by the encoder. The encoder handle passed to this function is no longer valid after a call to this function.

```
void dirac_encoder_close (dirac_encoder_t *encoder);
```

Parameters

encoder

The encoder handle, i.e. the pointer to the dirac_encoder_t returned by dirac_encoder_init.

Return Values

None.

[▲top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Encoder API Contents](#)

Decoder API

Decoder API contents:

- [API Overview](#)
- [API Reference](#)
- [Code Examples](#)

[top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Back to Decoder API Contents](#) | [API Reference](#)

Decoder API Overview

Though Dirac is implemented in C++, the public API of the decode engine is implemented in "C". The idea behind this is to have a stable public interface even if the underlying C++ implementation changes radically.

The Decoder API can be divided into the following categories:

- Base Data Structures
- Setup/Teardown functions
- Decoding

Base Data Structures

There are several structures that hold information required by the decoder during the decoding process. These structures are defined in `libdirac_common/dirac_types.h`, `libdirac_decoder/decoder_types.h` and `libdirac_decoder/dirac_decoder.h`.

<code>dirac_decoder_state_t</code>	State of the decoder
<code>dirac_parseparams_t</code>	Parameters describing the version of the Encoding software used to create the input Dirac bitstream.
<code>dirac_sourceparams_t</code>	Parameters that help the end user application correctly display the output of the decoder.
<code>dirac_framebuf_t</code>	Uncompressed/Decoded frame
<code>dirac_decoder_t</code>	Decoder handle

[top](#)

Setup/Teardown Functions

<code>dirac_decoder_init</code>	Initialises the Dirac decoder. This function must be called before the decoding functions are used. It returns a <code>dirac_decoder_t</code> structure that should be used in the subsequent calls to decoding and teardown functions.
<code>dirac_decoder_close</code>	Frees up the Dirac decoder resources. It must be called after the decoding process. The <code>dirac_decoder_t</code> structure returned by <code>dirac_decoder_init</code> is passed as a parameter to this function. After this function returns, the <code>dirac_decoder_t</code> structure is no longer valid and cannot be used for other decoding/teardown operations.

[top](#)

Decoding functions

<u>dirac_buffer</u>	This function loads a chunk of the bitstream passed to it into the Dirac decoder. It accepts a decoder handle and start and end addresses of the bitstream chunk in memory as inputs.
<u>dirac_parse</u>	Parses the bitstream loaded into the decoder. It accepts a decoder handle as input. It returns a <code>dirac_decoder_state_t</code> value that reflects the state the decoder is currently in.
<u>dirac_set_buf</u>	Set the buffer that the decoder will write the decoded frame to. It accepts a decoder handle and an array of YUV component buffers as input.

[!\[\]\(9b800325684b184be8e88ceef387e61b_img.jpg\)top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Back to Decoder API Contents](#) | [API Overview](#) | [Code Example](#)

Base Data Structures

[dirac_decoder_state_t](#)

[dirac_parseparams_t](#)

[dirac_sourceparams_t](#)

[dirac_framebuf_t](#)

[dirac_decoder_t](#)

Setup/Teardown Functions

[dirac_decoder_init](#)

[dirac_decoder_close](#)

Decoding functions

[dirac_buffer](#)

[dirac_parse](#)

[dirac_set_buf](#)

[top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Back to Decoder API Contents](#) | [API Reference](#)

Decoder API Example

The following code describes a simple example on how to use the Dirac Decoder API. For a more detailed usage refer to the sample decode application, `decoder/decmain.cpp`, in the Dirac distribution. The following sample reads a Dirac bitstream from an input file, and writes the decoded output to an output file.

```
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

/*
 * Include the Dirac decoder file
 */
#include <libdirac_decoder/dirac_parser.h>

extern int main (int argc, char **argv)
{
    FILE *in;
    FILE *out;

    int bytes_read;

    /* buffer to store data read from bitstream */
    unsigned char bits_buf[4096];

    /* Buffer to hold decoded output */
    unsigned char *buf[3];

    dirac_decoder_t *decoder; /* Decoder handle */
    dirac_decoder_state_t state; /* Decoder State */

    if (argc < 3)
    {
        printf ("Usage : %s input-file output-file\n", argv[0]);
        exit(0);
    }

    /* open input file */
    if ( (in = fopen (argv[1], "rb")) == NULL)
    {
        perror (argv[1]);
        exit(errno);
    }

    /* open output file */
    if ( (out = fopen (argv[2], "wb")) == NULL)
    {
        perror (argv[2]);
        exit(errno);
    }

    /* Initialise the Dirac decoder */
    if ( (decoder = dirac_decoder_init(0)) == NULL)
    {
        printf ("Error initialising the Dirac decoder\n");
        exit(-1);
    }
}
```

```

/* Main loop*/
do
{
    /* Call the parse function */
    state = dirac_parse(decoder);

    /*
    * Take appropriate action depending on state returned by parse function
    */
    switch (state)
    {
    case STATE_BUFFER:
        /*
        * Decoder needs more data to continue. Read more data from
        * input file
        */
        bytes_read = fread (bits_buf, 1, sizeof(bits_buf), in);
        if (bytes_read)
        {
            /* load data into decoder */
            dirac_buffer (decoder, bits_buf, bits_buf + bytes_read);
        }
        break;
    case STATE_SEQUENCE:
        /*
        * Decoder has detected and parsed a sequence header. The sequence
        * decoder->src_params and the parse parameters are available in decoder->parse_params
        * Allocate for frame buffers
        */

        /* Y Component */
        buf[0] = (unsigned char *)
            malloc(decoder->src_params.width * decoder->src_params.height);

        /* U Component */
        buf[1] = (unsigned char *)
            malloc(decoder->src_params.chroma_width * decoder->src_params.chroma_height);

        /* V Component */
        buf[2] = (unsigned char *)
            malloc(decoder->src_params.chroma_width * decoder->src_params.chroma_height);

        /* Set the decode frame buffer in Decoder handle */
        dirac_set_buf (decoder, buf, NULL);
        break;
    case STATE_PICTURE_AVAIL:
        /*
        * Next frame in display order is available in the decoder buffer
        * decoder->fbuf. Frame number of the frame decoded is available in
        * decoder->frame_num.
        */

        printf ("Frame %d Available\n", decoder->frame_num);
        /* Write Y Component to output file */
        fwrite (decoder->fbuf->buf[0],
            decoder->src_params.width*decoder->src_params.height, 1, out);

        /* Write U Component */
        fwrite (decoder->fbuf->buf[1],
            decoder->src_params.chroma_width*decoder->src_params.chroma_height, 1, out);
    }
}

```

```

    /* Write V Component */
    fwrite (decoder->fbuf->buf[2],
            decoder->src_params.chroma_width*decoder->src_params.chroma_height, 1, out);

    break;

case STATE_SEQUENCE_END:
    /*
     * Decoder detected end of sequence. Free resources allocated
     * for this sequence
     */
    free (buf[0]);
    free (buf[1]);
    free (buf[2]);
    break;

case STATE_INVALID:
    /* Invalid state. Stop all processing */
    printf ("Error processing file %s\n", argv[1]);
    break;

default:
    continue;
}
} while (bytes_read > 0 && state != STATE_INVALID);

/* Free the encoder resources */
dirac_decoder_close(decoder);

/* close files */
fclose (in);
fclose (out);
}

```

[top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Decoder API Overview](#)

Decoder Base Data Structures

The following structures are used in the Decoder API

dirac_decoder_state_t

Declared in libdirac_decoder/decoder_types.h

This enumerated type defines the state the Dirac Decoder is currently in.

```
typedef enum {
    STATE_BUFFER,
    STATE_SEQUENCE,
    STATE_PICTURE_AVAIL,
    STATE_SEQUENCE_END,
    STATE_INVALID,
} DecoderState;
typedef DecoderState dirac_decoder_state_t;
```

Enumerated type values

STATE_BUFFER

The Dirac decoder has insufficient data to continue processing any further. The user needs to provide data to be decoded if the decoder is in this state

STATE_SEQUENCE

The decoder detected a start of sequence in the input bitstream and has processed the source parameters in the bitstream.

STATE_PICTURE_AVAIL

The decoder has successfully decoded a frame and it is available to the end user

STATE_SEQUENCE_END

The decoder detected the end of the current sequence of encoded frames in the bitstream.

STATE_INVALID

The decoder has come across an irrecoverable error and is in an invalid state. Stop all further processing

[top](#)

dirac_decoder_t

Declared in libdirac_decoder/dirac_parser.h

The `dirac_decoder_t` structure defines a handle to the Dirac decoder. It is used by all the Dirac decoder routines. Before it can be used, it must be initialised by a call to [dirac_decoder_init](#). After use, the `dirac_decoder_t` structure must be freed with a call to [dirac_decoder_close](#).

```
typedef struct
```

```

{
    dirac_decoder_state_t state;
    dirac_parseparams_t parse_params,
    dirac_sourceparams_t src_params;
    unsigned int frame_num;
    void *parser;
    dirac_framebuf_t *fbuf;
    int frame_avail;
    int verbose;
} dirac_decoder_t;

```

Structure Members

state

The state the decoder is currently in. It is of type [dirac_decoder_state_t](#). The user should interpret the other fields in the decoder handle based on the value of this field.

parse_params

Parse parameters of the sequence currently being decoded in the Dirac bitstream. This structure contains information like the version number of the Encoder Software used to create the bitstream, the profile and level used. It is of type [dirac_parseparams_t](#). The decoder application uses the fields in this structure to determine if it is capable of decoding the bistream.

src_params

Parameters of the sequence currently being decoded in the Dirac bitstream. It is of type [dirac_sourceparams_t](#). This field must be used by the user only after the decoder returns a state of STATE_SEQUENCE.

frame_num

Frame number of the decoded frame available . This field is populated with the frame number of the decoded frame available to the user if the decoder returns a STATE_PICTURE_AVAIL state.

parser

Internal to the Dirac decoder. The user must not use/modify this field.

fbuf

Buffer holding the luma (Y) and chroma (U & V) components of the frame decoded in display order. It is of type [dirac_framebuf_t](#). This field must be used by the user only after the decoder returns a state of STATE_PICTURE_AVAIL.

frame_avail

This field is set by the decoder when a decoded frame in display order is available to the user in fbuf .

verbose

This field, if set, causes the decoder to display debugging information to standard error.

[▲top](#)



Dirac video codec: A programmer's guide

[Table of Contents](#) | [Decoder API Overview](#)

Decoder API Functions

The following functions are part of the Decoder API

dirac_decoder_init

Declared in libdirac_decoder/dirac_parser.h

This function initialises the Dirac decoder. It must be called before the decoding or teardown functions are used. It returns a dirac_decoder_t structure that must be used in subsequent calls to other Decoder functions.

```
dirac_decoder_t *dirac_decoder_init (int verbose);
```

Parameters

verbose

If set, the Decoder will write verbose debug information to standard. It is useful to set this flag to during the development phase for debugging purposes.

Return Values

A pointer to a dirac_decoder_t structure if successful.
NULL on failure

[top](#)

dirac_buffer

Declared in libdirac_decoder/dirac_parser.h

This function loads a chunk of the input bitstream into the Dirac decoder.

```
void dirac_buffer (dirac_decoder_t *decoder, unsigned char *start,  
unsigned char *end);
```

Parameters

decoder

A pointer to the dirac_decoder_t returned by dirac_decoder_init.

start

A pointer to the start of the bitstream buffer chunk.

end

A pointer to the end of the bitstream buffer chunk.

Return Values

None.

[top](#)

dirac_parse

Declared in libdirac_decoder/dirac_parser.h

This function parses the data loaded into it by [dirac_buffer](#) and returns the state of the decoder. The user must monitor the return value of this function and take action depending on it.

```
dirac_decoder_state_t dirac_parse (dirac_decoder_t *decoder);
```

Parameters

decoder

A pointer to the [dirac_decoder_t](#) returned by [dirac_decoder_init](#).

Return Values

STATE_BUFFER

Needs more data. The user should load more data into the decoder using the [dirac_buffer](#) function.

STATE_SEQUENCE

The decoder has detected and decoded a sequence header. The decoder sets up the [src_params](#) structure in [decoder_handle](#) with the information from the sequence header. In response to this state, the user must allocate the buffers to receive decoded frames and call [dirac_set_buf](#) to set up the frame buffer in the decoder handle. On successfully decoding a frame, the decoder will write the decoded frame to this buffer.

STATE_PICTURE_AVAIL

The decoder has successfully decoded a frame. The decoder sets up the `frame_num` field in [decoder_handle](#) with the frame number of the decoded frame. It copies the decoded frame into the frame buffer, [fbuf](#), in the handle.

STATE_SEQUENCE_END

The decoder has successfully decoded a full video sequence. The user must now free the frame buffers allocated earlier.

STATE_INVALID

The decoder has encountered an irrecoverable error. The user must stop all further processing and call [dirac_decoder_close](#) to free the decoder resources.

[top](#)

dirac_set_buf

Declared in libdirac_decoder/dirac_parser.h

This function sets up the frame buffer in the decoder handle. The decoder will write the decoded frame to this buffer after successfully decoding a frame of data.

```
void dirac_set_buf (dirac_decoder_t *decoder, unsigned char *buf[3],
void *id);
```

Parameters

decoder

A pointer to the [dirac_decoder_t](#) returned by [dirac_decoder_init](#).

buf

Array of 3 buffers for holding the decoded frame data, one for each component (Y, U, V) of the frame. It is the user's responsibility to allocate enough memory for each of the component buffers based on the dimensions of the frame and the chroma format. This information is available in the [src_params](#) field of the handle. For e.g. the user may allocate the component buffers as follows -

```
buf[0] = (unsigned char *)malloc (decoder->src_params.width*decoder->src_params.height);
buf[1] = (unsigned char *)malloc
(decoder->src_params.chroma_width*decoder->src_params.chroma_height);
buf[2] = (unsigned char *)malloc
(decoder->src_params.chroma_width*decoder->src_params.chroma_height);
```

id

User data. The decoder will not use/modify this field in any way. The user can set it to NULL.

Return Values

None.

[▲](#)[top](#)

dirac_decoder_close

Declared in `libdirac_decoder/dirac_parser.h`

This function frees the resources held by the decoder. The decoder handle is no longer valid after a call to this function.

```
void dirac_buffer (dirac_decoder_t *decoder);
```

Parameters

decoder

A pointer to the [dirac_decoder_t](#) returned by [dirac_decoder_init](#).

Return Values

None.

[▲top](#)

