DNG SDK 1.5

Generated by Doxygen 1.8.15

1 Adobe Digital Negative SDK 1.5

1.1 Introduction

Digital Negative (DNG) is a non-proprietary file format for camera raw image data and metadata. A wide variety of cameras and sensor types are supported by DNG, using the same documented file layout.

This SDK provides support for reading and writing DNG files as well as support for converting DNG data into a displayable or processible image. This SDK is intended to serve as a starting point for adding DNG support to existing applications that use and manipulate images.

1.2 Command line validation: dng_validate

A good place to start investigating the DNG SDK is the dng_validate command line tool, which can read, validate and convert an existing DNG file. The dng_validate.cpp file demonstrates a number of common uses of the SDK.

1.3 Starting points

- dng_host Used to customize memory allocation, to communicate progress updates and test for cancellation.
- dng_negative Main container for metadata and image data in a DNG file.
- dng_image Class used to hold and manipualte image data.
- dng_render Class used to convert DNG RAW data to displayable image data.
- dng_image_writer Class used to write DNG files.

1.4 Related documentation

- The Adobe Digital Negative specification: `https://helpx.adobe.com/photoshop/digital-negative.html`
- TIFF 6 specification: `https://www.adobe.io/content/dam/udp/en/open/standards/tiff/TIFF6.pdf`
- TIFF/EP specification: `http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=29377&ICS1=37&ICS2=40&ICS3=99`
- EXIF specification: `http://www.cipa.jp/std/documents/e/DC-008-Translation-2016-E.pdf`
- IPTC specification: `http://www.iptc.org/IPTC7901/`

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

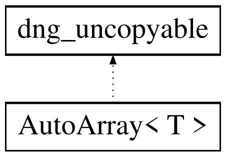
5 Class Documentation

5.1 AutoArray< T > Class Template Reference

A class intended to be used similarly to AutoPtr but for arrays.

```
#include <dng_auto_ptr.h>
```

Inheritance diagram for AutoArray< T >:



Public Member Functions

- AutoArray (T ∗p_=0)
- ∼AutoArray ()

  Reset is called on destruction.
- T ∗ Release ()
- void Reset (T ∗p_=0)
- T & operator[ ] (ptrdiff_t i) const
- T ∗ Get () const

5.1.1 Detailed Description

template<typename T>
class AutoArray< T >

A class intended to be used similarly to AutoPtr but for arrays.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 AutoArray()

```
template<typename T>
AutoArray< T >::AutoArray (
            T * p_ = 0 )  [inline], [explicit]
```

Construct an AutoArray which owns the argument pointer.

Parameters

p↩ _↩	array pointer which constructed AutoArray takes ownership of. p_ will be deleted on destruction or Reset unless Release is called first.

5.1.3 Member Function Documentation

5.1.3.1 Get()

```
template<typename T>
T* AutoArray< T >::Get ( ) const  [inline]
```

Return the owned pointer of this AutoArray, NULL if none. No change in ownership or other effects occur.

5.1.3.2 operator[]()

```
template<typename T>
T& AutoArray< T >::operator[] (
             ptrdiff_t i ) const  [inline]
```

Allows indexing into the AutoArray. It is an error to call this if the AutoArray has NULL as its value.

5.1.3.3 Release()

```
template<typename T>
T* AutoArray< T >::Release ( )  [inline]
```

Return the owned array pointer of this AutoArray, NULL if none. The AutoArray gives up ownership and takes NULL as its value.

5.1.3.4 Reset()

```
template<typename T>
void AutoArray< T >::Reset (
             T * p_ = 0 )  [inline]
```

If an array pointer is owned, it is deleted. Ownership is taken of the passed in pointer p_.

Parameters

p↩ _↩	array pointer which constructed AutoArray takes ownership of. p_ will be deleted on destruction or Reset unless Release is called first.

Referenced by dng_find_new_raw_image_digest_task::Start().

The documentation for this class was generated from the following file:

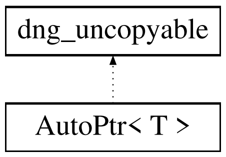
- dng_auto_ptr.h

5.2 AutoPtr< T > Class Template Reference

A class intended to be used in stack scope to hold a pointer from new. The held pointer will be deleted automatically if the scope is left without calling Release on the AutoPtr first.

```
#include <dng_auto_ptr.h>
```

Inheritance diagram for AutoPtr< T >:



Public Member Functions

- AutoPtr ()

  Construct an AutoPtr with no referent.
- AutoPtr (T ∗p)
- ∼AutoPtr ()

  Reset is called on destruction.
- void Alloc ()

  Call Reset with a pointer from new. Uses T's default constructor.
- T ∗ Get () const
- T ∗ Release ()
- void Reset (T ∗p)
- void Reset ()
- T ∗ operator-> () const
- T & operator ∗ () const

Friends

- void Swap (AutoPtr< T > &x, AutoPtr< T > &y)

  Swap with another auto ptr.

5.2.1 Detailed Description

template<class T>
class AutoPtr< T >

A class intended to be used in stack scope to hold a pointer from new. The held pointer will be deleted automatically if the scope is left without calling Release on the AutoPtr first.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 AutoPtr()

```
template<class T>
AutoPtr< T >::AutoPtr (
              T * p )  [inline], [explicit]
```

Construct an AutoPtr which owns the argument pointer.

Parameters

p	pointer which constructed AutoPtr takes ownership of. p will be deleted on destruction or Reset unless Release is called first.

5.2.3 Member Function Documentation

5.2.3.1 Get()

```
template<class T>
T* AutoPtr< T >::Get ( ) const  [inline]
```

Return the owned pointer of this AutoPtr, NULL if none. No change in ownership or other effects occur.

Referenced by dng_opcode_list::Apply(), dng_linearization_info::ColumnBlack(), dng_linearization_info::Column↩ BlackCount(), dng_render::dng_render(), dng_info::IsValidDNG(), dng_linearization_info::MaxBlackLevel(), dng_info↩ ::Parse(), dng_info::PostParse(), dng_opcode_FixVignetteRadial::Prepare(), dng_read_tiles_task::Process(), dng_↩ opcode_MapTable::ProcessArea(), dng_opcode_Unknown::PutData(), dng_render::Render(), dng_linearization_info↩ ::RowBlack(), and dng_linearization_info::RowBlackCount().

5.2.3.2 operator *()

```
template<class T>
T& AutoPtr< T >::operator * ( ) const  [inline]
```

Returns a reference to the object that the owned pointer points to. It is an error to call this if the AutoPtr has NULL as its value.

5.2.3.3 operator->()

```
template<class T>
T* AutoPtr< T >::operator-> ( ) const  [inline]
```

Allows members of the owned pointer to be accessed directly. It is an error to call this if the AutoPtr has NULL as its value.

5.2.3.4 Release()

```
template<class T >
T * AutoPtr< T >::Release ( )
```

Return the owned pointer of this AutoPtr, NULL if none. The AutoPtr gives up ownership and takes NULL as its value.

Referenced by dng_filter_opcode::Apply(), dng_opcode_WarpRectilinear::Apply(), dng_opcode_WarpFisheye::Apply(), dng_gain_map::GetStream(), dng_hue_sat_map::Interpolate(), and dng_render::Render().

5.2.3.5 Reset() [1/2]

```
template<class T>
void AutoPtr< T >::Reset (
             T * p )
```

If a pointer is owned, it is deleted. Ownership is taken of passed in pointer.

Parameters

p	pointer which constructed AutoPtr takes ownership of. p will be deleted on destruction or Reset unless Release is called first.

Referenced by dng_filter_opcode::Apply(), dng_opcode_WarpRectilinear::Apply(), dng_opcode_WarpFisheye::Apply(), dng_render::dng_render(), dng_1d_table::Initialize(), dng_info::Parse(), dng_opcode_MapTable::Prepare(), dng↵_opcode_FixVignetteRadial::Prepare(), dng_jpeg_image_encode_task::Process(), dng_read_tiles_task::Process(), dng_write_tiles_task::Process(), dng_render::Render(), and dng_find_new_raw_image_digest_task::Start().

5.2.3.6 Reset() [2/2]

```
template<class T>
void AutoPtr< T >::Reset ( )
```

If a pointer is owned, it is deleted and the AutoPtr takes NULL as its value.

The documentation for this class was generated from the following file:

- dng_auto_ptr.h

5.3 color_tag_set Class Reference

Public Member Functions

- **color_tag_set** (dng_tiff_directory &directory, const dng_negative &negative)

The documentation for this class was generated from the following file:

- dng_image_writer.cpp

5.4 DecompressInfo Struct Reference

Public Attributes

- int32 **imageWidth**
- int32 **imageHeight**
- int32 **dataPrecision**
- JpegComponentInfo ∗ **compInfo**
- int16 **numComponents**
- JpegComponentInfo ∗ **curCompInfo** [4]
- int16 **compsInScan**
- int16 **MCUmembership** [10]
- HuffmanTable ∗ **dcHuffTblPtrs** [4]
- int32 **Ss**
- int32 **Pt**
- int32 **restartInterval**
- int32 **restartInRows**
- int32 **restartRowsToGo**
- int16 **nextRestartNum**

The documentation for this struct was generated from the following file:

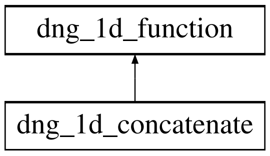
- dng_lossless_jpeg.cpp

5.5 dng_1d_concatenate Class Reference

A dng_1d_function that represents the composition (curry) of two other dng_1d_functions.

```
#include <dng_1d_function.h>
```

Inheritance diagram for dng_1d_concatenate:



Public Member Functions

- dng_1d_concatenate (const dng_1d_function &function1, const dng_1d_function &function2)
- virtual bool IsIdentity () const

  Only true if both function1 and function2 have IsIdentity equal to true.
- virtual real64 Evaluate (real64 x) const
- virtual real64 EvaluateInverse (real64 y) const

Protected Attributes

- const dng_1d_function & **fFunction1**
- const dng_1d_function & **fFunction2**

5.5.1 Detailed Description

A dng_1d_function that represents the composition (curry) of two other dng_1d_functions.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 dng_1d_concatenate()

```
dng_1d_concatenate::dng_1d_concatenate (
                const dng_1d_function & function1,
                const dng_1d_function & function2 )
```

Create a dng_1d_function which computes y = function2.Evaluate(function1.Evaluate(x)). Compose function1 and function2 to compute y = function2.Evaluate(function1.Evaluate(x)). The range of function1.Evaluate must be a subset of 0.0 to 1.0 inclusive, otherwise the result of function1(x) will be pinned (clipped) to 0.0 if <0.0 and to 1.0 if > 1.0 .

Parameters

function1	Inner function of composition.
function2	Outer function of composition.

5.5.3 Member Function Documentation

5.5.3.1 Evaluate()

```
real64 dng_1d_concatenate::Evaluate (
            real64 x ) const  [virtual]
```

Return the composed mapping for value x.

Parameters

x	A value between 0.0 and 1.0 (inclusive).

Return values

function2.Evaluate(function1.Evaluate(x)).	

Implements dng_1d_function.

References dng_1d_function::Evaluate().

5.5.3.2 EvaluateInverse()

```
real64 dng_1d_concatenate::EvaluateInverse (
            real64 y ) const  [virtual]
```

Return the reverse mapped value for y. Be careful using this method with compositions where the inner function does not have a range 0.0 to 1.0 . (Or better yet, do not use such functions.)

Parameters

y	A value to reverse map. Should be within the range of function2.Evaluate.

Return values

A	value x such that function2.Evaluate(function1.Evaluate(x)) == y (to very close approximation).

Reimplemented from dng_1d_function.

References dng_1d_function::EvaluateInverse().

The documentation for this class was generated from the following files:

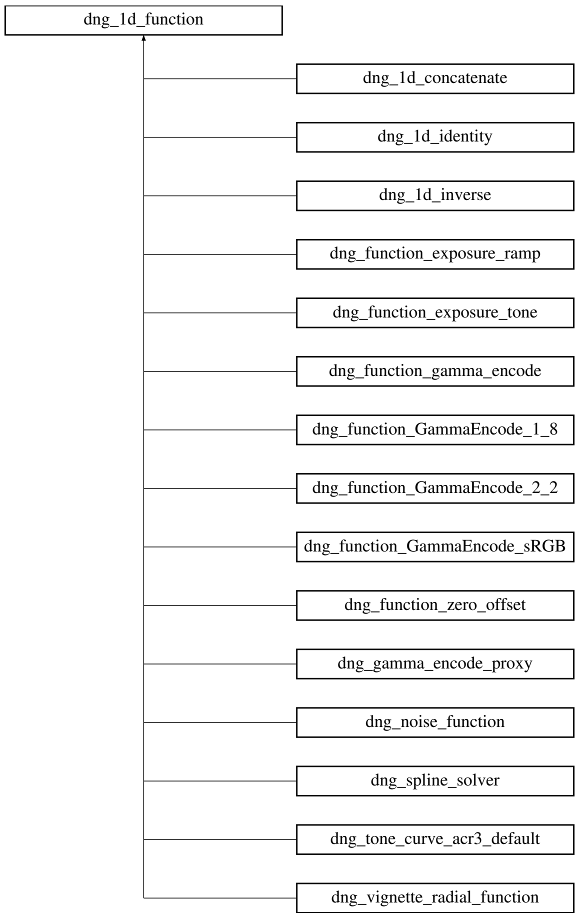
- dng_1d_function.h
- dng_1d_function.cpp

5.6 dng_1d_function Class Reference

A 1D floating-point function.

```
#include <dng_1d_function.h>
```

Inheritance diagram for dng_1d_function:


dng_1d_function
dng_1d_concatenate
dng_1d_identity
dng_1d_inverse
dng_function_exposure_ramp
dng_function_exposure_tone
dng_function_gamma_encode
dng_function_GammaEncode_1_8
dng_function_GammaEncode_2_2
dng_function_GammaEncode_sRGB
dng_function_zero_offset
dng_gamma_encode_proxy
dng_noise_function
dng_spline_solver
dng_tone_curve_acr3_default
dng_vignette_radial_function

Public Member Functions

- virtual bool IsIdentity () const

  Returns true if this function is the map x -> y such that x == y for all x . That is if Evaluate(x) == x for all x.
- virtual real64 Evaluate (real64 x) const =0
- virtual real64 EvaluateInverse (real64 y) const

5.6.1 Detailed Description

A 1D floating-point function.

The domain (input) is always from 0.0 to 1.0, while the range (output) can be an arbitrary interval.

5.6.2 Member Function Documentation

5.6.2.1 Evaluate()

```
virtual real64 dng_1d_function::Evaluate (
            real64 x ) const  [pure virtual]
```

Return the mapping for value x. This method must be implemented by a derived class of dng_1d_function and the derived class determines the lookup method and function used.

Parameters

x	A value between 0.0 and 1.0 (inclusive).

Return values

Mapped	value for x

Implemented in dng_gamma_encode_proxy, dng_vignette_radial_function, dng_1d_inverse, dng_function_gamma_encode, dng_1d_concatenate, dng_tone_curve_acr3_default, dng_noise_function, dng_function_exposure_tone, dng_1d_identity, dng_function_exposure_ramp, dng_spline_solver, dng_function_GammaEncode_2_2, dng_function_GammaEncode_1_8, dng_function_zero_offset, and dng_function_GammaEncode_sRGB.

Referenced by dng_1d_concatenate::Evaluate(), EvaluateInverse(), dng_1d_inverse::EvaluateInverse(), and dng_↩
color_space::GammaEncode().

5.6.2.2 EvaluateInverse()

```
real64 dng_1d_function::EvaluateInverse (
            real64 y ) const  [virtual]
```

Return the reverse mapped value for y. This method can be implemented by derived classes. The default implementation uses Newton's method to solve for x such that Evaluate(x) == y.

Parameters

y	A value to reverse map. Should be within the range of the function implemented by this dng_1d_function .

Return values

A	value x such that Evaluate(x) == y (to very close approximation).

Reimplemented in dng_1d_inverse, dng_1d_concatenate, dng_tone_curve_acr3_default, dng_1d_identity, dng_function_GammaEncode_
dng_function_GammaEncode_1_8, and dng_function_GammaEncode_sRGB.

References Evaluate().

Referenced by dng_1d_inverse::Evaluate(), dng_function_GammaEncode_1_8::EvaluateInverse(), dng_function_↩ GammaEncode_2_2::EvaluateInverse(), dng_1d_concatenate::EvaluateInverse(), and dng_color_space::Gamma↩ Decode().

The documentation for this class was generated from the following files:

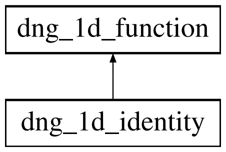
- dng_1d_function.h
- dng_1d_function.cpp

5.7 dng_1d_identity Class Reference

An identity (x -> y such that x == y for all x) mapping function.

```
#include <dng_1d_function.h>
```

Inheritance diagram for dng_1d_identity:



Public Member Functions

- virtual bool IsIdentity () const

  Always returns true for this class.
- virtual real64 Evaluate (real64 x) const

  Always returns x for this class.
- virtual real64 EvaluateInverse (real64 y) const

  Always returns y for this class.

Static Public Member Functions

- static const dng_1d_function & Get ()

  This class is a singleton, and is entirely threadsafe. Use this method to get an instance of the class.

5.7.1 Detailed Description

An identity (x -> y such that x == y for all x) mapping function.

The documentation for this class was generated from the following files:

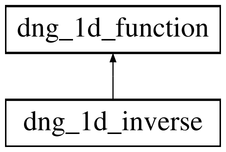
- dng_1d_function.h
- dng_1d_function.cpp

5.8 dng_1d_inverse Class Reference

A dng_1d_function that represents the inverse of another dng_1d_function.

`#include <dng_1d_function.h>`

Inheritance diagram for dng_1d_inverse:



Public Member Functions

- **dng_1d_inverse** (const dng_1d_function &f)
- virtual bool IsIdentity () const

  Returns true if this function is the map x -> y such that x == y for all x . That is if Evaluate(x) == x for all x.
- virtual real64 Evaluate (real64 x) const
- virtual real64 EvaluateInverse (real64 y) const

Protected Attributes

- const dng_1d_function & **fFunction**

5.8.1 Detailed Description

A dng_1d_function that represents the inverse of another dng_1d_function.

5.8.2 Member Function Documentation

5.8.2.1 Evaluate()

```
real64 dng_1d_inverse::Evaluate (
            real64 x ) const  [virtual]
```

Return the mapping for value x. This method must be implemented by a derived class of dng_1d_function and the derived class determines the lookup method and function used.

Parameters

x	A value between 0.0 and 1.0 (inclusive).

Return values

Mapped	value for x

Implements dng_1d_function.

References dng_1d_function::EvaluateInverse().

5.8.2.2 EvaluateInverse()

```
real64 dng_1d_inverse::EvaluateInverse (
             real64 y ) const  [virtual]
```

Return the reverse mapped value for y. This method can be implemented by derived classes. The default implementation uses Newton's method to solve for x such that Evaluate(x) == y.

Parameters

y	A value to reverse map. Should be within the range of the function implemented by this dng_1d_function .

Return values

A	value x such that Evaluate(x) == y (to very close approximation).

Reimplemented from dng_1d_function.

References dng_1d_function::Evaluate().

The documentation for this class was generated from the following files:

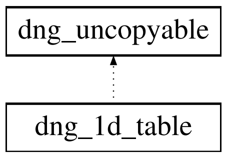
- dng_1d_function.h
- dng_1d_function.cpp

5.9 dng_1d_table Class Reference

A 1D floating-point lookup table using linear interpolation.

`#include <dng_1d_table.h>`

Inheritance diagram for dng_1d_table:



Public Member Functions

- dng_1d_table (uint32 count=kDefaultTableSize)
- uint32 Count () const

  Number of table entries.
- void Initialize (dng_memory_allocator &allocator, const dng_1d_function &function, bool subSample=false)
- real32 Interpolate (real32 x) const
- const real32 ∗ Table () const

  Direct access function for table data.
- void Expand16 (uint16 ∗table16) const

  Expand the table to a 16-bit to 16-bit table.

Static Public Attributes

- static const uint32 kMinTableSize = 512

  Constant denoting minimum size of table.

Protected Attributes

- AutoPtr< dng_memory_block > **fBuffer**
- real32 ∗ **fTable**
- const uint32 **fTableCount**

5.9.1 Detailed Description

A 1D floating-point lookup table using linear interpolation.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 dng_1d_table()

```
dng_1d_table::dng_1d_table (
             uint32 count = kDefaultTableSize )  [explicit]
```

Table constructor. count must be a power of two and at least kMinTableSize.

References DNG_REQUIRE, and kMinTableSize.

5.9.3 Member Function Documentation

5.9.3.1 Initialize()

```
void dng_1d_table::Initialize (
             dng_memory_allocator & allocator,
             const dng_1d_function & function,
             bool subSample = false )
```

Set up table, initialize entries using functiion. This method can throw an exception, e.g. if there is not enough memory.

Parameters

allocator	Memory allocator from which table memory is allocated.
function	Table is initialized with values of finction.Evalluate(0.0) to function.Evaluate(1.0).
subSample	If true, only sample the function a limited number of times and interpolate.

References dng_memory_allocator::Allocate(), dng_memory_block::Buffer_real32(), and AutoPtr< T >::Reset().

Referenced by dng_opcode_FixVignetteRadial::Prepare(), and dng_render_task::Start().

5.9.3.2 Interpolate()

```
real32 dng_1d_table::Interpolate (
             real32 x ) const  [inline]
```

Lookup and interpolate mapping for an input.

Parameters

x	value from 0.0 to 1.0 used as input for mapping

Return values

Approximation	of function.Evaluate(x)

References DNG_ASSERT.

Referenced by dng_opcode_FixVignetteRadial::Prepare().

The documentation for this class was generated from the following files:

- dng_1d_table.h
- dng_1d_table.cpp

5.10 dng_abort_sniffer Class Reference

Class for signaling user cancellation and receiving progress updates.

```
#include <dng_abort_sniffer.h>
```

Public Member Functions

- dng_priority Priority () const
  Getter for priority level.
- void SetPriority (dng_priority priority)
  Setter for priority level.
- void **SniffNoPriorityWait** ()
- virtual bool **ThreadSafe** () const
- virtual bool **SupportsPriorityWait** () const

Static Public Member Functions

- static void SniffForAbort (dng_abort_sniffer ∗sniffer)

Protected Member Functions

- virtual void Sniff ()=0
- virtual void StartTask (const char ∗name, real64 fract)
- virtual void EndTask ()

  Signals the end of the innermost task that has been started.
- virtual void UpdateProgress (real64 fract)

Friends

- class **dng_sniffer_task**

5.10.1 Detailed Description

Class for signaling user cancellation and receiving progress updates.

DNG SDK clients should derive a host application specific implementation from this class.

5.10.2 Member Function Documentation

5.10.2.1 Sniff()

```
virtual void dng_abort_sniffer::Sniff ( )  [protected], [pure virtual]
```

Should be implemented by derived classes to check for an user cancellation.

Referenced by SniffForAbort().

5.10.2.2 SniffForAbort()

```
void dng_abort_sniffer::SniffForAbort (
             dng_abort_sniffer * sniffer )  [static]
```

Check for pending user cancellation or other abort. ThrowUserCanceled will be called if one is pending. This static method is provided as a convenience for quickly testing for an abort and throwing an exception if one is pending.

Parameters

sniffer	The dng_sniffer to test for a pending abort. Can be NULL, in which case there an abort is never signalled.

References Sniff().

Referenced by dng_stream::Flush(), dng_jpeg_image_encode_task::Process(), dng_jpeg_image_find_digest_task::←
Process(), dng_area_task::ProcessOnThread(), dng_sniffer_task::Sniff(), and dng_host::SniffForAbort().

5.10.2.3 StartTask()

```
void dng_abort_sniffer::StartTask (
             const char * name,
             real64 fract )  [protected], [virtual]
```

Signals the start of a named task withn processing in the DNG SDK. Tasks may be nested.

Parameters

name	of the task
fract	Percentage of total processing this task is expected to take. From 0.0 to 1.0 .

Referenced by dng_sniffer_task::dng_sniffer_task().

5.10.2.4 UpdateProgress()

```
void dng_abort_sniffer::UpdateProgress (
             real64 fract )  [protected], [virtual]
```

Signals progress made on current task.

Parameters

fract	percentage of processing completed on current task. From 0.0 to 1.0 .

Referenced by dng_sniffer_task::UpdateProgress().

The documentation for this class was generated from the following files:

- dng_abort_sniffer.h
- dng_abort_sniffer.cpp

5.11 dng_area_spec Class Reference

A class to describe an area of an image, including a pixel subrectangle, plane range, and row/column pitch (e.g., for mosaic images). Useful for specifying opcodes that only apply to specific color planes or pixel types (e.g., only one of the two green Bayer pixels).

```
#include <dng_misc_opcodes.h>
```

Public Types

- enum { **kDataSize** = 32 }

Public Member Functions

- dng_area_spec (const dng_rect &area=dng_rect(), uint32 plane=0, uint32 planes=1, uint32 rowPitch=1, uint32 colPitch=1)

  Create an empty area.
- const dng_rect & Area () const

  The pixel area.
- uint32 Plane () const

  The first plane.
- uint32 Planes () const

  The total number of planes.
- uint32 RowPitch () const

  The row pitch (i.e., stride). A pitch of 1 means all rows.
- uint32 ColPitch () const

  The column pitch (i.e., stride). A pitch of 1 means all columns.
- void GetData (dng_stream &stream)

  Read area data from the specified stream.
- void PutData (dng_stream &stream) const

  Write area data to the specified stream.
- dng_rect Overlap (const dng_rect &tile) const

5.11.1 Detailed Description

A class to describe an area of an image, including a pixel subrectangle, plane range, and row/column pitch (e.g., for mosaic images). Useful for specifying opcodes that only apply to specific color planes or pixel types (e.g., only one of the two green Bayer pixels).

5.11.2 Member Function Documentation

5.11.2.1 Overlap()

```
dng_rect dng_area_spec::Overlap (
            const dng_rect & tile ) const
```

Compute and return pixel area overlap (i.e., intersection) between this area and the specified tile.

Referenced by dng_opcode_GainMap::ModifiedBounds(), dng_opcode_MapTable::ModifiedBounds(), dng_opcode←_MapPolynomial::ModifiedBounds(), dng_opcode_DeltaPerRow::ModifiedBounds(), dng_opcode_DeltaPerColumn::←ModifiedBounds(), dng_opcode_ScalePerRow::ModifiedBounds(), dng_opcode_ScalePerColumn::ModifiedBounds(), dng_opcode_GainMap::ProcessArea(), dng_opcode_MapTable::ProcessArea(), dng_opcode_MapPolynomial::←ProcessArea(), dng_opcode_DeltaPerRow::ProcessArea(), dng_opcode_DeltaPerColumn::ProcessArea(), dng_←opcode_ScalePerRow::ProcessArea(), and dng_opcode_ScalePerColumn::ProcessArea().

The documentation for this class was generated from the following files:

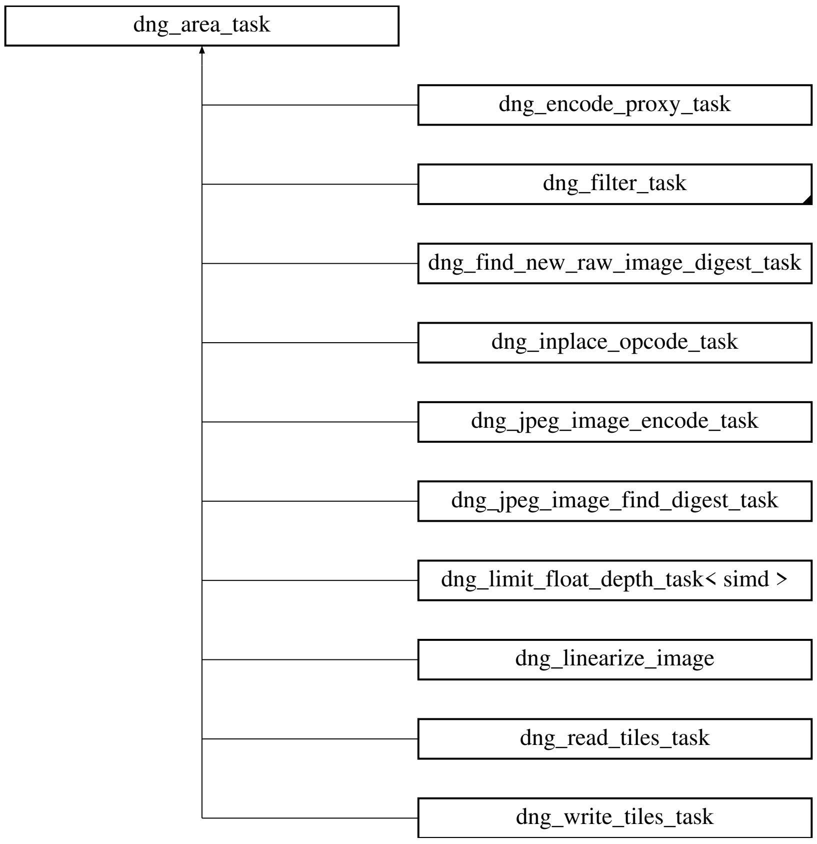
- dng_misc_opcodes.h
- dng_misc_opcodes.cpp

5.12 dng_area_task Class Reference

Abstract class for rectangular processing operations with support for partitioning across multiple processing resources and observing memory constraints.

```
#include <dng_area_task.h>
```

Inheritance diagram for dng_area_task:



Public Member Functions

- **dng_area_task** (const char ∗name="unnamed dng_area_task")
- const char ∗ **Name** () const
- virtual uint32 MaxThreads () const
- virtual uint32 MinTaskArea () const

  operation. (Partitions can be smaller due to small inputs and edge cases.)
- virtual dng_point UnitCell () const
- virtual dng_point MaxTileSize () const
- virtual dng_rect RepeatingTile1 () const
- virtual dng_rect RepeatingTile2 () const
- virtual dng_rect RepeatingTile3 () const
- virtual void Start (uint32 threadCount, const dng_rect &dstArea, const dng_point &tileSize, dng_memory_allocator ∗allocator, dng_abort_sniffer ∗sniffer)
- virtual void Process (uint32 threadIndex, const dng_rect &tile, dng_abort_sniffer ∗sniffer)=0

  and progress updates.
- virtual void Finish (uint32 threadCount)
- dng_point FindTileSize (const dng_rect &area) const
- void ProcessOnThread (uint32 threadIndex, const dng_rect &area, const dng_point &tileSize, dng_abort_sniffer ∗sniffer, dng_area_task_progress ∗progress)
- virtual dng_base_tile_iterator ∗ MakeTileIterator (uint32 threadIndex, const dng_rect &tile, const dng_rect &area) const
- virtual dng_base_tile_iterator ∗ MakeTileIterator (uint32 threadIndex, const dng_point &tileSize, const dng_rect &area) const

Static Public Member Functions

- static void Perform (dng_area_task &task, const dng_rect &area, dng_memory_allocator ∗allocator, dng_abort_sniffer ∗sniffer, dng_area_task_progress ∗progress)

Protected Attributes

- uint32 **fMaxThreads**
- uint32 **fMinTaskArea**
- dng_point **fUnitCell**
- dng_point **fMaxTileSize**
- dng_string **fName**

5.12.1 Detailed Description

Abstract class for rectangular processing operations with support for partitioning across multiple processing resources and observing memory constraints.

5.12.2 Member Function Documentation

5.12.2.1 FindTileSize()

```
dng_point dng_area_task::FindTileSize (
              const dng_rect & area ) const
```

Find tile size taking into account repeating tiles, unit cell, and maximum tile size.

Parameters

area	Computation area for which to find tile size.

Return values

Tile	size as height and width in point.

References MaxTileSize(), RepeatingTile1(), RepeatingTile2(), RepeatingTile3(), and UnitCell().

Referenced by Perform().

5.12.2.2 Finish()

```
void dng_area_task::Finish (
             uint32 threadCount )  [virtual]
```

Task computation finalization and teardown method. Called after all resources have completed processing. Can be overridden to accumulate results and free resources allocated in Start.

Parameters

threadCount	Number of threads used for processing. Same as value passed to Start.

Referenced by Perform().

5.12.2.3 MakeTileIterator() [1/2]

```
dng_base_tile_iterator * dng_area_task::MakeTileIterator (
             uint32 threadIndex,
             const dng_rect & tile,
             const dng_rect & area ) const  [virtual]
```

Factory method to make a tile iterator. This iterator will be used by a thread to process tiles in an area in a specific order. The default implementation uses a forward iterator that visits tiles from left to right (inner), top down (outer). Subclasses can override this method to produce tile iterators that visit tiles in different orders.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is.
tile	The tile to be traversed within the tile area.
area	Tile area partitioned to this resource.

Referenced by ProcessOnThread().

5.12.2.4 MakeTileIterator() [2/2]

```
dng_base_tile_iterator * dng_area_task::MakeTileIterator (
             uint32 threadIndex,
             const dng_point & tileSize,
             const dng_rect & area ) const  [virtual]
```

Factory method to make a tile iterator. This iterator will be used by a thread to process tiles in an area in a specific order. The default implementation uses a forward iterator that visits tiles from left to right (inner), top down (outer). Subclasses can override this method to produce tile iterators that visit tiles in different orders.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is.
tileSize	The tile size to be traversed within the tile area.
area	Tile area partitioned to this resource.

5.12.2.5 MaxThreads()

```
virtual uint32 dng_area_task::MaxThreads ( ) const  [inline], [virtual]
```

Getter for the maximum number of threads (resources) that can be used for processing

Return values

Number	of threads, minimum of 1, that can be used for this task.

5.12.2.6 MaxTileSize()

```
virtual dng_point dng_area_task::MaxTileSize ( ) const  [inline], [virtual]
```

Getter for maximum size of a tile for processing. Often processing will need to allocate temporary buffers or use other resources that are either fixed or in limited supply. The maximum tile size forces further partitioning if the tile is bigger than this size.

Return values

Maximum	tile size allowed for this area task.

Referenced by FindTileSize().

5.12.2.7 MinTaskArea()

```
virtual uint32 dng_area_task::MinTaskArea ( ) const  [inline], [virtual]
```

operation. (Partitions can be smaller due to small inputs and edge cases.)

Getter for minimum area of a partitioned rectangle. Often it is not profitable to use more resources if it requires partitioning the input into chunks that are too small, as the overhead increases more than the speedup. This method can be ovreridden for a specific task to indicate the smallest area for partitioning. Default is 256x256 pixels.

Return values

Minimum	area for a partitoned tile in order to give performant

5.12.2.8 Perform()

```
void dng_area_task::Perform (
             dng_area_task & task,
             const dng_rect & area,
             dng_memory_allocator ∗ allocator,
             dng_abort_sniffer ∗ sniffer,
             dng_area_task_progress ∗ progress )  [static]
```

Default resource partitioner that assumes a single resource to be used for processing. Implementations that are aware of multiple processing resources should override (replace) this method. This is usually done in dng_host::PerformAreaTask.

Parameters

task	The task to perform.
area	The area on which mage processing should be performed.
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.
sniffer	dng_abort_sniffer to use to check for user cancellation and progress updates.
progress	optional pointer to progress reporting object.

References FindTileSize(), Finish(), ProcessOnThread(), and Start().

Referenced by dng_host::PerformAreaTask().

5.12.2.9 Process()

```
virtual void dng_area_task::Process (
             uint32 threadIndex,
             const dng_rect & tile,
             dng_abort_sniffer ∗ sniffer )  [pure virtual]
```

and progress updates.

Process one tile or fully partitioned area. This method is overridden by derived classes to implement the actual image processing. Note that the sniffer can be ignored if it is certain that a processing task will complete very quickly. This method should never be called directly but rather accessed via Process. There is no allocator parameter as all allocation should be done in Start.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the Start method.)
tile	Area to process.
sniffer	dng_abort_sniffer to use to check for user cancellation

Implemented in dng_encode_proxy_task, dng_find_new_raw_image_digest_task, dng_write_tiles_task, dng_linearize_image, dng_limit_float_depth_task< simd >, dng_inplace_opcode_task, dng_jpeg_image_find_digest_task, dng_read_tiles_task, dng_filter_task, and dng_jpeg_image_encode_task.

Referenced by ProcessOnThread().

5.12.2.10 ProcessOnThread()

```
void dng_area_task::ProcessOnThread (
            uint32 threadIndex,
            const dng_rect & area,
            const dng_point & tileSize,
            dng_abort_sniffer * sniffer,
            dng_area_task_progress * progress )
```

Handle one resource's worth of partitioned tiles. Called after thread partitioning has already been done. Area may be further subdivided to handle maximum tile size, etc. It will be rare to override this method.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is.
area	Tile area partitioned to this resource.
tileSize	size of tiles to use for processing.
sniffer	dng_abort_sniffer to use to check for user cancellation and progress updates.
progress	optional pointer to progress reporting object.

References MakeTileIterator(), Process(), RepeatingTile1(), RepeatingTile2(), RepeatingTile3(), and dng_abort_↩
sniffer::SniffForAbort().

Referenced by Perform().

5.12.2.11 RepeatingTile1()

```
dng_rect dng_area_task::RepeatingTile1 ( ) const  [virtual]
```

Getter for RepeatingTile1. RepeatingTile1, RepeatingTile2, and RepeatingTile3 are used to establish a set of 0 to 3 tile patterns for which the resulting partitions that the final Process method is called on will not cross tile boundaries in any of the tile patterns. This can be used for a processing routine that needs to read from two tiles and write to a third such that all the tiles are aligned and sized in a certain way. A RepeatingTile value is valid if it is non-empty. Higher numbered RepeatingTile patterns are only used if all lower ones are non-empty. A RepeatingTile pattern must be a multiple of UnitCell in size for all constraints of the partitionerr to be met.

Reimplemented in dng_encode_proxy_task, dng_linearize_image, and dng_limit_float_depth_task< simd >.

Referenced by FindTileSize(), and ProcessOnThread().

5.12.2.12 RepeatingTile2()

```
dng_rect dng_area_task::RepeatingTile2 ( ) const  [virtual]
```

Getter for RepeatingTile2. RepeatingTile1, RepeatingTile2, and RepeatingTile3 are used to establish a set of 0 to 3 tile patterns for which the resulting partitions that the final Process method is called on will not cross tile boundaries in any of the tile patterns. This can be used for a processing routine that needs to read from two tiles and write to a third such that all the tiles are aligned and sized in a certain way. A RepeatingTile value is valid if it is non-empty. Higher numbered RepeatingTile patterns are only used if all lower ones are non-empty. A RepeatingTile pattern must be a multiple of UnitCell in size for all constraints of the partitionerr to be met.

Reimplemented in dng_encode_proxy_task, dng_linearize_image, and dng_limit_float_depth_task< simd >.

Referenced by FindTileSize(), and ProcessOnThread().

5.12.2.13 RepeatingTile3()

```
dng_rect dng_area_task::RepeatingTile3 ( ) const  [virtual]
```

Getter for RepeatingTile3. RepeatingTile1, RepeatingTile2, and RepeatingTile3 are used to establish a set of 0 to 3 tile patterns for which the resulting partitions that the final Process method is called on will not cross tile boundaries in any of the tile patterns. This can be used for a processing routine that needs to read from two tiles and write to a third such that all the tiles are aligned and sized in a certain way. A RepeatingTile value is valid if it is non-empty. Higher numbered RepeatingTile patterns are only used if all lower ones are non-empty. A RepeatingTile pattern must be a multiple of UnitCell in size for all constraints of the partitionerr to be met.

Referenced by FindTileSize(), and ProcessOnThread().

5.12.2.14 Start()

```
void dng_area_task::Start (
             uint32 threadCount,
             const dng_rect & dstArea,
             const dng_point & tileSize,
             dng_memory_allocator * allocator,
             dng_abort_sniffer * sniffer )  [virtual]
```

Task startup method called before any processing is done on partitions. The Start method is called before any processing is done and can be overridden to allocate temporary buffers, etc.

Parameters

threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads.
dstArea	Area to be processed in the current run of the task.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.
sniffer	Sniffer to test for user cancellation and to set up progress.

Reimplemented in dng_find_new_raw_image_digest_task, dng_render_task, dng_resample_task, dng_inplace_opcode_task, dng_filter_opcode_task, and dng_filter_task.

Referenced by Perform().

5.12.2.15 UnitCell()

```
virtual dng_point dng_area_task::UnitCell ( ) const  [inline], [virtual]
```

Getter for dimensions of which partitioned tiles should be a multiple. Various methods of processing prefer certain alignments. The partitioning attempts to construct tiles such that the sizes are a multiple of the dimensions of this point.

Return values

a	point giving preferred alignment in x and y

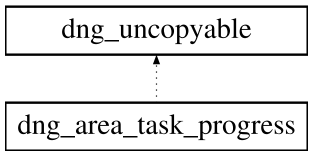
Referenced by FindTileSize().

The documentation for this class was generated from the following files:

- dng_area_task.h
- dng_area_task.cpp

5.13 dng_area_task_progress Class Reference

Inheritance diagram for dng_area_task_progress:



Public Member Functions

- virtual void **FinishedTile** (const dng_rect &)=0

The documentation for this class was generated from the following file:

- dng_area_task.h

5.14 dng_bad_pixel_list Class Reference

A list of bad pixels and rectangles (usually single rows or columns).

```
#include <dng_bad_pixels.h>
```

Public Types

- enum { **kNoIndex** = 0xFFFFFFFF }

Public Member Functions

- dng_bad_pixel_list ()

  Create an empty bad pixel list.
- uint32 PointCount () const

  Returns the number of bad single pixels.
- const dng_point & Point (uint32 index) const
- uint32 RectCount () const

  Returns the number of bad rectangles.
- const dng_rect & Rect (uint32 index) const
- bool IsEmpty () const
- bool NotEmpty () const
- void AddPoint (const dng_point &pt)
- void AddRect (const dng_rect &r)
- void Sort ()
- bool IsPointIsolated (uint32 index, uint32 radius) const
- bool IsRectIsolated (uint32 index, uint32 radius) const
- bool IsPointValid (const dng_point &pt, const dng_rect &imageBounds, uint32 index=kNoIndex) const

5.14.1 Detailed Description

A list of bad pixels and rectangles (usually single rows or columns).

5.14.2 Member Function Documentation

5.14.2.1 AddPoint()

```
void dng_bad_pixel_list::AddPoint (
             const dng_point & pt )
```

Add the specified coordinate to the list of bad single pixels.

Parameters

pt	The bad single pixel to add.

5.14.2.2 AddRect()

```
void dng_bad_pixel_list::AddRect (
             const dng_rect & r )
```

Add the specified rectangle to the list of bad rectangles.

Parameters

r	The bad rectangle to add.

5.14.2.3 IsEmpty()

```
bool dng_bad_pixel_list::IsEmpty ( ) const  [inline]
```

Returns true iff there are zero bad single pixels and zero bad rectangles.

References PointCount(), and RectCount().

Referenced by NotEmpty().

5.14.2.4 IsPointIsolated()

```
bool dng_bad_pixel_list::IsPointIsolated (
             uint32 index,
             uint32 radius ) const
```

Returns true iff the specified bad single pixel is isolated, i.e., there is no other bad single pixel or bad rectangle that lies within radius pixels of this bad single pixel.

Parameters

index	The index of the bad single pixel to test.
radius	The pixel radius to test for isolation.

References Point(), PointCount(), Rect(), and RectCount().

Referenced by dng_opcode_FixBadPixelsList::ProcessArea().

5.14.2.5 IsPointValid()

```
bool dng_bad_pixel_list::IsPointValid (
            const dng_point & pt,
            const dng_rect & imageBounds,
            uint32 index = kNoIndex ) const
```

Returns true iff the specified point is valid, i.e., lies within the specified image bounds, is different from all other bad single pixels, and is not contained in any bad rectangle. The second and third conditions are only checked if provided with a starting search index.

Parameters

pt	The point to test for validity.
imageBounds	The pt must lie within imageBounds to be valid. \index The search index to use (or kNoIndex, to avoid a search) for checking for validity.

References Point(), PointCount(), Rect(), and RectCount().

5.14.2.6 IsRectIsolated()

```
bool dng_bad_pixel_list::IsRectIsolated (
            uint32 index,
            uint32 radius ) const
```

Returns true iff the specified bad rectangle is isolated, i.e., there is no other bad single pixel or bad rectangle that lies within radius pixels of this bad rectangle.

Parameters

index	The index of the bad rectangle to test.
radius	The pixel radius to test for isolation.

References Rect(), and RectCount().

Referenced by dng_opcode_FixBadPixelsList::ProcessArea().

5.14.2.7 NotEmpty()

```
bool dng_bad_pixel_list::NotEmpty ( ) const  [inline]
```

Returns true iff there is at least one bad single pixel or at least one bad rectangle.

References IsEmpty().

5.14.2.8 Point()

```
const dng_point& dng_bad_pixel_list::Point (
             uint32 index ) const  [inline]
```

Retrieves the bad single pixel coordinate via the specified list index.

Parameters

index	The list index from which to retrieve the bad single pixel coordinate.

Referenced by IsPointIsolated(), IsPointValid(), dng_opcode_FixBadPixelsList::ProcessArea(), and dng_opcode_Fix↩ BadPixelsList::PutData().

5.14.2.9 Rect()

```
const dng_rect& dng_bad_pixel_list::Rect (
             uint32 index ) const  [inline]
```

Retrieves the bad rectangle via the specified list index.

Parameters

index	The list index from which to retrieve the bad rectangle coordinates.

Referenced by IsPointIsolated(), IsPointValid(), IsRectIsolated(), dng_opcode_FixBadPixelsList::ProcessArea(), and dng_opcode_FixBadPixelsList::PutData().

5.14.2.10 Sort()

```
void dng_bad_pixel_list::Sort ( )
```

Sort the bad single pixels and bad rectangles by coordinates (top to bottom, then left to right).

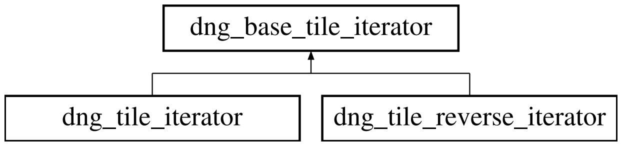
References PointCount(), and RectCount().

The documentation for this class was generated from the following files:

- dng_bad_pixels.h
- dng_bad_pixels.cpp

5.15 dng_base_tile_iterator Class Reference

Inheritance diagram for dng_base_tile_iterator:



Public Member Functions

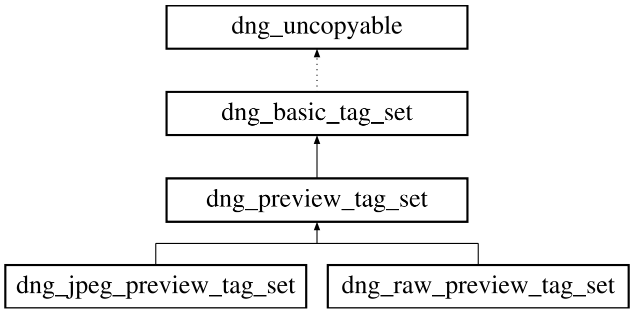
- virtual bool **GetOneTile** (dng_rect &tile)=0

The documentation for this class was generated from the following file:

- dng_tile_iterator.h

5.16 dng_basic_tag_set Class Reference

Inheritance diagram for dng_basic_tag_set:



Public Member Functions

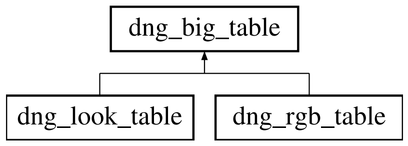
- **dng_basic_tag_set** (dng_tiff_directory &directory, const dng_ifd &info)
- void **SetTileOffset** (uint32 index, uint32 offset)
- void **SetTileByteCount** (uint32 index, uint32 count)
- bool **WritingStrips** () const

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.17 dng_big_table Class Reference

Inheritance diagram for dng_big_table:



Public Member Functions

- bool **IsMissing** () const
- void **SetMissing** ()
- virtual bool **IsValid** () const =0
- const dng_fingerprint & **Fingerprint** () const
- bool **DecodeFromBinary** (const uint8 ∗compressedData, uint32 compressedSize, dng_memory_allocator &allocator)
- bool **DecodeFromString** (const dng_string &block1, dng_memory_allocator &allocator)
- dng_memory_block ∗ **EncodeAsBinary** (dng_memory_allocator &allocator, uint32 &compressedSize) const
- dng_memory_block ∗ **EncodeAsString** (dng_memory_allocator &allocator) const
- bool **ExtractFromCache** (const dng_fingerprint &fingerprint)
- bool **ReadTableFromXMP** (const dng_xmp &xmp, const char ∗ns, const dng_fingerprint &fingerprint)
- bool **ReadFromXMP** (const dng_xmp &xmp, const char ∗ns, const char ∗path, dng_big_table_storage &storage)
- void **WriteToXMP** (dng_xmp &xmp, const char ∗ns, const char ∗path, dng_big_table_storage &storage) const

Protected Types

- enum **BigTableTypeEnum** { **btt_LookTable** = 0, **btt_RGBTable** = 1 }

Protected Member Functions

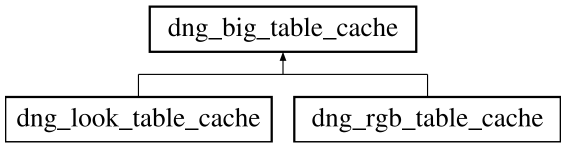
- **dng_big_table** (dng_big_table_cache ∗cache)
- **dng_big_table** (const dng_big_table &table)
- dng_big_table & **operator=** (const dng_big_table &table)
- void **RecomputeFingerprint** ()
- virtual void **GetStream** (dng_stream &stream)=0
- virtual void **PutStream** (dng_stream &stream, bool forFingerprint) const =0

The documentation for this class was generated from the following files:

- dng_big_table.h
- dng_big_table.cpp

5.18 dng_big_table_cache Class Reference

Inheritance diagram for dng_big_table_cache:



Public Member Functions

- void **FlushRecentlyUsed** ()

Static Public Member Functions

- static void **Increment** (dng_big_table_cache ∗cache, const dng_fingerprint &fingerprint)
- static void **Decrement** (dng_big_table_cache ∗cache, const dng_fingerprint &fingerprint)
- static void **Add** (dng_big_table_cache ∗cache, const dng_big_table &table)
- static bool **Extract** (dng_big_table_cache ∗cache, const dng_fingerprint &fingerprint, dng_big_table &table)

Protected Types

- enum { **kDefaultRecentlyUsedLimit** = 5 }

Protected Member Functions

- void **UseTable** (dng_lock_std_mutex &lock, const dng_fingerprint &fingerprint)
- virtual void **CacheIncrement** (dng_lock_std_mutex &lock, const dng_fingerprint &fingerprint)
- virtual void **CacheDecrement** (dng_lock_std_mutex &lock, const dng_fingerprint &fingerprint)
- virtual void **CacheAdd** (dng_lock_std_mutex &lock, const dng_big_table &table)
- virtual bool **CacheExtract** (dng_lock_std_mutex &lock, const dng_fingerprint &fingerprint, dng_big_table &table)
- virtual void **InsertTableData** (dng_lock_std_mutex &lock, const dng_big_table &table)=0
- virtual void **EraseTableData** (dng_lock_std_mutex &lock, const dng_fingerprint &fingerprint)=0
- virtual void **ExtractTableData** (dng_lock_std_mutex &lock, const dng_fingerprint &fingerprint, dng_big_table &table)=0

Protected Attributes

- uint32 **fRecentlyUsedLimit**

The documentation for this class was generated from the following file:

- dng_big_table.cpp

5.19 dng_big_table_storage Class Reference

Public Member Functions

- virtual bool **ReadTable** (dng_big_table &table, const dng_fingerprint &fingerprint, dng_memory_allocator &allocator)
- virtual bool **WriteTable** (const dng_big_table &table, const dng_fingerprint &fingerprint, dng_memory_allocator &allocator)
- virtual void **MissingTable** (const dng_fingerprint &fingerprint)

The documentation for this class was generated from the following files:

- dng_big_table.h
- dng_big_table.cpp

5.20 dng_bilinear_interpolator Class Reference

Public Member Functions

- **dng_bilinear_interpolator** (const dng_mosaic_info &info, int32 rowStep, int32 colStep)
- void **Interpolate** (dng_pixel_buffer &srcBuffer, dng_pixel_buffer &dstBuffer)

The documentation for this class was generated from the following file:

- dng_mosaic_info.cpp

5.21 dng_bilinear_kernel Class Reference

Public Types

- enum { **kMaxCount** = 8 }

Public Member Functions

- void **Add** (const dng_point &delta, real32 weight)
- void **Finalize** (const dng_point &scale, uint32 patRow, uint32 patCol, int32 rowStep, int32 colStep)

Public Attributes

- uint32 **fCount**
- dng_point **fDelta** [kMaxCount]
- real32 **fWeight32** [kMaxCount]
- uint16 **fWeight16** [kMaxCount]
- int32 **fOffset** [kMaxCount]

The documentation for this class was generated from the following file:

- dng_mosaic_info.cpp

5.22 dng_bilinear_pattern Class Reference

Public Types

- enum { **kMaxPattern** = kMaxCFAPattern ∗ 2 }

Public Member Functions

- void **Calculate** (const dng_mosaic_info &info, uint32 dstPlane, int32 rowStep, int32 colStep)

Public Attributes

- dng_point **fScale**
- uint32 **fPatRows**
- uint32 **fPatCols**
- dng_bilinear_kernel **fKernel** [kMaxPattern][kMaxPattern]
- uint32 **fCounts** [kMaxPattern][kMaxPattern]
- int32 ∗ **fOffsets** [kMaxPattern][kMaxPattern]
- uint16 ∗ **fWeights16** [kMaxPattern][kMaxPattern]
- real32 ∗ **fWeights32** [kMaxPattern][kMaxPattern]

The documentation for this class was generated from the following file:

- dng_mosaic_info.cpp

5.23 dng_camera_profile Class Reference

Container for DNG camera color profile and calibration data.

```
#include <dng_camera_profile.h>
```

Public Member Functions

- void SetName (const char ∗name)
- const dng_string & Name () const
- bool NameIsEmbedded () const
- void SetCalibrationIlluminant1 (uint32 light)
- void SetCalibrationIlluminant2 (uint32 light)
- uint32 CalibrationIlluminant1 () const
- uint32 CalibrationIlluminant2 () const
- real64 CalibrationTemperature1 () const
- real64 CalibrationTemperature2 () const
- void SetColorMatrix1 (const dng_matrix &m)
- void SetColorMatrix2 (const dng_matrix &m)
- bool HasColorMatrix1 () const

Predicate to test if first camera matrix is set.
- bool HasColorMatrix2 () const

  Predicate to test if second camera matrix is set.
- const dng_matrix & ColorMatrix1 () const

  Getter for first of up to two color matrices used for calibrations.
- const dng_matrix & ColorMatrix2 () const

  Getter for second of up to two color matrices used for calibrations.
- void SetForwardMatrix1 (const dng_matrix &m)

  Setter for first of up to two forward matrices used for calibrations.
- void SetForwardMatrix2 (const dng_matrix &m)

  Setter for second of up to two forward matrices used for calibrations.
- const dng_matrix & ForwardMatrix1 () const

  Getter for first of up to two forward matrices used for calibrations.
- const dng_matrix & ForwardMatrix2 () const

  Getter for second of up to two forward matrices used for calibrations.
- void SetReductionMatrix1 (const dng_matrix &m)
- void SetReductionMatrix2 (const dng_matrix &m)
- const dng_matrix & ReductionMatrix1 () const

  Getter for first of up to two dimensionality reduction hints for four color cameras.
- const dng_matrix & ReductionMatrix2 () const

  Getter for second of up to two dimensionality reduction hints for four color cameras.
- const dng_fingerprint & Fingerprint () const

  Getter function from profile fingerprint.
- dng_fingerprint UniqueID () const
- dng_camera_profile_id ProfileID () const
- void SetCopyright (const char ∗copyright)
- const dng_string & Copyright () const
- void SetEmbedPolicy (uint32 policy)
- uint32 EmbedPolicy () const
- bool IsLegalToEmbed () const
- bool HasHueSatDeltas () const

  Returns true iff the profile has a valid HueSatMap color table.
- const dng_hue_sat_map & HueSatDeltas1 () const

  Getter for first HueSatMap color table (for calibration illuminant 1).
- void SetHueSatDeltas1 (const dng_hue_sat_map &deltas1)

  Setter for first HueSatMap color table (for calibration illuminant 1).
- const dng_hue_sat_map & HueSatDeltas2 () const

  Getter for second HueSatMap color table (for calibration illuminant 2).
- void SetHueSatDeltas2 (const dng_hue_sat_map &deltas2)

  Setter for second HueSatMap color table (for calibration illuminant 2).
- uint32 HueSatMapEncoding () const

  Returns the hue sat map encoding (see ProfileHueSatMapEncoding tag).
- void SetHueSatMapEncoding (uint32 encoding)
- bool HasLookTable () const

  Returns true if the profile has a LookTable.
- const dng_hue_sat_map & LookTable () const

  Getter for LookTable.
- void SetLookTable (const dng_hue_sat_map &table)

Setter for LookTable.
- uint32 LookTableEncoding () const

  Returns the LookTable encoding (see ProfileLookTableEncoding tag).
- void SetLookTableEncoding (uint32 encoding)
- void SetBaselineExposureOffset (real64 exposureOffset)
- const dng_srational & BaselineExposureOffset () const
- void SetDefaultBlackRender (uint32 defaultBlackRender)
- uint32 DefaultBlackRender () const
- const dng_tone_curve & ToneCurve () const

  Returns the tone curve of the profile.
- void SetToneCurve (const dng_tone_curve &curve)

  Sets the tone curve of the profile to the specified curve.
- void SetProfileCalibrationSignature (const char ∗signature)
- const dng_string & ProfileCalibrationSignature () const
- void SetUniqueCameraModelRestriction (const char ∗camera)
- const dng_string & UniqueCameraModelRestriction () const
- void SetWasReadFromDNG (bool state=true)
- bool WasReadFromDNG () const

  Was this profile read from a DNG?
- void SetWasReadFromDisk (bool state=true)
- bool WasReadFromDisk () const

  Was this profile read from disk?
- void SetWasBuiltinMatrix (bool state=true)
- bool WasBuiltinMatrix () const

  Was this profile a built-in matrix profile?
- bool IsValid (uint32 channels) const
- bool EqualData (const dng_camera_profile &profile) const
- void Parse (dng_stream &stream, dng_camera_profile_info &profileInfo)

  Parse profile from dng_camera_profile_info data.
- bool ParseExtended (dng_stream &stream)
- virtual void SetFourColorBayer ()

  Convert from a three-color to a four-color Bayer profile.
- dng_hue_sat_map ∗ HueSatMapForWhite (const dng_xy_coord &white) const
- void Stub ()

  Stub out the profile (free memory used by large tables).
- bool WasStubbed () const

  Was this profile stubbed?

Static Public Member Functions

- static void NormalizeColorMatrix (dng_matrix &m)

  Utility function to normalize the scale of the color matrix.
- static void NormalizeForwardMatrix (dng_matrix &m)

  Utility function to normalize the scale of the forward matrix.

Protected Member Functions

- void **ClearFingerprint** ()
- void **CalculateFingerprint** () const

Static Protected Member Functions

- static real64 **IlluminantToTemperature** (uint32 light)
- static bool **ValidForwardMatrix** (const dng_matrix &m)
- static void **ReadHueSatMap** (dng_stream &stream, dng_hue_sat_map &hueSatMap, uint32 hues, uint32 sats, uint32 vals, bool skipSat0)

Protected Attributes

- dng_string **fName**
- uint32 **fCalibrationIlluminant1**
- uint32 **fCalibrationIlluminant2**
- dng_matrix **fColorMatrix1**
- dng_matrix **fColorMatrix2**
- dng_matrix **fForwardMatrix1**
- dng_matrix **fForwardMatrix2**
- dng_matrix **fReductionMatrix1**
- dng_matrix **fReductionMatrix2**
- dng_fingerprint **fFingerprint**
- dng_string **fCopyright**
- uint32 **fEmbedPolicy**
- dng_hue_sat_map **fHueSatDeltas1**
- dng_hue_sat_map **fHueSatDeltas2**
- uint32 **fHueSatMapEncoding**
- dng_hue_sat_map **fLookTable**
- uint32 **fLookTableEncoding**
- dng_srational **fBaselineExposureOffset**
- uint32 **fDefaultBlackRender**
- dng_tone_curve **fToneCurve**
- dng_string **fProfileCalibrationSignature**
- dng_string **fUniqueCameraModelRestriction**
- bool **fWasReadFromDNG**
- bool **fWasReadFromDisk**
- bool **fWasBuiltinMatrix**
- bool **fWasStubbed**

5.23.1 Detailed Description

Container for DNG camera color profile and calibration data.

5.23.2 Member Function Documentation

5.23.2.1 BaselineExposureOffset()

```
const dng_srational& dng_camera_profile::BaselineExposureOffset ( ) const  [inline]
```

Returns the baseline exposure offset of the profile (see BaselineExposureOffset tag).

Referenced by dng_negative::TotalBaselineExposure().

5.23.2.2 CalibrationIlluminant1()

```
uint32 dng_camera_profile::CalibrationIlluminant1 ( ) const  [inline]
```

Getter for first of up to two light sources used for calibration. Uses the EXIF encodings for illuminant and is used to distinguish which matrix to use. Corresponds to the DNG CalibrationIlluminant1 tag.

Referenced by CalibrationTemperature1().

5.23.2.3 CalibrationIlluminant2()

```
uint32 dng_camera_profile::CalibrationIlluminant2 ( ) const  [inline]
```

Getter for second of up to two light sources used for calibration. Uses the EXIF encodings for illuminant and is used to distinguish which matrix to use. Corresponds to the DNG CalibrationIlluminant2 tag.

Referenced by CalibrationTemperature2().

5.23.2.4 CalibrationTemperature1()

```
real64 dng_camera_profile::CalibrationTemperature1 ( ) const  [inline]
```

Getter for first of up to two light sources used for calibration, returning result as color temperature.

References CalibrationIlluminant1().

Referenced by dng_color_spec::dng_color_spec(), and HueSatMapForWhite().

5.23.2.5 CalibrationTemperature2()

```
real64 dng_camera_profile::CalibrationTemperature2 ( ) const  [inline]
```

Getter for second of up to two light sources used for calibration, returning result as color temperature.

References CalibrationIlluminant2().

Referenced by dng_color_spec::dng_color_spec(), and HueSatMapForWhite().

5.23.2.6 Copyright()

```
const dng_string& dng_camera_profile::Copyright ( ) const  [inline]
```

Getter for camera profile copyright.

Return values

Copyright	string for profile.

5.23.2.7 DefaultBlackRender()

```
uint32 dng_camera_profile::DefaultBlackRender ( ) const  [inline]
```

Returns the default black render of the profile (see DefaultBlackRender tag).

Referenced by dng_render::dng_render().

5.23.2.8 EmbedPolicy()

```
uint32 dng_camera_profile::EmbedPolicy ( ) const  [inline]
```

Getter for camera profile embed policy.

Return values

Policy	for profile.

Referenced by IsLegalToEmbed().

5.23.2.9 EqualData()

```
bool dng_camera_profile::EqualData (
             const dng_camera_profile & profile ) const
```

Predicate to check if two camera profiles are colorwise equal, thus ignores the profile name.

Parameters

profile	Camera profile to compare to.

5.23.2.10 HueSatMapForWhite()

```
dng_hue_sat_map * dng_camera_profile::HueSatMapForWhite (
             const dng_xy_coord & white ) const
```

Find the hue/sat table to use for a given white point, if any. The calling routine owns the resulting table.

References CalibrationTemperature1(), CalibrationTemperature2(), HueSatDeltas1(), HueSatDeltas2(), dng_hue_sat↩_map::Interpolate(), and dng_hue_sat_map::IsValid().

5.23.2.11 IsLegalToEmbed()

```
bool dng_camera_profile::IsLegalToEmbed ( ) const  [inline]
```

Returns true iff the profile is legal to embed in a DNG, per the profile's embed policy.

References EmbedPolicy(), and WasReadFromDNG().

5.23.2.12 IsValid()

```
bool dng_camera_profile::IsValid (
             uint32 channels ) const
```

Determines if this a valid profile for this number of color channels?

Return values

true	if the profile is valid.

References ReportError().

Referenced by dng_color_spec::dng_color_spec(), dng_info::Parse(), and SetFourColorBayer().

5.23.2.13 Name()

```
const dng_string& dng_camera_profile::Name ( ) const  [inline]
```

Getter for camera profile name.

Return values

Name	of profile.

Referenced by ProfileID().

5.23.2.14 NameIsEmbedded()

```
bool dng_camera_profile::NameIsEmbedded ( ) const  [inline]
```

Test if this name is embedded.

Return values

true	if the name matches the name of the embedded camera profile.

5.23.2.15 ParseExtended()

```
bool dng_camera_profile::ParseExtended (
            dng_stream & stream )
```

Parse from an extended profile stream, which is similar to stand alone TIFF file.

References Parse().

5.23.2.16 ProfileCalibrationSignature()

```
const dng_string& dng_camera_profile::ProfileCalibrationSignature ( ) const  [inline]
```

Returns the profile calibration signature (see ProfileCalibrationSignature tag) of the profile.

Referenced by dng_color_spec::dng_color_spec().

5.23.2.17 ProfileID()

```
dng_camera_profile_id dng_camera_profile::ProfileID ( ) const  [inline]
```

Getter for camera profile id.

Return values

ID	of profile.

References Fingerprint(), and Name().

5.23.2.18 SetBaselineExposureOffset()

```
void dng_camera_profile::SetBaselineExposureOffset (
            real64 exposureOffset )  [inline]
```

Sets the baseline exposure offset of the profile (see BaselineExposureOffset tag) to the specified value.

5.23.2.19 SetCalibrationIlluminant1()

```
void dng_camera_profile::SetCalibrationIlluminant1 (
            uint32 light )  [inline]
```

Setter for first of up to two light sources used for calibration. Uses the EXIF encodings for illuminant and is used to distinguish which matrix to use. Corresponds to the DNG CalibrationIlluminant1 tag.

5.23.2.20 SetCalibrationIlluminant2()

```
void dng_camera_profile::SetCalibrationIlluminant2 (
            uint32 light )  [inline]
```

Setter for second of up to two light sources used for calibration. Uses the EXIF encodings for illuminant and is used to distinguish which matrix to use. Corresponds to the DNG CalibrationIlluminant2 tag.

5.23.2.21 SetColorMatrix1()

```
void dng_camera_profile::SetColorMatrix1 (
            const dng_matrix & m )
```

Setter for first of up to two color matrices used for reference camera calibrations. These matrices map XYZ values to camera values. The DNG SDK needs to map colors that direction in order to determine the clipping points for highlight recovery logic based on the white point. If cameras were all three-color, the matrix could be stored as a forward matrix. The inverse matrix is requried to support four-color cameras.

References NormalizeColorMatrix().

5.23.2.22 SetColorMatrix2()

```
void dng_camera_profile::SetColorMatrix2 (
            const dng_matrix & m )
```

Setter for second of up to two color matrices used for reference camera calibrations. These matrices map XYZ values to camera values. The DNG SDK needs to map colors that direction in order to determine the clipping points for highlight recovery logic based on the white point. If cameras were all three-color, the matrix could be stored as a forward matrix. The inverse matrix is requried to support four-color cameras.

References NormalizeColorMatrix().

5.23.2.23 SetCopyright()

```
void dng_camera_profile::SetCopyright (
            const char * copyright )  [inline]
```

Setter for camera profile copyright.

Parameters

copyright	Copyright string to use for this camera profile.

5.23.2.24 SetDefaultBlackRender()

```
void dng_camera_profile::SetDefaultBlackRender (
             uint32 defaultBlackRender )  [inline]
```

Sets the default black render of the profile (see DefaultBlackRender tag) to the specified option.

5.23.2.25 SetEmbedPolicy()

```
void dng_camera_profile::SetEmbedPolicy (
             uint32 policy )  [inline]
```

Setter for camera profile embed policy.

Parameters

policy	Policy to use for this camera profile.

5.23.2.26 SetHueSatMapEncoding()

```
void dng_camera_profile::SetHueSatMapEncoding (
             uint32 encoding )  [inline]
```

Sets the hue sat map encoding (see ProfileHueSatMapEncoding tag) to the specified encoding.

5.23.2.27 SetLookTableEncoding()

```
void dng_camera_profile::SetLookTableEncoding (
             uint32 encoding )  [inline]
```

Sets the LookTable encoding (see ProfileLookTableEncoding tag) to the specified encoding.

5.23.2.28 SetName()

```
void dng_camera_profile::SetName (
             const char * name )  [inline]
```

Setter for camera profile name.

Parameters

name	Name to use for this camera profile.

5.23.2.29 SetProfileCalibrationSignature()

```
void dng_camera_profile::SetProfileCalibrationSignature (
             const char * signature )  [inline]
```

Sets the profile calibration signature (see ProfileCalibrationSignature tag) to the specified string.

5.23.2.30 SetReductionMatrix1()

```
void dng_camera_profile::SetReductionMatrix1 (
             const dng_matrix & m )
```

Setter for first of up to two dimensionality reduction hints for four-color cameras. This is an optional matrix that maps four components to three. See Appendix 6 of the DNG 1.1.0 specification.

5.23.2.31 SetReductionMatrix2()

```
void dng_camera_profile::SetReductionMatrix2 (
             const dng_matrix & m )
```

Setter for second of up to two dimensionality reduction hints for four-color cameras. This is an optional matrix that maps four components to three. See Appendix 6 of the DNG 1.1.0 specification.

5.23.2.32 SetUniqueCameraModelRestriction()

```
void dng_camera_profile::SetUniqueCameraModelRestriction (
             const char * camera )  [inline]
```

Setter for camera unique model name to restrict use of this profile.

Parameters

camera	Camera unique model name designating only camera this profile can be used with. (Empty string for no restriction.)

5.23.2.33 SetWasBuiltinMatrix()

```
void dng_camera_profile::SetWasBuiltinMatrix (
             bool state = true )  [inline]
```

Sets internal flag to indicate this profile was originally a built-in matrix profile.

5.23.2.34 SetWasReadFromDisk()

```
void dng_camera_profile::SetWasReadFromDisk (
             bool state = true )  [inline]
```

Sets internal flag to indicate this profile was originally read from disk.

5.23.2.35 SetWasReadFromDNG()

```
void dng_camera_profile::SetWasReadFromDNG (
             bool state = true )  [inline]
```

Sets internal flag to indicate this profile was originally read from a DNG file.

5.23.2.36 UniqueCameraModelRestriction()

```
const dng_string& dng_camera_profile::UniqueCameraModelRestriction ( ) const  [inline]
```

Getter for camera unique model name to restrict use of this profile.

Return values

Unique	model name of only camera this profile can be used with or empty if no restriction.

5.23.2.37 UniqueID()

```
dng_fingerprint dng_camera_profile::UniqueID ( ) const
```

Getter for camera profile unique ID. Use this ID for uniquely identifying profiles (e.g., for syncing purposes).

References dng_fingerprint::IsValid(), dng_stream::Put(), and dng_stream::SetLittleEndian().

The documentation for this class was generated from the following files:

- dng_camera_profile.h
- dng_camera_profile.cpp

5.24 dng_camera_profile_id Class Reference

An ID for a camera profile consisting of a name and optional fingerprint.

```
#include <dng_camera_profile.h>
```

Public Member Functions

- dng_camera_profile_id ()

  Construct an invalid camera profile ID (empty name and fingerprint).
- dng_camera_profile_id (const char ∗name)
- dng_camera_profile_id (const dng_string &name)
- dng_camera_profile_id (const char ∗name, const dng_fingerprint &fingerprint)
- dng_camera_profile_id (const dng_string &name, const dng_fingerprint &fingerprint)
- const dng_string & Name () const
- const dng_fingerprint & Fingerprint () const
- bool operator== (const dng_camera_profile_id &id) const
- bool operator!= (const dng_camera_profile_id &id) const
- bool IsValid () const

  Returns true iff the camera profile ID is valid.
- void Clear ()

5.24.1 Detailed Description

An ID for a camera profile consisting of a name and optional fingerprint.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 dng_camera_profile_id() [1/4]

```
dng_camera_profile_id::dng_camera_profile_id (
             const char * name )  [inline]
```

Construct a camera profile ID with the specified name and no fingerprint.

Parameters

name	The name of the camera profile ID.

5.24.2.2 dng_camera_profile_id() [2/4]

```
dng_camera_profile_id::dng_camera_profile_id (
             const dng_string & name )  [inline]
```

Construct a camera profile ID with the specified name and no fingerprint.

Parameters

name	The name of the camera profile ID.

5.24.2.3 dng_camera_profile_id() [3/4]

```
dng_camera_profile_id::dng_camera_profile_id (
            const char * name,
            const dng_fingerprint & fingerprint )  [inline]
```

Construct a camera profile ID with the specified name and fingerprint.

Parameters

name	The name of the camera profile ID.
fingerprint	The fingerprint of the camera profile ID.

References DNG_ASSERT, and dng_fingerprint::IsValid().

5.24.2.4 dng_camera_profile_id() [4/4]

```
dng_camera_profile_id::dng_camera_profile_id (
            const dng_string & name,
            const dng_fingerprint & fingerprint )  [inline]
```

Construct a camera profile ID with the specified name and fingerprint.

Parameters

name	The name of the camera profile ID.
fingerprint	The fingerprint of the camera profile ID.

References DNG_ASSERT, and dng_fingerprint::IsValid().

5.24.3 Member Function Documentation

5.24.3.1 Clear()

```
void dng_camera_profile_id::Clear ( )  [inline]
```

Resets the name and fingerprint, thereby making this camera profile ID invalid.

References dng_camera_profile_id().

5.24.3.2 Fingerprint()

```
const dng_fingerprint& dng_camera_profile_id::Fingerprint ( ) const  [inline]
```

Getter for the fingerprint of the camera profile ID.

Return values

The	fingerprint of the camera profile ID.

5.24.3.3 Name()

```
const dng_string& dng_camera_profile_id::Name ( ) const  [inline]
```

Getter for the name of the camera profile ID.

Return values

The	name of the camera profile ID.

5.24.3.4 operator"!=()

```
bool dng_camera_profile_id::operator!= (
             const dng_camera_profile_id & id ) const  [inline]
```

Test for inequality of two camera profile IDs.

Parameters

id	The id of the camera profile ID to compare.

5.24.3.5 operator==()

```
bool dng_camera_profile_id::operator== (
             const dng_camera_profile_id & id ) const  [inline]
```

Test for equality of two camera profile IDs.

Parameters

id	The id of the camera profile ID to compare.

The documentation for this class was generated from the following file:

- dng_camera_profile.h

5.25 dng_camera_profile_info Class Reference

Public Member Functions

- bool **ParseTag** (dng_stream &stream, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)
- bool **ParseExtended** (dng_stream &stream)

Public Attributes

- bool **fBigEndian**
- uint32 **fColorPlanes**
- uint32 **fCalibrationIlluminant1**
- uint32 **fCalibrationIlluminant2**
- dng_matrix **fColorMatrix1**
- dng_matrix **fColorMatrix2**
- dng_matrix **fForwardMatrix1**
- dng_matrix **fForwardMatrix2**
- dng_matrix **fReductionMatrix1**
- dng_matrix **fReductionMatrix2**
- dng_string **fProfileCalibrationSignature**
- dng_string **fProfileName**
- dng_string **fProfileCopyright**
- uint32 **fEmbedPolicy**
- uint32 **fProfileHues**
- uint32 **fProfileSats**
- uint32 **fProfileVals**
- uint64 **fHueSatDeltas1Offset**
- uint32 **fHueSatDeltas1Count**
- uint64 **fHueSatDeltas2Offset**
- uint32 **fHueSatDeltas2Count**
- uint32 **fHueSatMapEncoding**
- uint32 **fLookTableHues**
- uint32 **fLookTableSats**
- uint32 **fLookTableVals**
- uint64 **fLookTableOffset**
- uint32 **fLookTableCount**
- uint32 **fLookTableEncoding**
- dng_srational **fBaselineExposureOffset**
- uint32 **fDefaultBlackRender**
- uint64 **fToneCurveOffset**
- uint32 **fToneCurveCount**
- dng_string **fUniqueCameraModel**

The documentation for this class was generated from the following files:

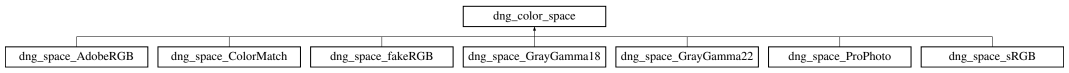
- dng_shared.h
- dng_shared.cpp

5.26 dng_color_space Class Reference

An abstract color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for dng_color_space:



Public Member Functions

- const dng_matrix & MatrixToPCS () const
- const dng_matrix & MatrixFromPCS () const
- bool IsMonochrome () const
- virtual const dng_1d_function & GammaFunction () const

  Getter for the gamma function for this color space.
- bool IsLinear () const

  Returns true if this color space is linear. (I.e. has gamma 1.0.)
- real64 GammaEncode (real64 x) const

  Map an input value through this color space's encoding gamma.
- real64 GammaDecode (real64 y) const
- virtual bool ICCProfile (uint32 &size, const uint8 $*$&data) const

Protected Member Functions

- void **SetMonochrome** ()
- void **SetMatrixToPCS** (const dng_matrix_3by3 &M)

Protected Attributes

- dng_matrix **fMatrixToPCS**
- dng_matrix **fMatrixFromPCS**

5.26.1 Detailed Description

An abstract color space.

5.26.2 Member Function Documentation

5.26.2.1 GammaDecode()

```
real64 dng_color_space::GammaDecode (
             real64 y ) const  [inline]
```

Map an input value through this color space's decoding gamma (inverse of the encoding gamma).

References dng_1d_function::EvaluateInverse(), and GammaFunction().

5.26.2.2 ICCProfile()

```
bool dng_color_space::ICCProfile (
             uint32 & size,
             const uint8 *& data ) const  [virtual]
```

Getter for ICC profile, if this color space has one.

Parameters

size	Out parameter which receives size on return.
data	Receives bytes of profile.

Return values

Returns	true if this color space has an ICC profile, false otherwise.

Reimplemented in dng_space_GrayGamma22, dng_space_GrayGamma18, dng_space_ProPhoto, dng_space_ColorMatch, dng_space_AdobeRGB, and dng_space_sRGB.

5.26.2.3 IsMonochrome()

```
bool dng_color_space::IsMonochrome ( ) const  [inline]
```

Predicate which is true if this color space is monochrome (has only a single column).

Referenced by dng_render::Render().

5.26.2.4 MatrixFromPCS()

```
const dng_matrix& dng_color_space::MatrixFromPCS ( ) const  [inline]
```

Return a matrix which transforms Profile Connection Space data into this color space.

5.26.2.5 MatrixToPCS()

```
const dng_matrix& dng_color_space::MatrixToPCS ( ) const  [inline]
```

Return a matrix which transforms source data in this color space into the Profile Connection Space.

The documentation for this class was generated from the following files:

- dng_color_space.h
- dng_color_space.cpp

5.27 dng_color_spec Class Reference

```
#include <dng_color_spec.h>
```

Public Member Functions

- dng_color_spec (const dng_negative &negative, const dng_camera_profile ∗profile)
- uint32 Channels () const
- void SetWhiteXY (const dng_xy_coord &white)
- const dng_xy_coord & WhiteXY () const
- const dng_vector & CameraWhite () const
- const dng_matrix & CameraToPCS () const
- const dng_matrix & PCStoCamera () const
- dng_xy_coord NeutralToXY (const dng_vector &neutral)

5.27.1 Detailed Description

Color transform taking into account white point and camera calibration and individual calibration from DNG negative.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 dng_color_spec()

```
dng_color_spec::dng_color_spec (
            const dng_negative & negative,
            const dng_camera_profile ∗ profile )
```

Read calibration info from DNG negative and construct a dng_color_spec.

References dng_negative::AnalogBalance(), dng_camera_profile::CalibrationTemperature1(), dng_camera_profile↩
::CalibrationTemperature2(), dng_negative::CameraCalibration1(), dng_negative::CameraCalibration2(), dng_↩
camera_profile::ColorMatrix1(), dng_camera_profile::ColorMatrix2(), dng_camera_profile::ForwardMatrix1(), dng↩
_camera_profile::ForwardMatrix2(), dng_camera_profile::HasColorMatrix2(), dng_camera_profile::IsValid(), dng_↩
camera_profile::NormalizeForwardMatrix(), dng_camera_profile::ProfileCalibrationSignature(), dng_camera_profile::↩
ReductionMatrix1(), dng_camera_profile::ReductionMatrix2(), ThrowBadFormat(), ThrowProgramError(), and dng_↩
camera_profile::WasStubbed().

5.27.3 Member Function Documentation

5.27.3.1 CameraToPCS()

```
const dng_matrix & dng_color_spec::CameraToPCS ( ) const
```

Getter for camera to Profile Connection Space color transform.

Return values

A	transform that takes into account all camera calibration transforms and white point.

References DNG_ASSERT.

5.27.3.2 CameraWhite()

```
const dng_vector & dng_color_spec::CameraWhite ( ) const
```

Return white point in camera native color coordinates.

Return values

A	dng_vector with components ranging from 0.0 to 1.0 that is normalized such that one component is equal to 1.0 .

References DNG_ASSERT.

5.27.3.3 Channels()

```
uint32 dng_color_spec::Channels ( ) const  [inline]
```

Number of channels used for this color transform. Three for most cameras.

5.27.3.4 NeutralToXY()

```
dng_xy_coord dng_color_spec::NeutralToXY (
             const dng_vector & neutral )
```

Return the XY value to use for SetWhiteXY for a given camera color space coordinate as the white point.

Parameters

neutral	A camera color space value to use for white point. Components range from 0.0 to 1.0 and should be normalized such that the largest value is 1.0 .

Return values

White	point in XY space that makes neutral map to this XY value as closely as possible.

5.27.3.5 PCStoCamera()

```
const dng_matrix & dng_color_spec::PCStoCamera ( ) const
```

Getter for Profile Connection Space to camera color transform.

Return values

A	transform that takes into account all camera calibration transforms and white point.

References DNG_ASSERT.

5.27.3.6 SetWhiteXY()

```
void dng_color_spec::SetWhiteXY (
             const dng_xy_coord & white )
```

Setter for white point. Value is as XY colorspace coordinate.

Parameters

white	White point to set as an XY value.

5.27.3.7 WhiteXY()

```
const dng_xy_coord & dng_color_spec::WhiteXY ( ) const
```

Getter for white point. Value is as XY colorspace coordinate.

Return values

XY	value of white point.

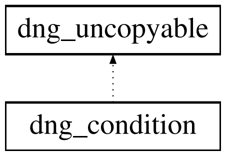
References DNG_ASSERT.

The documentation for this class was generated from the following files:

- dng_color_spec.h
- dng_color_spec.cpp

5.28 dng_condition Class Reference

Inheritance diagram for dng_condition:



Public Member Functions

- bool **Wait** (dng_mutex &mutex, double timeoutSecs=-1.0)
- void **Signal** ()
- void **Broadcast** ()

The documentation for this class was generated from the following files:

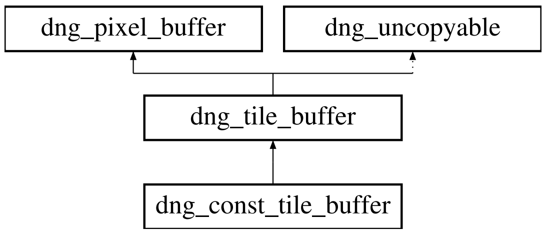
- dng_mutex.h
- dng_mutex.cpp

5.29 dng_const_tile_buffer Class Reference

Class to get resource acquisition is instantiation behavior for constant (read-only) tile buffers.

```
#include <dng_image.h>
```

Inheritance diagram for dng_const_tile_buffer:



Public Member Functions

- dng_const_tile_buffer (const dng_image &image, const dng_rect &tile)

Additional Inherited Members

5.29.1 Detailed Description

Class to get resource acquisition is instantiation behavior for constant (read-only) tile buffers.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 dng_const_tile_buffer()

```
dng_const_tile_buffer::dng_const_tile_buffer (
             const dng_image & image,
             const dng_rect & tile )
```

Obtain a read-only tile from an image.

Parameters

image	Image tile will come from.
tile	Rectangle denoting extent of tile.

The documentation for this class was generated from the following files:

- dng_image.h
- dng_image.cpp

5.30 dng_date_time Class Reference

Class for holding a date/time and converting to and from relevant date/time formats.

```
#include <dng_date_time.h>
```

Public Member Functions

- dng_date_time ()

  Construct an invalid date/time.
- dng_date_time (uint32 year, uint32 month, uint32 day, uint32 hour, uint32 minute, uint32 second)

- bool IsValid () const
- bool NotValid () const
- bool operator== (const dng_date_time &dt) const

  Equal operator.
- bool **operator!=** (const dng_date_time &dt) const
- void Clear ()

  Set date to an invalid value.
- bool Parse (const char ∗s)

Public Attributes

- uint32 **fYear**
- uint32 **fMonth**
- uint32 **fDay**
- uint32 **fHour**
- uint32 **fMinute**
- uint32 **fSecond**

5.30.1 Detailed Description

Class for holding a date/time and converting to and from relevant date/time formats.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 dng_date_time()

```
dng_date_time::dng_date_time (
             uint32 year,
             uint32 month,
             uint32 day,
             uint32 hour,
             uint32 minute,
             uint32 second )
```

Construct a date/time with specific values.

Parameters

year	Year to use as actual integer value, such as 2006.
month	Month to use from 1 - 12, where 1 is January.
day	Day of month to use from 1 -31, where 1 is the first.
hour	Hour of day to use from 0 - 23, where 0 is midnight.
minute	Minute of hour to use from 0 - 59.
second	Second of minute to use from 0 - 59.

5.30.3 Member Function Documentation

5.30.3.1 IsValid()

```
bool dng_date_time::IsValid ( ) const
```

Predicate to determine if a date is valid.

Return values

true	if all fields are within range.

Referenced by LocalTimeZone(), NotValid(), and Parse().

5.30.3.2 NotValid()

```
bool dng_date_time::NotValid ( ) const  [inline]
```

Predicate to determine if a date is invalid.

Return values

true	if any field is out of range.

References IsValid().

5.30.3.3 Parse()

```
bool dng_date_time::Parse (
             const char * s )
```

Parse an EXIF format date string.

Parameters

s	Input date string to parse.

Return values

true	if date was parsed successfully and date is valid.

References IsValid().

The documentation for this class was generated from the following files:

- dng_date_time.h
- dng_date_time.cpp

5.31 dng_date_time_info Class Reference

Class for holding complete data/time/zone information.

```
#include <dng_date_time.h>
```

Public Member Functions

- bool **IsValid** () const
- bool **NotValid** () const
- void **Clear** ()
- bool **IsDateOnly** () const
- const dng_date_time & **DateTime** () const
- void **SetDateTime** (const dng_date_time &dt)
- const dng_string & **Subseconds** () const
- void **SetSubseconds** (const dng_string &s)
- const dng_time_zone & **TimeZone** () const
- void **SetZone** (const dng_time_zone &zone)
- void **ClearZone** ()
- void **SetOffsetTime** (const dng_string &s)
- dng_string **OffsetTime** () const
- void **Decode_ISO_8601** (const char ∗s)
- dng_string **Encode_ISO_8601** () const
- void **Decode_IPTC_Date** (const char ∗s)
- dng_string **Encode_IPTC_Date** () const
- void **Decode_IPTC_Time** (const char ∗s)
- dng_string **Encode_IPTC_Time** () const

5.31.1 Detailed Description

Class for holding complete data/time/zone information.

The documentation for this class was generated from the following files:

- dng_date_time.h
- dng_date_time.cpp

5.32 dng_date_time_storage_info Class Reference

Store file offset from which date was read.

```
#include <dng_date_time.h>
```

Public Member Functions

- dng_date_time_storage_info ()

  The default constructor initializes to an invalid state.
- dng_date_time_storage_info (uint64 offset, dng_date_time_format format)

  Construct with file offset and date format.
- bool IsValid () const
- uint64 Offset () const
- dng_date_time_format Format () const

5.32.1 Detailed Description

Store file offset from which date was read.

Used internally by Adobe to update date in original file.

Warning

Use at your own risk.

5.32.2 Member Function Documentation

5.32.2.1 Format()

```
dng_date_time_format dng_date_time_storage_info::Format ( ) const
```

Get for format date was originally stored in file. Throws a dng_error_unknown exception if offset is invalid.

Exceptions

dng_exception	with fErrorCode equal to dng_error_unknown if offset is not valid.

References IsValid(), and ThrowProgramError().

5.32.2.2 IsValid()

```
bool dng_date_time_storage_info::IsValid ( ) const
```

Predicate to determine if an offset is valid.

Return values

true	if offset is valid.

Referenced by Format(), and Offset().

5.32.2.3 Offset()

```
uint64 dng_date_time_storage_info::Offset ( ) const
```

Getter for offset in file.

Exceptions

dng_exception	with fErrorCode equal to dng_error_unknown if offset is not valid.

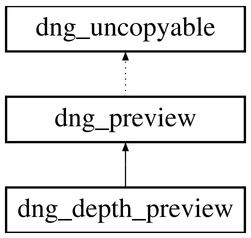
References IsValid(), and ThrowProgramError().

The documentation for this class was generated from the following files:

- dng_date_time.h
- dng_date_time.cpp

5.33 dng_depth_preview Class Reference

Inheritance diagram for dng_depth_preview:



Public Member Functions

- virtual dng_basic_tag_set ∗ **AddTagSet** (dng_tiff_directory &directory) const
- virtual void **WriteData** (dng_host &host, dng_image_writer &writer, dng_basic_tag_set &basic, dng_stream &stream) const

Public Attributes

- AutoPtr< dng_image > **fImage**
- int32 **fCompressionQuality**
- bool **fFullResolution**

The documentation for this class was generated from the following files:

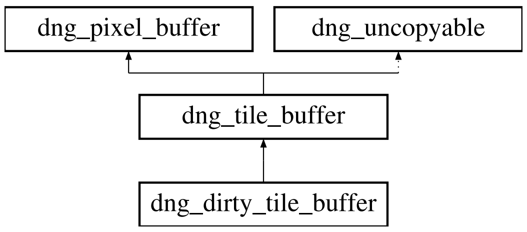
- dng_preview.h
- dng_preview.cpp

5.34 dng_dirty_tile_buffer Class Reference

Class to get resource acquisition is instantiation behavior for dirty (writable) tile buffers.

```
#include <dng_image.h>
```

Inheritance diagram for dng_dirty_tile_buffer:



Public Member Functions

- dng_dirty_tile_buffer (dng_image &image, const dng_rect &tile)

Additional Inherited Members

5.34.1 Detailed Description

Class to get resource acquisition is instantiation behavior for dirty (writable) tile buffers.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 dng_dirty_tile_buffer()

```
dng_dirty_tile_buffer::dng_dirty_tile_buffer (
                dng_image & image,
                const dng_rect & tile )
```

Obtain a writable tile from an image.

Parameters

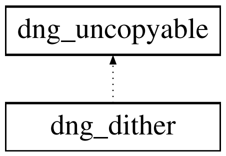
image	Image tile will come from.
tile	Rectangle denoting extent of tile.

The documentation for this class was generated from the following files:

- dng_image.h
- dng_image.cpp

5.35 dng_dither Class Reference

Inheritance diagram for dng_dither:



Public Member Functions

- const uint16 ∗ **NoiseBuffer16** () const

Static Public Member Functions

- static const dng_dither & **Get** ()

Static Public Attributes

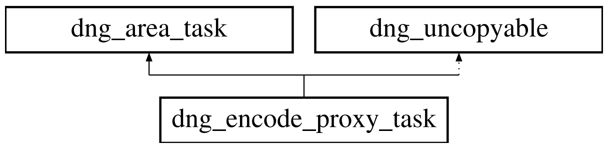
- static const uint32 **kRNGBits** = 7
- static const uint32 **kRNGSize** = 1 << kRNGBits
- static const uint32 **kRNGMask** = kRNGSize - 1
- static const uint32 **kRNGSize2D** = kRNGSize ∗ kRNGSize

The documentation for this class was generated from the following files:

- dng_utils.h
- dng_utils.cpp

5.36 dng_encode_proxy_task Class Reference

Inheritance diagram for dng_encode_proxy_task:



Public Member Functions

- **dng_encode_proxy_task** (dng_host &host, const dng_image &srcImage, dng_image &dstImage, const real64 ∗lower, const real64 ∗upper, bool isSceneReferred, real64 stage3BlackLevel, real64 ∗blackLevel)
- virtual dng_rect RepeatingTile1 () const
- virtual dng_rect RepeatingTile2 () const
- virtual void Process (uint32 threadIndex, const dng_rect &tile, dng_abort_sniffer ∗sniffer)

  and progress updates.

Additional Inherited Members

5.36.1 Member Function Documentation

5.36.1.1 Process()

```
void dng_encode_proxy_task::Process (
            uint32 threadIndex,
            const dng_rect & tile,
            dng_abort_sniffer * sniffer )  [virtual]
```

and progress updates.

Process one tile or fully partitioned area. This method is overridden by derived classes to implement the actual image processing. Note that the sniffer can be ignored if it is certain that a processing task will complete very quickly. This method should never be called directly but rather accessed via Process. There is no allocator parameter as all allocation should be done in Start.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the Start method.)
tile	Area to process.
sniffer	dng_abort_sniffer to use to check for user cancellation

Implements dng_area_task.

References dng_memory_block::Buffer_uint16(), dng_pixel_buffer::ConstPixel_uint16(), dng_pixel_buffer::DirtyPixel_↩ uint8(), and dng_image::Planes().

5.36.1.2 RepeatingTile1()

```
virtual dng_rect dng_encode_proxy_task::RepeatingTile1 ( ) const  [inline], [virtual]
```

Getter for RepeatingTile1. RepeatingTile1, RepeatingTile2, and RepeatingTile3 are used to establish a set of 0 to 3 tile patterns for which the resulting partitions that the final Process method is called on will not cross tile boundaries in any of the tile patterns. This can be used for a processing routine that needs to read from two tiles and write to a third such that all the tiles are aligned and sized in a certain way. A RepeatingTile value is valid if it is non-empty. Higher numbered RepeatingTile patterns are only used if all lower ones are non-empty. A RepeatingTile pattern must be a multiple of UnitCell in size for all constraints of the partitionerr to be met.

Reimplemented from dng_area_task.

References dng_image::RepeatingTile().

5.36.1.3 RepeatingTile2()

```
virtual dng_rect dng_encode_proxy_task::RepeatingTile2 ( ) const  [inline], [virtual]
```

Getter for RepeatingTile2. RepeatingTile1, RepeatingTile2, and RepeatingTile3 are used to establish a set of 0 to 3 tile patterns for which the resulting partitions that the final Process method is called on will not cross tile boundaries in any of the tile patterns. This can be used for a processing routine that needs to read from two tiles and write to a third such that all the tiles are aligned and sized in a certain way. A RepeatingTile value is valid if it is non-empty. Higher numbered RepeatingTile patterns are only used if all lower ones are non-empty. A RepeatingTile pattern must be a multiple of UnitCell in size for all constraints of the partitionerr to be met.

Reimplemented from dng_area_task.

References dng_image::RepeatingTile().

The documentation for this class was generated from the following file:

- dng_negative.cpp

5.37 dng_exception Class Reference

All exceptions thrown by the DNG SDK use this exception class.

```
#include <dng_exceptions.h>
```

Public Member Functions

- dng_exception (dng_error_code code)
- dng_error_code ErrorCode () const

5.37.1 Detailed Description

All exceptions thrown by the DNG SDK use this exception class.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 dng_exception()

```
dng_exception::dng_exception (
            dng_error_code code )  [inline]
```

Construct an exception representing the given error code.

Parameters

code	Error code this exception is for.

5.37.3 Member Function Documentation

5.37.3.1 ErrorCode()

```
dng_error_code dng_exception::ErrorCode ( ) const  [inline]
```

Getter for error code of this exception

Return values

The	error code of this exception.

The documentation for this class was generated from the following file:

- dng_exceptions.h

5.38 dng_exif Class Reference

Container class for parsing and holding EXIF tags.

```
#include <dng_exif.h>
```

Public Member Functions

- virtual dng_exif ∗ Clone () const

  Make clone.
- void SetEmpty ()

  Clear all EXIF fields.
- void CopyGPSFrom (const dng_exif &exif)
- void SetExposureTime (real64 et, bool snap=true)
- void SetShutterSpeedValue (real64 ss)
- void SetFNumber (real64 fs)
- void SetApertureValue (real64 av)
- void UpdateDateTime (const dng_date_time_info &dt)
- bool AtLeastVersion0230 () const

  Returns true iff the EXIF version is at least 2.3.
- bool AtLeastVersion0231 () const

  Returns true iff the EXIF version is at least 2.3.1.

- void SetVersion0231 ()

  Sets the EXIF version to 2.3.1.
- bool **HasLensDistortInfo** () const
- void **SetLensDistortInfo** (const dng_vector &params)
- virtual bool **ParseTag** (dng_stream &stream, dng_shared &shared, uint32 parentCode, bool isMainIFD, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)
- virtual void **PostParse** (dng_host &host, dng_shared &shared)

Static Public Member Functions

- static real64 SnapExposureTime (real64 et)
- static dng_urational EncodeFNumber (real64 fs)
- static real64 ApertureValueToFNumber (real64 av)
- static real64 ApertureValueToFNumber (const dng_urational &av)
- static real64 FNumberToApertureValue (real64 fNumber)
- static real64 FNumberToApertureValue (const dng_urational &fNumber)

Public Attributes

- dng_string **fImageDescription**
- dng_string **fMake**
- dng_string **fModel**
- dng_string **fSoftware**
- dng_string **fArtist**
- dng_string **fCopyright**
- dng_string **fCopyright2**
- dng_string **fUserComment**
- dng_date_time_info **fDateTime**
- dng_date_time_storage_info **fDateTimeStorageInfo**
- dng_date_time_info **fDateTimeOriginal**
- dng_date_time_storage_info **fDateTimeOriginalStorageInfo**
- dng_date_time_info **fDateTimeDigitized**
- dng_date_time_storage_info **fDateTimeDigitizedStorageInfo**
- uint32 **fTIFF_EP_StandardID**
- uint32 **fExifVersion**
- uint32 **fFlashPixVersion**
- dng_urational **fExposureTime**
- dng_urational **fFNumber**
- dng_srational **fShutterSpeedValue**
- dng_urational **fApertureValue**
- dng_srational **fBrightnessValue**
- dng_srational **fExposureBiasValue**
- dng_urational **fMaxApertureValue**
- dng_urational **fFocalLength**
- dng_urational **fDigitalZoomRatio**
- dng_urational **fExposureIndex**
- dng_urational **fSubjectDistance**
- dng_urational **fGamma**
- dng_urational **fBatteryLevelR**

- dng_string **fBatteryLevelA**
- uint32 **fExposureProgram**
- uint32 **fMeteringMode**
- uint32 **fLightSource**
- uint32 **fFlash**
- uint32 **fFlashMask**
- uint32 **fSensingMethod**
- uint32 **fColorSpace**
- uint32 **fFileSource**
- uint32 **fSceneType**
- uint32 **fCustomRendered**
- uint32 **fExposureMode**
- uint32 **fWhiteBalance**
- uint32 **fSceneCaptureType**
- uint32 **fGainControl**
- uint32 **fContrast**
- uint32 **fSaturation**
- uint32 **fSharpness**
- uint32 **fSubjectDistanceRange**
- uint32 **fSelfTimerMode**
- uint32 **fImageNumber**
- uint32 **fFocalLengthIn35mmFilm**
- uint32 **fISOSpeedRatings** [3]
- uint32 **fSensitivityType**
- uint32 **fStandardOutputSensitivity**
- uint32 **fRecommendedExposureIndex**
- uint32 **fISOSpeed**
- uint32 **fISOSpeedLatitudeyyy**
- uint32 **fISOSpeedLatitudezzz**
- uint32 **fSubjectAreaCount**
- uint32 **fSubjectArea** [4]
- uint32 **fComponentsConfiguration**
- dng_urational **fCompresssedBitsPerPixel**
- uint32 **fPixelXDimension**
- uint32 **fPixelYDimension**
- dng_urational **fFocalPlaneXResolution**
- dng_urational **fFocalPlaneYResolution**
- uint32 **fFocalPlaneResolutionUnit**
- uint32 **fCFARepeatPatternRows**
- uint32 **fCFARepeatPatternCols**
- uint8 **fCFAPattern** [kMaxCFAPattern][kMaxCFAPattern]
- dng_fingerprint **fImageUniqueID**
- uint32 **fGPSVersionID**
- dng_string **fGPSLatitudeRef**
- dng_urational **fGPSLatitude** [3]
- dng_string **fGPSLongitudeRef**
- dng_urational **fGPSLongitude** [3]
- uint32 **fGPSAltitudeRef**
- dng_urational **fGPSAltitude**
- dng_urational **fGPSTimeStamp** [3]
- dng_string **fGPSSatellites**

- dng_string **fGPSStatus**
- dng_string **fGPSMeasureMode**
- dng_urational **fGPSDOP**
- dng_string **fGPSSpeedRef**
- dng_urational **fGPSSpeed**
- dng_string **fGPSTrackRef**
- dng_urational **fGPSTrack**
- dng_string **fGPSImgDirectionRef**
- dng_urational **fGPSImgDirection**
- dng_string **fGPSMapDatum**
- dng_string **fGPSDestLatitudeRef**
- dng_urational **fGPSDestLatitude** [3]
- dng_string **fGPSDestLongitudeRef**
- dng_urational **fGPSDestLongitude** [3]
- dng_string **fGPSDestBearingRef**
- dng_urational **fGPSDestBearing**
- dng_string **fGPSDestDistanceRef**
- dng_urational **fGPSDestDistance**
- dng_string **fGPSProcessingMethod**
- dng_string **fGPSAreaInformation**
- dng_string **fGPSDateStamp**
- uint32 **fGPSDifferential**
- dng_urational **fGPSHPositioningError**
- dng_string **fInteroperabilityIndex**
- uint32 **fInteroperabilityVersion**
- dng_string **fRelatedImageFileFormat**
- uint32 **fRelatedImageWidth**
- uint32 **fRelatedImageLength**
- dng_string **fCameraSerialNumber**
- dng_urational **fLensInfo** [4]
- dng_string **fLensID**
- dng_string **fLensMake**
- dng_string **fLensName**
- dng_string **fLensSerialNumber**
- bool **fLensNameWasReadFromExif**
- dng_urational **fApproxFocusDistance**
- dng_srational **fFlashCompensation**
- dng_string **fOwnerName**
- dng_string **fFirmware**
- dng_srational **fTemperature**
- dng_urational **fHumidity**
- dng_urational **fPressure**
- dng_srational **fWaterDepth**
- dng_urational **fAcceleration**
- dng_srational **fCameraElevationAngle**
- dng_string **fTitle**
- dng_srational **fLensDistortInfo** [4]

Protected Member Functions

- virtual bool **Parse_ifd0** (dng_stream &stream, dng_shared &shared, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)
- virtual bool **Parse_ifd0_main** (dng_stream &stream, dng_shared &shared, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)
- virtual bool **Parse_ifd0_exif** (dng_stream &stream, dng_shared &shared, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)
- virtual bool **Parse_gps** (dng_stream &stream, dng_shared &shared, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)
- virtual bool **Parse_interoperability** (dng_stream &stream, dng_shared &shared, uint32 parentCode, uint32 tag↩ Code, uint32 tagType, uint32 tagCount, uint64 tagOffset)

5.38.1 Detailed Description

Container class for parsing and holding EXIF tags.

Public member fields are documented in EXIF specification.

5.38.2 Member Function Documentation

5.38.2.1 ApertureValueToFNumber() [1/2]

```
real64 dng_exif::ApertureValueToFNumber (
            real64 av ) [static]
```

Utility to convert aperture value (APEX units) to f-number.

Parameters

av	The aperture value (APEX units) to convert.

Referenced by ApertureValueToFNumber(), and SetApertureValue().

5.38.2.2 ApertureValueToFNumber() [2/2]

```
real64 dng_exif::ApertureValueToFNumber (
            const dng_urational & av ) [static]
```

Utility to convert aperture value (APEX units) to f-number.

Parameters

av	The aperture value (APEX units) to convert.

References ApertureValueToFNumber().

5.38.2.3 CopyGPSFrom()

```
void dng_exif::CopyGPSFrom (
             const dng_exif & exif )
```

Copy all GPS-related fields.

Parameters

exif	Source object from which to copy GPS fields.

5.38.2.4 EncodeFNumber()

```
dng_urational dng_exif::EncodeFNumber (
             real64 fs )  [static]
```

Utility to encode f-number as a rational.

Parameters

fs	The f-number to encode.

Referenced by SetFNumber().

5.38.2.5 FNumberToApertureValue() [1/2]

```
real64 dng_exif::FNumberToApertureValue (
             real64 fNumber )  [static]
```

Utility to convert f-number to aperture value (APEX units).

Parameters

fNumber	The f-number to convert.

Referenced by FNumberToApertureValue(), and SetFNumber().

5.38.2.6 FNumberToApertureValue() [2/2]

```
real64 dng_exif::FNumberToApertureValue (
            const dng_urational & fNumber )  [static]
```

Utility to convert f-number to aperture value (APEX units).

Parameters

fNumber	The f-number to convert.

References FNumberToApertureValue().

5.38.2.7 SetApertureValue()

```
void dng_exif::SetApertureValue (
            real64 av )
```

Set the FNumber and ApertureValue fields.

Parameters

av	The aperture value (APEX units).

References ApertureValueToFNumber(), and SetFNumber().

5.38.2.8 SetExposureTime()

```
void dng_exif::SetExposureTime (
            real64 et,
            bool snap = true )
```

Set exposure time and shutter speed fields. Optionally fix up common errors and rounding issues with EXIF exposure times.

Parameters

et	Exposure time in seconds.
snap	Set to true to fix up common errors and rounding issues with EXIF exposure times.

References SnapExposureTime().

Referenced by SetShutterSpeedValue().

5.38.2.9 SetFNumber()

```
void dng_exif::SetFNumber (
             real64 fs )
```

Set the FNumber and ApertureValue fields.

Parameters

fs	The f-number to set.

References EncodeFNumber(), and FNumberToApertureValue().

Referenced by SetApertureValue().

5.38.2.10 SetShutterSpeedValue()

```
void dng_exif::SetShutterSpeedValue (
             real64 ss )
```

Set shutter speed value (APEX units) and exposure time.

Parameters

ss	Shutter speed in APEX units.

References SetExposureTime().

5.38.2.11 SnapExposureTime()

```
real64 dng_exif::SnapExposureTime (
             real64 et )  [static]
```

Utility to fix up common errors and rounding issues with EXIF exposure times.

Referenced by SetExposureTime().

5.38.2.12 UpdateDateTime()

```
void dng_exif::UpdateDateTime (
             const dng_date_time_info & dt )
```

Set the DateTime field.

Parameters

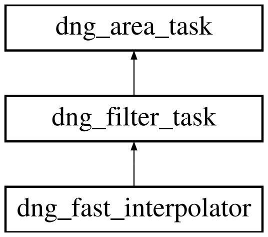
dt	The DateTime value.

The documentation for this class was generated from the following files:

- dng_exif.h
- dng_exif.cpp

5.39 dng_fast_interpolator Class Reference

Inheritance diagram for dng_fast_interpolator:



Public Member Functions

- **dng_fast_interpolator** (const dng_mosaic_info &info, const dng_image &srcImage, dng_image &dstImage, const dng_point &downScale, uint32 srcPlane)
- virtual dng_rect SrcArea (const dng_rect &dstArea)
- virtual void ProcessArea (uint32 threadIndex, dng_pixel_buffer &srcBuffer, dng_pixel_buffer &dstBuffer)

Protected Attributes

- const dng_mosaic_info & **fInfo**
- dng_point **fDownScale**
- uint32 **fFilterColor** [kMaxCFAPattern][kMaxCFAPattern]

Additional Inherited Members

5.39.1 Member Function Documentation

5.39.1.1 ProcessArea()

```
void dng_fast_interpolator::ProcessArea (
            uint32 threadIndex,
            dng_pixel_buffer & srcBuffer,
            dng_pixel_buffer & dstBuffer ) [virtual]
```

Implements filtering operation from one buffer to another. Source and destination pixels are set up in member fields of this class. Ideally, no allocation should be done in this routine.

Parameters

threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Start method.
srcBuffer	Input area and source pixels.
dstBuffer	Output area and destination pixels.

Implements dng_filter_task.

References dng_pixel_buffer::ConstPixel_uint16(), dng_pixel_buffer::DirtyPixel_uint16(), dng_mosaic_info::fCFA↩
PatternSize, dng_mosaic_info::fColorPlanes, and kMaxColorPlanes.

5.39.1.2 SrcArea()

```
dng_rect dng_fast_interpolator::SrcArea (
            const dng_rect & dstArea ) [virtual]
```

Compute the source area needed for a given destination area. Default implementation assumes destination area is equal to source area for all cases.

Parameters

dstArea	Area to for which pixels will be computed.

Return values

The	source area needed as input to calculate the requested destination area.

Reimplemented from dng_filter_task.

The documentation for this class was generated from the following file:

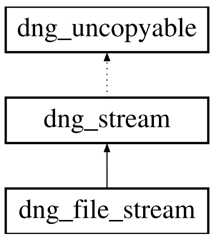
- dng_mosaic_info.cpp

5.40 dng_file_stream Class Reference

A stream to/from a disk file. See dng_stream for read/write interface.

```
#include <dng_file_stream.h>
```

Inheritance diagram for dng_file_stream:



Public Member Functions

- dng_file_stream (const char ∗filename, bool output=false, uint32 bufferSize=kDefaultBufferSize)

Protected Member Functions

- virtual uint64 **DoGetLength** ()
- virtual void **DoRead** (void ∗data, uint32 count, uint64 offset)
- virtual void **DoWrite** (const void ∗data, uint32 count, uint64 offset)

Additional Inherited Members

5.40.1 Detailed Description

A stream to/from a disk file. See dng_stream for read/write interface.

5.40.2 Constructor & Destructor Documentation

5.40.2.1 dng_file_stream()

```
dng_file_stream::dng_file_stream (
             const char * filename,
             bool output = false,
             uint32 bufferSize = kDefaultBufferSize )
```

Open a stream on a file.

Parameters

filename	Pathname in platform synax.
output	Set to true if writing, false otherwise.
bufferSize	size of internal buffer to use. Defaults to 4k.

References ReportError(), ThrowOpenFile(), and ThrowSilentError().

The documentation for this class was generated from the following files:

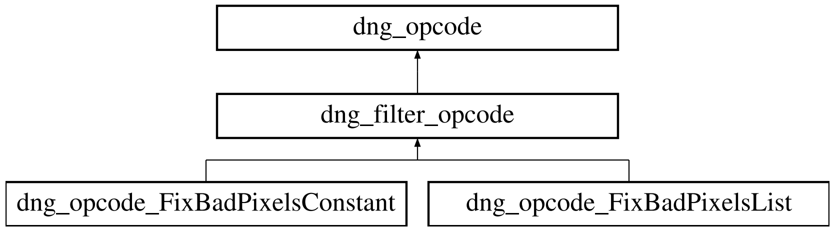
- dng_file_stream.h
- dng_file_stream.cpp

5.41 dng_filter_opcode Class Reference

Class to represent a filter opcode, such as a convolution.

```
#include <dng_opcodes.h>
```

Inheritance diagram for dng_filter_opcode:



Public Member Functions

- virtual uint32 BufferPixelType (uint32 imagePixelType)

  The pixel data type of this opcode.
- virtual dng_rect ModifiedBounds (const dng_rect &imageBounds)
- virtual dng_point SrcRepeat ()

  Returns the width and height (in pixels) of the repeating mosaic pattern.
- virtual dng_rect SrcArea (const dng_rect &dstArea, const dng_rect &imageBounds)
- virtual dng_point SrcTileSize (const dng_point &dstTileSize, const dng_rect &imageBounds)
- virtual void Prepare (dng_negative &negative, uint32 threadCount, const dng_point &tileSize, const dng_rect &imageBounds, uint32 imagePlanes, uint32 bufferPixelType, dng_memory_allocator &allocator)
- virtual void ProcessArea (dng_negative &negative, uint32 threadIndex, dng_pixel_buffer &srcBuffer, dng_pixel_buffer &dstBuffer, const dng_rect &dstArea, const dng_rect &imageBounds)=0
- virtual void Apply (dng_host &host, dng_negative &negative, AutoPtr< dng_image > &image)

  Apply this opcode to the specified image with associated negative.

Protected Member Functions

- **dng_filter_opcode** (uint32 opcodeID, uint32 minVersion, uint32 flags)
- **dng_filter_opcode** (uint32 opcodeID, dng_stream &stream, const char ∗name)

Additional Inherited Members

5.41.1 Detailed Description

Class to represent a filter opcode, such as a convolution.

5.41.2 Member Function Documentation

5.41.2.1 ModifiedBounds()

```
virtual dng_rect dng_filter_opcode::ModifiedBounds (
             const dng_rect & imageBounds )  [inline], [virtual]
```

The adjusted bounds (processing area) of this opcode. It is limited to the intersection of the specified image area and the GainMap area.

Referenced by Apply().

5.41.2.2 Prepare()

```
virtual void dng_filter_opcode::Prepare (
             dng_negative & negative,
             uint32 threadCount,
             const dng_point & tileSize,
             const dng_rect & imageBounds,
             uint32 imagePlanes,
             uint32 bufferPixelType,
             dng_memory_allocator & allocator )  [inline], [virtual]
```

Startup method called before any processing is performed on pixel areas. It can be used to allocate (per-thread) memory and setup tasks.

Parameters

negative	The negative object to be processed.
threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
imageBounds	Total size of image to be processed.
imagePlanes	Number of planes in the image. Less than or equal to kMaxColorPlanes.
bufferPixelType	Pixel type of image buffer (see dng_tag_types.h).
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.

Reimplemented in dng_opcode_FixBadPixelsList, and dng_opcode_FixBadPixelsConstant.

Referenced by dng_filter_opcode_task::Start().

5.41.2.3 ProcessArea()

```
virtual void dng_filter_opcode::ProcessArea (
             dng_negative & negative,
             uint32 threadIndex,
             dng_pixel_buffer & srcBuffer,
             dng_pixel_buffer & dstBuffer,
             const dng_rect & dstArea,
             const dng_rect & imageBounds )  [pure virtual]
```

Implements filtering operation from one buffer to another. Source and destination pixels are set up in member fields of this class. Ideally, no allocation should be done in this routine.

Parameters

negative	The negative associated with the pixels to be processed.
threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Prepare method.
srcBuffer	Input area and source pixels.
dstBuffer	Destination pixels.
dstArea	Destination pixel processing area.
imageBounds	Total image area to be processed; dstArea will always lie within these bounds.

Implemented in dng_opcode_FixBadPixelsList, and dng_opcode_FixBadPixelsConstant.

Referenced by dng_filter_opcode_task::ProcessArea().

5.41.2.4 SrcArea()

```
virtual dng_rect dng_filter_opcode::SrcArea (
            const dng_rect & dstArea,
            const dng_rect & imageBounds )  [inline], [virtual]
```

Returns the source pixel area needed to process a destination pixel area that lies within the specified bounds.

Parameters

dstArea	The destination pixel area to be computed.
imageBounds	The overall image area (dstArea will lie within these bounds).

Return values

The	source pixel area needed to process the specified dstArea.

Reimplemented in dng_opcode_FixBadPixelsList, and dng_opcode_FixBadPixelsConstant.

Referenced by dng_filter_opcode_task::SrcArea(), and SrcTileSize().

5.41.2.5 SrcTileSize()

```
virtual dng_point dng_filter_opcode::SrcTileSize (
            const dng_point & dstTileSize,
            const dng_rect & imageBounds )  [inline], [virtual]
```

Given a destination tile size, calculate input tile size. Simlar to SrcArea, and should seldom be overridden.

Parameters

dstTileSize	The destination tile size that is targeted for output.
imageBounds	The image bounds (the destination tile will always lie within these bounds).

Return values

The	source tile size needed to compute a tile of the destination size.

References SrcArea().

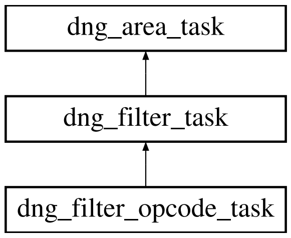
Referenced by dng_filter_opcode_task::SrcTileSize().

The documentation for this class was generated from the following files:

- dng_opcodes.h
- dng_opcodes.cpp

5.42 dng_filter_opcode_task Class Reference

Inheritance diagram for dng_filter_opcode_task:



Public Member Functions

- **dng_filter_opcode_task** (dng_filter_opcode &opcode, dng_negative &negative, const dng_image &srcImage, dng_image &dstImage)
- virtual dng_rect SrcArea (const dng_rect &dstArea)
- virtual dng_point SrcTileSize (const dng_point &dstTileSize)
- virtual void ProcessArea (uint32 threadIndex, dng_pixel_buffer &srcBuffer, dng_pixel_buffer &dstBuffer)
- virtual void Start (uint32 threadCount, const dng_rect &dstArea, const dng_point &tileSize, dng_memory_allocator ∗allocator, dng_abort_sniffer ∗sniffer)

Additional Inherited Members

5.42.1 Member Function Documentation

5.42.1.1 ProcessArea()

```
virtual void dng_filter_opcode_task::ProcessArea (
             uint32 threadIndex,
             dng_pixel_buffer & srcBuffer,
             dng_pixel_buffer & dstBuffer )  [inline], [virtual]
```

Implements filtering operation from one buffer to another. Source and destination pixels are set up in member fields of this class. Ideally, no allocation should be done in this routine.

Parameters

threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Start method.
srcBuffer	Input area and source pixels.
dstBuffer	Output area and destination pixels.

Implements dng_filter_task.

References dng_pixel_buffer::Area(), dng_image::Bounds(), and dng_filter_opcode::ProcessArea().

5.42.1.2 SrcArea()

```
virtual dng_rect dng_filter_opcode_task::SrcArea (
             const dng_rect & dstArea )  [inline], [virtual]
```

Compute the source area needed for a given destination area. Default implementation assumes destination area is equal to source area for all cases.

Parameters

dstArea	Area to for which pixels will be computed.

Return values

The	source area needed as input to calculate the requested destination area.

Reimplemented from dng_filter_task.

References dng_image::Bounds(), and dng_filter_opcode::SrcArea().

5.42.1.3 SrcTileSize()

```
virtual dng_point dng_filter_opcode_task::SrcTileSize (
             const dng_point & dstTileSize )  [inline], [virtual]
```

Given a destination tile size, calculate input tile size. Simlar to SrcArea, and should seldom be overridden.

Parameters

dstTileSize	The destination tile size that is targeted for output.

Return values

The	source tile size needed to compute a tile of the destination size.

Reimplemented from dng_filter_task.

References dng_image::Bounds(), and dng_filter_opcode::SrcTileSize().

5.42.1.4 Start()

```
virtual void dng_filter_opcode_task::Start (
             uint32 threadCount,
             const dng_rect & dstArea,
             const dng_point & tileSize,
             dng_memory_allocator * allocator,
             dng_abort_sniffer * sniffer )  [inline], [virtual]
```

Called prior to processing on specific threads. Can be used to allocate per-thread memory buffers, etc.

Parameters

threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads of dng_area_task.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.
sniffer	Sniffer to test for user cancellation and to set up progress.

Reimplemented from dng_filter_task.

References dng_image::Bounds(), dng_image::Planes(), dng_filter_opcode::Prepare(), and dng_filter_task::Start().

The documentation for this class was generated from the following file:

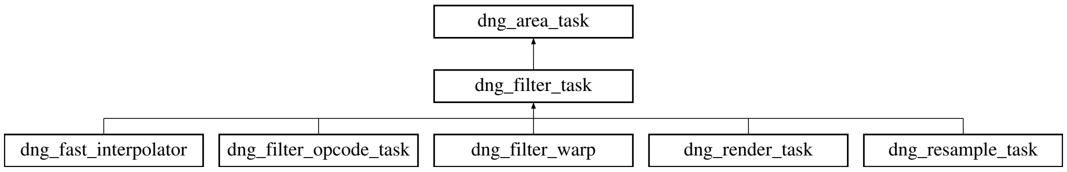
- dng_opcodes.cpp

5.43 dng_filter_task Class Reference

Represents a task which filters an area of a source dng_image to an area of a destination dng_image.

```
#include <dng_filter_task.h>
```

Inheritance diagram for dng_filter_task:



Public Member Functions

- dng_filter_task (const char ∗name, const dng_image &srcImage, dng_image &dstImage)
- virtual dng_rect SrcArea (const dng_rect &dstArea)
- virtual dng_point SrcTileSize (const dng_point &dstTileSize)
- virtual void ProcessArea (uint32 threadIndex, dng_pixel_buffer &srcBuffer, dng_pixel_buffer &dstBuffer)=0
- virtual void Start (uint32 threadCount, const dng_rect &dstArea, const dng_point &tileSize, dng_memory_allocator ∗allocator, dng_abort_sniffer ∗sniffer)
- virtual void Process (uint32 threadIndex, const dng_rect &area, dng_abort_sniffer ∗sniffer)

Protected Attributes

- const dng_image & **fSrcImage**
- dng_image & **fDstImage**
- uint32 **fSrcPlane**
- uint32 **fSrcPlanes**
- uint32 **fSrcPixelType**
- uint32 **fDstPlane**
- uint32 **fDstPlanes**
- uint32 **fDstPixelType**
- dng_point **fSrcRepeat**
- dng_point **fSrcTileSize**
- AutoPtr< dng_memory_block > **fSrcBuffer** [kMaxMPThreads]
- AutoPtr< dng_memory_block > **fDstBuffer** [kMaxMPThreads]

Additional Inherited Members

5.43.1 Detailed Description

Represents a task which filters an area of a source dng_image to an area of a destination dng_image.

5.43.2 Constructor & Destructor Documentation

5.43.2.1 dng_filter_task()

```
dng_filter_task::dng_filter_task (
              const char * name,
              const dng_image & srcImage,
              dng_image & dstImage )
```

Construct a filter task given a source and destination images.

Parameters

srcImage	Image from which source pixels are read.
dstImage	Image to which result pixels are written.

5.43.3 Member Function Documentation

5.43.3.1 Process()

```
void dng_filter_task::Process (
             uint32 threadIndex,
             const dng_rect & area,
             dng_abort_sniffer * sniffer )  [virtual]
```

Process one tile or partitioned area. Should not be overridden. Instead, override ProcessArea, which is where to implement filter processing for a specific type of dng_filter_task. There is no allocator parameter as all allocation should be done in Start.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the Start method.)
area	Size of tiles to be used for sizing buffers, etc. (Edges of processing can be smaller.)
sniffer	dng_abort_sniffer to use to check for user cancellation and progress updates.

Implements dng_area_task.

References dng_image::edge_repeat, dng_image::Get(), ProcessArea(), dng_image::Put(), SrcArea(), and Throw←MemoryFull().

5.43.3.2 ProcessArea()

```
virtual void dng_filter_task::ProcessArea (
             uint32 threadIndex,
             dng_pixel_buffer & srcBuffer,
             dng_pixel_buffer & dstBuffer )  [pure virtual]
```

Implements filtering operation from one buffer to another. Source and destination pixels are set up in member fields of this class. Ideally, no allocation should be done in this routine.

Parameters

threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Start method.
srcBuffer	Input area and source pixels.
dstBuffer	Output area and destination pixels.

Implemented in dng_fast_interpolator, dng_filter_warp, dng_render_task, dng_resample_task, and dng_filter_opcode_task.

Referenced by Process().

5.43.3.3 SrcArea()

```
virtual dng_rect dng_filter_task::SrcArea (
             const dng_rect & dstArea )  [inline], [virtual]
```

Compute the source area needed for a given destination area. Default implementation assumes destination area is equal to source area for all cases.

Parameters

dstArea	Area to for which pixels will be computed.

Return values

The	source area needed as input to calculate the requested destination area.

Reimplemented in dng_fast_interpolator, dng_filter_warp, dng_render_task, dng_resample_task, and dng_filter_opcode_task.

Referenced by Process(), and SrcTileSize().

5.43.3.4 SrcTileSize()

```
virtual dng_point dng_filter_task::SrcTileSize (
             const dng_point & dstTileSize )  [inline], [virtual]
```

Given a destination tile size, calculate input tile size. Simlar to SrcArea, and should seldom be overridden.

Parameters

dstTileSize	The destination tile size that is targeted for output.

Return values

The	source tile size needed to compute a tile of the destination size.

Reimplemented in dng_filter_warp, dng_resample_task, and dng_filter_opcode_task.

References SrcArea().

Referenced by Start().

5.43.3.5 Start()

```
void dng_filter_task::Start (
            uint32 threadCount,
            const dng_rect & dstArea,
            const dng_point & tileSize,
            dng_memory_allocator * allocator,
            dng_abort_sniffer * sniffer )  [virtual]
```

Called prior to processing on specific threads. Can be used to allocate per-thread memory buffers, etc.

Parameters

threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads of dng_area_task.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.
sniffer	Sniffer to test for user cancellation and to set up progress.

Reimplemented from dng_area_task.

Reimplemented in dng_render_task, dng_resample_task, and dng_filter_opcode_task.

References dng_memory_allocator::Allocate(), and SrcTileSize().

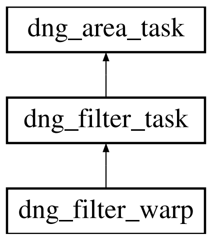
Referenced by dng_filter_opcode_task::Start(), dng_resample_task::Start(), and dng_render_task::Start().

The documentation for this class was generated from the following files:

- dng_filter_task.h
- dng_filter_task.cpp

5.44 dng_filter_warp Class Reference

Inheritance diagram for dng_filter_warp:



Public Member Functions

- **dng_filter_warp** (const dng_image &srcImage, dng_image &dstImage, const dng_negative &negative, AutoPtr< dng_warp_params > &params)
- virtual void **Initialize** (dng_host &host)
- virtual dng_rect SrcArea (const dng_rect &dstArea)
- virtual dng_point SrcTileSize (const dng_point &dstTileSize)
- virtual void ProcessArea (uint32 threadIndex, dng_pixel_buffer &srcBuffer, dng_pixel_buffer &dstBuffer)
- virtual dng_point_real64 **GetSrcPixelPosition** (const dng_point_real64 &dst, uint32 plane)

Protected Attributes

- AutoPtr< dng_warp_params > **fParams**
- dng_point_real64 **fCenter**
- dng_resample_weights_2d **fWeights**
- real64 **fNormRadius**
- real64 **fInvNormRadius**
- bool **fIsRadNOP**
- bool **fIsTanNOP**
- const real64 **fPixelScaleV**
- const real64 **fPixelScaleVInv**

Additional Inherited Members

5.44.1 Member Function Documentation

5.44.1.1 ProcessArea()

```
void dng_filter_warp::ProcessArea (
            uint32 threadIndex,
            dng_pixel_buffer & srcBuffer,
            dng_pixel_buffer & dstBuffer )  [virtual]
```

Implements filtering operation from one buffer to another. Source and destination pixels are set up in member fields of this class. Ideally, no allocation should be done in this routine.

Parameters

threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Start method.
srcBuffer	Input area and source pixels.
dstBuffer	Output area and destination pixels.

Implements dng_filter_task.

References dng_image::Bounds(), dng_pixel_buffer::ConstPixel_real32(), dng_pixel_buffer::DirtyPixel_real32(), dng_↩
pixel_buffer::RowStep(), and ThrowBadFormat().

5.44.1.2 SrcArea()

```
dng_rect dng_filter_warp::SrcArea (
             const dng_rect & dstArea )  [virtual]
```

Compute the source area needed for a given destination area. Default implementation assumes destination area is equal to source area for all cases.

Parameters

dstArea	Area to for which pixels will be computed.

Return values

The	source area needed as input to calculate the requested destination area.

Reimplemented from dng_filter_task.

References dng_image::Bounds().

Referenced by SrcTileSize().

5.44.1.3 SrcTileSize()

```
dng_point dng_filter_warp::SrcTileSize (
             const dng_point & dstTileSize )  [virtual]
```

Given a destination tile size, calculate input tile size. Simlar to SrcArea, and should seldom be overridden.

Parameters

dstTileSize	The destination tile size that is targeted for output.

Return values

The	source tile size needed to compute a tile of the destination size.

Reimplemented from dng_filter_task.

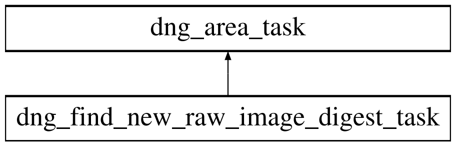
References dng_image::Bounds(), DNG_REQUIRE, dng_warp_params::MaxSrcRadiusGap(), dng_warp_params::↩
MaxSrcTanGap(), and SrcArea().

The documentation for this class was generated from the following file:

- dng_lens_correction.cpp

5.45 dng_find_new_raw_image_digest_task Class Reference

Inheritance diagram for dng_find_new_raw_image_digest_task:



Public Member Functions

- **dng_find_new_raw_image_digest_task** (const dng_image &image, uint32 pixelType)
- virtual void Start (uint32 threadCount, const dng_rect &, const dng_point &tileSize, dng_memory_allocator ∗allocator, dng_abort_sniffer ∗)
- virtual void Process (uint32 threadIndex, const dng_rect &tile, dng_abort_sniffer ∗)

  and progress updates.
- dng_fingerprint **Result** ()

Additional Inherited Members

5.45.1 Member Function Documentation

5.45.1.1 Process()

```
virtual void dng_find_new_raw_image_digest_task::Process (
             uint32 threadIndex,
             const dng_rect & tile,
             dng_abort_sniffer * sniffer )  [inline], [virtual]
```

and progress updates.

Process one tile or fully partitioned area. This method is overridden by derived classes to implement the actual image processing. Note that the sniffer can be ignored if it is certain that a processing task will complete very quickly. This method should never be called directly but rather accessed via Process. There is no allocator parameter as all allocation should be done in Start.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the Start method.)
tile	Area to process.
sniffer	dng_abort_sniffer to use to check for user cancellation

Implements dng_area_task.

References dng_image::Bounds(), dng_memory_block::Buffer(), DNG_ASSERT, dng_image::Get(), and dng_image::↩
Planes().

5.45.1.2 Start()

```
virtual void dng_find_new_raw_image_digest_task::Start (
             uint32 threadCount,
             const dng_rect & dstArea,
             const dng_point & tileSize,
             dng_memory_allocator * allocator,
             dng_abort_sniffer * sniffer )  [inline], [virtual]
```

Task startup method called before any processing is done on partitions. The Start method is called before any processing is done and can be overridden to allocate temporary buffers, etc.

Parameters

threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads.
dstArea	Area to be processed in the current run of the task.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.
sniffer	Sniffer to test for user cancellation and to set up progress.

Reimplemented from dng_area_task.

References dng_memory_allocator::Allocate(), dng_image::Bounds(), dng_image::Planes(), AutoPtr$<$ T $>$::Reset(), AutoArray$<$ T $>$::Reset(), and ThrowProgramError().

The documentation for this class was generated from the following file:

- dng_negative.cpp

5.46 dng_fingerprint Class Reference

Container fingerprint (MD5 only at present).

```
#include <dng_fingerprint.h>
```

Public Member Functions

- **dng_fingerprint** (const char ∗hex)
- bool IsNull () const

  Check if fingerprint is all zeros.
- bool IsValid () const

  Same as IsNull but expresses intention of testing validity.
- void Clear ()

  Set to all zeros, a value used to indicate an invalid fingerprint.
- bool operator== (const dng_fingerprint &print) const

  Test if two fingerprints are equal.
- bool operator!= (const dng_fingerprint &print) const

  Test if two fingerprints are not equal.
- bool operator< (const dng_fingerprint &print) const

  Comparision test for fingerprints.
- uint32 Collapse32 () const
- void ToUtf8HexString (char resultStr [2 ∗kDNGFingerprintSize+1]) const
- bool FromUtf8HexString (const char inputStr [2 ∗kDNGFingerprintSize+1])

Public Attributes

- uint8 **data** [kDNGFingerprintSize]

Static Public Attributes

- static const size_t **kDNGFingerprintSize** = 16

5.46.1 Detailed Description

Container fingerprint (MD5 only at present).

5.46.2 Member Function Documentation

5.46.2.1 Collapse32()

```
uint32 dng_fingerprint::Collapse32 ( ) const
```

Produce a 32-bit hash value from fingerprint used for faster hashing of fingerprints.

Referenced by dng_fingerprint_hash::operator()().

5.46.2.2 FromUtf8HexString()

```
bool dng_fingerprint::FromUtf8HexString (
             const char inputStr[2 *kDNGFingerprintSize+1] )
```

Convert UTF-8 string to fingerprint. Returns true on success, false on failure.

Parameters

inputStr	The input array from which the UTF-8 encoding of the fingerprint will be read.

Return values

True	indicates success.

5.46.2.3 ToUtf8HexString()

```
void dng_fingerprint::ToUtf8HexString (
             char resultStr[2 *kDNGFingerprintSize+1] ) const
```

Convert fingerprint to UTF-8 string.

Parameters

resultStr	The output array to which the UTF-8 encoding of the fingerprint will be written.

The documentation for this class was generated from the following files:

- dng_fingerprint.h
- dng_fingerprint.cpp

5.47 dng_fingerprint_hash Struct Reference

Utility to hash fingerprints (e.g., for hashtables).

```
#include <dng_fingerprint.h>
```

Public Member Functions

- size_t operator() (const dng_fingerprint &digest) const

    Hash function.

5.47.1 Detailed Description

Utility to hash fingerprints (e.g., for hashtables).

The documentation for this struct was generated from the following file:

- dng_fingerprint.h

5.48 dng_fingerprint_less_than Struct Reference

Utility to compare fingerprints (e.g., for sorting).

```
#include <dng_fingerprint.h>
```

Public Member Functions

- bool operator() (const dng_fingerprint &a, const dng_fingerprint &b) const
  Less-than comparison.

5.48.1 Detailed Description

Utility to compare fingerprints (e.g., for sorting).

The documentation for this struct was generated from the following file:

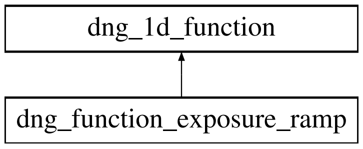
- dng_fingerprint.h

5.49 dng_function_exposure_ramp Class Reference

Curve for pre-exposure-compensation adjustment based on noise floor, shadows, and highlight level.

```
#include <dng_render.h>
```

Inheritance diagram for dng_function_exposure_ramp:



Public Member Functions

- **dng_function_exposure_ramp** (real64 white, real64 black, real64 minBlack)
- virtual real64 Evaluate (real64 x) const

Public Attributes

- real64 **fSlope**
- real64 **fBlack**
- real64 **fRadius**
- real64 **fQScale**

5.49.1 Detailed Description

Curve for pre-exposure-compensation adjustment based on noise floor, shadows, and highlight level.

5.49.2 Member Function Documentation

5.49.2.1 Evaluate()

```
real64 dng_function_exposure_ramp::Evaluate (
            real64 x ) const  [virtual]
```

Return the mapping for value x. This method must be implemented by a derived class of dng_1d_function and the derived class determines the lookup method and function used.

Parameters

x	A value between 0.0 and 1.0 (inclusive).

Return values

Mapped	value for x

Implements dng_1d_function.

The documentation for this class was generated from the following files:

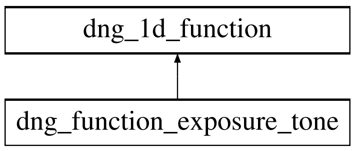
- dng_render.h
- dng_render.cpp

5.50 dng_function_exposure_tone Class Reference

Exposure compensation curve for a given compensation amount in stops using quadric for roll-off.

```
#include <dng_render.h>
```

Inheritance diagram for dng_function_exposure_tone:



Public Member Functions

- **dng_function_exposure_tone** (real64 exposure)
- virtual real64 Evaluate (real64 x) const

  Returns output value for a given input tone.

Protected Attributes

- bool **fIsNOP**
- real64 **fSlope**
- real64 **a**
- real64 **b**
- real64 **c**

5.50.1 Detailed Description

Exposure compensation curve for a given compensation amount in stops using quadric for roll-off.

The documentation for this class was generated from the following files:

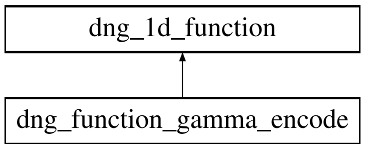
- dng_render.h
- dng_render.cpp

5.51 dng_function_gamma_encode Class Reference

Encoding gamma curve for a given color space.

```
#include <dng_render.h>
```

Inheritance diagram for dng_function_gamma_encode:



Public Member Functions

- **dng_function_gamma_encode** (const dng_color_space &space)
- virtual real64 Evaluate (real64 x) const

Protected Attributes

- const dng_color_space & **fSpace**

5.51.1 Detailed Description

Encoding gamma curve for a given color space.

5.51.2 Member Function Documentation

5.51.2.1 Evaluate()

```
virtual real64 dng_function_gamma_encode::Evaluate (
             real64 x ) const  [virtual]
```

Return the mapping for value x. This method must be implemented by a derived class of dng_1d_function and the derived class determines the lookup method and function used.

Parameters

x	A value between 0.0 and 1.0 (inclusive).

Return values

Mapped	value for x

Implements dng_1d_function.

The documentation for this class was generated from the following file:

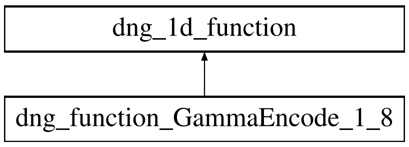
- dng_render.h

5.52 dng_function_GammaEncode_1_8 Class Reference

A dng_1d_function for gamma encoding with 1.8 gamma.

```
#include <dng_color_space.h>
```

Inheritance diagram for dng_function_GammaEncode_1_8:



Public Member Functions

- virtual real64 Evaluate (real64 x) const
- virtual real64 EvaluateInverse (real64 y) const

Static Public Member Functions

- static const dng_1d_function & **Get** ()

5.52.1 Detailed Description

A dng_1d_function for gamma encoding with 1.8 gamma.

5.52.2 Member Function Documentation

5.52.2.1 Evaluate()

```
real64 dng_function_GammaEncode_1_8::Evaluate (
             real64 x ) const  [virtual]
```

Return the mapping for value x. This method must be implemented by a derived class of dng_1d_function and the derived class determines the lookup method and function used.

Parameters

x	A value between 0.0 and 1.0 (inclusive).

Return values

Mapped	value for x

Implements dng_1d_function.

5.52.2.2 EvaluateInverse()

```
real64 dng_function_GammaEncode_1_8::EvaluateInverse (
             real64 y ) const  [virtual]
```

Return the reverse mapped value for y. This method can be implemented by derived classes. The default implementation uses Newton's method to solve for x such that Evaluate(x) == y.

Parameters

y	A value to reverse map. Should be within the range of the function implemented by this dng_1d_function .

Return values

A	value x such that Evaluate(x) == y (to very close approximation).

Reimplemented from dng_1d_function.

References dng_1d_function::EvaluateInverse().

The documentation for this class was generated from the following files:

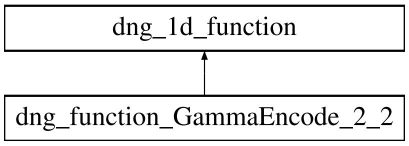
- dng_color_space.h
- dng_color_space.cpp

5.53 dng_function_GammaEncode_2_2 Class Reference

A dng_1d_function for gamma encoding with 2.2 gamma.

```
#include <dng_color_space.h>
```

Inheritance diagram for dng_function_GammaEncode_2_2:



Public Member Functions

- virtual real64 Evaluate (real64 x) const
- virtual real64 EvaluateInverse (real64 y) const

Static Public Member Functions

- static const dng_1d_function & **Get** ()

5.53.1 Detailed Description

A dng_1d_function for gamma encoding with 2.2 gamma.

5.53.2 Member Function Documentation

5.53.2.1 Evaluate()

```
real64 dng_function_GammaEncode_2_2::Evaluate (
            real64 x ) const  [virtual]
```

Return the mapping for value x. This method must be implemented by a derived class of dng_1d_function and the derived class determines the lookup method and function used.

Parameters

x	A value between 0.0 and 1.0 (inclusive).

Return values

Mapped	value for x

Implements dng_1d_function.

5.53.2.2 EvaluateInverse()

```
real64 dng_function_GammaEncode_2_2::EvaluateInverse (
            real64 y ) const [virtual]
```

Return the reverse mapped value for y. This method can be implemented by derived classes. The default implementation uses Newton's method to solve for x such that Evaluate(x) == y.

Parameters

y	A value to reverse map. Should be within the range of the function implemented by this dng_1d_function .

Return values

A	value x such that Evaluate(x) == y (to very close approximation).

Reimplemented from dng_1d_function.

References dng_1d_function::EvaluateInverse().

The documentation for this class was generated from the following files:

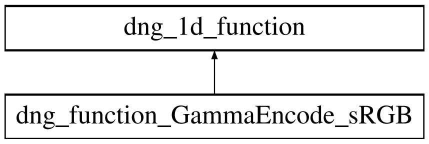
- dng_color_space.h
- dng_color_space.cpp

5.54 dng_function_GammaEncode_sRGB Class Reference

A dng_1d_function for gamma encoding in sRGB color space.

`#include <dng_color_space.h>`

Inheritance diagram for dng_function_GammaEncode_sRGB:



Public Member Functions

- virtual real64 Evaluate (real64 x) const
- virtual real64 EvaluateInverse (real64 y) const

Static Public Member Functions

- static const dng_1d_function & **Get** ()

5.54.1 Detailed Description

A dng_1d_function for gamma encoding in sRGB color space.

5.54.2 Member Function Documentation

5.54.2.1 Evaluate()

```
real64 dng_function_GammaEncode_sRGB::Evaluate (
             real64 x ) const  [virtual]
```

Return the mapping for value x. This method must be implemented by a derived class of dng_1d_function and the derived class determines the lookup method and function used.

Parameters

x	A value between 0.0 and 1.0 (inclusive).

Return values

Mapped	value for x

Implements dng_1d_function.

5.54.2.2 EvaluateInverse()

```
real64 dng_function_GammaEncode_sRGB::EvaluateInverse (
             real64 y ) const  [virtual]
```

Return the reverse mapped value for y. This method can be implemented by derived classes. The default implementation uses Newton's method to solve for x such that Evaluate(x) == y.

Parameters

y	A value to reverse map. Should be within the range of the function implemented by this dng_1d_function .

Return values

A	value x such that Evaluate(x) == y (to very close approximation).

Reimplemented from dng_1d_function.

The documentation for this class was generated from the following files:

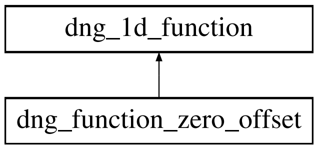
- dng_color_space.h
- dng_color_space.cpp

5.55 dng_function_zero_offset Class Reference

Curve for removing zero offset from stage3 image.

```
#include <dng_render.h>
```

Inheritance diagram for dng_function_zero_offset:



Public Member Functions

- **dng_function_zero_offset** (real64 zeroOffset)
- virtual real64 Evaluate (real64 x) const

Public Attributes

- real64 **fZeroOffset**
- real64 **fScale**

5.55.1 Detailed Description

Curve for removing zero offset from stage3 image.

5.55.2 Member Function Documentation

5.55.2.1 Evaluate()

```
real64 dng_function_zero_offset::Evaluate (
             real64 x ) const  [virtual]
```

Return the mapping for value x. This method must be implemented by a derived class of dng_1d_function and the derived class determines the lookup method and function used.

Parameters

x	A value between 0.0 and 1.0 (inclusive).

Return values

Mapped	value for x

Implements dng_1d_function.

The documentation for this class was generated from the following files:

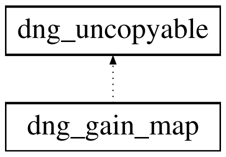
- dng_render.h
- dng_render.cpp

5.56 dng_gain_map Class Reference

Holds a discrete (i.e., sampled) 2D representation of a gain map. This is effectively an image containing scale factors.

```
#include <dng_gain_map.h>
```

Inheritance diagram for dng_gain_map:



Public Member Functions

- dng_gain_map (dng_memory_allocator &allocator, const dng_point &points, const dng_point_real64 &spacing, const dng_point_real64 &origin, uint32 planes)
- const dng_point & Points () const

  The number of samples in the horizontal and vertical directions.
- const dng_point_real64 & Spacing () const
- const dng_point_real64 & Origin () const

  The 2D coordinate for the first (i.e., top-left-most) sample.
- uint32 Planes () const

  The number of color planes.
- real32 & Entry (uint32 rowIndex, uint32 colIndex, uint32 plane)

  Getter for a gain map sample (specified by row, column, and plane).
- const real32 & Entry (uint32 rowIndex, uint32 colIndex, uint32 plane) const
- real32 Interpolate (int32 row, int32 col, uint32 plane, const dng_rect &bounds) const
- uint32 PutStreamSize () const

  The number of bytes needed to hold the gain map data.
- void PutStream (dng_stream &stream) const

  Write the gain map to the specified stream.

Static Public Member Functions

- static dng_gain_map ∗ GetStream (dng_host &host, dng_stream &stream)

  Read a gain map from the specified stream.

5.56.1 Detailed Description

Holds a discrete (i.e., sampled) 2D representation of a gain map. This is effectively an image containing scale factors.

5.56.2 Constructor & Destructor Documentation

5.56.2.1 dng_gain_map()

```
dng_gain_map::dng_gain_map (
             dng_memory_allocator & allocator,
             const dng_point & points,
             const dng_point_real64 & spacing,
             const dng_point_real64 & origin,
             uint32 planes )
```

Construct a gain map with the specified memory allocator, number of samples (points), sample spacing, origin, and number of color planes.

5.56.3 Member Function Documentation

5.56.3.1 Entry()

```
const real32& dng_gain_map::Entry (
             uint32 rowIndex,
             uint32 colIndex,
             uint32 plane ) const  [inline]
```

Getter for a gain map sample (specified by row index, column index, and plane index).

References dng_memory_block::Buffer_real32().

5.56.3.2 Interpolate()

```
real32 dng_gain_map::Interpolate (
            int32 row,
            int32 col,
            uint32 plane,
            const dng_rect & bounds ) const
```

Compute the interpolated gain (i.e., scale factor) at the specified pixel position and color plane, within the specified image bounds (in pixels).

5.56.3.3 Spacing()

```
const dng_point_real64& dng_gain_map::Spacing ( ) const  [inline]
```

The space between adjacent samples in the horizontal and vertical directions.

The documentation for this class was generated from the following files:

- dng_gain_map.h
- dng_gain_map.cpp

5.57 dng_gain_map_interpolator Class Reference

Public Member Functions

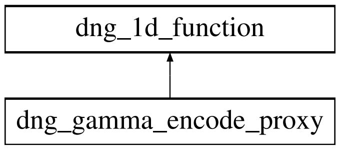
- **dng_gain_map_interpolator** (const dng_gain_map &map, const dng_rect &mapBounds, int32 row, int32 column, uint32 plane)
- real32 **Interpolate** () const
- void **Increment** ()

The documentation for this class was generated from the following file:

- dng_gain_map.cpp

5.58 dng_gamma_encode_proxy Class Reference

Inheritance diagram for dng_gamma_encode_proxy:



Public Member Functions

- **dng_gamma_encode_proxy** (real64 lower, real64 upper, bool isSceneReferred, real64 stage3BlackLevel, real64 blackLevel)
- virtual real64 Evaluate (real64 x) const

5.58.1 Member Function Documentation

5.58.1.1 Evaluate()

```
virtual real64 dng_gamma_encode_proxy::Evaluate (
             real64 x ) const  [inline], [virtual]
```

Return the mapping for value x. This method must be implemented by a derived class of dng_1d_function and the derived class determines the lookup method and function used.

Parameters

x	A value between 0.0 and 1.0 (inclusive).

Return values

Mapped	value for x

Implements dng_1d_function.

The documentation for this class was generated from the following file:

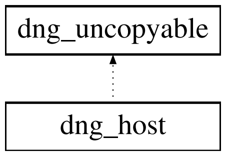
- dng_negative.cpp

5.59 dng_host Class Reference

The main class for communication between the application and the DNG SDK. Used to customize memory allocation and other behaviors.

`#include <dng_host.h>`

Inheritance diagram for dng_host:



Public Member Functions

- dng_host (dng_memory_allocator ∗allocator=NULL, dng_abort_sniffer ∗sniffer=NULL)
- virtual ∼dng_host ()
- dng_memory_allocator & Allocator ()

  Getter for host's memory allocator.
- virtual dng_memory_block ∗ Allocate (uint32 logicalSize)
- void SetSniffer (dng_abort_sniffer ∗sniffer)

  Setter for host's abort sniffer.
- dng_abort_sniffer ∗ Sniffer ()

  Getter for host's abort sniffer.
- virtual void SniffForAbort ()
- void SetNeedsMeta (bool needs)
- bool NeedsMeta () const

  Getter for flag determining whether all XMP metadata should be parsed.
- void SetNeedsImage (bool needs)
- bool NeedsImage () const

  Setter for flag determining whether DNG image data is needed.
- void SetForPreview (bool preview)
- bool ForPreview () const
- void SetMinimumSize (uint32 size)
- uint32 MinimumSize () const

  Getter for the minimum preview size.
- void SetPreferredSize (uint32 size)
- uint32 PreferredSize () const

  Getter for the preferred preview size.
- void SetMaximumSize (uint32 size)
- uint32 MaximumSize () const

  Getter for the maximum preview size.
- void SetForFastSaveToDNG (bool flag, uint32 size)
- bool ForFastSaveToDNG () const
- uint32 **FastSaveToDNGSize** () const
- void SetCropFactor (real64 cropFactor)
- real64 CropFactor () const

  Getter for the cropping factor.
- void ValidateSizes ()

  Makes sures minimum, preferred, and maximum sizes are reasonable.
- void SetSaveDNGVersion (uint32 version)
- virtual uint32 SaveDNGVersion () const

  Getter for what version to save DNG file compatible with.
- void SetSaveLinearDNG (bool linear)
- virtual bool SaveLinearDNG (const dng_negative &negative) const

  Getter for flag determining whether to save a linear DNG file.
- void SetKeepOriginalFile (bool keep)
- bool KeepOriginalFile ()

  Getter for flag determining whether to keep original RAW file data.
- virtual bool IsTransientError (dng_error_code code)
- virtual void PerformAreaTask (dng_area_task &task, const dng_rect &area, dng_area_task_progress ∗progress=NULL)

- virtual uint32 PerformAreaTaskThreads ()
- virtual dng_exif ∗ Make_dng_exif ()
- virtual dng_xmp ∗ Make_dng_xmp ()
- virtual dng_shared ∗ Make_dng_shared ()
- virtual dng_ifd ∗ Make_dng_ifd ()
- virtual dng_negative ∗ Make_dng_negative ()
- virtual dng_image ∗ Make_dng_image (const dng_rect &bounds, uint32 planes, uint32 pixelType)
- virtual dng_opcode ∗ Make_dng_opcode (uint32 opcodeID, dng_stream &stream)
- virtual void ApplyOpcodeList (dng_opcode_list &list, dng_negative &negative, AutoPtr< dng_image > &image)
- virtual void ResampleImage (const dng_image &srcImage, dng_image &dstImage)
- bool WantsPreserveStage2 () const
- void SetWantsPreserveStage2 (bool flag)

5.59.1 Detailed Description

The main class for communication between the application and the DNG SDK. Used to customize memory allocation and other behaviors.

dng_host allows setting parameters for the DNG conversion, mediates callback style interactions between the host application and the DNG SDK, and allows controlling certain internal behavior of the SDK such as memory allocation. Many applications will be able to use the default implementation of dng_host by just setting the dng_memory_allocator and dng_abort_sniffer in the constructor. More complex interactions will require deriving a class from dng_host.

Multiple dng_host objects can be allocated in a single process. This may be useful for DNG processing on separate threads. (Distinct dng_host objects are completely threadsafe for read/write. The application is responsible for establishing mutual exclusion for read/write access to a single dng_host object if it is used in multiple threads.)

5.59.2 Constructor & Destructor Documentation

5.59.2.1 dng_host()

```
dng_host::dng_host (
            dng_memory_allocator ∗ allocator = NULL,
            dng_abort_sniffer ∗ sniffer = NULL )
```

Allocate a dng_host object, possiblly with custom allocator and sniffer.

Parameters

allocator	Allows controlling all memory allocation done via this dng_host. Defaults to singleton global dng_memory_allocator, which calls new/delete dng_malloc_block for appropriate size.
sniffer	Used to periodically check if pending DNG conversions should be aborted and to communicate progress updates. Defaults to singleton global dng_abort_sniffer, which never aborts and ignores progress updated.

5.59.2.2 ∼dng_host()

```
dng_host::∼dng_host ( )  [virtual]
```

Clean up direct memory for dng_host. Memory allocator and abort sniffer are not deleted. Objects such as dng_image and others returned from host can still be used after host is deleted.

5.59.3 Member Function Documentation

5.59.3.1 Allocate()

```
dng_memory_block * dng_host::Allocate (
             uint32 logicalSize )  [virtual]
```

Alocate a new dng_memory_block using the host's memory allocator. Uses the Allocator() property of host to allocate a new block of memory. Will call ThrowMemoryFull if block cannot be allocated.

Parameters

logicalSize	Number of usable bytes returned dng_memory_block must contain.

References dng_memory_allocator::Allocate(), and Allocator().

Referenced by dng_mosaic_info::InterpolateGeneric(), dng_jpeg_image_encode_task::Process(), dng_read_tiles_↩ task::Process(), and dng_write_tiles_task::Process().

5.59.3.2 ApplyOpcodeList()

```
void dng_host::ApplyOpcodeList (
             dng_opcode_list & list,
             dng_negative & negative,
             AutoPtr< dng_image > & image )  [virtual]
```

Factory method to apply a dng_opcode_list. Can be used to override opcode list applications.

References dng_opcode_list::Apply().

5.59.3.3 ForFastSaveToDNG()

```
bool dng_host::ForFastSaveToDNG ( ) const  [inline]
```

Getter for the Boolean value that indicates whether this host is being used to perform a negative read for fast conversion to DNG.

5.59.3.4 ForPreview()

```
bool dng_host::ForPreview ( ) const  [inline]
```

Getter for flag determining whether image should be preview quality. Preview quality images may be rendered more quickly. Current DNG SDK does not change rendering behavior based on this flag, but derived versions may use this getter to choose between a slower more accurate path and a faster "good enough for preview" one. Data produce with ForPreview set to true should not be written back to a DNG file, except as a preview image.

Referenced by dng_opcode::AboutToApply().

5.59.3.5 IsTransientError()

```
bool dng_host::IsTransientError (
             dng_error_code code )  [virtual]
```

Determine if an error is the result of a temporary, but planned-for occurence such as user cancellation or memory exhaustion. This method is sometimes used to determine whether to try and continue processing a DNG file despite errors in the file format, etc. In such cases, processing will be continued if IsTransientError returns false. This is so that user cancellation and memory exhaustion always terminate processing.

Parameters

code	Error to test for transience.

References dng_error_memory, and dng_error_user_canceled.

5.59.3.6 Make_dng_exif()

```
dng_exif * dng_host::Make_dng_exif ( )  [virtual]
```

Factory method for dng_exif class. Can be used to customize allocation or to ensure a derived class is used instead of dng_exif.

References ThrowMemoryFull().

Referenced by dng_info::Parse().

5.59.3.7 Make_dng_ifd()

```
dng_ifd * dng_host::Make_dng_ifd ( )  [virtual]
```

Factory method for dng_ifd class. Can be used to customize allocation or to ensure a derived class is used instead of dng_ifd.

References ThrowMemoryFull().

Referenced by dng_info::Parse().

5.59.3.8 Make_dng_image()

```
dng_image * dng_host::Make_dng_image (
             const dng_rect & bounds,
             uint32 planes,
             uint32 pixelType )  [virtual]
```

Factory method for dng_image class. Can be used to customize allocation or to ensure a derived class is used instead of dng_simple_image.

References Allocator(), and ThrowMemoryFull().

Referenced by dng_filter_opcode::Apply(), dng_opcode_WarpRectilinear::Apply(), dng_opcode_WarpFisheye::Apply(), and dng_render::Render().

5.59.3.9 Make_dng_negative()

```
dng_negative * dng_host::Make_dng_negative ( )  [virtual]
```

Factory method for dng_negative class. Can be used to customize allocation or to ensure a derived class is used instead of dng_negative.

5.59.3.10 Make_dng_opcode()

```
dng_opcode * dng_host::Make_dng_opcode (
             uint32 opcodeID,
             dng_stream & stream )  [virtual]
```

Factory method for parsing dng_opcode based classs. Can be used to override opcode implementations.

Referenced by dng_opcode_list::Parse().

5.59.3.11 Make_dng_shared()

```
dng_shared * dng_host::Make_dng_shared ( )  [virtual]
```

Factory method for dng_shared class. Can be used to customize allocation or to ensure a derived class is used instead of dng_shared.

References ThrowMemoryFull().

Referenced by dng_info::Parse().

5.59.3.12 Make_dng_xmp()

```
dng_xmp * dng_host::Make_dng_xmp ( )  [virtual]
```

Factory method for dng_xmp class. Can be used to customize allocation or to ensure a derived class is used instead of dng_xmp.

References Allocator(), and ThrowMemoryFull().

5.59.3.13 PerformAreaTask()

```
void dng_host::PerformAreaTask (
             dng_area_task & task,
             const dng_rect & area,
             dng_area_task_progress * progress = NULL )  [virtual]
```

General top-level botttleneck for image processing tasks. Default implementation calls dng_area_task::PerformAreaTask method on task. Can be overridden in derived classes to support multiprocessing, for example.

Parameters

task	Image processing task to perform on area.
area	Rectangle over which to perform image processing task.

References Allocator(), dng_area_task::Perform(), and Sniffer().

Referenced by dng_filter_opcode::Apply(), dng_opcode_WarpRectilinear::Apply(), dng_inplace_opcode::Apply(), dng←_opcode_WarpFisheye::Apply(), dng_mosaic_info::InterpolateFast(), dng_linearization_info::Linearize(), and dng_←render::Render().

5.59.3.14 PerformAreaTaskThreads()

```
uint32 dng_host::PerformAreaTaskThreads ( )  [virtual]
```

How many multiprocessing threads does PerformAreaTask use? Default implementation always returns 1 since it is single threaded.

5.59.3.15 ResampleImage()

```
void dng_host::ResampleImage (
             const dng_image & srcImage,
             dng_image & dstImage )  [virtual]
```

Factory method to resample an image. Can be used to override image method used to resample images.

References dng_image::Bounds().

5.59.3.16 SetCropFactor()

```
void dng_host::SetCropFactor (
            real64 cropFactor )  [inline]
```

Setter for the cropping factor.

Parameters

cropFactor	Fraction of image to be used after crop.

5.59.3.17 SetForFastSaveToDNG()

```
void dng_host::SetForFastSaveToDNG (
            bool flag,
            uint32 size )  [inline]
```

Setter for the perform fast save to DNG.

Parameters

flag	True if the host is being used to perform a negative read for fast conversion to DNG, false otherwise.

5.59.3.18 SetForPreview()

```
void dng_host::SetForPreview (
            bool preview )  [inline]
```

Setter for flag determining whether image should be preview quality, or full quality.

Parameters

preview	If true, rendered images are for preview.

5.59.3.19 SetKeepOriginalFile()

```
void dng_host::SetKeepOriginalFile (
            bool keep )  [inline]
```

Setter for flag determining whether to keep original RAW file data.

Parameters

keep	If true, origianl RAW data will be kept.

5.59.3.20 SetMaximumSize()

```
void dng_host::SetMaximumSize (
             uint32 size )  [inline]
```

Setter for the maximum preview size.

Parameters

size	Maximum pixel size (long side of image).

5.59.3.21 SetMinimumSize()

```
void dng_host::SetMinimumSize (
             uint32 size )  [inline]
```

Setter for the minimum preview size.

Parameters

size	Minimum pixel size (long side of image).

Referenced by ValidateSizes().

5.59.3.22 SetNeedsImage()

```
void dng_host::SetNeedsImage (
             bool needs )  [inline]
```

Setter for flag determining whether DNG image data is needed. Defaults to true. Image data might not be needed for applications which only manipulate metadata.

Parameters

needs	If true, image data is needed.

5.59.3.23 SetNeedsMeta()

```
void dng_host::SetNeedsMeta (
             bool needs )  [inline]
```

Setter for flag determining whether all XMP metadata should be parsed. Defaults to true. One might not want metadata when doing a quick check to see if a file is readable.

Parameters

needs	If true, metadata is needed.

5.59.3.24 SetPreferredSize()

```
void dng_host::SetPreferredSize (
             uint32 size )  [inline]
```

Setter for the preferred preview size.

Parameters

size	Preferred pixel size (long side of image).

Referenced by ValidateSizes().

5.59.3.25 SetSaveDNGVersion()

```
void dng_host::SetSaveDNGVersion (
             uint32 version )  [inline]
```

Setter for what version to save DNG file compatible with.

Parameters

version	What version to save DNG file compatible with.

5.59.3.26 SetSaveLinearDNG()

```
void dng_host::SetSaveLinearDNG (
             bool linear )  [inline]
```

Setter for flag determining whether to force saving a linear DNG file.

Parameters

linear	If true, we should force saving a linear DNG file.

5.59.3.27 SetWantsPreserveStage2()

```
void dng_host::SetWantsPreserveStage2 (
            bool flag )  [inline]
```

Setter for flag determining whether we should preserve the stage 2 image after building the stage 3 image.

5.59.3.28 SniffForAbort()

```
void dng_host::SniffForAbort ( )  [virtual]
```

Check for pending abort. Should call ThrowUserCanceled if an abort is pending.

References Sniffer(), and dng_abort_sniffer::SniffForAbort().

Referenced by dng_mosaic_info::InterpolateGeneric().

5.59.3.29 WantsPreserveStage2()

```
bool dng_host::WantsPreserveStage2 ( ) const  [inline]
```

Getter for flag determining whether we should preserve the stage 2 image after building the stage 3 image.

The documentation for this class was generated from the following files:

- dng_host.h
- dng_host.cpp

5.60 dng_hue_sat_map Class Reference

A 3D table that maps HSV (hue, saturation, and value) floating-point input coordinates in the range [0,1] to delta signals. The table must have at least 1 sample in the hue dimension, at least 2 samples in the saturation dimension, and at least 1 sample in the value dimension. Tables are stored in value-hue-saturation order.

```
#include <dng_hue_sat_map.h>
```

Classes

- struct HSBModify

Public Member Functions

- dng_hue_sat_map ()

  Construct an empty (and invalid) hue sat map.
- dng_hue_sat_map (const dng_hue_sat_map &src)

  Copy an existing hue sat map.
- dng_hue_sat_map & operator= (const dng_hue_sat_map &rhs)

  Copy an existing hue sat map.
- virtual ∼dng_hue_sat_map ()

  Destructor.
- bool IsNull () const

  Is this hue sat map invalid?
- bool IsValid () const

  Is this hue sat map valid?
- void SetInvalid ()

  Clear the hue sat map, making it invalid.
- void GetDivisions (uint32 &hueDivisions, uint32 &satDivisions, uint32 &valDivisions) const

  Get the table dimensions (number of samples in each dimension).
- void SetDivisions (uint32 hueDivisions, uint32 satDivisions, uint32 valDivisions=1)
- void GetDelta (uint32 hueDiv, uint32 satDiv, uint32 valDiv, HSBModify &modify) const

  Get a specific table entry, specified by table indices.
- void EnsureWriteable ()

  Make sure the table is writeable.
- void SetDelta (uint32 hueDiv, uint32 satDiv, uint32 valDiv, const HSBModify &modify)

  Set a specific table entry, specified by table indices.
- void SetDeltaKnownWriteable (uint32 hueDiv, uint32 satDiv, uint32 valDiv, const HSBModify &modify)

  Same as SetDelta, without checking that the table is writeable.
- uint32 DeltasCount () const

  Get the total number of samples (across all dimensions).
- HSBModify ∗ GetDeltas ()
- const HSBModify ∗ GetConstDeltas () const
- void **AssignNewUniqueRuntimeFingerprint** ()
- void SetRuntimeFingerprint (const dng_fingerprint fingerprint)

  Set Fingerprint. Rare use cases want to set the fingerprint.
- const dng_fingerprint & RuntimeFingerprint () const

  Get the runtime fingerprint of this hue sat map.
- bool operator== (const dng_hue_sat_map &rhs) const

  Equality test.

Static Public Member Functions

- static dng_hue_sat_map ∗ Interpolate (const dng_hue_sat_map &map1, const dng_hue_sat_map &map2, real64 weight1)

5.60.1 Detailed Description

A 3D table that maps HSV (hue, saturation, and value) floating-point input coordinates in the range [0,1] to delta signals. The table must have at least 1 sample in the hue dimension, at least 2 samples in the saturation dimension, and at least 1 sample in the value dimension. Tables are stored in value-hue-saturation order.

5.60.2 Member Function Documentation

5.60.2.1 GetConstDeltas()

```
const HSBModify* dng_hue_sat_map::GetConstDeltas ( ) const  [inline]
```

Direct read-only access to table entries. The entries are stored in value-hue-saturation order (outer to inner).

References dng_ref_counted_block::Buffer_real32().

Referenced by GetDelta(), Interpolate(), and operator==().

5.60.2.2 GetDeltas()

```
HSBModify* dng_hue_sat_map::GetDeltas ( )  [inline]
```

Direct read/write access to table entries. The entries are stored in value-hue-saturation order (outer to inner).

References dng_ref_counted_block::Buffer_real32(), and EnsureWriteable().

5.60.2.3 Interpolate()

```
dng_hue_sat_map * dng_hue_sat_map::Interpolate (
             const dng_hue_sat_map & map1,
             const dng_hue_sat_map & map2,
             real64 weight1 )  [static]
```

Compute a linearly-interpolated hue sat map (i.e., delta and scale factors) from the specified tables, with the specified weight. map1 and map2 must have the same dimensions.

References DeltasCount(), dng_hue_sat_map(), DNG_REPORT, GetConstDeltas(), IsValid(), dng_md5_printer::↩ Process(), AutoPtr< T >::Release(), dng_md5_printer::Result(), RuntimeFingerprint(), SetDivisions(), SetRuntime↩ Fingerprint(), and ThrowProgramError().

Referenced by dng_camera_profile::HueSatMapForWhite().

5.60.2.4 SetDivisions()

```
void dng_hue_sat_map::SetDivisions (
            uint32 hueDivisions,
            uint32 satDivisions,
            uint32 valDivisions = 1 )
```

Set the table dimensions (number of samples in each dimension). This erases any existing table data.

References dng_ref_counted_block::Allocate(), DeltasCount(), and DNG_ASSERT.

Referenced by Interpolate().

The documentation for this class was generated from the following files:

- dng_hue_sat_map.h
- dng_hue_sat_map.cpp

5.61 dng_ifd Class Reference

Container for a single image file directory of a digital negative.

```
#include <dng_ifd.h>
```

Public Types

- enum { **kMaxTileInfo** = 32 }

Public Member Functions

- virtual dng_ifd ∗ **Clone** () const
- virtual bool **ParseTag** (dng_stream &stream, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tag↩ Count, uint64 tagOffset)
- virtual void **PostParse** ()
- virtual bool **IsValidDNG** (dng_shared &shared, uint32 parentCode)
- dng_rect **Bounds** () const
- uint32 **TilesAcross** () const
- uint32 **TilesDown** () const
- uint32 **TilesPerImage** () const
- dng_rect **TileArea** (uint32 rowIndex, uint32 colIndex) const
- virtual uint32 **TileByteCount** (const dng_rect &tile) const
- void **SetSingleStrip** ()
- void **FindTileSize** (uint32 bytesPerTile=128 ∗1024, uint32 cellH=16, uint32 cellV=16)
- void **FindStripSize** (uint32 bytesPerStrip=128 ∗1024, uint32 cellV=16)
- virtual uint32 **PixelType** () const
- virtual bool **IsBaselineJPEG** () const
- virtual bool **CanRead** () const
- virtual void **ReadImage** (dng_host &host, dng_stream &stream, dng_image &image, dng_jpeg_image ∗jpeg↩ Image=NULL, dng_fingerprint ∗jpegDigest=NULL) const

Public Attributes

- bool **fUsesNewSubFileType**
- uint32 **fNewSubFileType**
- uint32 **fImageWidth**
- uint32 **fImageLength**
- uint32 **fBitsPerSample** [kMaxSamplesPerPixel]
- uint32 **fCompression**
- uint32 **fPredictor**
- uint32 **fPhotometricInterpretation**
- uint32 **fFillOrder**
- uint32 **fOrientation**
- uint32 **fOrientationType**
- uint64 **fOrientationOffset**
- bool **fOrientationBigEndian**
- uint32 **fSamplesPerPixel**
- uint32 **fPlanarConfiguration**
- real64 **fXResolution**
- real64 **fYResolution**
- uint32 **fResolutionUnit**
- bool **fUsesStrips**
- bool **fUsesTiles**
- uint32 **fTileWidth**
- uint32 **fTileLength**
- uint32 **fTileOffsetsType**
- uint32 **fTileOffsetsCount**
- uint64 **fTileOffsetsOffset**
- uint64 **fTileOffset** [kMaxTileInfo]
- uint32 **fTileByteCountsType**
- uint32 **fTileByteCountsCount**
- uint64 **fTileByteCountsOffset**
- uint32 **fTileByteCount** [kMaxTileInfo]
- uint32 **fSubIFDsCount**
- uint64 **fSubIFDsOffset**
- uint32 **fExtraSamplesCount**
- uint32 **fExtraSamples** [kMaxSamplesPerPixel]
- uint32 **fSampleFormat** [kMaxSamplesPerPixel]
- uint32 **fJPEGTablesCount**
- uint64 **fJPEGTablesOffset**
- uint64 **fJPEGInterchangeFormat**
- uint32 **fJPEGInterchangeFormatLength**
- real64 **fYCbCrCoefficientR**
- real64 **fYCbCrCoefficientG**
- real64 **fYCbCrCoefficientB**
- uint32 **fYCbCrSubSampleH**
- uint32 **fYCbCrSubSampleV**
- uint32 **fYCbCrPositioning**
- real64 **fReferenceBlackWhite** [6]
- uint32 **fCFARepeatPatternRows**
- uint32 **fCFARepeatPatternCols**
- uint8 **fCFAPattern** [kMaxCFAPattern][kMaxCFAPattern]

- uint8 **fCFAPlaneColor** [kMaxColorPlanes]
- uint32 **fCFALayout**
- uint32 **fLinearizationTableType**
- uint32 **fLinearizationTableCount**
- uint64 **fLinearizationTableOffset**
- uint32 **fBlackLevelRepeatRows**
- uint32 **fBlackLevelRepeatCols**
- real64 **fBlackLevel** [kMaxBlackPattern][kMaxBlackPattern][kMaxSamplesPerPixel]
- uint32 **fBlackLevelDeltaHType**
- uint32 **fBlackLevelDeltaHCount**
- uint64 **fBlackLevelDeltaHOffset**
- uint32 **fBlackLevelDeltaVType**
- uint32 **fBlackLevelDeltaVCount**
- uint64 **fBlackLevelDeltaVOffset**
- real64 **fWhiteLevel** [kMaxSamplesPerPixel]
- dng_urational **fDefaultScaleH**
- dng_urational **fDefaultScaleV**
- dng_urational **fBestQualityScale**
- dng_urational **fDefaultCropOriginH**
- dng_urational **fDefaultCropOriginV**
- dng_urational **fDefaultCropSizeH**
- dng_urational **fDefaultCropSizeV**
- dng_urational **fDefaultUserCropT**
- dng_urational **fDefaultUserCropL**
- dng_urational **fDefaultUserCropB**
- dng_urational **fDefaultUserCropR**
- uint32 **fBayerGreenSplit**
- dng_urational **fChromaBlurRadius**
- dng_urational **fAntiAliasStrength**
- dng_rect **fActiveArea**
- uint32 **fMaskedAreaCount**
- dng_rect **fMaskedArea** [kMaxMaskedAreas]
- uint32 **fRowInterleaveFactor**
- uint32 **fSubTileBlockRows**
- uint32 **fSubTileBlockCols**
- dng_preview_info **fPreviewInfo**
- uint32 **fOpcodeList1Count**
- uint64 **fOpcodeList1Offset**
- uint32 **fOpcodeList2Count**
- uint64 **fOpcodeList2Offset**
- uint32 **fOpcodeList3Count**
- uint64 **fOpcodeList3Offset**
- dng_noise_profile **fNoiseProfile**
- dng_string **fEnhanceParams**
- dng_urational **fBaselineSharpness**
- dng_urational **fNoiseReductionApplied**
- bool **fLosslessJPEGBug16**
- uint32 **fSampleBitShift**
- uint64 **fThisIFD**
- uint64 **fNextIFD**
- int32 **fCompressionQuality**
- bool **fPatchFirstJPEGByte**

Protected Member Functions

- virtual bool **IsValidCFA** (dng_shared &shared, uint32 parentCode)

5.61.1 Detailed Description

Container for a single image file directory of a digital negative.

See DNG 1.1.0 specification for documentation of specific tags.

The documentation for this class was generated from the following files:

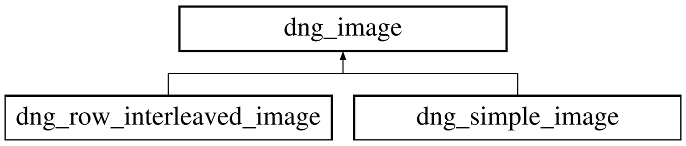
- dng_ifd.h
- dng_ifd.cpp

5.62 dng_image Class Reference

Base class for holding image data in DNG SDK. See dng_simple_image for derived class most often used in DNG SDK.

```
#include <dng_image.h>
```

Inheritance diagram for dng_image:



Public Types

- enum edge_option { edge_none, edge_zero, edge_repeat, edge_repeat_zero_last }

  How to handle requests to get image areas outside the image bounds.

Public Member Functions

- virtual dng_image ∗ **Clone** () const
- const dng_rect & Bounds () const

  Getter method for bounds of an image.
- dng_point Size () const

  Getter method for size of an image.
- uint32 Width () const

  Getter method for width of an image.
- uint32 Height () const

  Getter method for height of an image.
- uint32 Planes () const

  Getter method for number of planes in an image.
- uint32 PixelType () const
- virtual void SetPixelType (uint32 pixelType)
- uint32 PixelSize () const
- uint32 PixelRange () const
- virtual dng_rect RepeatingTile () const

  Getter for best "tile stride" for accessing image.
- void Get (dng_pixel_buffer &buffer, edge_option edgeOption=edge_none, uint32 repeatV=1, uint32 repeatH=1) const
- void Put (const dng_pixel_buffer &buffer)
- virtual void Trim (const dng_rect &r)
- virtual void Rotate (const dng_orientation &orientation)
- void CopyArea (const dng_image &src, const dng_rect &area, uint32 srcPlane, uint32 dstPlane, uint32 planes)
- void CopyArea (const dng_image &src, const dng_rect &area, uint32 plane, uint32 planes)
- virtual bool EqualArea (const dng_image &rhs, const dng_rect &area, uint32 plane, uint32 planes) const
- void **SetConstant_uint8** (uint8 value, const dng_rect &area)
- void **SetConstant_uint8** (uint8 value)
- void **SetConstant_uint16** (uint16 value, const dng_rect &area)
- void **SetConstant_uint16** (uint16 value)
- void **SetConstant_int16** (int16 value, const dng_rect &area)
- void **SetConstant_int16** (int16 value)
- void **SetConstant_uint32** (uint32 value, const dng_rect &area)
- void **SetConstant_uint32** (uint32 value)
- void **SetConstant_real32** (real32 value, const dng_rect &area)
- void **SetConstant_real32** (real32 value)
- virtual void **GetRepeat** (dng_pixel_buffer &buffer, const dng_rect &srcArea, const dng_rect &dstArea) const

Protected Member Functions

- **dng_image** (const dng_rect &bounds, uint32 planes, uint32 pixelType)
- virtual void **AcquireTileBuffer** (dng_tile_buffer &buffer, const dng_rect &area, bool dirty) const
- virtual void **ReleaseTileBuffer** (dng_tile_buffer &buffer) const
- virtual void **DoGet** (dng_pixel_buffer &buffer) const
- virtual void **DoPut** (const dng_pixel_buffer &buffer)
- virtual void **DoCopyArea** (const dng_image &src, const dng_rect &area, uint32 srcPlane, uint32 dstPlane, uint32 planes)
- void **GetEdge** (dng_pixel_buffer &buffer, edge_option edgeOption, const dng_rect &srcArea, const dng_rect &dstArea) const
- virtual void **SetConstant** (uint32 value, const dng_rect &area)

Protected Attributes

- dng_rect **fBounds**
- uint32 **fPlanes**
- uint32 **fPixelType**

Friends

- class **dng_tile_buffer**

5.62.1 Detailed Description

Base class for holding image data in DNG SDK. See dng_simple_image for derived class most often used in DNG SDK.

5.62.2 Member Enumeration Documentation

5.62.2.1 edge_option

```
enum dng_image::edge_option
```

How to handle requests to get image areas outside the image bounds.

Enumerator

edge_none	Leave edge pixels unchanged.
edge_zero	Pad with zeros.
edge_repeat	Repeat edge pixels.
edge_repeat_zero_last	Repeat edge pixels, except for last plane which is zero padded.

5.62.3 Member Function Documentation

5.62.3.1 CopyArea() [1/2]

```
void dng_image::CopyArea (
            const dng_image & src,
            const dng_rect & area,
            uint32 srcPlane,
            uint32 dstPlane,
            uint32 planes )  [inline]
```

Copy image data from an area of one image to same area of another.

Parameters

src	Image to copy from.
area	Rectangle of images to copy.
srcPlane	Plane to start copying in src.
dstPlane	Plane to start copying in this.
planes	Number of planes to copy.

5.62.3.2 CopyArea() [2/2]

```
void dng_image::CopyArea (
             const dng_image & src,
             const dng_rect & area,
             uint32 plane,
             uint32 planes )  [inline]
```

Copy image data from an area of one image to same area of another.

Parameters

src	Image to copy from.
area	Rectangle of images to copy.
plane	Plane to start copying in src and this.
planes	Number of planes to copy.

5.62.3.3 EqualArea()

```
bool dng_image::EqualArea (
             const dng_image & rhs,
             const dng_rect & area,
             uint32 plane,
             uint32 planes ) const  [virtual]
```

Return true if the contents of an area of the image are the same as those of another.

Parameters

rhs	Image to compare against.
area	Rectangle of image to test.
plane	Plane to start comparing.
planes	Number of planes to compare.

References dng_pixel_buffer::EqualArea().

5.62.3.4 Get()

```
void dng_image::Get (
            dng_pixel_buffer & buffer,
            edge_option edgeOption = edge_none,
            uint32 repeatV = 1,
            uint32 repeatH = 1 ) const
```

Get a pixel buffer of data on image with proper edge padding.

Parameters

buffer	Receives resulting pixel buffer.
edgeOption	edge_option describing how to pad edges.
repeatV	Amount of repeated padding needed in vertical for edge_repeat and edge_repeat_zero_last edgeOption cases.
repeatH	Amount of repeated padding needed in horizontal for edge_repeat and edge_repeat_zero_last edgeOption cases.

References dng_pixel_buffer::DirtyPixel(), and edge_none.

Referenced by dng_mosaic_info::InterpolateGeneric(), dng_filter_task::Process(), dng_inplace_opcode_task::↩ Process(), dng_find_new_raw_image_digest_task::Process(), and dng_render_task::ProcessArea().

5.62.3.5 PixelRange()

```
uint32 dng_image::PixelRange ( ) const
```

Getter for pixel range. For unsigned types, range is 0 to return value. For signed types, range is return value - 0x8000U. For ttFloat type, pixel range is 0.0 to 1.0 and this routine returns 1.

5.62.3.6 PixelSize()

```
uint32 dng_image::PixelSize ( ) const
```

Getter for pixel size.

Return values

Size,in	bytes, of pixel type for this image .

References PixelType().

Referenced by SetPixelType().

5.62.3.7 PixelType()

```
uint32 dng_image::PixelType ( ) const  [inline]
```

Getter for pixel type.

Return values

See	dng_tagtypes.h . Valid values are ttByte, ttShort, ttSShort, ttLong, ttFloat .

Referenced by dng_filter_opcode::Apply(), dng_opcode_WarpRectilinear::Apply(), dng_opcode_WarpFisheye::Apply(), dng_mosaic_info::InterpolateGeneric(), dng_linearization_info::Linearize(), PixelSize(), and dng_render::Render().

5.62.3.8 Put()

```
void dng_image::Put (
             const dng_pixel_buffer & buffer )
```

Put a pixel buffer into image.

Parameters

buffer	Pixel buffer to copy from.

References dng_pixel_buffer::ConstPixel(), and Planes().

Referenced by dng_mosaic_info::InterpolateGeneric(), dng_filter_task::Process(), and dng_inplace_opcode_task::↩ Process().

5.62.3.9 Rotate()

```
void dng_image::Rotate (
             const dng_orientation & orientation )  [virtual]
```

Rotate image to reflect given orientation change.

Parameters

orientation	Directive to rotate image in a certain way.

Reimplemented in dng_simple_image.

References ThrowProgramError().

5.62.3.10 SetPixelType()

```
void dng_image::SetPixelType (
             uint32 pixelType )  [virtual]
```

Setter for pixel type.

Parameters

pixelType	The new pixel type .

Reimplemented in dng_simple_image.

References PixelSize(), and ThrowProgramError().

Referenced by dng_simple_image::SetPixelType().

5.62.3.11 Trim()

```
void dng_image::Trim (
             const dng_rect & r )  [virtual]
```

Shrink bounds of image to given rectangle.

Parameters

r	Rectangle to crop to.

Reimplemented in dng_simple_image.

References Bounds(), and ThrowProgramError().

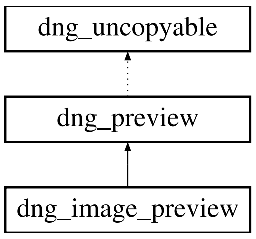
Referenced by dng_opcode_TrimBounds::Apply().

The documentation for this class was generated from the following files:

- dng_image.h
- dng_image.cpp

5.63 dng_image_preview Class Reference

Inheritance diagram for dng_image_preview:



Public Member Functions

- virtual dng_basic_tag_set ∗ **AddTagSet** (dng_tiff_directory &directory) const
- virtual void **WriteData** (dng_host &host, dng_image_writer &writer, dng_basic_tag_set &basic, dng_stream &stream) const

Public Attributes

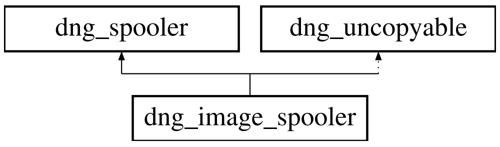
- AutoPtr< dng_image > **fImage**

The documentation for this class was generated from the following files:

- dng_preview.h
- dng_preview.cpp

5.64 dng_image_spooler Class Reference

Inheritance diagram for dng_image_spooler:



Public Member Functions

- **dng_image_spooler** (dng_host &host, const dng_ifd &ifd, dng_image &image, const dng_rect &tileArea, uint32 plane, uint32 planes, dng_memory_block &block, AutoPtr< dng_memory_block > &subTileBuffer)
- virtual void **Spool** (const void ∗data, uint32 count)

The documentation for this class was generated from the following file:

- dng_read_image.cpp

5.65 dng_image_writer Class Reference

Support for writing dng_image or dng_negative instances to a dng_stream in TIFF or DNG format.

```
#include <dng_image_writer.h>
```

Public Member Functions

- virtual void **EncodeJPEGPreview** (dng_host &host, const dng_image &image, dng_jpeg_preview &preview, int32 quality=-1)
- virtual void **WriteImage** (dng_host &host, const dng_ifd &ifd, dng_basic_tag_set &basic, dng_stream &stream, const dng_image &image, uint32 fakeChannels=1)
- void WriteTIFF (dng_host &host, dng_stream &stream, const dng_image &image, uint32 photometric←Interpretation, uint32 compression, dng_negative ∗negative, const dng_color_space ∗space=NULL, const dng_resolution ∗resolution=NULL, const dng_jpeg_preview ∗thumbnail=NULL, const dng_memory_block ∗imageResources=NULL, dng_metadata_subset metadataSubset=kMetadataSubset_All, bool hasTransparency=false)
- void **WriteTIFF** (dng_host &host, dng_stream &stream, const dng_image &image, uint32 photometric←Interpretation=piBlackIsZero, uint32 compression=ccUncompressed, const dng_metadata ∗metadata=NU←LL, const dng_color_space ∗space=NULL, const dng_resolution ∗resolution=NULL, const dng_jpeg_preview ∗thumbnail=NULL, const dng_memory_block ∗imageResources=NULL, dng_metadata_subset metadata←Subset=kMetadataSubset_All, bool hasTransparency=false)
- void WriteTIFFWithProfile (dng_host &host, dng_stream &stream, const dng_image &image, uint32 photometricInterpretation, uint32 compression, dng_negative ∗negative, const void ∗profileData=NULL, uint32 profileSize=0, const dng_resolution ∗resolution=NULL, const dng_jpeg_preview ∗thumbnail=NULL, const dng_memory_block ∗imageResources=NULL, dng_metadata_subset metadataSubset=kMetadataSubset_All, bool hasTransparency=false)
- virtual void **WriteTIFFWithProfile** (dng_host &host, dng_stream &stream, const dng_image &image, uint32 photometricInterpretation=piBlackIsZero, uint32 compression=ccUncompressed, const dng_metadata ∗metadata=NULL, const void ∗profileData=NULL, uint32 profileSize=0, const dng_resolution ∗resolution=N←ULL, const dng_jpeg_preview ∗thumbnail=NULL, const dng_memory_block ∗imageResources=NULL, dng_←metadata_subset metadataSubset=kMetadataSubset_All, bool hasTransparency=false)
- void WriteDNG (dng_host &host, dng_stream &stream, dng_negative &negative, const dng_preview_list ∗previewList=NULL, uint32 maxBackwardVersion=dngVersion_SaveDefault, bool uncompressed=false)
- virtual void WriteDNGWithMetadata (dng_host &host, dng_stream &stream, const dng_negative &negative, const dng_metadata &metadata, const dng_preview_list ∗previewList=NULL, uint32 maxBackwardVersion=dng←Version_SaveDefault, bool uncompressed=false)
- virtual void CleanUpMetadata (dng_host &host, dng_metadata &metadata, dng_metadata_subset metadata←Subset, const char ∗dstMIME, const char ∗software=NULL)

Protected Types

- enum { **kImageBufferSize** = 128 ∗ 1024 }

Protected Member Functions

- virtual void **UpdateExifColorSpaceTag** (dng_metadata &metadata, const void ∗profileData, const uint32 profileSize)
- virtual uint32 **CompressedBufferSize** (const dng_ifd &ifd, uint32 uncompressedSize)
- virtual void **EncodePredictor** (dng_host &host, const dng_ifd &ifd, dng_pixel_buffer &buffer, AutoPtr< dng_memory_block > &tempBuffer)
- virtual void **ByteSwapBuffer** (dng_host &host, dng_pixel_buffer &buffer)
- void **ReorderSubTileBlocks** (const dng_ifd &ifd, dng_pixel_buffer &buffer, AutoPtr< dng_memory_block > &uncompressedBuffer, AutoPtr< dng_memory_block > &subTileBlockBuffer)
- virtual void **WriteData** (dng_host &host, const dng_ifd &ifd, dng_stream &stream, dng_pixel_buffer &buffer, AutoPtr< dng_memory_block > &compressedBuffer, bool usingMultipleThreads)

- virtual void **WriteTile** (dng_host &host, const dng_ifd &ifd, dng_stream &stream, const dng_image &image, const dng_rect &tileArea, uint32 fakeChannels, AutoPtr< dng_memory_block > &compressedBuffer, AutoPtr< dng_memory_block > &uncompressedBuffer, AutoPtr< dng_memory_block > &subTileBlockBuffer, AutoPtr< dng_memory_block > &tempBuffer, bool usingMultipleThreads)
- virtual void **DoWriteTiles** (dng_host &host, const dng_ifd &ifd, dng_basic_tag_set &basic, dng_stream &stream, const dng_image &image, uint32 fakeChannels, uint32 tilesDown, uint32 tilesAcross, uint32 compressedSize, const dng_safe_uint32 &uncompressedSize)

Friends

- class **dng_jpeg_image**
- class **dng_jpeg_image_encode_task**
- class **dng_write_tiles_task**

5.65.1 Detailed Description

Support for writing dng_image or dng_negative instances to a dng_stream in TIFF or DNG format.

5.65.2 Member Function Documentation

5.65.2.1 CleanUpMetadata()

```
void dng_image_writer::CleanUpMetadata (
            dng_host & host,
            dng_metadata & metadata,
            dng_metadata_subset metadataSubset,
            const char * dstMIME,
            const char * software = NULL )  [virtual]
```

Resolve metadata conflicts and apply metadata policies in keeping with Metadata Working Group (MWG) guidelines.

Referenced by WriteDNGWithMetadata().

5.65.2.2 WriteDNG()

```
void dng_image_writer::WriteDNG (
            dng_host & host,
            dng_stream & stream,
            dng_negative & negative,
            const dng_preview_list * previewList = NULL,
            uint32 maxBackwardVersion = dngVersion_SaveDefault,
            bool uncompressed = false )
```

Write a dng_image to a dng_stream in DNG format.

Parameters

host	Host interface used for progress updates, abort testing, buffer allocation, etc.
stream	The dng_stream on which to write the TIFF.
negative	The image data and metadata (EXIF, IPTC, XMP) to be written.
previewList	List of previews (not counting thumbnail) to write to the file. Defaults to empty.
maxBackwardVersion	The DNG file should be readable by readers at least back to this version.
uncompressed	True to force uncompressed images. Otherwise use normal compression.

References dng_negative::Metadata(), and WriteDNGWithMetadata().

5.65.2.3 WriteDNGWithMetadata()

```
void dng_image_writer::WriteDNGWithMetadata (
             dng_host & host,
             dng_stream & stream,
             const dng_negative & negative,
             const dng_metadata & metadata,
             const dng_preview_list * previewList = NULL,
             uint32 maxBackwardVersion = dngVersion_SaveDefault,
             bool uncompressed = false )  [virtual]
```

Write a dng_image to a dng_stream in DNG format.

Parameters

host	Host interface used for progress updates, abort testing, buffer allocation, etc.
stream	The dng_stream on which to write the TIFF.
negative	The image data to be written.
metadata	The metadata (EXIF, IPTC, XMP) to be written.
previewList	List of previews (not counting thumbnail) to write to the file. Defaults to empty.
maxBackwardVersion	The DNG file should be readable by readers at least back to this version.
uncompressed	True to force uncompressed images. Otherwise use normal compression.

References dng_host::Allocator(), CleanUpMetadata(), and dng_metadata::Clone().

Referenced by WriteDNG().

5.65.2.4 WriteTIFF()

```
void dng_image_writer::WriteTIFF (
             dng_host & host,
             dng_stream & stream,
             const dng_image & image,
             uint32 photometricInterpretation,
             uint32 compression,
             dng_negative * negative,
             const dng_color_space * space = NULL,
             const dng_resolution * resolution = NULL,
             const dng_jpeg_preview * thumbnail = NULL,
             const dng_memory_block * imageResources = NULL,
             dng_metadata_subset metadataSubset = kMetadataSubset_All,
             bool hasTransparency = false )
```

Write a dng_image to a dng_stream in TIFF format.

Parameters

host	Host interface used for progress updates, abort testing, buffer allocation, etc.
stream	The dng_stream on which to write the TIFF.
image	The actual image data to be written.
photometricInterpretation	Either piBlackIsZero for monochrome or piRGB for RGB images.
compression	Must be ccUncompressed.
negative	or metadata If non-NULL, EXIF, IPTC, and XMP metadata from this negative is written to TIFF.
space	If non-null and color space has an ICC profile, TIFF will be tagged with this profile. No color space conversion of image data occurs.
resolution	If non-NULL, TIFF will be tagged with this resolution.
thumbnail	If non-NULL, will be stored in TIFF as preview image.
imageResources	If non-NULL, will image resources be stored in TIFF as well.
metadataSubset	The subset of metadata (e.g., copyright only) to include in the TIFF.

References dng_negative::Metadata().

5.65.2.5 WriteTIFFWithProfile()

```
void dng_image_writer::WriteTIFFWithProfile (
            dng_host & host,
            dng_stream & stream,
            const dng_image & image,
            uint32 photometricInterpretation,
            uint32 compression,
            dng_negative * negative,
            const void * profileData = NULL,
            uint32 profileSize = 0,
            const dng_resolution * resolution = NULL,
            const dng_jpeg_preview * thumbnail = NULL,
            const dng_memory_block * imageResources = NULL,
            dng_metadata_subset metadataSubset = kMetadataSubset_All,
            bool hasTransparency = false )
```

Write a dng_image to a dng_stream in TIFF format.

Parameters

host	Host interface used for progress updates, abort testing, buffer allocation, etc.
stream	The dng_stream on which to write the TIFF.
image	The actual image data to be written.
photometricInterpretation	Either piBlackIsZero for monochrome or piRGB for RGB images.
compression	Must be ccUncompressed.
negative	or metadata If non-NULL, EXIF, IPTC, and XMP metadata from this negative is written to TIFF.
profileData	If non-null, TIFF will be tagged with this profile. No color space conversion of image data occurs.
profileSize	The size for the profile data.
resolution	If non-NULL, TIFF will be tagged with this resolution.
thumbnail	If non-NULL, will be stored in TIFF as preview image.
imageResources	If non-NULL, will image resources be stored in TIFF as well.
metadataSubset	The subset of metadata (e.g., copyright only) to include in the TIFF.

References dng_negative::Metadata().

The documentation for this class was generated from the following files:

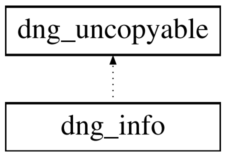
- dng_image_writer.h
- dng_image_writer.cpp

5.66 dng_info Class Reference

Top-level structure of DNG file with access to metadata.

```
#include <dng_info.h>
```

Inheritance diagram for dng_info:



Public Member Functions

- uint32 IFDCount () const

  Returns the number of parsed SubIFDs (including the main IFD).
- uint32 ChainedIFDCount () const

  Returns the number of chained IFDs.
- uint32 ChainedSubIFDCount (uint32 chainIndex) const

  Returns number SubIFDs for a chained IFD.
- virtual void Parse (dng_host &host, dng_stream &stream)
- virtual void PostParse (dng_host &host)

  Must be called immediately after a successful Parse operation.
- virtual bool IsValidDNG ()

Public Attributes

- uint64 **fTIFFBlockOffset**
- uint64 **fTIFFBlockOriginalOffset**
- bool **fBigEndian**
- uint32 **fMagic**
- AutoPtr< dng_exif > **fExif**
- AutoPtr< dng_shared > **fShared**
- int32 **fMainIndex**
- int32 **fMaskIndex**
- int32 **fDepthIndex**
- int32 **fEnhancedIndex**
- std::vector< dng_ifd ∗ > **fIFD**
- std::vector< dng_ifd ∗ > **fChainedIFD**
- std::vector< std::vector< dng_ifd ∗ > > **fChainedSubIFD**

Protected Member Functions

- virtual void **ValidateMagic** ()
- virtual void **ParseTag** (dng_host &host, dng_stream &stream, dng_exif ∗exif, dng_shared ∗shared, dng_ifd ∗ifd, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset, int64 offsetDelta)
- virtual bool **ValidateIFD** (dng_stream &stream, uint64 ifdOffset, int64 offsetDelta)
- virtual void **ParseIFD** (dng_host &host, dng_stream &stream, dng_exif ∗exif, dng_shared ∗shared, dng_ifd ∗ifd, uint64 ifdOffset, int64 offsetDelta, uint32 parentCode)
- virtual bool **ParseMakerNoteIFD** (dng_host &host, dng_stream &stream, uint64 ifdSize, uint64 ifdOffset, int64 offsetDelta, uint64 minOffset, uint64 maxOffset, uint32 parentCode)
- virtual void **ParseMakerNote** (dng_host &host, dng_stream &stream, uint32 makerNoteCount, uint64 maker↩NoteOffset, int64 offsetDelta, uint64 minOffset, uint64 maxOffset)
- virtual void **ParseSonyPrivateData** (dng_host &host, dng_stream &stream, uint64 count, uint64 oldOffset, uint64 newOffset)
- virtual void **ParseDNGPrivateData** (dng_host &host, dng_stream &stream)

Protected Attributes

- uint32 **fMakerNoteNextIFD**

5.66.1 Detailed Description

Top-level structure of DNG file with access to metadata.

See DNG 1.1.0 specification for information on member fields of this class.

5.66.2 Member Function Documentation

5.66.2.1 IsValidDNG()

```
bool dng_info::IsValidDNG ( )  [virtual]
```

Test validity of DNG data.

Return values

true	if stream provided a valid DNG.

References AutoPtr< T >::Get(), IFDCount(), and ReportError().

5.66.2.2 Parse()

```
void dng_info::Parse (
            dng_host & host,
            dng_stream & stream )  [virtual]
```

Read dng_info from a dng_stream

Parameters

host	DNG host used for progress updating, abort testing, buffer allocation, etc.
stream	Stream to read DNG data from.

References ChainedIFDCount(), AutoPtr< T >::Get(), dng_stream::Get_uint16(), dng_stream::Get_uint32(), IFD↩ Count(), dng_camera_profile::IsValid(), kMaxChainedIFDs, kMaxSubIFDs, dng_stream::Length(), dng_host::Make↩ _dng_exif(), dng_host::Make_dng_ifd(), dng_host::Make_dng_shared(), dng_camera_profile::Parse(), dng_stream::↩ Position(), dng_stream::PositionInOriginalFile(), ReportError(), ReportWarning(), AutoPtr< T >::Reset(), dng_stream↩ ::SetBigEndian(), dng_stream::SetLittleEndian(), dng_stream::SetReadPosition(), and ThrowBadFormat().

The documentation for this class was generated from the following files:

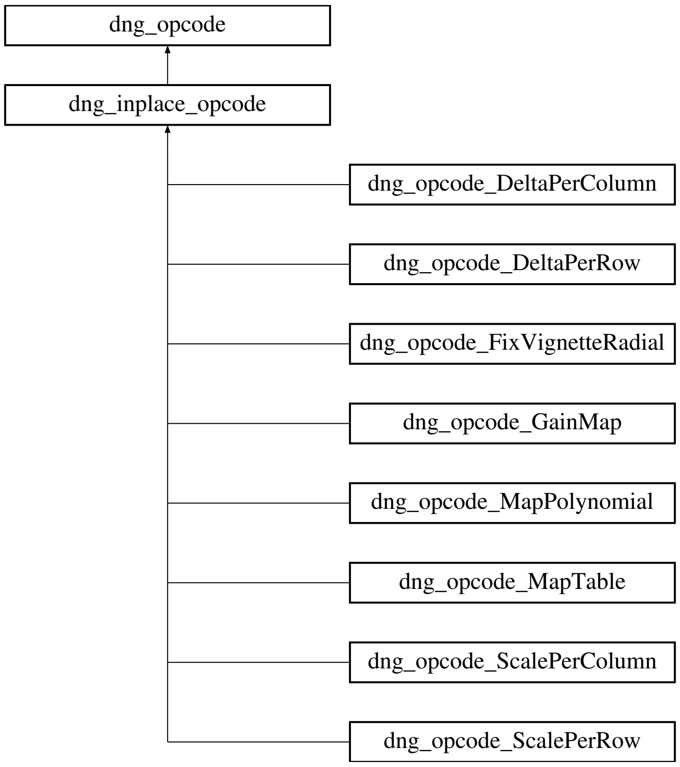
- dng_info.h
- dng_info.cpp

5.67 dng_inplace_opcode Class Reference

Class to represent an in-place (i.e., pointwise, per-pixel) opcode, such as a global tone curve.

```
#include <dng_opcodes.h>
```

Inheritance diagram for dng_inplace_opcode:



Public Member Functions

- virtual uint32 BufferPixelType (uint32 imagePixelType)

  The pixel data type of this opcode.
- virtual dng_rect ModifiedBounds (const dng_rect &imageBounds)
- virtual void Prepare (dng_negative &negative, uint32 threadCount, const dng_point &tileSize, const dng_rect &imageBounds, uint32 imagePlanes, uint32 bufferPixelType, dng_memory_allocator &allocator)
- virtual void ProcessArea (dng_negative &negative, uint32 threadIndex, dng_pixel_buffer &buffer, const dng_rect &dstArea, const dng_rect &imageBounds)=0
- virtual void Apply (dng_host &host, dng_negative &negative, AutoPtr< dng_image > &image)

  Apply this opcode to the specified image with associated negative.

Protected Member Functions

- **dng_inplace_opcode** (uint32 opcodeID, uint32 minVersion, uint32 flags)
- **dng_inplace_opcode** (uint32 opcodeID, dng_stream &stream, const char ∗name)

Additional Inherited Members

5.67.1 Detailed Description

Class to represent an in-place (i.e., pointwise, per-pixel) opcode, such as a global tone curve.

5.67.2 Member Function Documentation

5.67.2.1 ModifiedBounds()

```
virtual dng_rect dng_inplace_opcode::ModifiedBounds (
            const dng_rect & imageBounds )  [inline], [virtual]
```

The adjusted bounds (processing area) of this opcode. It is limited to the intersection of the specified image area and the GainMap area.

Reimplemented in dng_opcode_ScalePerColumn, dng_opcode_ScalePerRow, dng_opcode_DeltaPerColumn, dng_opcode_DeltaPerRow, dng_opcode_MapPolynomial, dng_opcode_MapTable, and dng_opcode_GainMap.

Referenced by Apply().

5.67.2.2 Prepare()

```
virtual void dng_inplace_opcode::Prepare (
            dng_negative & negative,
            uint32 threadCount,
            const dng_point & tileSize,
            const dng_rect & imageBounds,
            uint32 imagePlanes,
            uint32 bufferPixelType,
            dng_memory_allocator & allocator )  [inline], [virtual]
```

Startup method called before any processing is performed on pixel areas. It can be used to allocate (per-thread) memory and setup tasks.

Parameters

negative	The negative object to be processed.
threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
imageBounds	Total size of image to be processed.
imagePlanes	Number of planes in the image. Less than or equal to kMaxColorPlanes.
bufferPixelType	Pixel type of image buffer (see dng_tag_types.h).
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.

Reimplemented in dng_opcode_FixVignetteRadial, and dng_opcode_MapTable.

Referenced by dng_inplace_opcode_task::Start().

5.67.2.3 ProcessArea()

```
virtual void dng_inplace_opcode::ProcessArea (
            dng_negative & negative,
            uint32 threadIndex,
            dng_pixel_buffer & buffer,
            const dng_rect & dstArea,
            const dng_rect & imageBounds ) [pure virtual]
```

Implements image processing operation in a single buffer. The source pixels are provided as input to the buffer, and this routine calculates and writes the destination pixels to the same buffer. Ideally, no allocation should be done in this routine.

Parameters

negative	The negative associated with the pixels to be processed.
threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Prepare method.
buffer	Source and Destination pixels.
dstArea	Destination pixel processing area.
imageBounds	Total image area to be processed; dstArea will always lie within these bounds.

Implemented in dng_opcode_FixVignetteRadial, dng_opcode_ScalePerColumn, dng_opcode_ScalePerRow, dng_opcode_DeltaPerColumn, dng_opcode_DeltaPerRow, dng_opcode_MapPolynomial, dng_opcode_MapTable, and dng_opcode_GainMap.

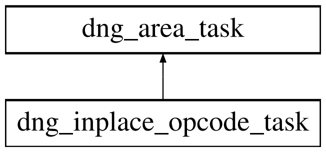
Referenced by dng_inplace_opcode_task::Process().

The documentation for this class was generated from the following files:

- dng_opcodes.h
- dng_opcodes.cpp

5.68 dng_inplace_opcode_task Class Reference

Inheritance diagram for dng_inplace_opcode_task:



Public Member Functions

- **dng_inplace_opcode_task** (dng_inplace_opcode &opcode, dng_negative &negative, dng_image &image)
- virtual void Start (uint32 threadCount, const dng_rect &, const dng_point &tileSize, dng_memory_allocator ∗allocator, dng_abort_sniffer ∗)
- virtual void Process (uint32 threadIndex, const dng_rect &tile, dng_abort_sniffer ∗)

    and progress updates.

Additional Inherited Members

5.68.1 Member Function Documentation

5.68.1.1 Process()

```
virtual void dng_inplace_opcode_task::Process (
             uint32 threadIndex,
             const dng_rect & tile,
             dng_abort_sniffer * sniffer )  [inline], [virtual]
```

and progress updates.

Process one tile or fully partitioned area. This method is overridden by derived classes to implement the actual image processing. Note that the sniffer can be ignored if it is certain that a processing task will complete very quickly. This method should never be called directly but rather accessed via Process. There is no allocator parameter as all allocation should be done in Start.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the Start method.)
tile	Area to process.
sniffer	dng_abort_sniffer to use to check for user cancellation

Implements dng_area_task.

References dng_image::Bounds(), dng_memory_block::Buffer(), dng_image::Get(), dng_image::Planes(), dng_↩ inplace_opcode::ProcessArea(), and dng_image::Put().

5.68.1.2 Start()

```
virtual void dng_inplace_opcode_task::Start (
             uint32 threadCount,
             const dng_rect & dstArea,
             const dng_point & tileSize,
             dng_memory_allocator * allocator,
             dng_abort_sniffer * sniffer )  [inline], [virtual]
```

Task startup method called before any processing is done on partitions. The Start method is called before any processing is done and can be overridden to allocate temporary buffers, etc.

Parameters

threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads.
dstArea	Area to be processed in the current run of the task.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.
sniffer	Sniffer to test for user cancellation and to set up progress.

Reimplemented from dng_area_task.

References dng_memory_allocator::Allocate(), dng_image::Bounds(), dng_image::Planes(), and dng_inplace_↩ opcode::Prepare().

The documentation for this class was generated from the following file:

- dng_opcodes.cpp

5.69 dng_iptc Class Reference

Class for reading and holding IPTC metadata associated with a DNG file.

```
#include <dng_iptc.h>
```

Public Member Functions

- bool IsEmpty () const
- bool NotEmpty () const
- void Parse (const void ∗blockData, uint32 blockSize, uint64 offsetInOriginalFile)
- dng_memory_block ∗ Spool (dng_memory_allocator &allocator, bool padForTIFF)

Public Attributes

- dng_string **fTitle**
- int32 **fUrgency**
- dng_string **fCategory**
- dng_string_list **fSupplementalCategories**
- dng_string_list **fKeywords**
- dng_string **fInstructions**
- dng_date_time_info **fDateTimeCreated**
- dng_date_time_info **fDigitalCreationDateTime**
- dng_string_list **fAuthors**
- dng_string **fAuthorsPosition**
- dng_string **fCity**
- dng_string **fState**
- dng_string **fCountry**
- dng_string **fCountryCode**
- dng_string **fLocation**
- dng_string **fTransmissionReference**
- dng_string **fHeadline**
- dng_string **fCredit**
- dng_string **fSource**
- dng_string **fCopyrightNotice**
- dng_string **fDescription**
- dng_string **fDescriptionWriter**

Protected Types

- enum **DataSet** {
  kRecordVersionSet = 0, **kObjectNameSet** = 5, **kUrgencySet** = 10, **kCategorySet** = 15,
  kSupplementalCategoriesSet = 20, **kKeywordsSet** = 25, **kSpecialInstructionsSet** = 40, **kDateCreatedSet** = 55,
  kTimeCreatedSet = 60, **kDigitalCreationDateSet** = 62, **kDigitalCreationTimeSet** = 63, **kBylineSet** = 80,
  kBylineTitleSet = 85, **kCitySet** = 90, **kSublocationSet** = 92, **kProvinceStateSet** = 95,
  kCountryCodeSet = 100, **kCountryNameSet** = 101, **kOriginalTransmissionReferenceSet** = 103, **kHeadline↩ Set** = 105,
  kCreditSet = 110, **kSourceSet** = 115, **kCopyrightNoticeSet** = 116, **kCaptionSet** = 120,
  kCaptionWriterSet = 122 }
- enum **CharSet** { **kCharSetUnknown** = 0, **kCharSetUTF8** = 1 }

Protected Member Functions

- void **ParseString** (dng_stream &stream, dng_string &s, CharSet charSet)
- void **SpoolString** (dng_stream &stream, const dng_string &s, uint8 dataSet, uint32 maxChars, CharSet charSet)

5.69.1 Detailed Description

Class for reading and holding IPTC metadata associated with a DNG file.

See the IPTC specification for information on member fields of this class.

5.69.2 Member Function Documentation

5.69.2.1 IsEmpty()

```
bool dng_iptc::IsEmpty ( ) const
```

Test if IPTC metadata exists.

Return values

true	if no IPTC metadata exists for this DNG.

References NotEmpty().

Referenced by NotEmpty().

5.69.2.2 NotEmpty()

```
bool dng_iptc::NotEmpty ( ) const  [inline]
```

Test if IPTC metadata exists.

Return values

true	if IPTC metadata exists for this DNG.

References IsEmpty().

Referenced by IsEmpty().

5.69.2.3 Parse()

```
void dng_iptc::Parse (
            const void * blockData,
            uint32 blockSize,
            uint64 offsetInOriginalFile )
```

Parse a complete block of IPTC data.

Parameters

blockData	The block of IPTC data.
blockSize	Size in bytes of data block.
offsetInOriginalFile	Used to enable certain file patching operations such as updating date/time in place.

References dng_memory_data::Buffer_char(), dng_stream::Get(), dng_stream::Get_int8(), dng_stream::Get_uint16(), dng_stream::Get_uint8(), dng_stream::Length(), dng_stream::Position(), dng_stream::SetBigEndian(), and dng_↩
stream::SetReadPosition().

5.69.2.4 Spool()

```
dng_memory_block * dng_iptc::Spool (
            dng_memory_allocator & allocator,
            bool padForTIFF )
```

Serialize IPTC data to a memory block.

Parameters

allocator	Memory allocator used to acquire memory block.
padForTIFF	Forces length of block to be a multiple of four bytes in accordance with TIFF standard.

Return values

Memory	block

References dng_stream::AsMemoryBlock(), DNG_ASSERT, dng_stream::Flush(), dng_stream::Length(), dng_stream↩
::Put(), dng_stream::Put_uint16(), dng_stream::Put_uint8(), and dng_stream::SetBigEndian().

The documentation for this class was generated from the following files:

- dng_iptc.h
- dng_iptc.cpp

5.70 dng_jpeg_image Class Reference

Public Member Functions

- uint32 **TilesAcross** () const
- uint32 **TilesDown** () const
- uint32 **TileCount** () const
- void **Encode** (dng_host &host, const dng_negative &negative, dng_image_writer &writer, const dng_image &image)
- dng_fingerprint **FindDigest** (dng_host &host) const

Public Attributes

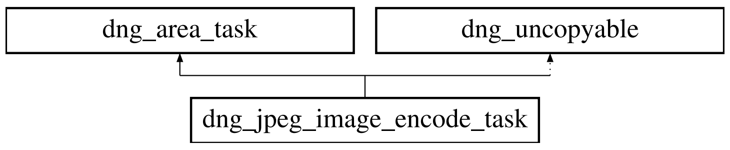
- dng_point **fImageSize**
- dng_point **fTileSize**
- bool **fUsesStrips**
- AutoPtr< dng_memory_block > **fJPEGTables**
- AutoArray< dng_jpeg_image_tile_ptr > **fJPEGData**

The documentation for this class was generated from the following files:

- dng_jpeg_image.h
- dng_jpeg_image.cpp

5.71 dng_jpeg_image_encode_task Class Reference

Inheritance diagram for dng_jpeg_image_encode_task:



Public Member Functions

- **dng_jpeg_image_encode_task** (dng_host &host, dng_image_writer &writer, const dng_image &image, dng_jpeg_image &jpegImage, uint32 tileCount, const dng_ifd &ifd)
- void Process (uint32, const dng_rect &, dng_abort_sniffer ∗sniffer)

    and progress updates.

Additional Inherited Members

5.71.1 Member Function Documentation

5.71.1.1 Process()

```
void dng_jpeg_image_encode_task::Process (
            uint32 threadIndex,
            const dng_rect & tile,
            dng_abort_sniffer ∗ sniffer )  [inline], [virtual]
```

and progress updates.

Process one tile or fully partitioned area. This method is overridden by derived classes to implement the actual image processing. Note that the sniffer can be ignored if it is certain that a processing task will complete very quickly. This method should never be called directly but rather accessed via Process. There is no allocator parameter as all allocation should be done in Start.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the Start method.)
tile	Area to process.
sniffer	dng_abort_sniffer to use to check for user cancellation

Implements dng_area_task.

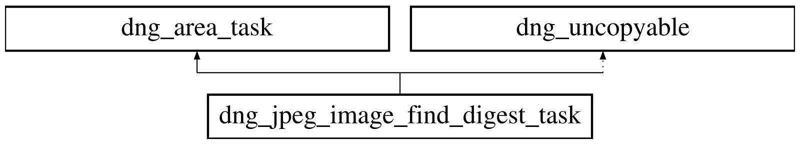
References dng_host::Allocate(), dng_host::Allocator(), AutoPtr< T >::Reset(), and dng_abort_sniffer::SniffForAbort().

The documentation for this class was generated from the following file:

- dng_jpeg_image.cpp

5.72 dng_jpeg_image_find_digest_task Class Reference

Inheritance diagram for dng_jpeg_image_find_digest_task:



Public Member Functions

- **dng_jpeg_image_find_digest_task** (const dng_jpeg_image &jpegImage, uint32 tileCount, dng_fingerprint ∗digests)
- void Process (uint32, const dng_rect &, dng_abort_sniffer ∗sniffer)

  and progress updates.

Additional Inherited Members

5.72.1 Member Function Documentation

5.72.1.1 Process()

```
void dng_jpeg_image_find_digest_task::Process (
            uint32 threadIndex,
            const dng_rect & tile,
            dng_abort_sniffer ∗ sniffer )  [inline], [virtual]
```

and progress updates.

Process one tile or fully partitioned area. This method is overridden by derived classes to implement the actual image processing. Note that the sniffer can be ignored if it is certain that a processing task will complete very quickly. This method should never be called directly but rather accessed via Process. There is no allocator parameter as all allocation should be done in Start.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the Start method.)
tile	Area to process.
sniffer	dng_abort_sniffer to use to check for user cancellation

Implements dng_area_task.

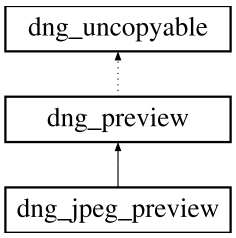
References dng_md5_printer::Process(), dng_md5_printer::Result(), and dng_abort_sniffer::SniffForAbort().

The documentation for this class was generated from the following file:

- dng_jpeg_image.cpp

5.73 dng_jpeg_preview Class Reference

Inheritance diagram for dng_jpeg_preview:



Public Member Functions

- virtual dng_basic_tag_set ∗ **AddTagSet** (dng_tiff_directory &directory) const
- virtual void **WriteData** (dng_host &host, dng_image_writer &writer, dng_basic_tag_set &basic, dng_stream &stream) const
- void **SpoolAdobeThumbnail** (dng_stream &stream) const

Public Attributes

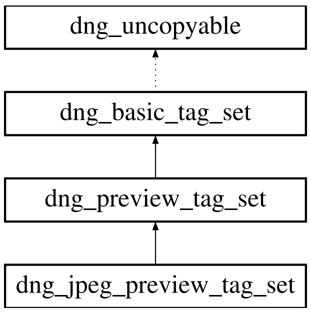
- dng_point **fPreviewSize**
- uint16 **fPhotometricInterpretation**
- dng_point **fYCbCrSubSampling**
- uint16 **fYCbCrPositioning**
- AutoPtr< dng_memory_block > **fCompressedData**

The documentation for this class was generated from the following files:

- dng_preview.h
- dng_preview.cpp

5.74 dng_jpeg_preview_tag_set Class Reference

Inheritance diagram for dng_jpeg_preview_tag_set:



Public Member Functions

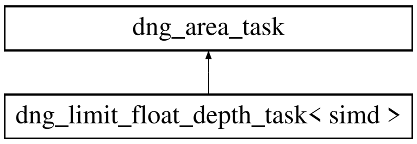
- **dng_jpeg_preview_tag_set** (dng_tiff_directory &directory, const dng_jpeg_preview &preview, const dng_ifd &ifd)

The documentation for this class was generated from the following file:

- dng_preview.cpp

5.75 dng_limit_float_depth_task< simd > Class Template Reference

Inheritance diagram for dng_limit_float_depth_task< simd >:



Public Member Functions

- **dng_limit_float_depth_task** (const dng_image &srcImage, dng_image &dstImage, uint32 bitDepth, real32 scale)
- virtual dng_rect RepeatingTile1 () const
- virtual dng_rect RepeatingTile2 () const
- virtual void Process (uint32 threadIndex, const dng_rect &tile, dng_abort_sniffer ∗sniffer)

    and progress updates.

Additional Inherited Members

5.75.1 Member Function Documentation

5.75.1.1 Process()

```
template<SIMDType simd>
void dng_limit_float_depth_task< simd >::Process (
            uint32 threadIndex,
            const dng_rect & tile,
            dng_abort_sniffer ∗ sniffer )  [virtual]
```

and progress updates.

Process one tile or fully partitioned area. This method is overridden by derived classes to implement the actual image processing. Note that the sniffer can be ignored if it is certain that a processing task will complete very quickly. This method should never be called directly but rather accessed via Process. There is no allocator parameter as all allocation should be done in Start.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the Start method.)
tile	Area to process.
sniffer	dng_abort_sniffer to use to check for user cancellation

Implements dng_area_task.

References dng_pixel_buffer::ConstPixel(), dng_pixel_buffer::DirtyPixel(), and OptimizeOrder().

5.75.1.2 RepeatingTile1()

```
template<SIMDType simd>
virtual dng_rect dng_limit_float_depth_task< simd >::RepeatingTile1 ( ) const  [inline], [virtual]
```

Getter for RepeatingTile1. RepeatingTile1, RepeatingTile2, and RepeatingTile3 are used to establish a set of 0 to 3 tile patterns for which the resulting partitions that the final Process method is called on will not cross tile boundaries in any of the tile patterns. This can be used for a processing routine that needs to read from two tiles and write to a third such that all the tiles are aligned and sized in a certain way. A RepeatingTile value is valid if it is non-empty. Higher numbered RepeatingTile patterns are only used if all lower ones are non-empty. A RepeatingTile pattern must be a multiple of UnitCell in size for all constraints of the partitionerr to be met.

Reimplemented from dng_area_task.

References dng_image::RepeatingTile().

5.75.1.3 RepeatingTile2()

```
template<SIMDType simd>
virtual dng_rect dng_limit_float_depth_task< simd >::RepeatingTile2 ( ) const  [inline], [virtual]
```

Getter for RepeatingTile2. RepeatingTile1, RepeatingTile2, and RepeatingTile3 are used to establish a set of 0 to 3 tile patterns for which the resulting partitions that the final Process method is called on will not cross tile boundaries in any of the tile patterns. This can be used for a processing routine that needs to read from two tiles and write to a third such that all the tiles are aligned and sized in a certain way. A RepeatingTile value is valid if it is non-empty. Higher numbered RepeatingTile patterns are only used if all lower ones are non-empty. A RepeatingTile pattern must be a multiple of UnitCell in size for all constraints of the partitionerr to be met.

Reimplemented from dng_area_task.

References dng_image::RepeatingTile().

The documentation for this class was generated from the following file:

- dng_utils.cpp

5.76 dng_linearization_info Class Reference

Class for managing data values related to DNG linearization.

```
#include <dng_linearization_info.h>
```

Public Member Functions

- void **RoundBlacks** ()
- virtual void **Parse** (dng_host &host, dng_stream &stream, dng_info &info)
- virtual void **PostParse** (dng_host &host, dng_negative &negative)
- real64 MaxBlackLevel (uint32 plane) const
- virtual void Linearize (dng_host &host, dng_negative &negative, const dng_image &srcImage, dng_image &dst←Image)
- dng_urational BlackLevel (uint32 row, uint32 col, uint32 plane) const
- uint32 RowBlackCount () const

  Number of per-row black level deltas in fBlackDeltaV.
- dng_srational RowBlack (uint32 row) const
- uint32 ColumnBlackCount () const

  Number of per-column black level deltas in fBlackDeltaV.
- dng_srational ColumnBlack (uint32 col) const

Public Attributes

- dng_rect fActiveArea
- uint32 fMaskedAreaCount

  Number of rectangles in fMaskedArea.
- dng_rect fMaskedArea [kMaxMaskedAreas]
- AutoPtr< dng_memory_block > fLinearizationTable
- uint32 fBlackLevelRepeatRows

  Actual number of rows in fBlackLevel pattern.
- uint32 fBlackLevelRepeatCols

  Actual number of columns in fBlackLevel pattern.
- real64 fBlackLevel [kMaxBlackPattern][kMaxBlackPattern][kMaxSamplesPerPixel]

  Repeating pattern of black level deltas fBlackLevelRepeatRows by fBlackLevelRepeatCols in size.
- AutoPtr< dng_memory_block > fBlackDeltaH

  Memory block of double-precision floating point deltas between baseline black level and a given column's black level.
- AutoPtr< dng_memory_block > fBlackDeltaV

  Memory block of double-precision floating point deltas between baseline black level and a given row's black level.
- real64 fWhiteLevel [kMaxSamplesPerPixel]

  Single white level (maximum sensor value) for each sample plane.

Protected Attributes

- int32 **fBlackDenom**

5.76.1 Detailed Description

Class for managing data values related to DNG linearization.

See LinearizationTable, BlackLevel, BlackLevelRepeatDim, BlackLevelDeltaH, BlackLevelDeltaV and WhiteLevel tags in the DNG 1.1.0 specification.

5.76.2 Member Function Documentation

5.76.2.1 BlackLevel()

```
dng_urational dng_linearization_info::BlackLevel (
            uint32 row,
            uint32 col,
            uint32 plane ) const
```

Compute black level for one coordinate and sample plane in the image.

Parameters

row	Row to compute black level for.
col	Column to compute black level for.
plane	Sample plane to compute black level for.

References fBlackLevel.

5.76.2.2 ColumnBlack()

```
dng_srational dng_linearization_info::ColumnBlack (
            uint32 col ) const
```

Lookup black level delta for a given column.

Parameters

col	Column to get black level for.

Return values

black	level for indicated column.

References dng_memory_block::Buffer_real64(), fBlackDeltaH, and AutoPtr< T >::Get().

5.76.2.3 Linearize()

```
void dng_linearization_info::Linearize (
            dng_host & host,
            dng_negative & negative,
            const dng_image & srcImage,
            dng_image & dstImage )  [virtual]
```

Convert raw data from in-file format to a true linear image using linearization data from DNG.

Parameters

host	Used to allocate buffers, check for aborts, and post progress updates.
negative	Used to remember preserved black point.
srcImage	Input pre-linearization RAW samples.
dstImage	Output linearized image.

References fActiveArea, fWhiteLevel, kMaxStage3BlackLevelNormalized, MaxBlackLevel(), dng_host::PerformAreaTask(), dng_image::PixelType(), and dng_image::Planes().

5.76.2.4 MaxBlackLevel()

```
real64 dng_linearization_info::MaxBlackLevel (
            uint32 plane ) const
```

Compute the maximum black level for a given sample plane taking into account base black level, repeated black level patter, and row/column delta maps.

References dng_memory_block::Buffer_real64(), DNG_REQUIRE, fBlackDeltaH, fBlackDeltaV, fBlackLevel, fBlackLevelRepeatCols, fBlackLevelRepeatRows, AutoPtr< T >::Get(), kMaxBlackPattern, and dng_memory_block::LogicalSize().

Referenced by Linearize().

5.76.2.5 RowBlack()

```
dng_srational dng_linearization_info::RowBlack (
            uint32 row ) const
```

Lookup black level delta for a given row.

Parameters

row	Row to get black level for.

Return values

black	level for indicated row.

References dng_memory_block::Buffer_real64(), fBlackDeltaV, and AutoPtr< T >::Get().

5.76.3 Member Data Documentation

5.76.3.1 fActiveArea

```
dng_rect dng_linearization_info::fActiveArea
```

This rectangle defines the active (non-masked) pixels of the sensor. The order of the rectangle coordinates is: top, left, bottom, right.

Referenced by Linearize().

5.76.3.2 fLinearizationTable

```
AutoPtr<dng_memory_block> dng_linearization_info::fLinearizationTable
```

A lookup table that maps stored values into linear values. This tag is typically used to increase compression ratios by storing the raw data in a non-linear, more visually uniform space with fewer total encoding levels. If SamplesPerPixel is not equal to one, e.g. Fuji S3 type sensor, this single table applies to all the samples for each pixel.

5.76.3.3 fMaskedArea

```
dng_rect dng_linearization_info::fMaskedArea[kMaxMaskedAreas]
```

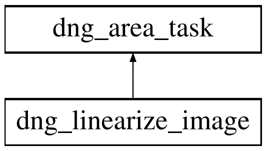
List of non-overlapping rectangle coordinates of fully masked pixels. Can be optionally used by DNG readers to measure the black encoding level. The order of each rectangle's coordinates is: top, left, bottom, right. If the raw image data has already had its black encoding level subtracted, then this tag should not be used, since the masked pixels are no longer useful. Note that DNG writers are still required to include an estimate and store the black encoding level using the black level DNG tags. Support for the MaskedAreas tag is not required of DNG readers.

The documentation for this class was generated from the following files:

- dng_linearization_info.h
- dng_linearization_info.cpp

5.77 dng_linearize_image Class Reference

Inheritance diagram for dng_linearize_image:



Public Member Functions

- **dng_linearize_image** (dng_host &host, dng_linearization_info &info, uint16 dstBlackLevel, bool forceClipBlack←Level, const dng_image &srcImage, dng_image &dstImage)
- virtual dng_rect RepeatingTile1 () const
- virtual dng_rect RepeatingTile2 () const
- virtual void Process (uint32 threadIndex, const dng_rect &tile, dng_abort_sniffer ∗sniffer)

    and progress updates.

Additional Inherited Members

5.77.1 Member Function Documentation

5.77.1.1 Process()

```
void dng_linearize_image::Process (
            uint32 threadIndex,
            const dng_rect & tile,
            dng_abort_sniffer * sniffer )  [virtual]
```

and progress updates.

Process one tile or fully partitioned area. This method is overridden by derived classes to implement the actual image processing. Note that the sniffer can be ignored if it is certain that a processing task will complete very quickly. This method should never be called directly but rather accessed via Process. There is no allocator parameter as all allocation should be done in Start.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the Start method.)
tile	Area to process.
sniffer	dng_abort_sniffer to use to check for user cancellation

Implements dng_area_task.

References dng_image::Planes().

5.77.1.2 RepeatingTile1()

```
dng_rect dng_linearize_image::RepeatingTile1 ( ) const  [virtual]
```

Getter for RepeatingTile1. RepeatingTile1, RepeatingTile2, and RepeatingTile3 are used to establish a set of 0 to 3 tile patterns for which the resulting partitions that the final Process method is called on will not cross tile boundaries in any of the tile patterns. This can be used for a processing routine that needs to read from two tiles and write to a third such that all the tiles are aligned and sized in a certain way. A RepeatingTile value is valid if it is non-empty. Higher numbered RepeatingTile patterns are only used if all lower ones are non-empty. A RepeatingTile pattern must be a multiple of UnitCell in size for all constraints of the partitionerr to be met.

Reimplemented from dng_area_task.

References dng_image::RepeatingTile().

5.77.1.3 RepeatingTile2()

```
dng_rect dng_linearize_image::RepeatingTile2 ( ) const  [virtual]
```

Getter for RepeatingTile2. RepeatingTile1, RepeatingTile2, and RepeatingTile3 are used to establish a set of 0 to 3 tile patterns for which the resulting partitions that the final Process method is called on will not cross tile boundaries in any of the tile patterns. This can be used for a processing routine that needs to read from two tiles and write to a third such that all the tiles are aligned and sized in a certain way. A RepeatingTile value is valid if it is non-empty. Higher numbered RepeatingTile patterns are only used if all lower ones are non-empty. A RepeatingTile pattern must be a multiple of UnitCell in size for all constraints of the partitionerr to be met.

Reimplemented from dng_area_task.

References dng_image::RepeatingTile().

The documentation for this class was generated from the following file:

- dng_linearization_info.cpp

5.78 dng_linearize_plane Class Reference

Public Member Functions

- **dng_linearize_plane** (dng_host &host, dng_linearization_info &info, uint16 dstBlackLevel, bool forceClipBlack↩ Level, const dng_image &srcImage, dng_image &dstImage, uint32 plane)
- void **Process** (const dng_rect &tile)

The documentation for this class was generated from the following file:

- dng_linearization_info.cpp

5.79 dng_local_string Class Reference

Public Member Functions

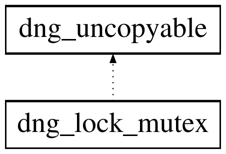
- **dng_local_string** (const dng_string &s)
- void **Clear** ()
- void **SetDefaultText** (const dng_string &s)
- void **AddTranslation** (const dng_string &language, const dng_string &translation)
- void **Set** (const char ∗s)
- const dng_string & **DefaultText** () const
- dng_string & **DefaultText** ()
- uint32 **TranslationCount** () const
- const dng_string & **Language** (uint32 index) const
- const dng_string & **Translation** (uint32 index) const
- const dng_string & **LocalText** (const dng_string &locale) const
- bool **IsEmpty** () const
- bool **NotEmpty** () const
- bool **operator==** (const dng_local_string &s) const
- bool **operator!=** (const dng_local_string &s) const
- void **Truncate** (uint32 maxBytes)

The documentation for this class was generated from the following files:

- dng_local_string.h
- dng_local_string.cpp

5.80 dng_lock_mutex Class Reference

Inheritance diagram for dng_lock_mutex:



Public Member Functions

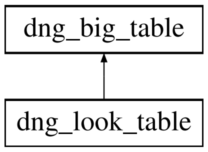
- **dng_lock_mutex** (dng_mutex ∗mutex)
- **dng_lock_mutex** (dng_mutex &mutex)

The documentation for this class was generated from the following files:

- dng_mutex.h
- dng_mutex.cpp

5.81 dng_look_table Class Reference

Inheritance diagram for dng_look_table:



Public Types

- enum { **kMaxTotalSamples** = 36 ∗ 32 ∗ 16, **kMaxHueSamples** = 360, **kMaxSatSamples** = 256, **kMaxVal**↩ **Samples** = 256 }

Public Member Functions

- **dng_look_table** (const dng_look_table &table)
- dng_look_table & **operator=** (const dng_look_table &table)
- bool **operator==** (const dng_look_table &table) const
- bool **operator!=** (const dng_look_table &table) const
- void **Set** (const dng_hue_sat_map &map, uint32 encoding)
- virtual bool **IsValid** () const
- void **SetInvalid** ()
- real64 **MinAmount** () const
- real64 **MaxAmount** () const
- void **SetAmountRange** (real64 minAmount, real64 maxAmount)
- real64 **Amount** () const
- void **SetAmount** (real64 amount)
- const dng_hue_sat_map & **Map** () const
- uint32 **Encoding** () const
- bool **Monochrome** () const

Protected Member Functions

- virtual void **GetStream** (dng_stream &stream)
- virtual void **PutStream** (dng_stream &stream, bool forFingerprint) const

Friends

- class **dng_look_table_cache**

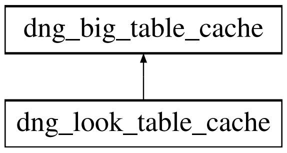
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_big_table.h
- dng_big_table.cpp

5.82 dng_look_table_cache Class Reference

Inheritance diagram for dng_look_table_cache:



Public Member Functions

- virtual void **InsertTableData** (dng_lock_std_mutex &, const dng_big_table &table)
- virtual void **EraseTableData** (dng_lock_std_mutex &, const dng_fingerprint &fingerprint)
- virtual void **ExtractTableData** (dng_lock_std_mutex &, const dng_fingerprint &fingerprint, dng_big_table &table)

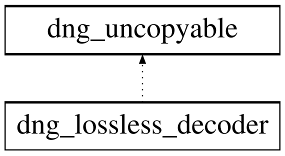
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_big_table.cpp

5.83 dng_lossless_decoder Class Reference

Inheritance diagram for dng_lossless_decoder:



Public Member Functions

- **dng_lossless_decoder** (dng_stream ∗stream, dng_spooler ∗spooler, bool bug16)
- void **StartRead** (uint32 &imageWidth, uint32 &imageHeight, uint32 &imageChannels)
- void **FinishRead** ()
- bool **IsHasselblad3FR** ()

The documentation for this class was generated from the following file:

- dng_lossless_jpeg.cpp

5.84 dng_lossless_encoder Class Reference

Public Member Functions

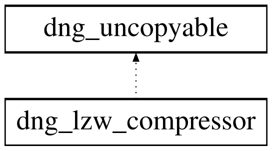
- **dng_lossless_encoder** (const uint16 ∗srcData, uint32 srcRows, uint32 srcCols, uint32 srcChannels, uint32 srcBitDepth, int32 srcRowStep, int32 srcColStep, dng_stream &stream)
- void **Encode** ()

The documentation for this class was generated from the following file:

- dng_lossless_jpeg.cpp

5.85 dng_lzw_compressor Class Reference

Inheritance diagram for dng_lzw_compressor:



Public Member Functions

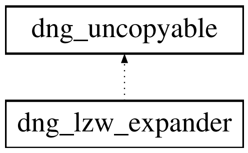
- void **Compress** (const uint8 ∗sPtr, uint8 ∗dPtr, uint32 sCount, uint32 &dCount)

The documentation for this class was generated from the following file:

- dng_image_writer.cpp

5.86 dng_lzw_expander Class Reference

Inheritance diagram for dng_lzw_expander:



Public Member Functions

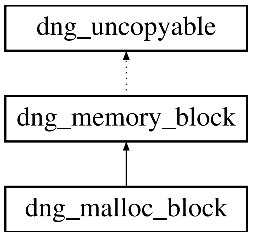
- bool **Expand** (const uint8 ∗sPtr, uint8 ∗dPtr, int32 sCount, int32 dCount)

The documentation for this class was generated from the following file:

- dng_read_image.cpp

5.87 dng_malloc_block Class Reference

Inheritance diagram for dng_malloc_block:



Public Member Functions

- **dng_malloc_block** (uint32 logicalSize)

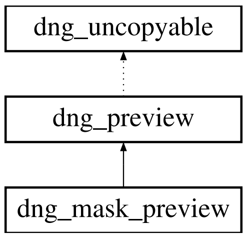
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_memory.h
- dng_memory.cpp

5.88 dng_mask_preview Class Reference

Inheritance diagram for dng_mask_preview:



Public Member Functions

- virtual dng_basic_tag_set ∗ **AddTagSet** (dng_tiff_directory &directory) const
- virtual void **WriteData** (dng_host &host, dng_image_writer &writer, dng_basic_tag_set &basic, dng_stream &stream) const

Public Attributes

- AutoPtr< dng_image > **fImage**
- int32 **fCompressionQuality**

The documentation for this class was generated from the following files:

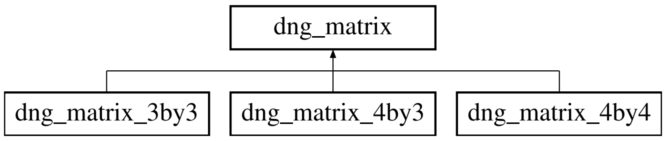
- dng_preview.h
- dng_preview.cpp

5.89 dng_matrix Class Reference

Class to represent 2D matrix up to kMaxColorPlanes x kMaxColorPlanes in size.

```
#include <dng_matrix.h>
```

Inheritance diagram for dng_matrix:



Public Member Functions

- **dng_matrix** (uint32 rows, uint32 cols)
- **dng_matrix** (const dng_matrix &m)
- void **Clear** ()
- void **SetIdentity** (uint32 count)
- uint32 **Rows** () const
- uint32 **Cols** () const
- real64 ∗ **operator [ ]** (uint32 row)
- const real64 ∗ **operator [ ]** (uint32 row) const
- bool **operator==** (const dng_matrix &m) const
- bool **operator!=** (const dng_matrix &m) const
- bool **IsEmpty** () const
- bool **NotEmpty** () const
- bool **IsDiagonal** () const
- bool **IsIdentity** () const
- real64 **MaxEntry** () const
- real64 **MinEntry** () const
- void **Scale** (real64 factor)
- void **Round** (real64 factor)
- void **SafeRound** (real64 factor)
- bool **AlmostEqual** (const dng_matrix &m, real64 slop=1.0e-8) const
- bool **AlmostIdentity** (real64 slop=1.0e-8) const

Protected Attributes

- uint32 **fRows**
- uint32 **fCols**
- real64 **fData** [kMaxColorPlanes][kMaxColorPlanes]

5.89.1 Detailed Description

Class to represent 2D matrix up to kMaxColorPlanes x kMaxColorPlanes in size.

The documentation for this class was generated from the following files:

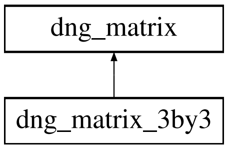
- dng_matrix.h
- dng_matrix.cpp

5.90 dng_matrix_3by3 Class Reference

A 3x3 matrix.

```
#include <dng_matrix.h>
```

Inheritance diagram for dng_matrix_3by3:



Public Member Functions

- **dng_matrix_3by3** (const dng_matrix &m)
- **dng_matrix_3by3** (real64 a00, real64 a01, real64 a02, real64 a10, real64 a11, real64 a12, real64 a20, real64 a21, real64 a22)
- **dng_matrix_3by3** (real64 a00, real64 a11, real64 a22)

Additional Inherited Members

5.90.1 Detailed Description

A 3x3 matrix.

The documentation for this class was generated from the following files:

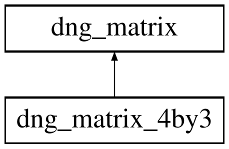
- dng_matrix.h
- dng_matrix.cpp

5.91 dng_matrix_4by3 Class Reference

A 4x3 matrix. Handy for working with 4-color cameras.

```
#include <dng_matrix.h>
```

Inheritance diagram for dng_matrix_4by3:



Public Member Functions

- **dng_matrix_4by3** (const dng_matrix &m)
- **dng_matrix_4by3** (real64 a00, real64 a01, real64 a02, real64 a10, real64 a11, real64 a12, real64 a20, real64 a21, real64 a22, real64 a30, real64 a31, real64 a32)

Additional Inherited Members

5.91.1 Detailed Description

A 4x3 matrix. Handy for working with 4-color cameras.

The documentation for this class was generated from the following files:

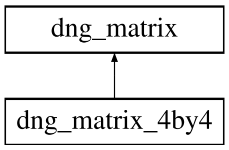
- dng_matrix.h
- dng_matrix.cpp

5.92 dng_matrix_4by4 Class Reference

A 4x4 matrix. Handy for GPU APIs.

```
#include <dng_matrix.h>
```

Inheritance diagram for dng_matrix_4by4:



Public Member Functions

- **dng_matrix_4by4** (const dng_matrix &m)
- **dng_matrix_4by4** (real64 a00, real64 a01, real64 a02, real64 a03, real64 a10, real64 a11, real64 a12, real64 a13, real64 a20, real64 a21, real64 a22, real64 a23, real64 a30, real64 a31, real64 a32, real64 a33)
- **dng_matrix_4by4** (real64 a00, real64 a11, real64 a22, real64 a33)

Additional Inherited Members

5.92.1 Detailed Description

A 4x4 matrix. Handy for GPU APIs.

The documentation for this class was generated from the following files:

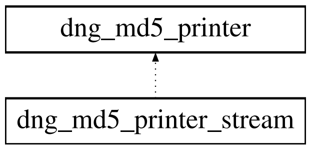
- dng_matrix.h
- dng_matrix.cpp

5.93 dng_md5_printer Class Reference

Class to hash binary data to a fingerprint using the MD5 Message-Digest Algorithm.

```
#include <dng_fingerprint.h>
```

Inheritance diagram for dng_md5_printer:



Public Member Functions

- void Reset ()

  Reset the fingerprint.
- void Process (const void ∗data, uint32 inputLen)
- void Process (const char ∗text)
- const dng_fingerprint & Result ()

  Get the fingerprint (i.e., result of the hash).

5.93.1 Detailed Description

Class to hash binary data to a fingerprint using the MD5 Message-Digest Algorithm.

5.93.2 Member Function Documentation

5.93.2.1 Process() [1/2]

```
void dng_md5_printer::Process (
             const void * data,
             uint32 inputLen )
```

Append the data to the stream to be hashed.

Parameters

data	The data to be hashed.
inputLen	The length of data, in bytes.

References DNG_ASSERT.

Referenced by dng_hue_sat_map::Interpolate(), Process(), dng_jpeg_image_find_digest_task::Process(), and Result().

5.93.2.2 Process() [2/2]

```
void dng_md5_printer::Process (
             const char * text )  [inline]
```

Append the string to the stream to be hashed.

Parameters

text	The string to be hashed.

References Process().

The documentation for this class was generated from the following files:

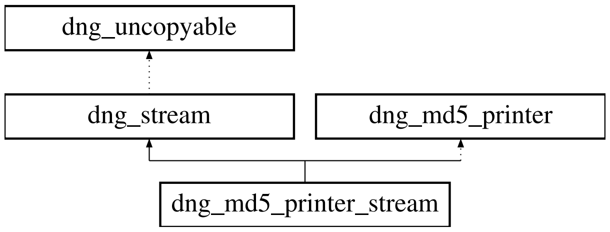
- dng_fingerprint.h
- dng_fingerprint.cpp

5.94 dng_md5_printer_stream Class Reference

A dng_stream based interface to the MD5 printing logic.

```
#include <dng_fingerprint.h>
```

Inheritance diagram for dng_md5_printer_stream:



Public Member Functions

- dng_md5_printer_stream ()

    Create an empty MD5 printer stream.
- virtual uint64 **DoGetLength** ()
- virtual void **DoRead** (void ∗, uint32, uint64)
- virtual void **DoSetLength** (uint64 length)
- virtual void **DoWrite** (const void ∗data, uint32 count2, uint64 offset)
- const dng_fingerprint & **Result** ()

Additional Inherited Members

5.94.1 Detailed Description

A dng_stream based interface to the MD5 printing logic.

The documentation for this class was generated from the following file:

- dng_fingerprint.h

5.95 dng_memory_allocator Class Reference

Interface for dng_memory_block allocator.

```
#include <dng_memory.h>
```

Public Member Functions

- virtual dng_memory_block ∗ Allocate (uint32 size)
- virtual void ∗ Malloc (size_t size)
- virtual void Free (void ∗ptr)

5.95.1 Detailed Description

Interface for dng_memory_block allocator.

5.95.2 Member Function Documentation

5.95.2.1 Allocate()

```
dng_memory_block * dng_memory_allocator::Allocate (
             uint32 size )  [virtual]
```

Allocate a dng_memory block.

Parameters

size	Number of bytes in memory block.

Return values

A	dng_memory_block with at least size bytes of valid storage.

Exceptions

dng_exception	with fErrorCode equal to dng_error_memory.

References ThrowMemoryFull().

Referenced by dng_host::Allocate(), dng_stream::AsMemoryBlock(), dng_1d_table::Initialize(), dng_opcode_Map↩
Table::Prepare(), dng_opcode_FixVignetteRadial::Prepare(), dng_filter_task::Start(), dng_inplace_opcode_task::↩
Start(), dng_resample_task::Start(), and dng_find_new_raw_image_digest_task::Start().

5.95.2.2 Free()

```
void dng_memory_allocator::Free (
             void * ptr )  [virtual]
```

Free the specified block of memory previously allocated with Malloc. Default implementation uses standard library 'free' routine.

5.95.2.3 Malloc()

```
void * dng_memory_allocator::Malloc (
             size_t size )  [virtual]
```

Directly allocate a block of at least 'size' bytes.

Parameters

size	Number of bytes in memory block.

Return values

A	pointer to a contiguous block of memory with at least size bytes of valid storage. Caller is responsible for freeing the memory with Free. Default implementation uses standard library 'malloc' routine.

The documentation for this class was generated from the following files:

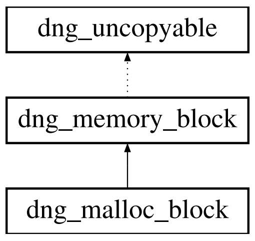
- dng_memory.h
- dng_memory.cpp

5.96 dng_memory_block Class Reference

Class to provide resource acquisition is instantiation discipline for image buffers and other larger memory allocations.

```
#include <dng_memory.h>
```

Inheritance diagram for dng_memory_block:



Public Member Functions

- dng_memory_block ∗ **Clone** (dng_memory_allocator &allocator) const
- uint32 LogicalSize () const
- void ∗ Buffer ()
- const void ∗ Buffer () const
- char ∗ Buffer_char ()
- const char ∗ Buffer_char () const
- uint8 ∗ Buffer_uint8 ()
- const uint8 ∗ Buffer_uint8 () const
- uint16 ∗ Buffer_uint16 ()
- const uint16 ∗ Buffer_uint16 () const
- int16 ∗ Buffer_int16 ()
- const int16 ∗ Buffer_int16 () const
- uint32 ∗ Buffer_uint32 ()
- const uint32 ∗ Buffer_uint32 () const
- int32 ∗ Buffer_int32 ()
- const int32 ∗ Buffer_int32 () const
- real32 ∗ Buffer_real32 ()
- const real32 ∗ Buffer_real32 () const
- real64 ∗ Buffer_real64 ()
- const real64 ∗ Buffer_real64 () const

Protected Member Functions

- **dng_memory_block** (uint32 logicalSize)
- uint32 **PhysicalSize** ()
- void **SetBuffer** (void ∗p)

5.96.1 Detailed Description

Class to provide resource acquisition is instantiation discipline for image buffers and other larger memory allocations.

This class requires a dng_memory_allocator for allocation.

5.96.2 Member Function Documentation

5.96.2.1 Buffer() [1/2]

```
void* dng_memory_block::Buffer ( )  [inline]
```

Return pointer to allocated memory as a void ∗..

Return values

void	∗ valid for as many bytes as were allocated.

Referenced by Buffer_char(), Buffer_int16(), Buffer_int32(), Buffer_real32(), Buffer_real64(), Buffer_uint16(), Buffer↩ _uint32(), Buffer_uint8(), dng_inplace_opcode_task::Process(), dng_find_new_raw_image_digest_task::Process(), and dng_opcode_Unknown::PutData().

5.96.2.2 Buffer() [2/2]

```
const void* dng_memory_block::Buffer ( ) const  [inline]
```

Return pointer to allocated memory as a const void ∗.

Return values

const	void ∗ valid for as many bytes as were allocated.

5.96.2.3 Buffer_char() [1/2]

```
char* dng_memory_block::Buffer_char ( )  [inline]
```

Return pointer to allocated memory as a char ∗.

Return values

char	∗ valid for as many bytes as were allocated.

References Buffer().

5.96.2.4 Buffer_char() [2/2]

```
const char* dng_memory_block::Buffer_char ( ) const  [inline]
```

Return pointer to allocated memory as a const char ∗.

Return values

const	char ∗ valid for as many bytes as were allocated.

References Buffer().

5.96.2.5 Buffer_int16() [1/2]

```
int16* dng_memory_block::Buffer_int16 ( )  [inline]
```

Return pointer to allocated memory as a int16 ∗.

Return values

int16	∗ valid for as many bytes as were allocated.

References Buffer().

5.96.2.6 Buffer_int16() [2/2]

```
const int16* dng_memory_block::Buffer_int16 ( ) const  [inline]
```

Return pointer to allocated memory as a const int16 ∗.

Return values

const	int16 ∗ valid for as many bytes as were allocated.

References Buffer().

5.96.2.7 Buffer_int32() [1/2]

```
int32* dng_memory_block::Buffer_int32 ( )  [inline]
```

Return pointer to allocated memory as a int32 ∗.

Return values

int32	∗ valid for as many bytes as were allocated.

References Buffer().

5.96.2.8 Buffer_int32() [2/2]

```
const int32* dng_memory_block::Buffer_int32 ( ) const  [inline]
```

Return pointer to allocated memory as a const int32 ∗.

Return values

const	int32 ∗ valid for as many bytes as were allocated.

References Buffer().

5.96.2.9 Buffer_real32() [1/2]

```
real32* dng_memory_block::Buffer_real32 ( )  [inline]
```

Return pointer to allocated memory as a real32 ∗.

Return values

real32	∗ valid for as many bytes as were allocated.

References Buffer().

Referenced by dng_gain_map::Entry(), dng_1d_table::Initialize(), dng_opcode_DeltaPerRow::ProcessArea(), dng↩_opcode_DeltaPerColumn::ProcessArea(), dng_opcode_ScalePerRow::ProcessArea(), dng_opcode_ScalePer↩Column::ProcessArea(), dng_resample_task::ProcessArea(), dng_render_task::ProcessArea(), dng_opcode_Delta↩PerRow::PutData(), dng_opcode_DeltaPerColumn::PutData(), dng_opcode_ScalePerRow::PutData(), and dng_↩opcode_ScalePerColumn::PutData().

5.96.2.10 Buffer_real32() [2/2]

```
const real32* dng_memory_block::Buffer_real32 ( ) const  [inline]
```

Return pointer to allocated memory as a const real32 ∗.

Return values

const	real32 ∗ valid for as many bytes as were allocated.

References Buffer().

5.96.2.11 Buffer_real64() [1/2]

```
real64* dng_memory_block::Buffer_real64 ( )  [inline]
```

Return pointer to allocated memory as a real64 ∗.

Return values

real64	∗ valid for as many bytes as were allocated.

References Buffer().

Referenced by dng_linearization_info::ColumnBlack(), dng_linearization_info::MaxBlackLevel(), and dng_linearization↩_info::RowBlack().

5.96.2.12 Buffer_real64() [2/2]

```
const real64* dng_memory_block::Buffer_real64 ( ) const  [inline]
```

Return pointer to allocated memory as a const real64 ∗.

Return values

const	real64 ∗ valid for as many bytes as were allocated.

References Buffer().

5.96.2.13 Buffer_uint16() [1/2]

```
uint16* dng_memory_block::Buffer_uint16 ( )  [inline]
```

Return pointer to allocated memory as a uint16 ∗.

Return values

uint16	∗ valid for as many bytes as were allocated.

References Buffer().

Referenced by dng_opcode_MapTable::Prepare(), dng_opcode_FixVignetteRadial::Prepare(), dng_encode_proxy_↩ task::Process(), dng_opcode_MapTable::ProcessArea(), and dng_opcode_MapTable::PutData().

5.96.2.14 Buffer_uint16() [2/2]

```
const uint16* dng_memory_block::Buffer_uint16 ( ) const  [inline]
```

Return pointer to allocated memory as a const uint16 ∗.

Return values

const	uint16 ∗ valid for as many bytes as were allocated.

References Buffer().

5.96.2.15 Buffer_uint32() [1/2]

```
uint32* dng_memory_block::Buffer_uint32 ( )  [inline]
```

Return pointer to allocated memory as a uint32 ∗.

Return values

uint32	∗ valid for as many bytes as were allocated.

References Buffer().

5.96.2.16 Buffer_uint32() [2/2]

```
const uint32* dng_memory_block::Buffer_uint32 ( ) const  [inline]
```

Return pointer to allocated memory as a const uint32 ∗.

Return values

const	uint32 ∗ valid for as many bytes as were allocated.

References Buffer().

5.96.2.17 Buffer_uint8() [1/2]

```
uint8* dng_memory_block::Buffer_uint8 ( )  [inline]
```

Return pointer to allocated memory as a uint8 ∗.

Return values

uint8	∗ valid for as many bytes as were allocated.

References Buffer().

Referenced by dng_memory_stream::CopyToStream().

5.96.2.18 Buffer_uint8() [2/2]

```
const uint8* dng_memory_block::Buffer_uint8 ( ) const  [inline]
```

Return pointer to allocated memory as a const uint8 ∗.

Return values

const	uint8 ∗ valid for as many bytes as were allocated.

References Buffer().

5.96.2.19 LogicalSize()

```
uint32 dng_memory_block::LogicalSize ( ) const  [inline]
```

Getter for available size, in bytes, of memory block.

Return values

size	in bytes of available memory in memory block.

Referenced by dng_linearization_info::ColumnBlackCount(), dng_linearization_info::MaxBlackLevel(), dng_opcode_↩ Unknown::PutData(), and dng_linearization_info::RowBlackCount().

The documentation for this class was generated from the following files:

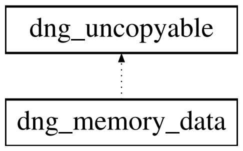
- dng_memory.h
- dng_memory.cpp

5.97 dng_memory_data Class Reference

Class to provide resource acquisition is instantiation discipline for small memory allocations.

```
#include <dng_memory.h>
```

Inheritance diagram for dng_memory_data:



Public Member Functions

- dng_memory_data ()
- dng_memory_data (uint32 size)
- **dng_memory_data** (const dng_safe_uint32 &size)
- dng_memory_data (uint32 count, std::size_t elementSize)
- ∼dng_memory_data ()

  Release memory buffer using free.
- void Allocate (uint32 size)
- void **Allocate** (const dng_safe_uint32 &size)
- void Allocate (uint32 count, std::size_t elementSize)
- void **Allocate** (const dng_safe_uint32 &count, std::size_t elementSize)
- void Clear ()
- void ∗ Buffer ()
- const void ∗ Buffer () const
- char ∗ Buffer_char ()
- const char ∗ Buffer_char () const
- uint8 ∗ Buffer_uint8 ()
- const uint8 ∗ Buffer_uint8 () const
- uint16 ∗ Buffer_uint16 ()
- const uint16 ∗ Buffer_uint16 () const
- int16 ∗ Buffer_int16 ()
- const int16 ∗ Buffer_int16 () const
- uint32 ∗ Buffer_uint32 ()
- const uint32 ∗ Buffer_uint32 () const
- int32 ∗ Buffer_int32 ()

- const int32 ∗ Buffer_int32 () const
- uint64 ∗ Buffer_uint64 ()
- const uint64 ∗ Buffer_uint64 () const
- int64 ∗ Buffer_int64 ()
- const int64 ∗ Buffer_int64 () const
- real32 ∗ Buffer_real32 ()
- const real32 ∗ Buffer_real32 () const
- real64 ∗ Buffer_real64 ()
- const real64 ∗ Buffer_real64 () const

5.97.1 Detailed Description

Class to provide resource acquisition is instantiation discipline for small memory allocations.

This class does not use dng_memory_allocator for memory allocation.

5.97.2 Constructor & Destructor Documentation

5.97.2.1 dng_memory_data() [1/3]

```
dng_memory_data::dng_memory_data ( )
```

Construct an empty memory buffer using malloc.

Exceptions

dng_memory_full	with fErrorCode equal to dng_error_memory.

5.97.2.2 dng_memory_data() [2/3]

```
dng_memory_data::dng_memory_data (
             uint32 size )
```

Construct memory buffer of size bytes using malloc.

Parameters

size	Number of bytes of memory needed.

Exceptions

dng_memory_full	with fErrorCode equal to dng_error_memory.

References Allocate().

5.97.2.3 dng_memory_data() [3/3]

```
dng_memory_data::dng_memory_data (
            uint32 count,
            std::size_t elementSize )
```

Note: This constructor is for internal use only and should not be considered part of the DNG SDK API.

Construct memory buffer of count elements of elementSize bytes each.

Parameters

count	Number of elements.
elementSize	Size of each element.

Exceptions

dng_memory_full	with fErrorCode equal to dng_error_memory.

References Allocate().

5.97.3 Member Function Documentation

5.97.3.1 Allocate() [1/2]

```
void dng_memory_data::Allocate (
            uint32 size )
```

Clear existing memory buffer and allocate new memory of size bytes.

Parameters

size	Number of bytes of memory needed.

Exceptions

dng_memory_full	with fErrorCode equal to dng_error_memory.

References Clear(), and ThrowMemoryFull().

Referenced by Allocate(), and dng_memory_data().

5.97.3.2 Allocate() [2/2]

```
void dng_memory_data::Allocate (
             uint32 count,
             std::size_t elementSize )
```

Note: This method is for internal use only and should not be considered part of the DNG SDK API.

Clear existing memory buffer and allocate new memory of count elements of elementSize bytes each.

Parameters

count	Number of elements.
elementSize	Size of each element.

Exceptions

dng_memory_full	with fErrorCode equal to dng_error_memory.

References Allocate(), and ThrowOverflow().

5.97.3.3 Buffer() [1/2]

```
void* dng_memory_data::Buffer ( )  [inline]
```

Return pointer to allocated memory as a void ∗..

Return values

void	∗ valid for as many bytes as were allocated.

Referenced by Buffer_char(), Buffer_int16(), Buffer_int32(), Buffer_int64(), Buffer_real32(), Buffer_real64(), Buffer_↩ uint16(), Buffer_uint32(), Buffer_uint64(), Buffer_uint8(), and dng_stream::CopyToStream().

5.97.3.4 Buffer() [2/2]

```
const void* dng_memory_data::Buffer ( ) const  [inline]
```

Return pointer to allocated memory as a const void ∗.

Return values

const	void ∗ valid for as many bytes as were allocated.

5.97.3.5 Buffer_char() [1/2]

```
char* dng_memory_data::Buffer_char ( )  [inline]
```

Return pointer to allocated memory as a char ∗.

Return values

char	∗ valid for as many bytes as were allocated.

References Buffer().

Referenced by dng_iptc::Parse().

5.97.3.6 Buffer_char() [2/2]

```
const char* dng_memory_data::Buffer_char ( ) const  [inline]
```

Return pointer to allocated memory as a const char ∗.

Return values

const	char ∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.7 Buffer_int16() [1/2]

```
int16* dng_memory_data::Buffer_int16 ( )  [inline]
```

Return pointer to allocated memory as a int16 ∗.

Return values

int16	∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.8 Buffer_int16() [2/2]

```
const int16* dng_memory_data::Buffer_int16 ( ) const  [inline]
```

Return pointer to allocated memory as a const int16 ∗.

Return values

const	int16 ∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.9 Buffer_int32() [1/2]

```
int32* dng_memory_data::Buffer_int32 ( ) [inline]
```

Return pointer to allocated memory as a const int32 ∗.

Return values

const	int32 ∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.10 Buffer_int32() [2/2]

```
const int32* dng_memory_data::Buffer_int32 ( ) const [inline]
```

Return pointer to allocated memory as a const int32 ∗.

Return values

const	int32 ∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.11 Buffer_int64() [1/2]

```
int64* dng_memory_data::Buffer_int64 ( ) [inline]
```

Return pointer to allocated memory as a const int64 ∗.

Return values

const	int64 ∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.12 Buffer_int64() [2/2]

```
const int64* dng_memory_data::Buffer_int64 ( ) const  [inline]
```

Return pointer to allocated memory as a const int64 ∗.

Return values

const	int64 ∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.13 Buffer_real32() [1/2]

```
real32* dng_memory_data::Buffer_real32 ( )  [inline]
```

Return pointer to allocated memory as a real32 ∗.

Return values

real32	∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.14 Buffer_real32() [2/2]

```
const real32* dng_memory_data::Buffer_real32 ( ) const  [inline]
```

Return pointer to allocated memory as a const real32 ∗.

Return values

const	real32 ∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.15 Buffer_real64() [1/2]

```
real64* dng_memory_data::Buffer_real64 ( )  [inline]
```

Return pointer to allocated memory as a real64 ∗.

Return values

real64	∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.16 Buffer_real64() [2/2]

```
const real64* dng_memory_data::Buffer_real64 ( ) const  [inline]
```

Return pointer to allocated memory as a const real64 ∗.

Return values

const	real64 ∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.17 Buffer_uint16() [1/2]

```
uint16* dng_memory_data::Buffer_uint16 ( )  [inline]
```

Return pointer to allocated memory as a uint16 ∗.

Return values

uint16	∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.18 Buffer_uint16() [2/2]

```
const uint16* dng_memory_data::Buffer_uint16 ( ) const  [inline]
```

Return pointer to allocated memory as a const uint16 ∗.

Return values

const	uint16 ∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.19 Buffer_uint32() [1/2]

```
uint32* dng_memory_data::Buffer_uint32 ( )  [inline]
```

Return pointer to allocated memory as a uint32 ∗.

Return values

uint32	∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.20 Buffer_uint32() [2/2]

```
const uint32* dng_memory_data::Buffer_uint32 ( ) const  [inline]
```

Return pointer to allocated memory as a uint32 ∗.

Return values

uint32	∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.21 Buffer_uint64() [1/2]

```
uint64* dng_memory_data::Buffer_uint64 ( )  [inline]
```

Return pointer to allocated memory as a uint64 ∗.

Return values

uint64	∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.22 Buffer_uint64() [2/2]

```
const uint64* dng_memory_data::Buffer_uint64 ( ) const  [inline]
```

Return pointer to allocated memory as a uint64 ∗.

Return values

uint64	∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.23 Buffer_uint8() [1/2]

```
uint8* dng_memory_data::Buffer_uint8 ( )  [inline]
```

Return pointer to allocated memory as a uint8 ∗.

Return values

uint8	∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.24 Buffer_uint8() [2/2]

```
const uint8* dng_memory_data::Buffer_uint8 ( ) const  [inline]
```

Return pointer to allocated memory as a const uint8 ∗.

Return values

const	uint8 ∗ valid for as many bytes as were allocated.

References Buffer().

5.97.3.25 Clear()

```
void dng_memory_data::Clear ( )
```

Release any allocated memory using free. Object is still valid and Allocate can be called again.

Referenced by Allocate(), and ∼dng_memory_data().

The documentation for this class was generated from the following files:

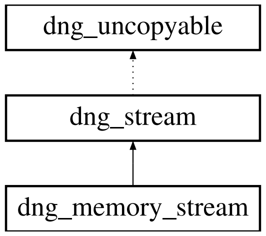
- dng_memory.h
- dng_memory.cpp

5.98 dng_memory_stream Class Reference

A dng_stream which can be read from or written to memory.

`#include <dng_memory_stream.h>`

Inheritance diagram for dng_memory_stream:



Public Member Functions

- dng_memory_stream (dng_memory_allocator &allocator, dng_abort_sniffer ∗sniffer=NULL, uint32 pageSize=64 ∗1024)
- void SetLengthLimit (uint64 limit)

  Sets a maximum length limit.
- virtual void CopyToStream (dng_stream &dstStream, uint64 count)

Protected Member Functions

- virtual uint64 **DoGetLength** ()
- virtual void **DoRead** (void ∗data, uint32 count, uint64 offset)
- virtual void **DoSetLength** (uint64 length)
- virtual void **DoWrite** (const void ∗data, uint32 count, uint64 offset)

Protected Attributes

- dng_memory_allocator & **fAllocator**
- uint32 **fPageSize**
- uint32 **fPageCount**
- uint32 **fPagesAllocated**
- dng_memory_block ∗∗ **fPageList**
- uint64 **fMemoryStreamLength**
- uint64 **fLengthLimit**

Additional Inherited Members

5.98.1 Detailed Description

A dng_stream which can be read from or written to memory.

Stream is populated via writing and either read or accessed by asking for contents as a pointer.

5.98.2 Constructor & Destructor Documentation

5.98.2.1 dng_memory_stream()

```
dng_memory_stream::dng_memory_stream (
             dng_memory_allocator & allocator,
             dng_abort_sniffer * sniffer = NULL,
             uint32 pageSize = 64 * 1024 )
```

Construct a new memory-based stream.

Parameters

allocator	Allocator to use to allocate memory in stream as needed.
sniffer	If non-NULL used to check for user cancellation.
pageSize	Unit of allocation for data stored in stream.

5.98.3 Member Function Documentation

5.98.3.1 CopyToStream()

```
void dng_memory_stream::CopyToStream (
             dng_stream & dstStream,
             uint64 count )  [virtual]
```

Copy a specified number of bytes to a target stream.

Parameters

dstStream	The target stream.
count	The number of bytes to copy.

Reimplemented from dng_stream.

References dng_memory_block::Buffer_uint8(), dng_stream::CopyToStream(), dng_stream::Flush(), dng_stream::↩
Length(), dng_stream::Position(), dng_stream::Put(), dng_stream::SetReadPosition(), and ThrowEndOfFile().

The documentation for this class was generated from the following files:

- dng_memory_stream.h
- dng_memory_stream.cpp

5.99 dng_metadata Class Reference

Main class for holding metadata.

```
#include <dng_negative.h>
```

Public Member Functions

- **dng_metadata** (dng_host &host)
- **dng_metadata** (const dng_metadata &rhs, dng_memory_allocator &allocator)
- virtual dng_metadata ∗ Clone (dng_memory_allocator &allocator) const

  Copy this metadata.
- void SetBaseOrientation (const dng_orientation &orientation)

  Setter for BaseOrientation.
- bool HasBaseOrientation () const

  Has BaseOrientation been set?
- const dng_orientation & BaseOrientation () const

  Getter for BaseOrientation.
- void ApplyOrientation (const dng_orientation &orientation)
- void **SetIPTC** (AutoPtr< dng_memory_block > &block, uint64 offset)
- void **SetIPTC** (AutoPtr< dng_memory_block > &block)
- void **ClearIPTC** ()
- const void ∗ **IPTCData** () const
- uint32 **IPTCLength** () const
- uint64 **IPTCOffset** () const
- dng_fingerprint **IPTCDigest** (bool includePadding=true) const
- void **RebuildIPTC** (dng_memory_allocator &allocator, bool padForTIFF)
- void **SetMakerNoteSafety** (bool safe)
- bool **IsMakerNoteSafe** () const
- void **SetMakerNote** (AutoPtr< dng_memory_block > &block)
- void **ClearMakerNote** ()
- const void ∗ **MakerNoteData** () const
- uint32 **MakerNoteLength** () const
- dng_exif ∗ **GetExif** ()
- const dng_exif ∗ **GetExif** () const
- template<class E >
  E & **Exif** ()
- template<class E >
  const E & **Exif** () const
- void **ResetExif** (dng_exif ∗newExif)
- dng_memory_block ∗ **BuildExifBlock** (dng_memory_allocator &allocator, const dng_resolution ∗resolution=N←ULL, bool includeIPTC=false, const dng_jpeg_preview ∗thumbnail=NULL) const
- dng_exif ∗ **GetOriginalExif** ()
- const dng_exif ∗ **GetOriginalExif** () const
- bool **SetXMP** (dng_host &host, const void ∗buffer, uint32 count, bool xmpInSidecar=false, bool xmpIs←Newer=false)
- void **SetEmbeddedXMP** (dng_host &host, const void ∗buffer, uint32 count)
- dng_xmp ∗ **GetXMP** ()
- const dng_xmp ∗ **GetXMP** () const

- template<class X >
  X & **XMP** ()
- template<class X >
  const X & **XMP** () const
- bool **XMPinSidecar** () const
- const dng_fingerprint & **EmbeddedXMPDigest** () const
- bool **HaveValidEmbeddedXMP** () const
- void **ResetXMP** (dng_xmp ∗newXMP)
- void **ResetXMPSidecarNewer** (dng_xmp ∗newXMP, bool inSidecar, bool isNewer)
- void **SynchronizeMetadata** ()
- void **UpdateDateTime** (const dng_date_time_info &dt)
- void **UpdateDateTimeToNow** ()
- void **UpdateMetadataDateTimeToNow** ()
- void **SetSourceMIME** (const char ∗s)
- const dng_string & **SourceMIME** () const

5.99.1 Detailed Description

Main class for holding metadata.

5.99.2 Member Function Documentation

5.99.2.1 ApplyOrientation()

```
void dng_metadata::ApplyOrientation (
            const dng_orientation & orientation )
```

Logically rotates the image by changing the orientation values. This will also update the XMP data.

Referenced by dng_negative::ApplyOrientation().

The documentation for this class was generated from the following files:

- dng_negative.h
- dng_negative.cpp

5.100 dng_mosaic_info Class Reference

Support for describing color filter array patterns and manipulating mosaic sample data.

```
#include <dng_mosaic_info.h>
```

Public Member Functions

- virtual void **Parse** (dng_host &host, dng_stream &stream, dng_info &info)
- virtual void **PostParse** (dng_host &host, dng_negative &negative)
- bool IsColorFilterArray () const
- virtual bool SetFourColorBayer ()
- virtual dng_point FullScale () const
- virtual dng_point DownScale (uint32 minSize, uint32 prefSize, real64 cropFactor) const
- virtual dng_point DstSize (const dng_point &downScale) const
- virtual void InterpolateGeneric (dng_host &host, dng_negative &negative, const dng_image &srcImage, dng_image &dstImage, uint32 srcPlane=0) const
- virtual void InterpolateFast (dng_host &host, dng_negative &negative, const dng_image &srcImage, dng_image &dstImage, const dng_point &downScale, uint32 srcPlane=0) const
- virtual void Interpolate (dng_host &host, dng_negative &negative, const dng_image &srcImage, dng_image &dstImage, const dng_point &downScale, uint32 srcPlane=0, dng_matrix ∗scaleTransforms=NULL) const
- virtual bool **SupportsPreservedBlackLevels** () const

Public Attributes

- dng_point fCFAPatternSize

  Size of fCFAPattern.
- uint8 fCFAPattern [kMaxCFAPattern][kMaxCFAPattern]

  CFA pattern from CFAPattern tag in the TIFF/EP specification..
- uint32 fColorPlanes

  Number of color planes in DNG input.
- uint8 **fCFAPlaneColor** [kMaxColorPlanes]
- uint32 fCFALayout
- uint32 fBayerGreenSplit

Protected Member Functions

- virtual bool **IsSafeDownScale** (const dng_point &downScale) const
- uint32 **SizeForDownScale** (const dng_point &downScale) const
- virtual bool **ValidSizeDownScale** (const dng_point &downScale, uint32 minSize) const

Protected Attributes

- dng_point **fSrcSize**
- dng_point **fCroppedSize**
- real64 **fAspectRatio**

5.100.1 Detailed Description

Support for describing color filter array patterns and manipulating mosaic sample data.

See CFAPattern tag in TIFF/EP specification and CFAPlaneColor, CFALayout, and BayerGreenSplit tags in the DNG 1.1.0 specification.

5.100.2 Member Function Documentation

5.100.2.1 DownScale()

```
dng_point dng_mosaic_info::DownScale (
             uint32 minSize,
             uint32 prefSize,
             real64 cropFactor ) const  [virtual]
```

Returns integer factors by which mosaic data must be downsampled to produce an image which is as close to prefSize as possible in longer dimension, but no smaller than minSize.

Parameters

minSize	Number of pixels as minium for longer dimension of downsampled image.
prefSize	Number of pixels as target for longer dimension of downsampled image.
cropFactor	Faction of the image to be used after cropping.

Return values

Point	containing integer factors by which image must be downsampled.

References IsColorFilterArray().

5.100.2.2 DstSize()

```
dng_point dng_mosaic_info::DstSize (
             const dng_point & downScale ) const  [virtual]
```

Return size of demosaiced image for passed in downscaling factor.

Parameters

downScale	Integer downsampling factor obtained from DownScale method.

Return values

Size	of resulting demosaiced image.

References FullScale().

5.100.2.3 FullScale()

```
dng_point dng_mosaic_info::FullScale ( ) const  [virtual]
```

Returns scaling factor relative to input size needed to capture output data. Staggered (or rotated) sensing arrays are produced to a larger output than the number of input samples. This method indicates how much larger.

Return values

a	point with integer scaling factors for the horizotal and vertical dimensions.

References fCFALayout.

Referenced by DstSize(), and InterpolateGeneric().

5.100.2.4 Interpolate()

```
void dng_mosaic_info::Interpolate (
            dng_host & host,
            dng_negative & negative,
            const dng_image & srcImage,
            dng_image & dstImage,
            const dng_point & downScale,
            uint32 srcPlane = 0,
            dng_matrix * scaleTransforms = NULL ) const  [virtual]
```

Demosaic interpolation of a single plane. Chooses between generic and fast interpolators based on parameters.

Parameters

host	dng_host to use for buffer allocation requests, user cancellation testing, and progress updates.
negative	DNG negative of mosaiced data.
srcImage	Source image for mosaiced data.
dstImage	Destination image for resulting interpolated data.
downScale	Amount (in horizontal and vertical) by which to subsample image.
srcPlane	Which plane to interpolate.

References InterpolateFast(), InterpolateGeneric(), and dng_image::Planes().

5.100.2.5 InterpolateFast()

```
void dng_mosaic_info::InterpolateFast (
            dng_host & host,
            dng_negative & negative,
            const dng_image & srcImage,
            dng_image & dstImage,
            const dng_point & downScale,
            uint32 srcPlane = 0 ) const  [virtual]
```

Demosaic interpolation of a single plane for downsampled case.

Parameters

host	dng_host to use for buffer allocation requests, user cancellation testing, and progress updates.
negative	DNG negative of mosaiced data.
srcImage	Source image for mosaiced data.
dstImage	Destination image for resulting interpolated data.
downScale	Amount (in horizontal and vertical) by which to subsample image.
srcPlane	Which plane to interpolate.

References dng_image::Bounds(), and dng_host::PerformAreaTask().

Referenced by Interpolate().

5.100.2.6 InterpolateGeneric()

```
void dng_mosaic_info::InterpolateGeneric (
            dng_host & host,
            dng_negative & negative,
            const dng_image & srcImage,
            dng_image & dstImage,
            uint32 srcPlane = 0 ) const  [virtual]
```

Demosaic interpolation of a single plane for non-downsampled case.

Parameters

host	dng_host to use for buffer allocation requests, user cancellation testing, and progress updates.
negative	DNG negative of mosaiced data.
srcImage	Source image for mosaiced data.
dstImage	Destination image for resulting interpolated data.
srcPlane	Which plane to interpolate.

References dng_host::Allocate(), dng_image::Bounds(), dng_image::edge_repeat, fCFAPatternSize, fColorPlanes, FullScale(), dng_image::Get(), dng_image::PixelType(), dng_image::Put(), dng_image::RepeatingTile(), and dng_↩ host::SniffForAbort().

Referenced by Interpolate().

5.100.2.7 IsColorFilterArray()

```
bool dng_mosaic_info::IsColorFilterArray ( ) const  [inline]
```

Returns whether the RAW data in this DNG file from a color filter array (mosaiced) source.

Return values

true	if this DNG file is from a color filter array (mosiaced) source.

References fCFAPatternSize.

Referenced by DownScale().

5.100.2.8 SetFourColorBayer()

```
bool dng_mosaic_info::SetFourColorBayer ( )  [virtual]
```

Enable generating four-plane output from three-plane Bayer input. Extra plane is a second version of the green channel. First green is produced using green mosaic samples from one set of rows/columns (even/odd) and the second green channel is produced using the other set of rows/columns. One can compare the two versions to judge whether Bayer↩ GreenSplit needs to be set for a given input source.

References fCFAPattern, fCFAPatternSize, and fColorPlanes.

5.100.3 Member Data Documentation

5.100.3.1 fBayerGreenSplit

```
uint32 dng_mosaic_info::fBayerGreenSplit
```

Value of BayerGreeSplit tag in DNG file. BayerGreenSplit only applies to CFA images using a Bayer pattern filter array. This tag specifies, in arbitrary units, how closely the values of the green pixels in the blue/green rows track the values of the green pixels in the red/green rows.

A value of zero means the two kinds of green pixels track closely, while a non-zero value means they sometimes diverge. The useful range for this tag is from 0 (no divergence) to about 5000 (large divergence).

5.100.3.2 fCFALayout

`uint32 dng_mosaic_info::fCFALayout`

Value of CFALayout tag in the DNG 1.3 specification. CFALayout describes the spatial layout of the CFA. The currently defined values are:

- 1 = Rectangular (or square) layout.
- 2 = Staggered layout A: even columns are offset down by 1/2 row.
- 3 = Staggered layout B: even columns are offset up by 1/2 row.
- 4 = Staggered layout C: even rows are offset right by 1/2 column.
- 5 = Staggered layout D: even rows are offset left by 1/2 column.
- 6 = Staggered layout E: even rows are offset up by 1/2 row, even columns are offset left by 1/2 column.
- 7 = Staggered layout F: even rows are offset up by 1/2 row, even columns are offset right by 1/2 column.
- 8 = Staggered layout G: even rows are offset down by 1/2 row, even columns are offset left by 1/2 column.
- 9 = Staggered layout H: even rows are offset down by 1/2 row, even columns are offset right by 1/2 column.

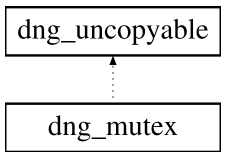
Referenced by FullScale().

The documentation for this class was generated from the following files:

- dng_mosaic_info.h
- dng_mosaic_info.cpp

5.101 dng_mutex Class Reference

Inheritance diagram for dng_mutex:



Public Types

- enum { **kDNGMutexLevelLeaf** = 0x70000000u, **kDNGMutexLevelIgnore** = 0x7FFFFFFFu }

Public Member Functions

- **dng_mutex** (const char ∗mutexName, uint32 mutexLevel=kDNGMutexLevelLeaf)
- void **Lock** ()
- void **Unlock** ()
- const char ∗ **MutexName** () const

The documentation for this class was generated from the following files:

- dng_mutex.h
- dng_mutex.cpp

5.102 dng_negative Class Reference

Main class for holding DNG image data and associated metadata.

```
#include <dng_negative.h>
```

Public Types

- enum **RawImageStageEnum** {
  rawImageStagePreOpcode1, **rawImageStagePostOpcode1**, **rawImageStagePostOpcode2**, **rawImage**↩
  StagePreOpcode3,
  rawImageStagePostOpcode3, **rawImageStageNone** }

Public Member Functions

- dng_memory_allocator & Allocator () const

  Provide access to the memory allocator used for this object.
- void SetModelName (const char ∗name)

  Getter for ModelName.
- const dng_string & ModelName () const

  Setter for ModelName.
- void SetLocalName (const char ∗name)

  Setter for LocalName.
- const dng_string & LocalName () const

  Getter for LocalName.
- dng_metadata & Metadata ()

  Getter for metadata.
- dng_metadata ∗ CloneInternalMetadata () const
- void SetBaseOrientation (const dng_orientation &orientation)

  Setter for BaseOrientation.
- bool HasBaseOrientation () METACONST

  Has BaseOrientation been set?
- const dng_orientation & BaseOrientation () METACONST

  Getter for BaseOrientation.

- virtual dng_orientation ComputeOrientation (const dng_metadata &metadata) const

  Hook to allow SDK host code to add additional rotations.
- dng_orientation Orientation ()

  For non-const negatives, we simply default to using the metadata attached to the negative.
- void ApplyOrientation (const dng_orientation &orientation)
- void SetDefaultCropSize (const dng_urational &sizeH, const dng_urational &sizeV)

  Setter for DefaultCropSize.
- void SetDefaultCropSize (uint32 sizeH, uint32 sizeV)

  Setter for DefaultCropSize.
- const dng_urational & DefaultCropSizeH () const

  Getter for DefaultCropSize horizontal.
- const dng_urational & DefaultCropSizeV () const

  Getter for DefaultCropSize vertical.
- void SetDefaultCropOrigin (const dng_urational &originH, const dng_urational &originV)

  Setter for DefaultCropOrigin.
- void SetDefaultCropOrigin (uint32 originH, uint32 originV)

  Setter for DefaultCropOrigin.
- void SetDefaultCropCentered (const dng_point &rawSize)

  Set default crop around center of image.
- const dng_urational & DefaultCropOriginH () const

  Get default crop origin horizontal value.
- const dng_urational & DefaultCropOriginV () const

  Get default crop origin vertical value.
- bool HasDefaultUserCrop () const

  Is there a default user crop?
- const dng_urational & DefaultUserCropT () const

  Getter for top coordinate of default user crop.
- const dng_urational & DefaultUserCropL () const

  Getter for left coordinate of default user crop.
- const dng_urational & DefaultUserCropB () const

  Getter for bottom coordinate of default user crop.
- const dng_urational & DefaultUserCropR () const

  Getter for right coordinate of default user crop.
- void ResetDefaultUserCrop ()

  Reset default user crop to default crop area.
- void SetDefaultUserCrop (const dng_urational &t, const dng_urational &l, const dng_urational &b, const dng_urational &r)

  Setter for all 4 coordinates of default user crop.
- void SetDefaultUserCropT (const dng_urational &value)

  Setter for top coordinate of default user crop.
- void SetDefaultUserCropL (const dng_urational &value)

  Setter for left coordinate of default user crop.
- void SetDefaultUserCropB (const dng_urational &value)

  Setter for bottom coordinate of default user crop.
- void SetDefaultUserCropR (const dng_urational &value)

  Setter for right coordinate of default user crop.
- void SetDefaultScale (const dng_urational &scaleH, const dng_urational &scaleV)

Setter for DefaultScale.
- const dng_urational & DefaultScaleH () const

  Get default scale horizontal value.
- const dng_urational & DefaultScaleV () const

  Get default scale vertical value.
- void SetBestQualityScale (const dng_urational &scale)

  Setter for BestQualityScale.
- const dng_urational & BestQualityScale () const

  Getter for BestQualityScale.
- bool HasBestQualityScale () const

  Is the best quality scale different than the default scale?
- real64 RawToFullScaleH () const

  API for raw to full image scaling factors horizontal.
- real64 RawToFullScaleV () const

  API for raw to full image scaling factors vertical.
- void SetRawToFullScale (real64 scaleH, real64 scaleV)

  Setter for raw to full scales.
- real64 DefaultScale () const
- real64 SquareWidth () const

  Default cropped image size (at scale == 1.0) width.
- real64 SquareHeight () const

  Default cropped image size (at scale == 1.0) height.
- real64 AspectRatio () const

  Default cropped image aspect ratio.
- real64 PixelAspectRatio () const

  Pixel aspect ratio of stage 3 image.
- uint32 FinalWidth (real64 scale) const

  Default cropped image size at given scale factor width.
- uint32 FinalHeight (real64 scale) const

  Default cropped image size at given scale factor height.
- uint32 DefaultFinalWidth () const

  Default cropped image size at default scale factor width.
- uint32 DefaultFinalHeight () const

  Default cropped image size at default scale factor height.
- uint32 BestQualityFinalWidth () const
- uint32 BestQualityFinalHeight () const
- const dng_point & OriginalDefaultFinalSize () const
- void SetOriginalDefaultFinalSize (const dng_point &size)

  Setter for OriginalDefaultFinalSize.
- const dng_point & OriginalBestQualityFinalSize () const
- void SetOriginalBestQualityFinalSize (const dng_point &size)

  Setter for OriginalBestQualityFinalSize.
- const dng_urational & OriginalDefaultCropSizeH () const
- const dng_urational & **OriginalDefaultCropSizeV** () const
- void SetOriginalDefaultCropSize (const dng_urational &sizeH, const dng_urational &sizeV)

  Setter for OriginalDefaultCropSize.
- void SetDefaultOriginalSizes ()
- void SetOriginalSizes (const dng_point &size)

Set all the original size fields to a specific size.
- dng_rect DefaultCropArea () const

  The default crop area in the stage 3 image coordinates.
- void SetBaselineNoise (real64 noise)

  Setter for BaselineNoise.
- const dng_urational & BaselineNoiseR () const

  Getter for BaselineNoise as dng_urational.
- real64 BaselineNoise () const

  Getter for BaselineNoise as real64.
- void SetNoiseReductionApplied (const dng_urational &value)

  Setter for NoiseReductionApplied.
- const dng_urational & NoiseReductionApplied () const

  Getter for NoiseReductionApplied.
- void **SetRawNoiseReductionApplied** ()
- const dng_urational & **RawNoiseReductionApplied** () const
- void SetNoiseProfile (const dng_noise_profile &noiseProfile)

  Setter for noise profile.
- bool HasNoiseProfile () const

  Does this negative have a valid noise profile?
- const dng_noise_profile & NoiseProfile () const

  Getter for noise profile.
- bool **HasRawNoiseProfile** () const
- void **SetRawNoiseProfile** ()
- const dng_noise_profile & **RawNoiseProfile** () const
- void SetBaselineExposure (real64 exposure)

  Setter for BaselineExposure.
- const dng_srational & BaselineExposureR () const

  Getter for BaselineExposure as dng_urational.
- real64 BaselineExposure () const

  Getter for BaselineExposure as real64.
- real64 TotalBaselineExposure (const dng_camera_profile_id &profileID) const
- void SetBaselineSharpness (real64 sharpness)

  Setter for BaselineSharpness.
- const dng_urational & BaselineSharpnessR () const

  Getter for BaselineSharpness as dng_urational.
- real64 BaselineSharpness () const

  Getter for BaselineSharpness as real64.
- void **SetRawBaselineSharpness** ()
- const dng_urational & **RawBaselineSharpness** () const
- void SetChromaBlurRadius (const dng_urational &radius)

  Setter for ChromaBlurRadius.
- const dng_urational & ChromaBlurRadius () const

  Getter for ChromaBlurRadius as dng_urational.
- void SetAntiAliasStrength (const dng_urational &strength)

  Setter for AntiAliasStrength.
- const dng_urational & AntiAliasStrength () const

  Getter for AntiAliasStrength as dng_urational.
- void SetLinearResponseLimit (real64 limit)

Setter for LinearResponseLimit.
- const dng_urational & LinearResponseLimitR () const

    Getter for LinearResponseLimit as dng_urational.
- real64 LinearResponseLimit () const

    Getter for LinearResponseLimit as real64.
- void SetShadowScale (const dng_urational &scale)

    Setter for ShadowScale.
- const dng_urational & ShadowScaleR () const

    Getter for ShadowScale as dng_urational.
- real64 ShadowScale () const

    Getter for ShadowScale as real64.
- void **SetColorimetricReference** (uint32 ref)
- uint32 **ColorimetricReference** () const
- void **SetFloatingPoint** (bool isFloatingPoint)
- bool **IsFloatingPoint** () const
- bool **IsHighDynamicRange** () const
- bool **IsNormalDynamicRange** () const
- void SetColorChannels (uint32 channels)

    Setter for ColorChannels.
- uint32 ColorChannels () const

    Getter for ColorChannels.
- void SetMonochrome ()

    Setter for Monochrome.
- bool IsMonochrome () const

    Getter for Monochrome.
- void SetAnalogBalance (const dng_vector &b)

    Setter for AnalogBalance.
- dng_urational AnalogBalanceR (uint32 channel) const

    Getter for AnalogBalance as dng_urational.
- real64 AnalogBalance (uint32 channel) const

    Getter for AnalogBalance as real64.
- void SetCameraNeutral (const dng_vector &n)

    Setter for CameraNeutral.
- void ClearCameraNeutral ()

    Clear CameraNeutral.
- bool HasCameraNeutral () const

    Determine if CameraNeutral has been set but not cleared.
- const dng_vector & CameraNeutral () const

    Getter for CameraNeutral.
- dng_urational **CameraNeutralR** (uint32 channel) const
- void SetCameraWhiteXY (const dng_xy_coord &coord)

    Setter for CameraWhiteXY.
- bool **HasCameraWhiteXY** () const
- const dng_xy_coord & **CameraWhiteXY** () const
- void **GetCameraWhiteXY** (dng_urational &x, dng_urational &y) const
- void SetCameraCalibration1 (const dng_matrix &m)
- void SetCameraCalibration2 (const dng_matrix &m)
- const dng_matrix & CameraCalibration1 () const

Getter for first of up to two color matrices used for individual camera calibrations.
- const dng_matrix & CameraCalibration2 () const

  Getter for second of up to two color matrices used for individual camera calibrations.
- void **SetCameraCalibrationSignature** (const char ∗signature)
- const dng_string & **CameraCalibrationSignature** () const
- void **AddProfile** (AutoPtr< dng_camera_profile > &profile)
- void **ClearProfiles** ()
- void **ClearProfiles** (bool clearBuiltinMatrixProfiles, bool clearReadFromDisk)
- uint32 **ProfileCount** () const
- const dng_camera_profile & **ProfileByIndex** (uint32 index) const
- virtual const dng_camera_profile ∗ **ProfileByID** (const dng_camera_profile_id &id, bool useDefaultIfNo←Match=true) const
- bool **HasProfileID** (const dng_camera_profile_id &id) const
- virtual const dng_camera_profile ∗ **ComputeCameraProfileToEmbed** (const dng_metadata &metadata) const
- const dng_camera_profile ∗ **CameraProfileToEmbed** ()
- void **SetAsShotProfileName** (const char ∗name)
- const dng_string & **AsShotProfileName** () const
- virtual dng_color_spec ∗ **MakeColorSpec** (const dng_camera_profile_id &id) const
- void **SetRawImageDigest** (const dng_fingerprint &digest)
- void **SetNewRawImageDigest** (const dng_fingerprint &digest)
- void **ClearRawImageDigest** () const
- const dng_fingerprint & **RawImageDigest** () const
- const dng_fingerprint & **NewRawImageDigest** () const
- void **FindRawImageDigest** (dng_host &host) const
- void **FindNewRawImageDigest** (dng_host &host) const
- void **ValidateRawImageDigest** (dng_host &host)
- void **SetRawDataUniqueID** (const dng_fingerprint &id)
- dng_fingerprint **RawDataUniqueID** () const
- void **FindRawDataUniqueID** (dng_host &host) const
- virtual void **RecomputeRawDataUniqueID** (dng_host &host)
- void **SetOriginalRawFileName** (const char ∗name)
- bool **HasOriginalRawFileName** () const
- const dng_string & **OriginalRawFileName** () const
- void **SetHasOriginalRawFileData** (bool hasData)
- bool **CanEmbedOriginalRaw** () const
- void **SetOriginalRawFileData** (AutoPtr< dng_memory_block > &data)
- const void ∗ **OriginalRawFileData** () const
- uint32 **OriginalRawFileDataLength** () const
- void **SetOriginalRawFileDigest** (const dng_fingerprint &digest)
- const dng_fingerprint & **OriginalRawFileDigest** () const
- void **FindOriginalRawFileDigest** () const
- void **ValidateOriginalRawFileDigest** ()
- void **SetPrivateData** (AutoPtr< dng_memory_block > &block)
- void **ClearPrivateData** ()
- const uint8 ∗ **PrivateData** () const
- uint32 **PrivateLength** () const
- void **SetMakerNoteSafety** (bool safe)
- bool **IsMakerNoteSafe** () METACONST
- void **SetMakerNote** (AutoPtr< dng_memory_block > &block)
- void **ClearMakerNote** ()
- const void ∗ **MakerNoteData** () METACONST

- uint32 **MakerNoteLength** () METACONST
- dng_exif ∗ **GetExif** ()
- void **ResetExif** (dng_exif ∗newExif)
- dng_exif ∗ **GetOriginalExif** ()
- void **SetIPTC** (AutoPtr< dng_memory_block > &block, uint64 offset)
- void **SetIPTC** (AutoPtr< dng_memory_block > &block)
- void **ClearIPTC** ()
- const void ∗ **IPTCData** () METACONST
- uint32 **IPTCLength** () METACONST
- uint64 **IPTCOffset** () METACONST
- dng_fingerprint **IPTCDigest** (bool includePadding=true) METACONST
- void **RebuildIPTC** (bool padForTIFF)
- bool **SetXMP** (dng_host &host, const void ∗buffer, uint32 count, bool xmpInSidecar=false, bool xmpIs←Newer=false)
- dng_xmp ∗ **GetXMP** ()
- bool **XMPinSidecar** () METACONST
- void **ResetXMP** (dng_xmp ∗newXMP)
- void **ResetXMPSidecarNewer** (dng_xmp ∗newXMP, bool inSidecar, bool isNewer)
- bool **HaveValidEmbeddedXMP** () METACONST
- void **SetSourceMIME** (const char ∗s)
- const dng_linearization_info ∗ **GetLinearizationInfo** () const
- void **ClearLinearizationInfo** ()
- void **SetLinearization** (AutoPtr< dng_memory_block > &curve)
- void **SetActiveArea** (const dng_rect &area)
- void **SetMaskedAreas** (uint32 count, const dng_rect ∗area)
- void **SetMaskedArea** (const dng_rect &area)
- void **SetBlackLevel** (real64 black, int32 plane=-1)
- void **SetQuadBlacks** (real64 black0, real64 black1, real64 black2, real64 black3, int32 plane=-1)
- void **Set6x6Blacks** (real64 blacks6x6 [36], int32 plane=-1)
- void **SetRowBlacks** (const real64 ∗blacks, uint32 count)
- void **SetColumnBlacks** (const real64 ∗blacks, uint32 count)
- uint32 **WhiteLevel** (uint32 plane=0) const
- void **SetWhiteLevel** (uint32 white, int32 plane=-1)
- const dng_mosaic_info ∗ **GetMosaicInfo** () const
- void **ClearMosaicInfo** ()
- void **SetColorKeys** (ColorKeyCode color0, ColorKeyCode color1, ColorKeyCode color2, ColorKeyCode color3=colorKeyMaxEnum)
- void **SetRGB** ()
- void **SetCMY** ()
- void **SetGMCY** ()
- void **SetBayerMosaic** (uint32 phase)
- void **SetFujiMosaic** (uint32 phase)
- void **SetFujiMosaic6x6** (uint32 phase)
- void **SetQuadMosaic** (uint32 pattern)
- void **SetGreenSplit** (uint32 split)
- const dng_opcode_list & **OpcodeList1** () const
- dng_opcode_list & **OpcodeList1** ()
- const dng_opcode_list & **OpcodeList2** () const
- dng_opcode_list & **OpcodeList2** ()
- const dng_opcode_list & **OpcodeList3** () const
- dng_opcode_list & **OpcodeList3** ()

- virtual void **Parse** (dng_host &host, dng_stream &stream, dng_info &info)
- virtual void **PostParse** (dng_host &host, dng_stream &stream, dng_info &info)
- void **SynchronizeMetadata** ()
- void **UpdateDateTime** (const dng_date_time_info &dt)
- void **UpdateDateTimeToNow** ()
- virtual bool **SetFourColorBayer** ()
- const dng_image ∗ **Stage1Image** () const
- const dng_image ∗ **Stage2Image** () const
- const dng_image ∗ **Stage3Image** () const
- RawImageStageEnum **RawImageStage** () const
- const dng_image & **RawImage** () const
- uint16 **RawImageBlackLevel** () const
- uint32 **RawFloatBitDepth** () const
- void **SetRawFloatBitDepth** (uint32 bitDepth)
- const dng_jpeg_image ∗ **RawJPEGImage** () const
- void **SetRawJPEGImage** (AutoPtr< dng_jpeg_image > &jpegImage)
- void **ClearRawJPEGImage** ()
- void **SetRawJPEGImageDigest** (const dng_fingerprint &digest)
- void **ClearRawJPEGImageDigest** () const
- const dng_fingerprint & **RawJPEGImageDigest** () const
- void **FindRawJPEGImageDigest** (dng_host &host) const
- virtual void **ReadOpcodeLists** (dng_host &host, dng_stream &stream, dng_info &info)
- virtual void **ReadStage1Image** (dng_host &host, dng_stream &stream, dng_info &info)
- virtual void **ReadEnhancedImage** (dng_host &host, dng_stream &stream, dng_info &info)
- void **SetStage1Image** (AutoPtr< dng_image > &image)
- void **SetStage2Image** (AutoPtr< dng_image > &image)
- void **SetStage3Image** (AutoPtr< dng_image > &image)
- void **BuildStage2Image** (dng_host &host)
- void **BuildStage3Image** (dng_host &host, int32 srcPlane=-1)
- void **SetStage3Gain** (real64 gain)
- real64 **Stage3Gain** () const
- void **SetStage3BlackLevel** (uint16 level)
- uint16 **Stage3BlackLevel** () const
- real64 **Stage3BlackLevelNormalized** () const
- virtual bool **SupportsPreservedBlackLevels** (dng_host &host)
- dng_image ∗ **EncodeRawProxy** (dng_host &host, const dng_image &srcImage, dng_opcode_list &opcodeList, real64 ∗blackLevel) const
- void **ConvertToProxy** (dng_host &host, dng_image_writer &writer, uint32 proxySize=0, uint64 proxyCount=0)
- bool **IsProxy** () const
- void **SetIsPreview** (bool preview)
- bool **IsPreview** () const
- void **SetIsDamaged** (bool damaged)
- bool **IsDamaged** () const
- void **SetTransparencyMask** (AutoPtr< dng_image > &image, uint32 bitDepth=0)
- const dng_image ∗ **TransparencyMask** () const
- const dng_image ∗ **RawTransparencyMask** () const
- uint32 **RawTransparencyMaskBitDepth** () const
- void **ReadTransparencyMask** (dng_host &host, dng_stream &stream, dng_info &info)
- virtual void **ResizeTransparencyToMatchStage3** (dng_host &host, bool convertTo8Bit=false)
- virtual bool **NeedFlattenTransparency** (dng_host &host)
- virtual void **FlattenTransparency** (dng_host &host)

- const dng_image ∗ **UnflattenedStage3Image** () const
- bool **HasDepthMap** () const
- void **SetHasDepthMap** (bool hasDepthMap)
- const dng_image ∗ **DepthMap** () const
- void **SetDepthMap** (AutoPtr< dng_image > &depthMap)
- bool **HasDepthMapImage** () const
- const dng_image ∗ **RawDepthMap** () const
- void **ReadDepthMap** (dng_host &host, dng_stream &stream, dng_info &info)
- virtual void **ResizeDepthToMatchStage3** (dng_host &host)
- uint32 **DepthFormat** () const
- void **SetDepthFormat** (uint32 format)
- const dng_urational & **DepthNear** () const
- void **SetDepthNear** (const dng_urational &dist)
- const dng_urational & **DepthFar** () const
- void **SetDepthFar** (const dng_urational &dist)
- uint32 **DepthUnits** () const
- void **SetDepthUnits** (uint32 units)
- uint32 **DepthMeasureType** () const
- void **SetDepthMeasureType** (uint32 measure)
- const dng_string & **EnhanceParams** () const
- void **SetEnhanceParams** (const dng_string &s)
- void **SetEnhanceParams** (const char ∗s)

Static Public Member Functions

- static dng_negative ∗ **Make** (dng_host &host)
- static dng_fingerprint **FindImageDigest** (dng_host &host, const dng_image &image)

Protected Member Functions

- const dng_metadata & InternalMetadata () const
- **dng_negative** (dng_host &host)
- virtual void **Initialize** ()
- virtual dng_linearization_info ∗ **MakeLinearizationInfo** ()
- void **NeedLinearizationInfo** ()
- virtual dng_mosaic_info ∗ **MakeMosaicInfo** ()
- void **NeedMosaicInfo** ()
- virtual void **DoBuildStage2** (dng_host &host)
- virtual void **DoPostOpcodeList2** (dng_host &host)
- virtual bool **NeedDefloatStage2** (dng_host &host)
- virtual void **DefloatStage2** (dng_host &host)
- virtual void **DoInterpolateStage3** (dng_host &host, int32 srcPlane, dng_matrix ∗scaleTransforms)
- virtual void **DoMergeStage3** (dng_host &host, dng_matrix ∗scaleTransforms)
- virtual void **DoBuildStage3** (dng_host &host, int32 srcPlane, dng_matrix ∗scaleTransforms)
- virtual void **AdjustProfileForStage3** ()

Protected Attributes

- dng_memory_allocator & **fAllocator**
- dng_string **fModelName**
- dng_string **fLocalName**
- dng_urational **fDefaultCropSizeH**
- dng_urational **fDefaultCropSizeV**
- dng_urational **fDefaultCropOriginH**
- dng_urational **fDefaultCropOriginV**
- dng_urational **fDefaultUserCropT**
- dng_urational **fDefaultUserCropL**
- dng_urational **fDefaultUserCropB**
- dng_urational **fDefaultUserCropR**
- dng_urational **fDefaultScaleH**
- dng_urational **fDefaultScaleV**
- dng_urational **fBestQualityScale**
- dng_point **fOriginalDefaultFinalSize**
- dng_point **fOriginalBestQualityFinalSize**
- dng_urational **fOriginalDefaultCropSizeH**
- dng_urational **fOriginalDefaultCropSizeV**
- real64 **fRawToFullScaleH**
- real64 **fRawToFullScaleV**
- dng_urational **fBaselineNoise**
- dng_urational **fNoiseReductionApplied**
- dng_urational **fRawNoiseReductionApplied**
- dng_noise_profile **fNoiseProfile**
- dng_noise_profile **fRawNoiseProfile**
- dng_srational **fBaselineExposure**
- dng_urational **fBaselineSharpness**
- dng_urational **fRawBaselineSharpness**
- dng_urational **fChromaBlurRadius**
- dng_urational **fAntiAliasStrength**
- dng_urational **fLinearResponseLimit**
- dng_urational **fShadowScale**
- uint32 **fColorimetricReference**
- bool **fFloatingPoint**
- uint32 **fColorChannels**
- dng_vector **fAnalogBalance**
- dng_vector **fCameraNeutral**
- dng_xy_coord **fCameraWhiteXY**
- dng_matrix **fCameraCalibration1**
- dng_matrix **fCameraCalibration2**
- dng_string **fCameraCalibrationSignature**
- dng_std_vector< dng_camera_profile ∗ > **fCameraProfile**
- dng_string **fAsShotProfileName**
- dng_fingerprint **fRawImageDigest**
- dng_fingerprint **fNewRawImageDigest**
- dng_fingerprint **fRawDataUniqueID**
- dng_std_mutex **fRawDataUniqueIDMutex**
- dng_string **fOriginalRawFileName**
- bool **fHasOriginalRawFileData**

- AutoPtr< dng_memory_block > **fOriginalRawFileData**
- dng_fingerprint **fOriginalRawFileDigest**
- AutoPtr< dng_memory_block > **fDNGPrivateData**
- dng_metadata **fMetadata**
- AutoPtr< dng_linearization_info > **fLinearizationInfo**
- AutoPtr< dng_mosaic_info > **fMosaicInfo**
- dng_opcode_list **fOpcodeList1**
- dng_opcode_list **fOpcodeList2**
- dng_opcode_list **fOpcodeList3**
- AutoPtr< dng_image > **fStage1Image**
- AutoPtr< dng_image > **fStage2Image**
- AutoPtr< dng_image > **fStage3Image**
- real64 **fStage3Gain**
- uint16 **fStage3BlackLevel**
- bool **fIsPreview**
- bool **fIsDamaged**
- RawImageStageEnum **fRawImageStage**
- AutoPtr< dng_image > **fRawImage**
- uint16 **fRawImageBlackLevel**
- uint32 **fRawFloatBitDepth**
- AutoPtr< dng_jpeg_image > **fRawJPEGImage**
- dng_fingerprint **fRawJPEGImageDigest**
- AutoPtr< dng_image > **fTransparencyMask**
- AutoPtr< dng_image > **fRawTransparencyMask**
- uint32 **fRawTransparencyMaskBitDepth**
- AutoPtr< dng_image > **fUnflattenedStage3Image**
- bool **fHasDepthMap**
- AutoPtr< dng_image > **fDepthMap**
- AutoPtr< dng_image > **fRawDepthMap**
- uint32 **fDepthFormat**
- dng_urational **fDepthNear**
- dng_urational **fDepthFar**
- uint32 **fDepthUnits**
- uint32 **fDepthMeasureType**
- dng_string **fEnhanceParams**

5.102.1 Detailed Description

Main class for holding DNG image data and associated metadata.

5.102.2 Member Function Documentation

5.102.2.1 ApplyOrientation()

```
void dng_negative::ApplyOrientation (
             const dng_orientation & orientation )  [inline]
```

Logically rotates the image by changing the orientation values. This will also update the XMP data.

References dng_metadata::ApplyOrientation(), and Metadata().

5.102.2.2 BestQualityFinalHeight()

```
uint32 dng_negative::BestQualityFinalHeight ( ) const  [inline]
```

Get best quality height. For a naive conversion, one could use either the default size, or the best quality size.

References BestQualityScale(), DefaultScale(), and FinalHeight().

Referenced by SetDefaultOriginalSizes().

5.102.2.3 BestQualityFinalWidth()

```
uint32 dng_negative::BestQualityFinalWidth ( ) const  [inline]
```

Get best quality width. For a naive conversion, one could use either the default size, or the best quality size.

References BestQualityScale(), DefaultScale(), and FinalWidth().

Referenced by SetDefaultOriginalSizes().

5.102.2.4 CloneInternalMetadata()

```
dng_metadata * dng_negative::CloneInternalMetadata ( ) const
```

Make a copy of the internal metadata generally as a basis for further changes.

References Allocator(), dng_metadata::Clone(), and InternalMetadata().

5.102.2.5 DefaultScale()

```
real64 dng_negative::DefaultScale ( ) const  [inline]
```

Get default scale factor. When specifing a single scale factor, we use the horizontal scale factor, and let the vertical scale factor be calculated based on the pixel aspect ratio.

References DefaultScaleH().

Referenced by BestQualityFinalHeight(), BestQualityFinalWidth(), DefaultFinalHeight(), and DefaultFinalWidth().

5.102.2.6 InternalMetadata()

```
const dng_metadata& dng_negative::InternalMetadata ( ) const  [inline], [protected]
```

An accessor for the internal metadata that works even when we have general access turned off. This is needed to provide access to EXIF ISO information.

Referenced by CloneInternalMetadata().

5.102.2.7 OriginalBestQualityFinalSize()

```
const dng_point& dng_negative::OriginalBestQualityFinalSize ( ) const  [inline]
```

Best quality size of original (non-proxy) image. For non-proxy images, this is equal to BestQualityFinalWidth/Best↩QualityFinalHeight. For proxy images, this is equal to the BestQualityFinalWidth/BestQualityFinalHeight of the image this proxy was derived from.

Referenced by SetDefaultOriginalSizes().

5.102.2.8 OriginalDefaultCropSizeH()

```
const dng_urational& dng_negative::OriginalDefaultCropSizeH ( ) const  [inline]
```

DefaultCropSize for original (non-proxy) image. For non-proxy images, this is equal to the DefaultCropSize. for proxy images, this is equal size of the DefaultCropSize of the image this proxy was derived from.

Referenced by SetDefaultOriginalSizes().

5.102.2.9 OriginalDefaultFinalSize()

```
const dng_point& dng_negative::OriginalDefaultFinalSize ( ) const  [inline]
```

Default size of original (non-proxy) image. For non-proxy images, this is equal to DefaultFinalWidth/DefaultFinalHight. For proxy images, this is equal to the DefaultFinalWidth/DefaultFinalHeight of the image this proxy was derived from.

Referenced by SetDefaultOriginalSizes().

5.102.2.10 SetCameraCalibration1()

```
void dng_negative::SetCameraCalibration1 (
             const dng_matrix & m )
```

Setter for first of up to two color matrices used for individual camera calibrations.

The sequence of matrix transforms is: Camera data --> camera calibration --> "inverse" of color matrix

This will be a 4x4 matrix for a four-color camera. The defaults are almost always the identity matrix, and for the cases where they aren't, they are diagonal matrices.

5.102.2.11 SetCameraCalibration2()

```
void dng_negative::SetCameraCalibration2 (
             const dng_matrix & m )
```

Setter for second of up to two color matrices used for individual camera calibrations.

The sequence of matrix transforms is: Camera data --> camera calibration --> "inverse" of color matrix

This will be a 4x4 matrix for a four-color camera. The defaults are almost always the identity matrix, and for the cases where they aren't, they are diagonal matrices.

5.102.2.12 SetDefaultOriginalSizes()

```
void dng_negative::SetDefaultOriginalSizes ( )
```

If the original size fields are undefined, set them to the current sizes.

References BestQualityFinalHeight(), BestQualityFinalWidth(), DefaultCropSizeH(), DefaultCropSizeV(), DefaultFinal←Height(), DefaultFinalWidth(), OriginalBestQualityFinalSize(), OriginalDefaultCropSizeH(), OriginalDefaultFinalSize(), SetOriginalBestQualityFinalSize(), SetOriginalDefaultCropSize(), and SetOriginalDefaultFinalSize().

5.102.2.13 TotalBaselineExposure()

```
real64 dng_negative::TotalBaselineExposure (
             const dng_camera_profile_id & profileID ) const
```

Compute total baseline exposure (sum of negative's BaselineExposure and profile's BaselineExposureOffset).

References BaselineExposure(), and dng_camera_profile::BaselineExposureOffset().

The documentation for this class was generated from the following files:

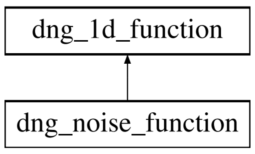
- dng_negative.h
- dng_negative.cpp

5.103 dng_noise_function Class Reference

Noise model for photon and sensor read noise, assuming that they are independent random variables and spatially invariant.

`#include <dng_negative.h>`

Inheritance diagram for dng_noise_function:



Public Member Functions

- dng_noise_function ()

    Create empty and invalid noise function.
- dng_noise_function (real64 scale, real64 offset)

    Create noise function with the specified scale and offset.
- virtual real64 Evaluate (real64 x) const
- real64 Scale () const

    The scale (slope, gain) of the noise function.
- real64 Offset () const

    The offset (square of the noise floor) of the noise function.
- void SetScale (real64 scale)

    Set the scale (slope, gain) of the noise function.
- void SetOffset (real64 offset)

    Set the offset (square of the noise floor) of the noise function.
- bool IsValid () const

    Is the noise function valid?

Protected Attributes

- real64 **fScale**
- real64 **fOffset**

5.103.1 Detailed Description

Noise model for photon and sensor read noise, assuming that they are independent random variables and spatially invariant.

The noise model is N (x) = sqrt (scale∗x + offset), where x represents a linear signal value in the range [0,1], and N (x) is the standard deviation (i.e., noise). The parameters scale and offset are both sensor-dependent and ISO-dependent. scale must be positive, and offset must be non-negative.

5.103.2 Member Function Documentation

5.103.2.1 Evaluate()

```
virtual real64 dng_noise_function::Evaluate (
             real64 x ) const  [inline], [virtual]
```

Compute noise (standard deviation) at the specified average signal level x.

Implements dng_1d_function.

The documentation for this class was generated from the following file:

- dng_negative.h

5.104 dng_noise_profile Class Reference

Noise profile for a negative.

```
#include <dng_negative.h>
```

Public Member Functions

- dng_noise_profile ()

  Create empty (invalid) noise profile.
- dng_noise_profile (const dng_std_vector< dng_noise_function > &functions)

  Create noise profile with the specified noise functions (1 per plane).
- bool IsValid () const

  Is the noise profile valid?
- bool IsValidForNegative (const dng_negative &negative) const

  Is the noise profile valid for the specified negative?
- const dng_noise_function & NoiseFunction (uint32 plane) const

  The noise function for the specified plane.
- uint32 NumFunctions () const

  The number of noise functions in this profile.
- bool operator== (const dng_noise_profile &profile) const

  Equality test.
- bool **operator!=** (const dng_noise_profile &profile) const

Protected Attributes

- dng_std_vector< dng_noise_function > **fNoiseFunctions**

5.104.1 Detailed Description

Noise profile for a negative.

For mosaiced negatives, the noise profile describes the approximate noise characteristics of a mosaic negative after linearization, but prior to demosaicing. For demosaiced negatives (i.e., linear DNGs), the noise profile describes the approximate noise characteristics of the image data immediately following the demosaic step, prior to the processing of opcode list 3.

A noise profile may contain 1 or N noise functions, where N is the number of color planes for the negative. Otherwise the noise profile is considered to be invalid for that negative. If the noise profile contains 1 noise function, then it is assumed that this single noise function applies to all color planes of the negative. Otherwise, the N noise functions map to the N planes of the negative in order specified in the CFAPlaneColor tag.

The documentation for this class was generated from the following files:

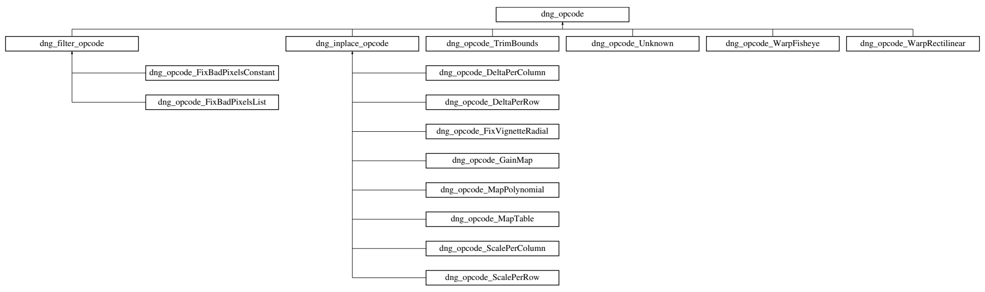
- dng_negative.h
- dng_negative.cpp

5.105 dng_opcode Class Reference

Virtual base class for opcode.

`#include <dng_opcodes.h>`

Inheritance diagram for dng_opcode:



Public Types

- enum { kFlag_None = 0, kFlag_Optional = 1, kFlag_SkipIfPreview = 2 }

  Opcode flags.

Public Member Functions

- uint32 OpcodeID () const

  The ID of this opcode.
- uint32 MinVersion () const

  The first DNG version that supports this opcode.
- uint32 Flags () const

  The flags for this opcode.
- bool Optional () const

  Is this opcode optional?
- bool SkipIfPreview () const

  Should the opcode be skipped when rendering preview images?
- bool WasReadFromStream () const

  Was this opcode read from a data stream?
- uint32 Stage () const
- void SetStage (uint32 stage)
- virtual bool IsNOP () const
- virtual bool IsValidForNegative (const dng_negative &) const

  Is this opcode valid for the specified negative?
- virtual void PutData (dng_stream &stream) const
- bool AboutToApply (dng_host &host, dng_negative &negative, const dng_rect &imageBounds, uint32 image↩ Planes)
- virtual void Apply (dng_host &host, dng_negative &negative, AutoPtr< dng_image > &image)=0

  Apply this opcode to the specified image with associated negative.

Protected Member Functions

- **dng_opcode** (uint32 opcodeID, uint32 minVersion, uint32 flags)
- **dng_opcode** (uint32 opcodeID, dng_stream &stream, const char ∗name)
- virtual void **DoAboutToApply** (dng_host &, dng_negative &, const dng_rect &, uint32)

5.105.1 Detailed Description

Virtual base class for opcode.

5.105.2 Member Enumeration Documentation

5.105.2.1 anonymous enum

```
anonymous enum
```

Opcode flags.

Enumerator

kFlag_None	No flag.
kFlag_Optional	This opcode is optional.
kFlag_SkipIfPreview	May skip opcode for preview images.

5.105.3 Member Function Documentation

5.105.3.1 AboutToApply()

```
bool dng_opcode::AboutToApply (
            dng_host & host,
            dng_negative & negative,
            const dng_rect & imageBounds,
            uint32 imagePlanes )
```

Perform error checking prior to applying this opcode to the specified negative. Returns true if this opcode should be applied to the negative, false otherwise.

References dng_host::ForPreview(), IsNOP(), IsValidForNegative(), MinVersion(), Optional(), SkipIfPreview(), Throw↩BadFormat(), and WasReadFromStream().

Referenced by dng_opcode_list::Apply().

5.105.3.2 IsNOP()

```
virtual bool dng_opcode::IsNOP ( ) const  [inline], [virtual]
```

Is the opcode a NOP (i.e., does nothing)? An opcode could be a NOP for some specific parameters.

Reimplemented in dng_opcode_FixVignetteRadial, dng_opcode_WarpFisheye, and dng_opcode_WarpRectilinear.

Referenced by AboutToApply().

5.105.3.3 PutData()

```
void dng_opcode::PutData (
             dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented in dng_opcode_FixVignetteRadial, dng_opcode_WarpFisheye, dng_opcode_WarpRectilinear, dng_opcode_ScalePerColumn, dng_opcode_ScalePerRow, dng_opcode_DeltaPerColumn, dng_opcode_DeltaPerRow, dng_opcode_Unknown, dng_opcode_FixBadPixelsList, dng_opcode_MapPolynomial, dng_opcode_MapTable, dng_opcode_GainMap, dng_opcode_FixBadPixelsConstant, and dng_opcode_TrimBounds.

References dng_stream::Put_uint32().

5.105.3.4 SetStage()

```
void dng_opcode::SetStage (
             uint32 stage )  [inline]
```

Set the image processing stage (1, 2, 3) for this opcode. Stage 1 is the original image data, including masked areas. Stage 2 is linearized image data and trimmed to the active area. Stage 3 is demosaiced and trimmed to the active area.

5.105.3.5 Stage()

```
uint32 dng_opcode::Stage ( ) const  [inline]
```

Which image processing stage (1, 2, 3) is associated with this opcode?

Referenced by dng_opcode_MapPolynomial::BufferPixelType(), dng_opcode_MapTable::Prepare(), dng_opcode_↩
GainMap::ProcessArea(), dng_opcode_MapPolynomial::ProcessArea(), dng_opcode_DeltaPerRow::ProcessArea(), dng_opcode_DeltaPerColumn::ProcessArea(), dng_opcode_ScalePerRow::ProcessArea(), and dng_opcode_Scale↩
PerColumn::ProcessArea().

The documentation for this class was generated from the following files:

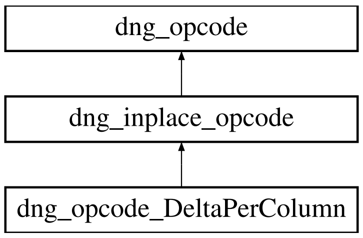
- dng_opcodes.h
- dng_opcodes.cpp

5.106 dng_opcode_DeltaPerColumn Class Reference

An opcode to apply a delta (i.e., offset) that varies per column. Within a column, the same delta value is applied to all specified pixels.

```
#include <dng_misc_opcodes.h>
```

Inheritance diagram for dng_opcode_DeltaPerColumn:



Public Member Functions

- dng_opcode_DeltaPerColumn (const dng_area_spec &areaSpec, AutoPtr< dng_memory_block > &table)
- **dng_opcode_DeltaPerColumn** (dng_host &host, dng_stream &stream)
- virtual void PutData (dng_stream &stream) const
- virtual uint32 BufferPixelType (uint32 imagePixelType)

  The pixel data type of this opcode.
- virtual dng_rect ModifiedBounds (const dng_rect &imageBounds)
- virtual void ProcessArea (dng_negative &negative, uint32 threadIndex, dng_pixel_buffer &buffer, const dng_rect &dstArea, const dng_rect &imageBounds)

Additional Inherited Members

5.106.1 Detailed Description

An opcode to apply a delta (i.e., offset) that varies per column. Within a column, the same delta value is applied to all specified pixels.

5.106.2 Constructor & Destructor Documentation

5.106.2.1 dng_opcode_DeltaPerColumn()

```
dng_opcode_DeltaPerColumn::dng_opcode_DeltaPerColumn (
            const dng_area_spec & areaSpec,
            AutoPtr< dng_memory_block > & table )
```

Create a DeltaPerColumn opcode with the specified area and column deltas (specified as a table of 32-bit floats).

5.106.3 Member Function Documentation

5.106.3.1 ModifiedBounds()

```
dng_rect dng_opcode_DeltaPerColumn::ModifiedBounds (
            const dng_rect & imageBounds )  [virtual]
```

The adjusted bounds (processing area) of this opcode. It is limited to the intersection of the specified image area and the GainMap area.

Reimplemented from dng_inplace_opcode.

References dng_area_spec::Overlap().

5.106.3.2 ProcessArea()

```
void dng_opcode_DeltaPerColumn::ProcessArea (
            dng_negative & negative,
            uint32 threadIndex,
            dng_pixel_buffer & buffer,
            const dng_rect & dstArea,
            const dng_rect & imageBounds )  [virtual]
```

Implements image processing operation in a single buffer. The source pixels are provided as input to the buffer, and this routine calculates and writes the destination pixels to the same buffer. Ideally, no allocation should be done in this routine.

Parameters

negative	The negative associated with the pixels to be processed.
threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Prepare method.
buffer	Source and Destination pixels.
dstArea	Destination pixel processing area.
imageBounds	Total image area to be processed; dstArea will always lie within these bounds.

Implements dng_inplace_opcode.

References dng_area_spec::Area(), dng_memory_block::Buffer_real32(), dng_area_spec::ColPitch(), dng_pixel_↩ buffer::DirtyPixel_real32(), dng_area_spec::Overlap(), dng_area_spec::Plane(), dng_area_spec::Planes(), dng_pixel↩ _buffer::Planes(), dng_area_spec::RowPitch(), dng_pixel_buffer::RowStep(), and dng_opcode::Stage().

5.106.3.3 PutData()

```
void dng_opcode_DeltaPerColumn::PutData (
            dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References dng_area_spec::Area(), dng_memory_block::Buffer_real32(), dng_area_spec::ColPitch(), dng_stream::↩ Put_real32(), dng_stream::Put_uint32(), and dng_area_spec::PutData().

The documentation for this class was generated from the following files:

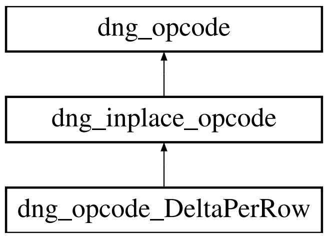
- dng_misc_opcodes.h
- dng_misc_opcodes.cpp

5.107 dng_opcode_DeltaPerRow Class Reference

An opcode to apply a delta (i.e., offset) that varies per row. Within a row, the same delta value is applied to all specified pixels.

```
#include <dng_misc_opcodes.h>
```

Inheritance diagram for dng_opcode_DeltaPerRow:



Public Member Functions

- dng_opcode_DeltaPerRow (const dng_area_spec &areaSpec, AutoPtr< dng_memory_block > &table)
- **dng_opcode_DeltaPerRow** (dng_host &host, dng_stream &stream)
- virtual void PutData (dng_stream &stream) const
- virtual uint32 BufferPixelType (uint32 imagePixelType)

  The pixel data type of this opcode.
- virtual dng_rect ModifiedBounds (const dng_rect &imageBounds)
- virtual void ProcessArea (dng_negative &negative, uint32 threadIndex, dng_pixel_buffer &buffer, const dng_rect &dstArea, const dng_rect &imageBounds)

Additional Inherited Members

5.107.1 Detailed Description

An opcode to apply a delta (i.e., offset) that varies per row. Within a row, the same delta value is applied to all specified pixels.

5.107.2 Constructor & Destructor Documentation

5.107.2.1 dng_opcode_DeltaPerRow()

```
dng_opcode_DeltaPerRow::dng_opcode_DeltaPerRow (
             const dng_area_spec & areaSpec,
             AutoPtr< dng_memory_block > & table )
```

Create a DeltaPerRow opcode with the specified area and row deltas (specified as a table of 32-bit floats).

5.107.3 Member Function Documentation

5.107.3.1 ModifiedBounds()

```
dng_rect dng_opcode_DeltaPerRow::ModifiedBounds (
             const dng_rect & imageBounds )  [virtual]
```

The adjusted bounds (processing area) of this opcode. It is limited to the intersection of the specified image area and the GainMap area.

Reimplemented from dng_inplace_opcode.

References dng_area_spec::Overlap().

5.107.3.2 ProcessArea()

```
void dng_opcode_DeltaPerRow::ProcessArea (
             dng_negative & negative,
             uint32 threadIndex,
             dng_pixel_buffer & buffer,
             const dng_rect & dstArea,
             const dng_rect & imageBounds )  [virtual]
```

Implements image processing operation in a single buffer. The source pixels are provided as input to the buffer, and this routine calculates and writes the destination pixels to the same buffer. Ideally, no allocation should be done in this routine.

Parameters

negative	The negative associated with the pixels to be processed.
threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Prepare method.
buffer	Source and Destination pixels.
dstArea	Destination pixel processing area.
imageBounds	Total image area to be processed; dstArea will always lie within these bounds.

Implements dng_inplace_opcode.

References dng_area_spec::Area(), dng_memory_block::Buffer_real32(), dng_area_spec::ColPitch(), dng_pixel_↩
buffer::DirtyPixel_real32(), dng_area_spec::Overlap(), dng_area_spec::Plane(), dng_area_spec::Planes(), dng_pixel↩
_buffer::Planes(), dng_area_spec::RowPitch(), and dng_opcode::Stage().

5.107.3.3 PutData()

```
void dng_opcode_DeltaPerRow::PutData (
            dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References dng_area_spec::Area(), dng_memory_block::Buffer_real32(), dng_stream::Put_real32(), dng_stream::↩
Put_uint32(), dng_area_spec::PutData(), and dng_area_spec::RowPitch().

The documentation for this class was generated from the following files:

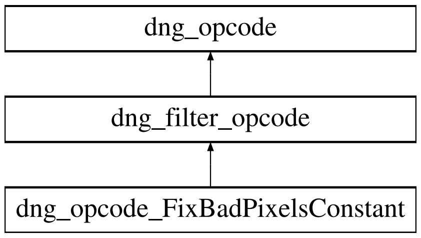
- dng_misc_opcodes.h
- dng_misc_opcodes.cpp

5.108 dng_opcode_FixBadPixelsConstant Class Reference

An opcode to fix individual bad pixels that are marked with a constant value (e.g., 0) in a Bayer image.

```
#include <dng_bad_pixels.h>
```

Inheritance diagram for dng_opcode_FixBadPixelsConstant:



Public Member Functions

- dng_opcode_FixBadPixelsConstant (uint32 constant, uint32 bayerPhase)
- **dng_opcode_FixBadPixelsConstant** (dng_stream &stream)
- virtual void PutData (dng_stream &stream) const
- virtual dng_point SrcRepeat ()

  Returns the width and height (in pixels) of the repeating mosaic pattern.
- virtual dng_rect SrcArea (const dng_rect &dstArea, const dng_rect &imageBounds)
- virtual void Prepare (dng_negative &negative, uint32 threadCount, const dng_point &tileSize, const dng_rect &imageBounds, uint32 imagePlanes, uint32 bufferPixelType, dng_memory_allocator &allocator)
- virtual void ProcessArea (dng_negative &negative, uint32 threadIndex, dng_pixel_buffer &srcBuffer, dng_pixel_buffer &dstBuffer, const dng_rect &dstArea, const dng_rect &imageBounds)

Protected Member Functions

- bool **IsGreen** (int32 row, int32 col) const

Additional Inherited Members

5.108.1 Detailed Description

An opcode to fix individual bad pixels that are marked with a constant value (e.g., 0) in a Bayer image.

5.108.2 Constructor & Destructor Documentation

5.108.2.1 dng_opcode_FixBadPixelsConstant()

```
dng_opcode_FixBadPixelsConstant::dng_opcode_FixBadPixelsConstant (
                uint32 constant,
                uint32 bayerPhase )
```

Construct an opcode to fix an individual bad pixels that are marked with a constant value in a Bayer image.

Parameters

constant	The constant value that indicates a bad pixel.
bayerPhase	The phase of the Bayer mosaic pattern (0, 1, 2, 3).

5.108.3 Member Function Documentation

5.108.3.1 Prepare()

```
void dng_opcode_FixBadPixelsConstant::Prepare (
             dng_negative & negative,
             uint32 threadCount,
             const dng_point & tileSize,
             const dng_rect & imageBounds,
             uint32 imagePlanes,
             uint32 bufferPixelType,
             dng_memory_allocator & allocator )  [virtual]
```

Startup method called before any processing is performed on pixel areas. It can be used to allocate (per-thread) memory and setup tasks.

Parameters

negative	The negative object to be processed.
threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
imageBounds	Total size of image to be processed.
imagePlanes	Number of planes in the image. Less than or equal to kMaxColorPlanes.
bufferPixelType	Pixel type of image buffer (see dng_tag_types.h).
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.

Reimplemented from dng_filter_opcode.

References ThrowBadFormat().

5.108.3.2 ProcessArea()

```
void dng_opcode_FixBadPixelsConstant::ProcessArea (
             dng_negative & negative,
             uint32 threadIndex,
             dng_pixel_buffer & srcBuffer,
             dng_pixel_buffer & dstBuffer,
             const dng_rect & dstArea,
             const dng_rect & imageBounds )  [virtual]
```

Implements filtering operation from one buffer to another. Source and destination pixels are set up in member fields of this class. Ideally, no allocation should be done in this routine.

Parameters

negative	The negative associated with the pixels to be processed.
threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Prepare method.
srcBuffer	Input area and source pixels.
dstBuffer	Destination pixels.
dstArea	Destination pixel processing area.
imageBounds	Total image area to be processed; dstArea will always lie within these bounds.

Implements dng_filter_opcode.

References dng_pixel_buffer::ConstPixel_uint16(), dng_pixel_buffer::CopyArea(), and dng_pixel_buffer::DirtyPixel_↩ uint16().

5.108.3.3 PutData()

```
void dng_opcode_FixBadPixelsConstant::PutData (
             dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References dng_stream::Put_uint32().

5.108.3.4 SrcArea()

```
dng_rect dng_opcode_FixBadPixelsConstant::SrcArea (
             const dng_rect & dstArea,
             const dng_rect & imageBounds )  [virtual]
```

Returns the source pixel area needed to process a destination pixel area that lies within the specified bounds.

Parameters

dstArea	The destination pixel area to be computed.
imageBounds	The overall image area (dstArea will lie within these bounds).

Return values

The	source pixel area needed to process the specified dstArea.

Reimplemented from dng_filter_opcode.

The documentation for this class was generated from the following files:

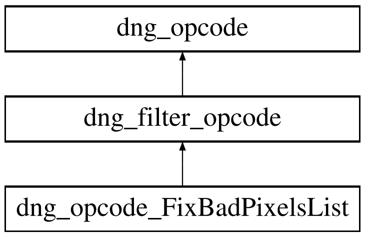
- dng_bad_pixels.h
- dng_bad_pixels.cpp

5.109 dng_opcode_FixBadPixelsList Class Reference

An opcode to fix lists of bad pixels (indicated by position) in a Bayer image.

```
#include <dng_bad_pixels.h>
```

Inheritance diagram for dng_opcode_FixBadPixelsList:



Public Member Functions

- dng_opcode_FixBadPixelsList (AutoPtr< dng_bad_pixel_list > &list, uint32 bayerPhase)
- **dng_opcode_FixBadPixelsList** (dng_stream &stream)
- virtual void PutData (dng_stream &stream) const
- virtual dng_point SrcRepeat ()

  Returns the width and height (in pixels) of the repeating mosaic pattern.
- virtual dng_rect SrcArea (const dng_rect &dstArea, const dng_rect &imageBounds)
- virtual void Prepare (dng_negative &negative, uint32 threadCount, const dng_point &tileSize, const dng_rect &imageBounds, uint32 imagePlanes, uint32 bufferPixelType, dng_memory_allocator &allocator)
- virtual void ProcessArea (dng_negative &negative, uint32 threadIndex, dng_pixel_buffer &srcBuffer, dng_pixel_buffer &dstBuffer, const dng_rect &dstArea, const dng_rect &imageBounds)

Protected Types

- enum { **kBadPointPadding** = 2, **kBadRectPadding** = 4 }

Protected Member Functions

- bool **IsGreen** (int32 row, int32 col) const
- virtual void **FixIsolatedPixel** (dng_pixel_buffer &buffer, dng_point &badPoint)
- virtual void **FixClusteredPixel** (dng_pixel_buffer &buffer, uint32 pointIndex, const dng_rect &imageBounds)
- virtual void **FixSingleColumn** (dng_pixel_buffer &buffer, const dng_rect &badRect)
- virtual void **FixSingleRow** (dng_pixel_buffer &buffer, const dng_rect &badRect)
- virtual void **FixClusteredRect** (dng_pixel_buffer &buffer, const dng_rect &badRect, const dng_rect &image↩ Bounds)

Additional Inherited Members

5.109.1 Detailed Description

An opcode to fix lists of bad pixels (indicated by position) in a Bayer image.

5.109.2 Constructor & Destructor Documentation

5.109.2.1 dng_opcode_FixBadPixelsList()

```
dng_opcode_FixBadPixelsList::dng_opcode_FixBadPixelsList (
            AutoPtr< dng_bad_pixel_list > & list,
            uint32 bayerPhase )
```

Construct an opcode to fix lists of bad pixels (indicated by position) in a Bayer image.

Parameters

list	The list of bad pixels to fix.
bayerPhase	The phase of the Bayer mosaic pattern (0, 1, 2, 3).

5.109.3 Member Function Documentation

5.109.3.1 Prepare()

```
void dng_opcode_FixBadPixelsList::Prepare (
            dng_negative & negative,
            uint32 threadCount,
            const dng_point & tileSize,
            const dng_rect & imageBounds,
            uint32 imagePlanes,
            uint32 bufferPixelType,
            dng_memory_allocator & allocator )  [virtual]
```

Startup method called before any processing is performed on pixel areas. It can be used to allocate (per-thread) memory and setup tasks.

Parameters

negative	The negative object to be processed.
threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
imageBounds	Total size of image to be processed.
imagePlanes	Number of planes in the image. Less than or equal to kMaxColorPlanes.
bufferPixelType	Pixel type of image buffer (see dng_tag_types.h).
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.

Reimplemented from dng_filter_opcode.

References ThrowBadFormat().

5.109.3.2 ProcessArea()

```
void dng_opcode_FixBadPixelsList::ProcessArea (
            dng_negative & negative,
            uint32 threadIndex,
            dng_pixel_buffer & srcBuffer,
            dng_pixel_buffer & dstBuffer,
            const dng_rect & dstArea,
            const dng_rect & imageBounds )  [virtual]
```

Implements filtering operation from one buffer to another. Source and destination pixels are set up in member fields of this class. Ideally, no allocation should be done in this routine.

Parameters

negative	The negative associated with the pixels to be processed.
threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Prepare method.
srcBuffer	Input area and source pixels.
dstBuffer	Destination pixels.
dstArea	Destination pixel processing area.
imageBounds	Total image area to be processed; dstArea will always lie within these bounds.

Implements dng_filter_opcode.

References dng_pixel_buffer::CopyArea(), dng_bad_pixel_list::IsPointIsolated(), dng_bad_pixel_list::IsRectIsolated(),

dng_bad_pixel_list::Point(), dng_bad_pixel_list::PointCount(), dng_bad_pixel_list::Rect(), dng_bad_pixel_list::Rect↩
Count(), dng_pixel_buffer::RepeatSubArea(), and SrcRepeat().

5.109.3.3 PutData()

```
void dng_opcode_FixBadPixelsList::PutData (
            dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References dng_bad_pixel_list::Point(), dng_bad_pixel_list::PointCount(), dng_stream::Put_int32(), dng_stream::Put↩
_uint32(), dng_bad_pixel_list::Rect(), and dng_bad_pixel_list::RectCount().

5.109.3.4 SrcArea()

```
dng_rect dng_opcode_FixBadPixelsList::SrcArea (
            const dng_rect & dstArea,
            const dng_rect & imageBounds )  [virtual]
```

Returns the source pixel area needed to process a destination pixel area that lies within the specified bounds.

Parameters

dstArea	The destination pixel area to be computed.
imageBounds	The overall image area (dstArea will lie within these bounds).

Return values

The	source pixel area needed to process the specified dstArea.

Reimplemented from dng_filter_opcode.

References dng_bad_pixel_list::PointCount(), and dng_bad_pixel_list::RectCount().

The documentation for this class was generated from the following files:

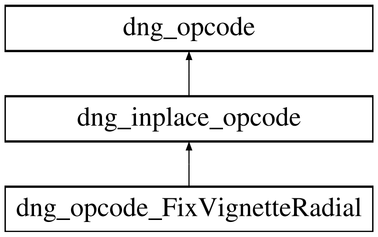
- dng_bad_pixels.h
- dng_bad_pixels.cpp

5.110 dng_opcode_FixVignetteRadial Class Reference

Radially-symmetric lens vignette correction opcode.

`#include <dng_lens_correction.h>`

Inheritance diagram for dng_opcode_FixVignetteRadial:



Public Member Functions

- **dng_opcode_FixVignetteRadial** (const dng_vignette_radial_params &params, uint32 flags)
- **dng_opcode_FixVignetteRadial** (dng_stream &stream)
- const dng_vignette_radial_params & **Params** () const
- virtual bool IsNOP () const
- virtual bool IsValidForNegative (const dng_negative &) const

    Is this opcode valid for the specified negative?
- virtual void PutData (dng_stream &stream) const
- virtual uint32 BufferPixelType (uint32)

    The pixel data type of this opcode.
- virtual void Prepare (dng_negative &negative, uint32 threadCount, const dng_point &tileSize, const dng_rect &imageBounds, uint32 imagePlanes, uint32 bufferPixelType, dng_memory_allocator &allocator)
- virtual void ProcessArea (dng_negative &negative, uint32 threadIndex, dng_pixel_buffer &buffer, const dng_rect &dstArea, const dng_rect &imageBounds)

Protected Member Functions

- virtual dng_vignette_radial_params **MakeParamsForRender** (const dng_negative &negative)

Static Protected Member Functions

- static uint32 **ParamBytes** ()

Protected Attributes

- dng_vignette_radial_params **fParams**
- uint32 **fImagePlanes**
- int64 **fSrcOriginH**
- int64 **fSrcOriginV**
- int64 **fSrcStepH**
- int64 **fSrcStepV**
- uint32 **fTableInputBits**
- uint32 **fTableOutputBits**
- AutoPtr< dng_memory_block > **fGainTable**
- AutoPtr< dng_memory_block > **fMaskBuffers** [kMaxMPThreads]

Additional Inherited Members

5.110.1 Detailed Description

Radially-symmetric lens vignette correction opcode.

5.110.2 Member Function Documentation

5.110.2.1 IsNOP()

```
bool dng_opcode_FixVignetteRadial::IsNOP ( ) const  [virtual]
```

Is the opcode a NOP (i.e., does nothing)? An opcode could be a NOP for some specific parameters.

Reimplemented from dng_opcode.

5.110.2.2 Prepare()

```
void dng_opcode_FixVignetteRadial::Prepare (
             dng_negative & negative,
             uint32 threadCount,
             const dng_point & tileSize,
             const dng_rect & imageBounds,
             uint32 imagePlanes,
             uint32 bufferPixelType,
             dng_memory_allocator & allocator )  [virtual]
```

Startup method called before any processing is performed on pixel areas. It can be used to allocate (per-thread) memory and setup tasks.

Parameters

negative	The negative object to be processed.
threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
imageBounds	Total size of image to be processed.
imagePlanes	Number of planes in the image. Less than or equal to kMaxColorPlanes.
bufferPixelType	Pixel type of image buffer (see dng_tag_types.h).
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.

Reimplemented from dng_inplace_opcode.

References dng_memory_allocator::Allocate(), dng_memory_block::Buffer_uint16(), DNG_ASSERT, AutoPtr< T >↩

::Get(), dng_1d_table::Initialize(), dng_1d_table::Interpolate(), kMaxColorPlanes, kMaxMPThreads, dng_negative::↩ PixelAspectRatio(), AutoPtr< T >::Reset(), ThrowBadFormat(), and ThrowProgramError().

5.110.2.3 ProcessArea()

```
void dng_opcode_FixVignetteRadial::ProcessArea (
            dng_negative & negative,
            uint32 threadIndex,
            dng_pixel_buffer & buffer,
            const dng_rect & dstArea,
            const dng_rect & imageBounds )  [virtual]
```

Implements image processing operation in a single buffer. The source pixels are provided as input to the buffer, and this routine calculates and writes the destination pixels to the same buffer. Ideally, no allocation should be done in this routine.

Parameters

negative	The negative associated with the pixels to be processed.
threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Prepare method.
buffer	Source and Destination pixels.
dstArea	Destination pixel processing area.
imageBounds	Total image area to be processed; dstArea will always lie within these bounds.

Implements dng_inplace_opcode.

5.110.2.4 PutData()

```
void dng_opcode_FixVignetteRadial::PutData (
            dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References DNG_REQUIRE, dng_stream::Put_real64(), and dng_stream::Put_uint32().

The documentation for this class was generated from the following files:

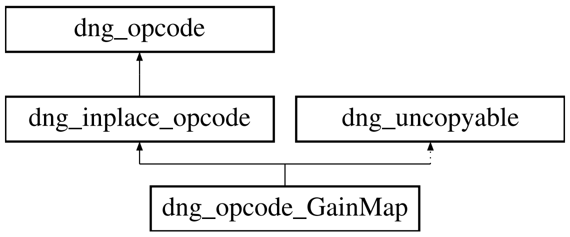
- dng_lens_correction.h
- dng_lens_correction.cpp

5.111 dng_opcode_GainMap Class Reference

An opcode to fix 2D spatially-varying light falloff or color casts (i.e., uniformity issues). This is commonly due to shading.

```
#include <dng_gain_map.h>
```

Inheritance diagram for dng_opcode_GainMap:



Public Member Functions

- dng_opcode_GainMap (const dng_area_spec &areaSpec, AutoPtr< dng_gain_map > &gainMap)
- dng_opcode_GainMap (dng_host &host, dng_stream &stream)

  Construct a GainMap opcode from the specified stream.
- virtual void PutData (dng_stream &stream) const

  Write the opcode to the specified stream.
- virtual uint32 BufferPixelType (uint32)

  The pixel data type of this opcode.
- virtual dng_rect ModifiedBounds (const dng_rect &imageBounds)
- virtual void ProcessArea (dng_negative &negative, uint32 threadIndex, dng_pixel_buffer &buffer, const dng_rect &dstArea, const dng_rect &imageBounds)

  Apply the gain map.

Additional Inherited Members

5.111.1 Detailed Description

An opcode to fix 2D spatially-varying light falloff or color casts (i.e., uniformity issues). This is commonly due to shading.

5.111.2 Constructor & Destructor Documentation

5.111.2.1 dng_opcode_GainMap()

```
dng_opcode_GainMap::dng_opcode_GainMap (
                const dng_area_spec & areaSpec,
                AutoPtr< dng_gain_map > & gainMap )
```

Construct a GainMap opcode for the specified image area and the specified gain map.

5.111.3 Member Function Documentation

5.111.3.1 ModifiedBounds()

```
virtual dng_rect dng_opcode_GainMap::ModifiedBounds (
            const dng_rect & imageBounds )  [inline], [virtual]
```

The adjusted bounds (processing area) of this opcode. It is limited to the intersection of the specified image area and the GainMap area.

Reimplemented from dng_inplace_opcode.

References dng_area_spec::Overlap().

The documentation for this class was generated from the following files:

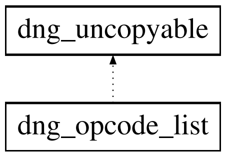
- dng_gain_map.h
- dng_gain_map.cpp

5.112 dng_opcode_list Class Reference

A list of opcodes.

```
#include <dng_opcode_list.h>
```

Inheritance diagram for dng_opcode_list:



Public Member Functions

- dng_opcode_list (uint32 stage)
  Create an empty opcode list for the specific image stage (1, 2, or 3).
- bool IsEmpty () const
  Is the opcode list empty?
- bool NotEmpty () const
  Does the list contain at least 1 opcode?
- bool AlwaysApply () const
  Should the opcode list always be applied to the image?
- void SetAlwaysApply ()
- uint32 Count () const

The number of opcodes in this list.
- dng_opcode & Entry (uint32 index)
- const dng_opcode & Entry (uint32 index) const
- void Clear ()

  Remove all opcodes from the list.
- void Swap (dng_opcode_list &otherList)

  Swap two opcode lists.
- uint32 MinVersion (bool includeOptional) const
- void Apply (dng_host &host, dng_negative &negative, AutoPtr< dng_image > &image)
- void Append (AutoPtr< dng_opcode > &opcode)

  Append the specified opcode to this list.
- dng_memory_block ∗ Spool (dng_host &host) const
- void FingerprintToStream (dng_stream &stream) const

  Write a fingerprint of this opcode list to the specified stream.
- void Parse (dng_host &host, dng_stream &stream, uint32 byteCount, uint64 streamOffset)

5.112.1 Detailed Description

A list of opcodes.

5.112.2 Member Function Documentation

5.112.2.1 Apply()

```
void dng_opcode_list::Apply (
            dng_host & host,
            dng_negative & negative,
            AutoPtr< dng_image > & image )
```

Apply this opcode list to the specified image with corresponding negative.

References dng_opcode::AboutToApply(), dng_opcode::Apply(), dng_image::Bounds(), Count(), DNG_REQUIRE, Entry(), AutoPtr< T >::Get(), and dng_image::Planes().

Referenced by dng_host::ApplyOpcodeList().

5.112.2.2 Entry() [1/2]

```
dng_opcode& dng_opcode_list::Entry (
            uint32 index ) [inline]
```

Retrieve read/write opcode by index (must be in the range 0 to Count () - 1).

Referenced by Apply().

5.112.2.3 Entry() [2/2]

```
const dng_opcode& dng_opcode_list::Entry (
            uint32 index ) const  [inline]
```

Retrieve read-only opcode by index (must be in the range 0 to Count () - 1).

5.112.2.4 MinVersion()

```
uint32 dng_opcode_list::MinVersion (
            bool includeOptional ) const
```

Return minimum DNG version required to support all opcodes in this list. If includeOptional is set to true, then this calculation will include optional opcodes.

Referenced by FingerprintToStream(), and Spool().

5.112.2.5 Parse()

```
void dng_opcode_list::Parse (
            dng_host & host,
            dng_stream & stream,
            uint32 byteCount,
            uint64 streamOffset )
```

Read an opcode list from the specified stream, starting at the specified offset (streamOffset, in bytes). byteCount is provided for error checking purposes. A bad format exception will be thrown if the length of the opcode stream does not exactly match byteCount.

References Append(), Clear(), dng_stream::Get_uint32(), dng_host::Make_dng_opcode(), dng_stream::Position(), dng_stream::SetReadPosition(), and ThrowBadFormat().

5.112.2.6 SetAlwaysApply()

```
void dng_opcode_list::SetAlwaysApply ( )  [inline]
```

Set internal flag to indicate this opcode list should always be applied.

5.112.2.7 Spool()

```
dng_memory_block * dng_opcode_list::Spool (
            dng_host & host ) const
```

Serialize this opcode list to a block of memory. The caller is responsible for deleting this block.

References dng_host::Allocator(), AlwaysApply(), IsEmpty(), MinVersion(), dng_stream::SetBigEndian(), and Throw↩ProgramError().

The documentation for this class was generated from the following files:

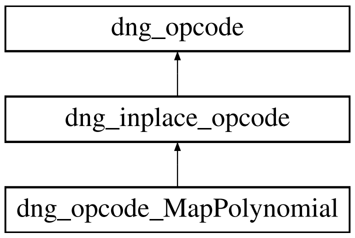
- dng_opcode_list.h
- dng_opcode_list.cpp

5.113 dng_opcode_MapPolynomial Class Reference

An opcode to apply a 1D function (represented as a polynomial) to an image area.

```
#include <dng_misc_opcodes.h>
```

Inheritance diagram for dng_opcode_MapPolynomial:



Public Types

- enum { **kMaxDegree** = 8 }

Public Member Functions

- dng_opcode_MapPolynomial (const dng_area_spec &areaSpec, uint32 degree, const real64 ∗coefficient)
- **dng_opcode_MapPolynomial** (dng_stream &stream)
- virtual void PutData (dng_stream &stream) const
- virtual uint32 BufferPixelType (uint32 imagePixelType)

    The pixel data type of this opcode.
- virtual dng_rect ModifiedBounds (const dng_rect &imageBounds)
- virtual void ProcessArea (dng_negative &negative, uint32 threadIndex, dng_pixel_buffer &buffer, const dng_rect &dstArea, const dng_rect &imageBounds)
- uint32 **Degree** () const
- const real64 ∗ **Coefficients** () const

Protected Member Functions

- virtual void **DoProcess** (dng_pixel_buffer &buffer, const dng_rect &area, const uint32 plane, const uint32 row↩ Pitch, const uint32 colPitch, const real32 ∗coefficients, const uint32 degree, uint16 blackLevel) const

Protected Attributes

- dng_area_spec **fAreaSpec**
- uint32 **fDegree**
- real64 **fCoefficient** [kMaxDegree+1]
- real32 **fCoefficient32** [kMaxDegree+1]

5.113.1 Detailed Description

An opcode to apply a 1D function (represented as a polynomial) to an image area.

5.113.2 Constructor & Destructor Documentation

5.113.2.1 dng_opcode_MapPolynomial()

```
dng_opcode_MapPolynomial::dng_opcode_MapPolynomial (
             const dng_area_spec & areaSpec,
             uint32 degree,
             const real64 * coefficient )
```

Create a MapPolynomial opcode with the specified area, polynomial degree, and polynomial coefficients. The function that will be applied to each pixel x is:

f (x) = coefficient [0] + ((x * coefficient [1]) + (x^2 * coefficient [2]) + (x^3 * coefficient [3]) + (x^4 * coefficient [4]) ...

5.113.3 Member Function Documentation

5.113.3.1 ModifiedBounds()

```
dng_rect dng_opcode_MapPolynomial::ModifiedBounds (
             const dng_rect & imageBounds )  [virtual]
```

The adjusted bounds (processing area) of this opcode. It is limited to the intersection of the specified image area and the GainMap area.

Reimplemented from dng_inplace_opcode.

References dng_area_spec::Overlap().

5.113.3.2 ProcessArea()

```
void dng_opcode_MapPolynomial::ProcessArea (
             dng_negative & negative,
             uint32 threadIndex,
             dng_pixel_buffer & buffer,
             const dng_rect & dstArea,
             const dng_rect & imageBounds )  [virtual]
```

Implements image processing operation in a single buffer. The source pixels are provided as input to the buffer, and this routine calculates and writes the destination pixels to the same buffer. Ideally, no allocation should be done in this routine.

Parameters

negative	The negative associated with the pixels to be processed.
threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Prepare method.
buffer	Source and Destination pixels.
dstArea	Destination pixel processing area.
imageBounds	Total image area to be processed; dstArea will always lie within these bounds.

Implements dng_inplace_opcode.

References dng_area_spec::ColPitch(), dng_area_spec::Overlap(), dng_area_spec::Plane(), dng_area_spec::Planes(), dng_pixel_buffer::Planes(), dng_area_spec::RowPitch(), and dng_opcode::Stage().

5.113.3.3 PutData()

```
void dng_opcode_MapPolynomial::PutData (
            dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References dng_stream::Put_real64(), dng_stream::Put_uint32(), and dng_area_spec::PutData().

The documentation for this class was generated from the following files:

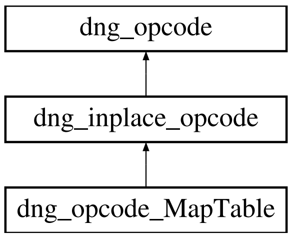
- dng_misc_opcodes.h
- dng_misc_opcodes.cpp

5.114 dng_opcode_MapTable Class Reference

An opcode to apply a 1D function (represented as a 16-bit table) to an image area.

```
#include <dng_misc_opcodes.h>
```

Inheritance diagram for dng_opcode_MapTable:



Public Member Functions

- dng_opcode_MapTable (dng_host &host, const dng_area_spec &areaSpec, const uint16 *table, uint32 count=0x10000)
- **dng_opcode_MapTable** (dng_host &host, dng_stream &stream)
- virtual void PutData (dng_stream &stream) const
- virtual uint32 BufferPixelType (uint32 imagePixelType)

    The pixel data type of this opcode.
- virtual dng_rect ModifiedBounds (const dng_rect &imageBounds)
- virtual void Prepare (dng_negative &negative, uint32 threadCount, const dng_point &tileSize, const dng_rect &imageBounds, uint32 imagePlanes, uint32 bufferPixelType, dng_memory_allocator &allocator)
- virtual void ProcessArea (dng_negative &negative, uint32 threadIndex, dng_pixel_buffer &buffer, const dng_rect &dstArea, const dng_rect &imageBounds)

Additional Inherited Members

5.114.1 Detailed Description

An opcode to apply a 1D function (represented as a 16-bit table) to an image area.

5.114.2 Constructor & Destructor Documentation

5.114.2.1 dng_opcode_MapTable()

```
dng_opcode_MapTable::dng_opcode_MapTable (
            dng_host & host,
            const dng_area_spec & areaSpec,
            const uint16 * table,
            uint32 count = 0x10000 )
```

Create a MapTable opcode with the specified area, table, and number of table entries.

5.114.3 Member Function Documentation

5.114.3.1 ModifiedBounds()

```
dng_rect dng_opcode_MapTable::ModifiedBounds (
            const dng_rect & imageBounds ) [virtual]
```

The adjusted bounds (processing area) of this opcode. It is limited to the intersection of the specified image area and the GainMap area.

Reimplemented from dng_inplace_opcode.

References dng_area_spec::Overlap().

5.114.3.2 Prepare()

```
void dng_opcode_MapTable::Prepare (
            dng_negative & negative,
            uint32 threadCount,
            const dng_point & tileSize,
            const dng_rect & imageBounds,
            uint32 imagePlanes,
            uint32 bufferPixelType,
            dng_memory_allocator & allocator )  [virtual]
```

Startup method called before any processing is performed on pixel areas. It can be used to allocate (per-thread) memory and setup tasks.

Parameters

negative	The negative object to be processed.
threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
imageBounds	Total size of image to be processed.
imagePlanes	Number of planes in the image. Less than or equal to kMaxColorPlanes.
bufferPixelType	Pixel type of image buffer (see dng_tag_types.h).
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.

Reimplemented from dng_inplace_opcode.

References dng_memory_allocator::Allocate(), dng_memory_block::Buffer_uint16(), AutoPtr< T >::Reset(), and dng↩_opcode::Stage().

5.114.3.3 ProcessArea()

```
void dng_opcode_MapTable::ProcessArea (
            dng_negative & negative,
            uint32 threadIndex,
            dng_pixel_buffer & buffer,
            const dng_rect & dstArea,
            const dng_rect & imageBounds )  [virtual]
```

Implements image processing operation in a single buffer. The source pixels are provided as input to the buffer, and this routine calculates and writes the destination pixels to the same buffer. Ideally, no allocation should be done in this routine.

Parameters

negative	The negative associated with the pixels to be processed.
threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Prepare method.
buffer	Source and Destination pixels.
dstArea	Destination pixel processing area.
imageBounds	Total image area to be processed; dstArea will always lie within these bounds.

Implements dng_inplace_opcode.

References dng_memory_block::Buffer_uint16(), AutoPtr< T >::Get(), dng_area_spec::Overlap(), dng_area_spec::↩
Plane(), dng_area_spec::Planes(), and dng_pixel_buffer::Planes().

5.114.3.4 PutData()

```
void dng_opcode_MapTable::PutData (
            dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References dng_memory_block::Buffer_uint16(), dng_stream::Put_uint16(), dng_stream::Put_uint32(), and dng_area↩
_spec::PutData().

The documentation for this class was generated from the following files:

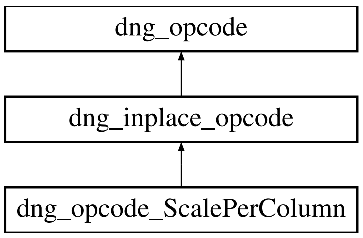
- dng_misc_opcodes.h
- dng_misc_opcodes.cpp

5.115 dng_opcode_ScalePerColumn Class Reference

An opcode to apply a scale factor that varies per column. Within a column, the same scale factor is applied to all specified pixels.

```
#include <dng_misc_opcodes.h>
```

Inheritance diagram for dng_opcode_ScalePerColumn:



Public Member Functions

- dng_opcode_ScalePerColumn (const dng_area_spec &areaSpec, AutoPtr< dng_memory_block > &table)
- **dng_opcode_ScalePerColumn** (dng_host &host, dng_stream &stream)
- virtual void PutData (dng_stream &stream) const
- virtual uint32 BufferPixelType (uint32 imagePixelType)

    The pixel data type of this opcode.
- virtual dng_rect ModifiedBounds (const dng_rect &imageBounds)
- virtual void ProcessArea (dng_negative &negative, uint32 threadIndex, dng_pixel_buffer &buffer, const dng_rect &dstArea, const dng_rect &imageBounds)

Additional Inherited Members

5.115.1 Detailed Description

An opcode to apply a scale factor that varies per column. Within a column, the same scale factor is applied to all specified pixels.

5.115.2 Constructor & Destructor Documentation

5.115.2.1 dng_opcode_ScalePerColumn()

```
dng_opcode_ScalePerColumn::dng_opcode_ScalePerColumn (
            const dng_area_spec & areaSpec,
            AutoPtr< dng_memory_block > & table )
```

Create a ScalePerColumn opcode with the specified area and column scale factors (specified as a table of 32-bit floats).

5.115.3 Member Function Documentation

5.115.3.1 ModifiedBounds()

```
dng_rect dng_opcode_ScalePerColumn::ModifiedBounds (
            const dng_rect & imageBounds ) [virtual]
```

The adjusted bounds (processing area) of this opcode. It is limited to the intersection of the specified image area and the GainMap area.

Reimplemented from dng_inplace_opcode.

References dng_area_spec::Overlap().

5.115.3.2 ProcessArea()

```
void dng_opcode_ScalePerColumn::ProcessArea (
            dng_negative & negative,
            uint32 threadIndex,
            dng_pixel_buffer & buffer,
            const dng_rect & dstArea,
            const dng_rect & imageBounds )  [virtual]
```

Implements image processing operation in a single buffer. The source pixels are provided as input to the buffer, and this routine calculates and writes the destination pixels to the same buffer. Ideally, no allocation should be done in this routine.

Parameters

negative	The negative associated with the pixels to be processed.
threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Prepare method.
buffer	Source and Destination pixels.
dstArea	Destination pixel processing area.
imageBounds	Total image area to be processed; dstArea will always lie within these bounds.

Implements dng_inplace_opcode.

References dng_area_spec::Area(), dng_memory_block::Buffer_real32(), dng_area_spec::ColPitch(), dng_pixel_↩ buffer::DirtyPixel_real32(), dng_area_spec::Overlap(), dng_area_spec::Plane(), dng_area_spec::Planes(), dng_pixel↩ _buffer::Planes(), dng_area_spec::RowPitch(), dng_pixel_buffer::RowStep(), and dng_opcode::Stage().

5.115.3.3 PutData()

```
void dng_opcode_ScalePerColumn::PutData (
            dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References dng_area_spec::Area(), dng_memory_block::Buffer_real32(), dng_area_spec::ColPitch(), dng_stream::↩ Put_real32(), dng_stream::Put_uint32(), and dng_area_spec::PutData().

The documentation for this class was generated from the following files:

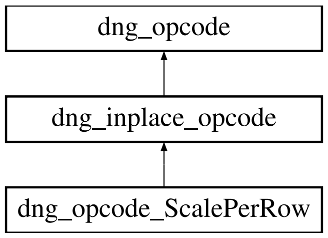
- dng_misc_opcodes.h
- dng_misc_opcodes.cpp

5.116 dng_opcode_ScalePerRow Class Reference

An opcode to apply a scale factor that varies per row. Within a row, the same scale factor is applied to all specified pixels.

`#include <dng_misc_opcodes.h>`

Inheritance diagram for dng_opcode_ScalePerRow:



Public Member Functions

- dng_opcode_ScalePerRow (const dng_area_spec &areaSpec, AutoPtr< dng_memory_block > &table)
- **dng_opcode_ScalePerRow** (dng_host &host, dng_stream &stream)
- virtual void PutData (dng_stream &stream) const
- virtual uint32 BufferPixelType (uint32 imagePixelType)

  The pixel data type of this opcode.
- virtual dng_rect ModifiedBounds (const dng_rect &imageBounds)
- virtual void ProcessArea (dng_negative &negative, uint32 threadIndex, dng_pixel_buffer &buffer, const dng_rect &dstArea, const dng_rect &imageBounds)

Additional Inherited Members

5.116.1 Detailed Description

An opcode to apply a scale factor that varies per row. Within a row, the same scale factor is applied to all specified pixels.

5.116.2 Constructor & Destructor Documentation

5.116.2.1 dng_opcode_ScalePerRow()

```
dng_opcode_ScalePerRow::dng_opcode_ScalePerRow (
            const dng_area_spec & areaSpec,
            AutoPtr< dng_memory_block > & table )
```

Create a ScalePerRow opcode with the specified area and row scale factors (specified as a table of 32-bit floats).

5.116.3 Member Function Documentation

5.116.3.1 ModifiedBounds()

```
dng_rect dng_opcode_ScalePerRow::ModifiedBounds (
             const dng_rect & imageBounds )  [virtual]
```

The adjusted bounds (processing area) of this opcode. It is limited to the intersection of the specified image area and the GainMap area.

Reimplemented from dng_inplace_opcode.

References dng_area_spec::Overlap().

5.116.3.2 ProcessArea()

```
void dng_opcode_ScalePerRow::ProcessArea (
             dng_negative & negative,
             uint32 threadIndex,
             dng_pixel_buffer & buffer,
             const dng_rect & dstArea,
             const dng_rect & imageBounds )  [virtual]
```

Implements image processing operation in a single buffer. The source pixels are provided as input to the buffer, and this routine calculates and writes the destination pixels to the same buffer. Ideally, no allocation should be done in this routine.

Parameters

negative	The negative associated with the pixels to be processed.
threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Prepare method.
buffer	Source and Destination pixels.
dstArea	Destination pixel processing area.
imageBounds	Total image area to be processed; dstArea will always lie within these bounds.

Implements dng_inplace_opcode.

References dng_area_spec::Area(), dng_memory_block::Buffer_real32(), dng_area_spec::ColPitch(), dng_pixel_↩ buffer::DirtyPixel_real32(), dng_area_spec::Overlap(), dng_area_spec::Plane(), dng_area_spec::Planes(), dng_pixel↩ _buffer::Planes(), dng_area_spec::RowPitch(), and dng_opcode::Stage().

5.116.3.3 PutData()

```
void dng_opcode_ScalePerRow::PutData (
            dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References dng_area_spec::Area(), dng_memory_block::Buffer_real32(), dng_stream::Put_real32(), dng_stream::↩ Put_uint32(), dng_area_spec::PutData(), and dng_area_spec::RowPitch().

The documentation for this class was generated from the following files:

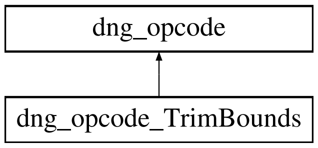
- dng_misc_opcodes.h
- dng_misc_opcodes.cpp

5.117 dng_opcode_TrimBounds Class Reference

Opcode to trim image to a specified rectangle.

```
#include <dng_misc_opcodes.h>
```

Inheritance diagram for dng_opcode_TrimBounds:



Public Member Functions

- dng_opcode_TrimBounds (const dng_rect &bounds)

  Create opcode to trim image to the specified bounds.
- **dng_opcode_TrimBounds** (dng_stream &stream)
- virtual void PutData (dng_stream &stream) const
- virtual void Apply (dng_host &host, dng_negative &negative, AutoPtr< dng_image > &image)

  Apply this opcode to the specified image with associated negative.

Additional Inherited Members

5.117.1 Detailed Description

Opcode to trim image to a specified rectangle.

5.117.2 Member Function Documentation

5.117.2.1 PutData()

```
void dng_opcode_TrimBounds::PutData (
            dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References dng_stream::Put_int32(), and dng_stream::Put_uint32().

The documentation for this class was generated from the following files:

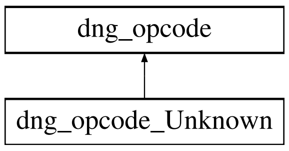
- dng_misc_opcodes.h
- dng_misc_opcodes.cpp

5.118 dng_opcode_Unknown Class Reference

Class to represent unknown opcodes (e.g, opcodes defined in future DNG versions).

```
#include <dng_opcodes.h>
```

Inheritance diagram for dng_opcode_Unknown:



Public Member Functions

- **dng_opcode_Unknown** (dng_host &host, uint32 opcodeID, dng_stream &stream)
- virtual void PutData (dng_stream &stream) const
- virtual void Apply (dng_host &host, dng_negative &negative, AutoPtr< dng_image > &image)

  Apply this opcode to the specified image with associated negative.

Additional Inherited Members

5.118.1 Detailed Description

Class to represent unknown opcodes (e.g, opcodes defined in future DNG versions).

5.118.2 Member Function Documentation

5.118.2.1 PutData()

```
void dng_opcode_Unknown::PutData (
            dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References dng_memory_block::Buffer(), AutoPtr< T >::Get(), dng_memory_block::LogicalSize(), dng_stream::Put(), and dng_stream::Put_uint32().

The documentation for this class was generated from the following files:

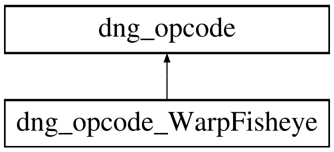
- dng_opcodes.h
- dng_opcodes.cpp

5.119 dng_opcode_WarpFisheye Class Reference

Warp opcode for fisheye camera model.

```
#include <dng_lens_correction.h>
```

Inheritance diagram for dng_opcode_WarpFisheye:



Public Member Functions

- **dng_opcode_WarpFisheye** (const dng_warp_params_fisheye &params, uint32 flags)
- **dng_opcode_WarpFisheye** (dng_stream &stream)
- virtual bool IsNOP () const
- virtual bool IsValidForNegative (const dng_negative &negative) const

  Is this opcode valid for the specified negative?
- virtual void PutData (dng_stream &stream) const
- virtual void Apply (dng_host &host, dng_negative &negative, AutoPtr< dng_image > &image)

  Apply this opcode to the specified image with associated negative.

Static Protected Member Functions

- static uint32 **ParamBytes** (uint32 planes)

Protected Attributes

- dng_warp_params_fisheye **fWarpParams**

Additional Inherited Members

5.119.1 Detailed Description

Warp opcode for fisheye camera model.

5.119.2 Member Function Documentation

5.119.2.1 IsNOP()

```
bool dng_opcode_WarpFisheye::IsNOP ( ) const  [virtual]
```

Is the opcode a NOP (i.e., does nothing)? An opcode could be a NOP for some specific parameters.

Reimplemented from dng_opcode.

References dng_warp_params::IsNOPAll().

5.119.2.2 PutData()

```
void dng_opcode_WarpFisheye::PutData (
             dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References dng_stream::Put_real64(), and dng_stream::Put_uint32().

The documentation for this class was generated from the following files:

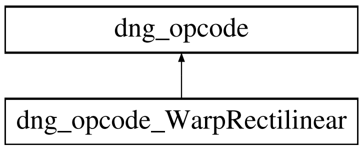
- dng_lens_correction.h
- dng_lens_correction.cpp

5.120 dng_opcode_WarpRectilinear Class Reference

Warp opcode for pinhole perspective (rectilinear) camera model.

```
#include <dng_lens_correction.h>
```

Inheritance diagram for dng_opcode_WarpRectilinear:



Public Member Functions

- **dng_opcode_WarpRectilinear** (const dng_warp_params_rectilinear &params, uint32 flags)
- **dng_opcode_WarpRectilinear** (dng_stream &stream)
- virtual bool IsNOP () const
- virtual bool IsValidForNegative (const dng_negative &negative) const

    Is this opcode valid for the specified negative?
- virtual void PutData (dng_stream &stream) const
- virtual void Apply (dng_host &host, dng_negative &negative, AutoPtr< dng_image > &image)

    Apply this opcode to the specified image with associated negative.
- bool **HasDistort** () const
- bool **HasLateralCA** () const
- const dng_warp_params_rectilinear & **Params** () const

Static Protected Member Functions

- static uint32 **ParamBytes** (uint32 planes)

Protected Attributes

- dng_warp_params_rectilinear **fWarpParams**

Additional Inherited Members

5.120.1 Detailed Description

Warp opcode for pinhole perspective (rectilinear) camera model.

5.120.2 Member Function Documentation

5.120.2.1 IsNOP()

```
bool dng_opcode_WarpRectilinear::IsNOP ( ) const  [virtual]
```

Is the opcode a NOP (i.e., does nothing)? An opcode could be a NOP for some specific parameters.

Reimplemented from dng_opcode.

References dng_warp_params::IsNOPAll().

5.120.2.2 PutData()

```
void dng_opcode_WarpRectilinear::PutData (
             dng_stream & stream ) const  [virtual]
```

Write opcode to a stream.

Parameters

stream	The stream to which to write the opcode data.

Reimplemented from dng_opcode.

References dng_stream::Put_real64(), and dng_stream::Put_uint32().

The documentation for this class was generated from the following files:

- dng_lens_correction.h
- dng_lens_correction.cpp

5.121 dng_orientation Class Reference

Public Types

- enum {
  kNormal = 0, **kRotate90CW** = 1, **kRotate180** = 2, **kRotate90CCW** = 3,
  kMirror = 4, **kMirror90CW** = 5, **kMirror180** = 6, **kMirror90CCW** = 7,
  kUnknown = 8 }

Public Member Functions

- void **SetAdobe** (uint32 adobe)
- uint32 **GetAdobe** () const
- void **SetTIFF** (uint32 tiff)
- uint32 **GetTIFF** () const
- bool **IsValid** () const
- bool **NotValid** () const
- bool **FlipD** () const
- bool **FlipH** () const
- bool **FlipV** () const
- bool **operator==** (const dng_orientation &b) const
- bool **operator!=** (const dng_orientation &b) const
- dng_orientation **operator-** () const
- dng_orientation **operator+** (const dng_orientation &b) const
- dng_orientation **operator-** (const dng_orientation &b) const
- void **operator+=** (const dng_orientation &b)
- void **operator-=** (const dng_orientation &b)
- bool **CalcForwardMatrix3by3** (dng_matrix &matrix, bool horizontalFirstRow) const
- bool **CalcForwardMatrix4by4** (dng_matrix &matrix, bool horizontalFirstRow) const

Static Public Member Functions

- static dng_orientation **AdobeToDNG** (uint32 adobe)
- static dng_orientation **TIFFtoDNG** (uint32 tiff)
- static dng_orientation **Normal** ()
- static dng_orientation **Rotate90CW** ()
- static dng_orientation **Rotate180** ()
- static dng_orientation **Rotate90CCW** ()
- static dng_orientation **Mirror** ()
- static dng_orientation **Mirror90CW** ()
- static dng_orientation **Mirror180** ()
- static dng_orientation **Mirror90CCW** ()
- static dng_orientation **Unknown** ()

The documentation for this class was generated from the following files:

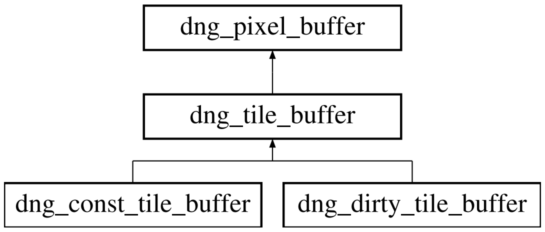
- dng_orientation.h
- dng_orientation.cpp

5.122 dng_pixel_buffer Class Reference

Holds a buffer of pixel data with "pixel geometry" metadata.

`#include <dng_pixel_buffer.h>`

Inheritance diagram for dng_pixel_buffer:



Public Member Functions

- dng_pixel_buffer (const dng_rect &area, uint32 plane, uint32 planes, uint32 pixelType, uint32 planarConfiguration, void ∗data)
- **dng_pixel_buffer** (const dng_pixel_buffer &buffer)
- dng_pixel_buffer & **operator=** (const dng_pixel_buffer &buffer)
- uint32 PixelRange () const
- const dng_rect & Area () const
- uint32 Planes () const
- int32 RowStep () const
- int32 PlaneStep () const
- const void ∗ ConstPixel (int32 row, int32 col, uint32 plane=0) const
- void ∗ DirtyPixel (int32 row, int32 col, uint32 plane=0)
- const uint8 ∗ ConstPixel_uint8 (int32 row, int32 col, uint32 plane=0) const
- const uint8 ∗ **ConstPixel_uint8_overrideType** (int32 row, int32 col, uint32 plane=0) const
- uint8 ∗ DirtyPixel_uint8 (int32 row, int32 col, uint32 plane=0)
- uint8 ∗ **DirtyPixel_uint8_overrideType** (int32 row, int32 col, uint32 plane=0)
- const int8 ∗ ConstPixel_int8 (int32 row, int32 col, uint32 plane=0) const
- int8 ∗ DirtyPixel_int8 (int32 row, int32 col, uint32 plane=0)
- const uint16 ∗ ConstPixel_uint16 (int32 row, int32 col, uint32 plane=0) const
- uint16 ∗ DirtyPixel_uint16 (int32 row, int32 col, uint32 plane=0)
- const int16 ∗ ConstPixel_int16 (int32 row, int32 col, uint32 plane=0) const
- int16 ∗ DirtyPixel_int16 (int32 row, int32 col, uint32 plane=0)
- const uint32 ∗ ConstPixel_uint32 (int32 row, int32 col, uint32 plane=0) const
- uint32 ∗ DirtyPixel_uint32 (int32 row, int32 col, uint32 plane=0)
- const int32 ∗ ConstPixel_int32 (int32 row, int32 col, uint32 plane=0) const
- int32 ∗ DirtyPixel_int32 (int32 row, int32 col, uint32 plane=0)
- const real32 ∗ ConstPixel_real32 (int32 row, int32 col, uint32 plane=0) const
- real32 ∗ DirtyPixel_real32 (int32 row, int32 col, uint32 plane=0)
- void SetConstant (const dng_rect &area, uint32 plane, uint32 planes, uint32 value)
- void SetConstant_uint8 (const dng_rect &area, uint32 plane, uint32 planes, uint8 value)
- void SetConstant_uint16 (const dng_rect &area, uint32 plane, uint32 planes, uint16 value)
- void SetConstant_int16 (const dng_rect &area, uint32 plane, uint32 planes, int16 value)

- void SetConstant_uint32 (const dng_rect &area, uint32 plane, uint32 planes, uint32 value)
- void SetConstant_real32 (const dng_rect &area, uint32 plane, uint32 planes, real32 value)
- void SetZero (const dng_rect &area, uint32 plane, uint32 planes)
- void CopyArea (const dng_pixel_buffer &src, const dng_rect &area, uint32 srcPlane, uint32 dstPlane, uint32 planes)
- void CopyArea (const dng_pixel_buffer &src, const dng_rect &area, uint32 plane, uint32 planes)
- void RepeatArea (const dng_rect &srcArea, const dng_rect &dstArea)
- void RepeatSubArea (const dng_rect subArea, uint32 repeatV=1, uint32 repeatH=1)

  Replicates a sub-area of a buffer to fill the entire buffer.
- void ShiftRight (uint32 shift)
- void FlipH ()
- void FlipV ()
- void FlipZ ()
- bool EqualArea (const dng_pixel_buffer &rhs, const dng_rect &area, uint32 plane, uint32 planes) const
- real64 MaximumDifference (const dng_pixel_buffer &rhs, const dng_rect &area, uint32 plane, uint32 planes) const

Static Public Member Functions

- static dng_point RepeatPhase (const dng_rect &srcArea, const dng_rect &dstArea)

Public Attributes

- dng_rect **fArea**
- uint32 **fPlane**
- uint32 **fPlanes**
- int32 **fRowStep**
- int32 **fColStep**
- int32 **fPlaneStep**
- uint32 **fPixelType**
- uint32 **fPixelSize**
- void ∗ **fData**
- bool **fDirty**

5.122.1 Detailed Description

Holds a buffer of pixel data with "pixel geometry" metadata.

The pixel geometry describes the layout in terms of how many planes, rows and columns plus the steps (in bytes) between each column, row and plane.

5.122.2 Constructor & Destructor Documentation

5.122.2.1 dng_pixel_buffer()

```
dng_pixel_buffer::dng_pixel_buffer (
            const dng_rect & area,
            uint32 plane,
            uint32 planes,
            uint32 pixelType,
            uint32 planarConfiguration,
            void * data )
```

Note: This constructor is for internal use only and should not be considered part of the DNG SDK API.

Initialize the pixel buffer according to the given parameters (see below). May throw an error if arithmetic overflow occurs when computing the row, column or plane step, or if an invalid value was passed for planarConfiguration.

Parameters

area	Area covered by the pixel buffer
plane	Index of the first plane
planes	Number of planes
pixelType	Pixel data type (one of the values defined in dng_tag_types.h)
planarConfiguration	Layout of the pixel planes in memory: One of pcInterleaved, pcPlanar, or pcRowInterleaved (defined in dng_tag_values.h)
data	Pointer to the pixel data

References ThrowOverflow(), and ThrowProgramError().

5.122.3 Member Function Documentation

5.122.3.1 Area()

```
const dng_rect& dng_pixel_buffer::Area ( ) const  [inline]
```

Get extent of pixels in buffer

Return values

Rectangle	giving valid extent of buffer.

Referenced by dng_filter_opcode_task::ProcessArea().

5.122.3.2 ConstPixel()

```
const void* dng_pixel_buffer::ConstPixel (
            int32 row,
            int32 col,
            uint32 plane = 0 ) const  [inline]
```

Get read-only untyped (void ∗) pointer to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as void ∗.

Referenced by CopyArea(), EqualArea(), MaximumDifference(), dng_limit_float_depth_task< simd >::Process(), dng←_image::Put(), and RepeatArea().

5.122.3.3 ConstPixel_int16()

```
const int16* dng_pixel_buffer::ConstPixel_int16 (
            int32 row,
            int32 col,
            uint32 plane = 0 ) const  [inline]
```

Get read-only int16 ∗ to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as int16 ∗.

5.122.3.4 ConstPixel_int32()

```
const int32* dng_pixel_buffer::ConstPixel_int32 (
            int32 row,
            int32 col,
            uint32 plane = 0 ) const  [inline]
```

Get read-only int32 ∗ to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as int32 ∗.

5.122.3.5 ConstPixel_int8()

```
const int8* dng_pixel_buffer::ConstPixel_int8 (
             int32 row,
             int32 col,
             uint32 plane = 0 ) const  [inline]
```

Get read-only int8 ∗ to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as int8 ∗.

5.122.3.6 ConstPixel_real32()

```
const real32* dng_pixel_buffer::ConstPixel_real32 (
             int32 row,
             int32 col,
             uint32 plane = 0 ) const  [inline]
```

Get read-only real32 ∗ to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as real32 ∗.

Referenced by dng_resample_task::ProcessArea(), and dng_filter_warp::ProcessArea().

5.122.3.7 ConstPixel_uint16()

```
const uint16* dng_pixel_buffer::ConstPixel_uint16 (
             int32 row,
             int32 col,
             uint32 plane = 0 ) const  [inline]
```

Get read-only uint16 ∗ to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as uint16 ∗.

Referenced by dng_encode_proxy_task::Process(), dng_opcode_FixBadPixelsConstant::ProcessArea(), and dng_↩
fast_interpolator::ProcessArea().

5.122.3.8 ConstPixel_uint32()

```
const uint32* dng_pixel_buffer::ConstPixel_uint32 (
             int32 row,
             int32 col,
             uint32 plane = 0 ) const  [inline]
```

Get read-only uint32 ∗ to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as uint32 ∗.

5.122.3.9 ConstPixel_uint8()

```
const uint8* dng_pixel_buffer::ConstPixel_uint8 (
             int32 row,
             int32 col,
             uint32 plane = 0 ) const  [inline]
```

Get read-only uint8 ∗ to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as uint8 ∗.

5.122.3.10 CopyArea() [1/2]

```
void dng_pixel_buffer::CopyArea (
             const dng_pixel_buffer & src,
             const dng_rect & area,
             uint32 srcPlane,
             uint32 dstPlane,
             uint32 planes )
```

Copy image data from an area of one pixel buffer to same area of another.

Parameters

src	Buffer to copy from.
area	Rectangle of pixel buffer to copy.
srcPlane	Plane to start copy in src.
dstPlane	Plane to start copy in dst.
planes	Number of planes to copy.

References ConstPixel(), DirtyPixel(), and OptimizeOrder().

Referenced by CopyArea(), dng_opcode_FixBadPixelsConstant::ProcessArea(), and dng_opcode_FixBadPixelsList::↩
ProcessArea().

5.122.3.11 CopyArea() [2/2]

```
void dng_pixel_buffer::CopyArea (
             const dng_pixel_buffer & src,
             const dng_rect & area,
             uint32 plane,
             uint32 planes )  [inline]
```

Copy image data from an area of one pixel buffer to same area of another.

Parameters

src	Buffer to copy from.
area	Rectangle of pixel buffer to copy.
plane	Plane to start copy in src and this.
planes	Number of planes to copy.

References CopyArea().

5.122.3.12 DirtyPixel()

```
void* dng_pixel_buffer::DirtyPixel (
             int32 row,
             int32 col,
             uint32 plane = 0 )  [inline]
```

Get a writable untyped (void ∗) pointer to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as void ∗.

References DNG_ASSERT.

Referenced by CopyArea(), dng_image::Get(), dng_limit_float_depth_task< simd >::Process(), dng_render_task::↩
ProcessArea(), RepeatArea(), dng_simple_image::Rotate(), SetConstant(), ShiftRight(), and dng_simple_image::Trim().

5.122.3.13 DirtyPixel_int16()

```
int16* dng_pixel_buffer::DirtyPixel_int16 (
             int32 row,
             int32 col,
             uint32 plane = 0 )  [inline]
```

Get a writable int16 * to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as int16 *.

5.122.3.14 DirtyPixel_int32()

```
int32* dng_pixel_buffer::DirtyPixel_int32 (
             int32 row,
             int32 col,
             uint32 plane = 0 )  [inline]
```

Get a writable int32 * to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as int32 *.

5.122.3.15 DirtyPixel_int8()

```
int8* dng_pixel_buffer::DirtyPixel_int8 (
             int32 row,
             int32 col,
             uint32 plane = 0 )  [inline]
```

Get a writable int8 ∗ to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as int8 ∗.

5.122.3.16 DirtyPixel_real32()

```
real32* dng_pixel_buffer::DirtyPixel_real32 (
            int32 row,
            int32 col,
            uint32 plane = 0 )  [inline]
```

Get a writable real32 ∗ to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as real32 ∗.

Referenced by dng_opcode_GainMap::ProcessArea(), dng_opcode_DeltaPerRow::ProcessArea(), dng_opcode←_DeltaPerColumn::ProcessArea(), dng_opcode_ScalePerRow::ProcessArea(), dng_opcode_ScalePerColumn::←ProcessArea(), and dng_filter_warp::ProcessArea().

5.122.3.17 DirtyPixel_uint16()

```
uint16* dng_pixel_buffer::DirtyPixel_uint16 (
            int32 row,
            int32 col,
            uint32 plane = 0 )  [inline]
```

Get a writable uint16 ∗ to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as uint16 ∗.

Referenced by dng_opcode_FixBadPixelsConstant::ProcessArea(), and dng_fast_interpolator::ProcessArea().

5.122.3.18 DirtyPixel_uint32()

```
uint32* dng_pixel_buffer::DirtyPixel_uint32 (
             int32 row,
             int32 col,
             uint32 plane = 0 )  [inline]
```

Get a writable uint32 ∗ to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as uint32 ∗.

5.122.3.19 DirtyPixel_uint8()

```
uint8* dng_pixel_buffer::DirtyPixel_uint8 (
             int32 row,
             int32 col,
             uint32 plane = 0 )  [inline]
```

Get a writable uint8 ∗ to pixel data starting at a specific pixel in the buffer.

Parameters

row	Start row for buffer pointer.
col	Start column for buffer pointer.
plane	Start plane for buffer pointer.

Return values

Pointer	to pixel data as uint8 ∗.

Referenced by dng_encode_proxy_task::Process().

5.122.3.20 EqualArea()

```
bool dng_pixel_buffer::EqualArea (
            const dng_pixel_buffer & rhs,
            const dng_rect & area,
            uint32 plane,
            uint32 planes ) const
```

Return true if the contents of an area of the pixel buffer area are the same as those of another.

Parameters

rhs	Buffer to compare against.
area	Rectangle of pixel buffer to test.
plane	Plane to start comparing.
planes	Number of planes to compare.

Return values

bool	true if areas are equal, false otherwise.

References ConstPixel().

Referenced by dng_image::EqualArea().

5.122.3.21 FlipH()

```
void dng_pixel_buffer::FlipH ( )
```

Change metadata so pixels are iterated in opposite horizontal order. This operation does not require movement of actual pixel data.

5.122.3.22 FlipV()

```
void dng_pixel_buffer::FlipV ( )
```

Change metadata so pixels are iterated in opposite vertical order. This operation does not require movement of actual pixel data.

5.122.3.23 FlipZ()

```
void dng_pixel_buffer::FlipZ ( )
```

Change metadata so pixels are iterated in opposite plane order. This operation does not require movement of actual pixel data.

5.122.3.24 MaximumDifference()

```
real64 dng_pixel_buffer::MaximumDifference (
             const dng_pixel_buffer & rhs,
             const dng_rect & area,
             uint32 plane,
             uint32 planes ) const
```

Return the absolute value of the maximum difference between two pixel buffers. Used for comparison testing with tolerance

Parameters

rhs	Buffer to compare against.
area	Rectangle of pixel buffer to test.
plane	Plane to start comparing.
planes	Number of planes to compare.

Return values

larges	absolute value difference between the corresponding pixels each buffer across area.

References ConstPixel(), ThrowNotYetImplemented(), and ThrowProgramError().

5.122.3.25 PixelRange()

```
uint32 dng_pixel_buffer::PixelRange ( ) const
```

Get the range of pixel values.

Return values

Range	of value a pixel can take. (Meaning [0, max] for unsigned case. Signed case is biased so [-32768, max - 32768].)

5.122.3.26 Planes()

```
uint32 dng_pixel_buffer::Planes ( ) const  [inline]
```

Number of planes of image data.

Return values

Number	of planes held in buffer.

Referenced by dng_opcode_GainMap::ProcessArea(), dng_opcode_MapTable::ProcessArea(), dng_opcode_Map↩
Polynomial::ProcessArea(), dng_opcode_DeltaPerRow::ProcessArea(), dng_opcode_DeltaPerColumn::ProcessArea(), dng_opcode_ScalePerRow::ProcessArea(), and dng_opcode_ScalePerColumn::ProcessArea().

5.122.3.27 PlaneStep()

```
int32 dng_pixel_buffer::PlaneStep ( ) const  [inline]
```

Step, in pixels not bytes, between planes of data in buffer.

Return values

plane	step in pixels. May be negative.

5.122.3.28 RepeatArea()

```
void dng_pixel_buffer::RepeatArea (
             const dng_rect & srcArea,
             const dng_rect & dstArea )
```

Repeat the image data in srcArea across dstArea. (Generally used for padding operations.)

Parameters

srcArea	Area to repeat from.
dstArea	Area to fill with data from srcArea.

References ConstPixel(), DirtyPixel(), and RepeatPhase().

Referenced by RepeatSubArea().

5.122.3.29 RepeatPhase()

```
dng_point dng_pixel_buffer::RepeatPhase (
             const dng_rect & srcArea,
             const dng_rect & dstArea )  [static]
```

Calculate the offset phase of destination rectangle relative to source rectangle. Phase is based on a 0,0 origin and the notion of repeating srcArea across dstArea. It is the number of pixels into srcArea to start repeating from when tiling dstArea.

Return values

dng_point	containing horizontal and vertical phase.

References DNG_REPORT.

Referenced by RepeatArea().

5.122.3.30 RowStep()

```
int32 dng_pixel_buffer::RowStep ( ) const  [inline]
```

Step, in pixels not bytes, between rows of data in buffer.

Return values

row	step in pixels. May be negative.

Referenced by dng_opcode_DeltaPerColumn::ProcessArea(), dng_opcode_ScalePerColumn::ProcessArea(), and dng_filter_warp::ProcessArea().

5.122.3.31 SetConstant()

```
void dng_pixel_buffer::SetConstant (
             const dng_rect & area,
             uint32 plane,
             uint32 planes,
             uint32 value )
```

Initialize a rectangular area of pixel buffer to a constant.

Parameters

area	Rectangle of pixel buffer to set.
plane	Plane to start filling on.
planes	Number of planes to fill.
value	Constant value to set pixels to.

References DirtyPixel(), and OptimizeOrder().

Referenced by SetConstant_int16(), SetConstant_real32(), SetConstant_uint16(), SetConstant_uint32(), SetConstant↩_uint8(), and SetZero().

5.122.3.32 SetConstant_int16()

```
void dng_pixel_buffer::SetConstant_int16 (
             const dng_rect & area,
             uint32 plane,
             uint32 planes,
             int16 value )  [inline]
```

Initialize a rectangular area of pixel buffer to a constant signed 16-bit value.

Parameters

area	Rectangle of pixel buffer to set.
plane	Plane to start filling on.
planes	Number of planes to fill.
value	Constant int16 value to set pixels to.

References DNG_ASSERT, and SetConstant().

5.122.3.33 SetConstant_real32()

```
void dng_pixel_buffer::SetConstant_real32 (
             const dng_rect & area,
             uint32 plane,
             uint32 planes,
             real32 value )  [inline]
```

Initialize a rectangular area of pixel buffer to a constant real 32-bit value.

Parameters

area	Rectangle of pixel buffer to set.
plane	Plane to start filling on.
planes	Number of planes to fill.
value	Constant real32 value to set pixels to.

References DNG_ASSERT, and SetConstant().

5.122.3.34 SetConstant_uint16()

```
void dng_pixel_buffer::SetConstant_uint16 (
             const dng_rect & area,
             uint32 plane,
             uint32 planes,
             uint16 value )  [inline]
```

Initialize a rectangular area of pixel buffer to a constant unsigned 16-bit value.

Parameters

area	Rectangle of pixel buffer to set.
plane	Plane to start filling on.
planes	Number of planes to fill.
value	Constant uint16 value to set pixels to.

References DNG_ASSERT, and SetConstant().

5.122.3.35 SetConstant_uint32()

```
void dng_pixel_buffer::SetConstant_uint32 (
             const dng_rect & area,
             uint32 plane,
             uint32 planes,
             uint32 value )  [inline]
```

Initialize a rectangular area of pixel buffer to a constant unsigned 32-bit value.

Parameters

area	Rectangle of pixel buffer to set.
plane	Plane to start filling on.
planes	Number of planes to fill.
value	Constant uint32 value to set pixels to.

References DNG_ASSERT, and SetConstant().

5.122.3.36 SetConstant_uint8()

```
void dng_pixel_buffer::SetConstant_uint8 (
             const dng_rect & area,
             uint32 plane,
             uint32 planes,
             uint8 value )  [inline]
```

Initialize a rectangular area of pixel buffer to a constant unsigned 8-bit value.

Parameters

area	Rectangle of pixel buffer to set.
plane	Plane to start filling on.
planes	Number of planes to fill.
value	Constant uint8 value to set pixels to.

References DNG_ASSERT, and SetConstant().

5.122.3.37 SetZero()

```
void dng_pixel_buffer::SetZero (
            const dng_rect & area,
            uint32 plane,
            uint32 planes )
```

Initialize a rectangular area of pixel buffer to zeros.

Parameters

area	Rectangle of pixel buffer to zero.
plane	Plane to start filling on.
planes	Number of planes to fill.

References SetConstant(), and ThrowNotYetImplemented().

5.122.3.38 ShiftRight()

```
void dng_pixel_buffer::ShiftRight (
            uint32 shift )
```

Apply a right shift (C++ oerpator $>>$) to all pixel values. Only implemented for 16-bit (signed or unsigned) pixel buffers.

Parameters

shift	Number of bits by which to right shift each pixel value.

References DirtyPixel(), OptimizeOrder(), and ThrowNotYetImplemented().

The documentation for this class was generated from the following files:

- dng_pixel_buffer.h
- dng_pixel_buffer.cpp

5.123 dng_point Class Reference

Public Member Functions

- **dng_point** (int32 vv, int32 hh)
- bool **operator==** (const dng_point &pt) const
- bool **operator!=** (const dng_point &pt) const
- real64 **Length** () const

Public Attributes

- int32 **v**
- int32 **h**

The documentation for this class was generated from the following file:

- dng_point.h

5.124 dng_point_real64 Class Reference

Public Member Functions

- **dng_point_real64** (real64 vv, real64 hh)
- **dng_point_real64** (const dng_point &pt)
- bool **operator==** (const dng_point_real64 &pt) const
- bool **operator!=** (const dng_point_real64 &pt) const
- dng_point **Round** () const
- real64 **Length** () const
- void **Scale** (real64 scale)
- void **Normalize** ()

Public Attributes

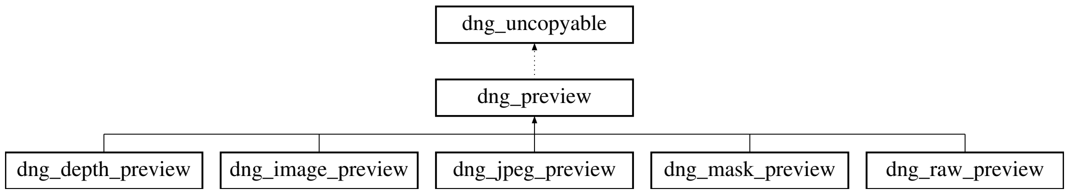
- real64 **v**
- real64 **h**

The documentation for this class was generated from the following file:

- dng_point.h

5.125 dng_preview Class Reference

Inheritance diagram for dng_preview:



Public Member Functions

- virtual dng_basic_tag_set ∗ **AddTagSet** (dng_tiff_directory &directory) const =0
- virtual void **WriteData** (dng_host &host, dng_image_writer &writer, dng_basic_tag_set &basic, dng_stream &stream) const =0

Public Attributes

- dng_preview_info **fInfo**

The documentation for this class was generated from the following files:

- dng_preview.h
- dng_preview.cpp

5.126 dng_preview_info Class Reference

Public Attributes

- bool **fIsPrimary**
- dng_string **fApplicationName**
- dng_string **fApplicationVersion**
- dng_string **fSettingsName**
- dng_fingerprint **fSettingsDigest**
- PreviewColorSpaceEnum **fColorSpace**
- dng_string **fDateTime**
- real64 **fRawToPreviewGain**
- uint32 **fCacheVersion**

The documentation for this class was generated from the following files:

- dng_ifd.h
- dng_ifd.cpp

5.127 dng_preview_list Class Reference

Public Member Functions

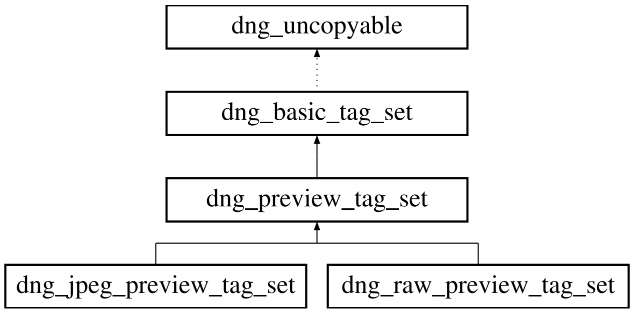
- uint32 **Count** () const
- const dng_preview & **Preview** (uint32 index) const
- void **Append** (AutoPtr< dng_preview > &preview)

The documentation for this class was generated from the following files:

- dng_preview.h
- dng_preview.cpp

5.128 dng_preview_tag_set Class Reference

Inheritance diagram for dng_preview_tag_set:



Public Member Functions

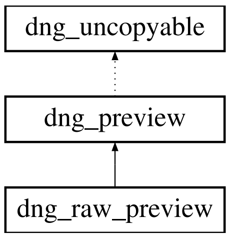
- **dng_preview_tag_set** (dng_tiff_directory &directory, const dng_preview &preview, const dng_ifd &ifd)

The documentation for this class was generated from the following file:

- dng_preview.cpp

5.129 dng_raw_preview Class Reference

Inheritance diagram for dng_raw_preview:



Public Member Functions

- virtual dng_basic_tag_set ∗ **AddTagSet** (dng_tiff_directory &directory) const
- virtual void **WriteData** (dng_host &host, dng_image_writer &writer, dng_basic_tag_set &basic, dng_stream &stream) const

Public Attributes

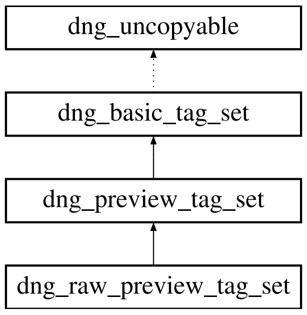
- AutoPtr< dng_image > **fImage**
- AutoPtr< dng_memory_block > **fOpcodeList2Data**
- real64 **fBlackLevel** [kMaxSamplesPerPixel]
- int32 **fCompressionQuality**

The documentation for this class was generated from the following files:

- dng_preview.h
- dng_preview.cpp

5.130 dng_raw_preview_tag_set Class Reference

Inheritance diagram for dng_raw_preview_tag_set:



Public Member Functions

- **dng_raw_preview_tag_set** (dng_tiff_directory &directory, const dng_raw_preview &preview, const dng_ifd &ifd)

The documentation for this class was generated from the following file:

- dng_preview.cpp

5.131 dng_read_image Class Reference

Public Member Functions

- virtual bool **CanRead** (const dng_ifd &ifd)
- virtual void **Read** (dng_host &host, const dng_ifd &ifd, dng_stream &stream, dng_image &image, dng_jpeg_image ∗jpegImage, dng_fingerprint ∗jpegDigest)

Protected Types

- enum { **kImageBufferSize** = 128 ∗ 1024 }

Protected Member Functions

- virtual bool **ReadUncompressed** (dng_host &host, const dng_ifd &ifd, dng_stream &stream, dng_image &image, const dng_rect &tileArea, uint32 plane, uint32 planes, AutoPtr< dng_memory_block > &uncompressed←Buffer, AutoPtr< dng_memory_block > &subTileBlockBuffer)
- virtual void **DecodeLossyJPEG** (dng_host &host, dng_image &image, const dng_rect &tileArea, uint32 plane, uint32 planes, uint32 photometricInterpretation, uint32 jpegDataSize, uint8 ∗jpegDataInMemory, bool using←MultipleThreads)
- virtual bool **ReadBaselineJPEG** (dng_host &host, const dng_ifd &ifd, dng_stream &stream, dng_image &image, const dng_rect &tileArea, uint32 plane, uint32 planes, uint32 tileByteCount, uint8 ∗jpegDataInMemory, bool usingMultipleThreads)
- virtual bool **ReadLosslessJPEG** (dng_host &host, const dng_ifd &ifd, dng_stream &stream, dng_image &image, const dng_rect &tileArea, uint32 plane, uint32 planes, uint32 tileByteCount, AutoPtr< dng_memory_block > &uncompressedBuffer, AutoPtr< dng_memory_block > &subTileBlockBuffer)
- virtual bool **CanReadTile** (const dng_ifd &ifd)
- virtual bool **NeedsCompressedBuffer** (const dng_ifd &ifd)
- virtual void **ByteSwapBuffer** (dng_host &host, dng_pixel_buffer &buffer)
- virtual void **DecodePredictor** (dng_host &host, const dng_ifd &ifd, dng_pixel_buffer &buffer)
- virtual void **ReadTile** (dng_host &host, const dng_ifd &ifd, dng_stream &stream, dng_image &image, const dng_rect &tileArea, uint32 plane, uint32 planes, uint32 tileByteCount, AutoPtr< dng_memory_block > &compressedBuffer, AutoPtr< dng_memory_block > &uncompressedBuffer, AutoPtr< dng_memory_block > &subTileBlockBuffer, bool usingMultipleThreads)
- virtual void **DoReadTiles** (dng_host &host, const dng_ifd &ifd, dng_stream &stream, dng_image &image, dng_jpeg_image ∗jpegImage, dng_fingerprint ∗jpegTileDigest, uint32 outerSamples, uint32 innerSamples, uint32 tilesDown, uint32 tilesAcross, uint64 ∗tileOffset, uint32 ∗tileByteCount, uint32 compressedSize, uint32 uncompressedSize)

Protected Attributes

- AutoPtr< dng_memory_block > **fJPEGTables**

Friends

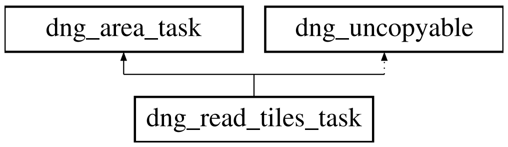
- class **dng_read_tiles_task**

The documentation for this class was generated from the following files:

- dng_read_image.h
- dng_read_image.cpp

5.132 dng_read_tiles_task Class Reference

Inheritance diagram for dng_read_tiles_task:



Public Member Functions

- **dng_read_tiles_task** (dng_read_image &readImage, dng_host &host, const dng_ifd &ifd, dng_stream &stream, dng_image &image, dng_jpeg_image ∗jpegImage, dng_fingerprint ∗jpegTileDigest, uint32 outerSamples, uint32 innerSamples, uint32 tilesDown, uint32 tilesAcross, uint64 ∗tileOffset, uint32 ∗tileByteCount, uint32 compressedSize, uint32 uncompressedSize)
- void Process (uint32 threadIndex, const dng_rect &tile, dng_abort_sniffer ∗sniffer)

  and progress updates.

Protected Member Functions

- void **ReadTask** (uint32 tileIndex, uint32 &byteCount, dng_memory_block ∗compressedBuffer)
- void **ProcessTask** (uint32 tileIndex, uint32 byteCount, dng_abort_sniffer ∗sniffer, AutoPtr< dng_memory_block > &compressedBuffer, AutoPtr< dng_memory_block > &uncompressedBuffer, AutoPtr< dng_memory_block > &subTileBlockBuffer)

Protected Attributes

- dng_read_image & **fReadImage**
- dng_host & **fHost**
- const dng_ifd & **fIFD**
- dng_stream & **fStream**
- dng_image & **fImage**
- dng_jpeg_image ∗ **fJPEGImage**
- dng_fingerprint ∗ **fJPEGTileDigest**
- uint32 **fOuterSamples**
- uint32 **fInnerSamples**
- uint32 **fTilesDown**
- uint32 **fTilesAcross**
- uint64 ∗ **fTileOffset**
- uint32 ∗ **fTileByteCount**
- uint32 **fCompressedSize**
- uint32 **fUncompressedSize**
- dng_mutex **fMutex**
- uint32 **fNextTileIndex**

Additional Inherited Members

5.132.1 Member Function Documentation

5.132.1.1 Process()

```
void dng_read_tiles_task::Process (
             uint32 threadIndex,
             const dng_rect & tile,
             dng_abort_sniffer * sniffer )  [virtual]
```

and progress updates.

Process one tile or fully partitioned area. This method is overridden by derived classes to implement the actual image processing. Note that the sniffer can be ignored if it is certain that a processing task will complete very quickly. This method should never be called directly but rather accessed via Process. There is no allocator parameter as all allocation should be done in Start.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the Start method.)
tile	Area to process.
sniffer	dng_abort_sniffer to use to check for user cancellation

Implements dng_area_task.

References dng_host::Allocate(), AutoPtr< T >::Get(), and AutoPtr< T >::Reset().

The documentation for this class was generated from the following files:

- dng_read_image.h
- dng_read_image.cpp

5.133 dng_rect Class Reference

Public Member Functions

- **dng_rect** (int32 tt, int32 ll, int32 bb, int32 rr)
- **dng_rect** (uint32 h, uint32 w)
- **dng_rect** (const dng_point &size)
- void **Clear** ()
- bool **operator==** (const dng_rect &rect) const
- bool **operator!=** (const dng_rect &rect) const
- bool **IsZero** () const
- bool **NotZero** () const

- bool **IsEmpty** () const
- bool **NotEmpty** () const
- uint32 **W** () const
- uint32 **H** () const
- dng_point **TL** () const
- dng_point **TR** () const
- dng_point **BL** () const
- dng_point **BR** () const
- dng_point **Size** () const
- uint32 **LongSide** () const
- uint32 **ShortSide** () const
- real64 **Diagonal** () const

Public Attributes

- int32 **t**
- int32 **l**
- int32 **b**
- int32 **r**

The documentation for this class was generated from the following files:

- dng_rect.h
- dng_rect.cpp

5.134 dng_rect_real64 Class Reference

Public Member Functions

- **dng_rect_real64** (real64 tt, real64 ll, real64 bb, real64 rr)
- **dng_rect_real64** (real64 h, real64 w)
- **dng_rect_real64** (const dng_point_real64 &size)
- **dng_rect_real64** (const dng_point_real64 &pt1, const dng_point_real64 &pt2)
- **dng_rect_real64** (const dng_rect &rect)
- void **Clear** ()
- bool **operator==** (const dng_rect_real64 &rect) const
- bool **operator!=** (const dng_rect_real64 &rect) const
- bool **IsZero** () const
- bool **NotZero** () const
- bool **IsEmpty** () const
- bool **NotEmpty** () const
- real64 **W** () const
- real64 **H** () const
- dng_point_real64 **TL** () const
- dng_point_real64 **TR** () const
- dng_point_real64 **BL** () const
- dng_point_real64 **BR** () const
- dng_point_real64 **Size** () const
- dng_rect **Round** () const
- real64 **LongSide** () const
- real64 **ShortSide** () const
- real64 **Diagonal** () const
- dng_point_real64 **Center** () const

Public Attributes

- real64 **t**
- real64 **l**
- real64 **b**
- real64 **r**

The documentation for this class was generated from the following files:

- dng_rect.h
- dng_rect.cpp

5.135 dng_ref_counted_block Class Reference

Class to provide resource acquisition is instantiation discipline for small memory allocations.

```
#include <dng_ref_counted_block.h>
```

Public Member Functions

- dng_ref_counted_block ()
- dng_ref_counted_block (uint32 size)
- ∼dng_ref_counted_block ()

  Release memory buffer using free.
- dng_ref_counted_block (const dng_ref_counted_block &data)

  Copy constructore, which takes a reference to data and does not copy the block.
- dng_ref_counted_block & operator= (const dng_ref_counted_block &data)

  Assignment operatore takes a reference to right hand side and does not copy the data.
- void Allocate (uint32 size)
- void Clear ()
- void EnsureWriteable ()

  If there is only one reference, do nothing, otherwise copy the data into a new block and return an object with that block as the data.
- uint32 LogicalSize () const
- void ∗ **Buffer** ()
- const void ∗ Buffer () const
- char ∗ Buffer_char ()
- const char ∗ Buffer_char () const
- uint8 ∗ Buffer_uint8 ()
- const uint8 ∗ Buffer_uint8 () const
- uint16 ∗ Buffer_uint16 ()
- const uint16 ∗ Buffer_uint16 () const
- int16 ∗ Buffer_int16 ()
- const int16 ∗ Buffer_int16 () const
- uint32 ∗ Buffer_uint32 ()
- const uint32 ∗ Buffer_uint32 () const
- int32 ∗ Buffer_int32 ()

- const int32 ∗ Buffer_int32 () const
- uint64 ∗ Buffer_uint64 ()
- const uint64 ∗ Buffer_uint64 () const
- int64 ∗ Buffer_int64 ()
- const int64 ∗ Buffer_int64 () const
- real32 ∗ Buffer_real32 ()
- const real32 ∗ Buffer_real32 () const
- real64 ∗ Buffer_real64 ()
- const real64 ∗ Buffer_real64 () const

5.135.1 Detailed Description

Class to provide resource acquisition is instantiation discipline for small memory allocations.

Support for a refcounted block, with optional copy-on-writeThis class does not use dng_memory_allocator for memory allocation.

5.135.2 Constructor & Destructor Documentation

5.135.2.1 dng_ref_counted_block() [1/2]

```
dng_ref_counted_block::dng_ref_counted_block ( )
```

Construct an empty memory buffer using malloc.

Exceptions

dng_memory_full	with fErrorCode equal to dng_error_memory.

5.135.2.2 dng_ref_counted_block() [2/2]

```
dng_ref_counted_block::dng_ref_counted_block (
             uint32 size )
```

Construct memory buffer of size bytes using malloc.

Parameters

size	Number of bytes of memory needed.

Exceptions

dng_memory_full	with fErrorCode equal to dng_error_memory.

References Allocate().

5.135.3 Member Function Documentation

5.135.3.1 Allocate()

```
void dng_ref_counted_block::Allocate (
             uint32 size )
```

Clear existing memory buffer and allocate new memory of size bytes.

Parameters

size	Number of bytes of memory needed.

Exceptions

dng_memory_full	with fErrorCode equal to dng_error_memory.

References Clear(), and ThrowMemoryFull().

Referenced by dng_ref_counted_block(), EnsureWriteable(), and dng_hue_sat_map::SetDivisions().

5.135.3.2 Buffer()

```
const void* dng_ref_counted_block::Buffer ( ) const  [inline]
```

Return pointer to allocated memory as a const void ∗.

Return values

const	void ∗ valid for as many bytes as were allocated.

5.135.3.3 Buffer_char() [1/2]

```
char* dng_ref_counted_block::Buffer_char ( )  [inline]
```

Return pointer to allocated memory as a char ∗.

Return values

char	∗ valid for as many bytes as were allocated.

5.135.3.4 Buffer_char() [2/2]

```
const char* dng_ref_counted_block::Buffer_char ( ) const  [inline]
```

Return pointer to allocated memory as a const char ∗.

Return values

const	char ∗ valid for as many bytes as were allocated.

5.135.3.5 Buffer_int16() [1/2]

```
int16* dng_ref_counted_block::Buffer_int16 ( )  [inline]
```

Return pointer to allocated memory as a int16 ∗.

Return values

int16	∗ valid for as many bytes as were allocated.

5.135.3.6 Buffer_int16() [2/2]

```
const int16* dng_ref_counted_block::Buffer_int16 ( ) const  [inline]
```

Return pointer to allocated memory as a const int16 ∗.

Return values

const	int16 ∗ valid for as many bytes as were allocated.

5.135.3.7 Buffer_int32() [1/2]

```
int32* dng_ref_counted_block::Buffer_int32 ( )  [inline]
```

Return pointer to allocated memory as a const int32 ∗.

Return values

const	int32 ∗ valid for as many bytes as were allocated.

5.135.3.8 Buffer_int32() [2/2]

```
const int32* dng_ref_counted_block::Buffer_int32 ( ) const  [inline]
```

Return pointer to allocated memory as a const int32 ∗.

Return values

const	int32 ∗ valid for as many bytes as were allocated.

5.135.3.9 Buffer_int64() [1/2]

```
int64* dng_ref_counted_block::Buffer_int64 ( )  [inline]
```

Return pointer to allocated memory as a const int64 ∗.

Return values

const	int64 ∗ valid for as many bytes as were allocated.

5.135.3.10 Buffer_int64() [2/2]

```
const int64* dng_ref_counted_block::Buffer_int64 ( ) const  [inline]
```

Return pointer to allocated memory as a const int64 ∗.

Return values

const	int64 ∗ valid for as many bytes as were allocated.

5.135.3.11 Buffer_real32() [1/2]

```
real32* dng_ref_counted_block::Buffer_real32 ( )  [inline]
```

Return pointer to allocated memory as a real32 ∗.

Return values

real32	∗ valid for as many bytes as were allocated.

Referenced by dng_hue_sat_map::GetConstDeltas(), and dng_hue_sat_map::GetDeltas().

5.135.3.12 Buffer_real32() [2/2]

```
const real32* dng_ref_counted_block::Buffer_real32 ( ) const  [inline]
```

Return pointer to allocated memory as a const real32 ∗.

Return values

const	real32 ∗ valid for as many bytes as were allocated.

5.135.3.13 Buffer_real64() [1/2]

```
real64* dng_ref_counted_block::Buffer_real64 ( )  [inline]
```

Return pointer to allocated memory as a real64 ∗.

Return values

real64	∗ valid for as many bytes as were allocated.

5.135.3.14 Buffer_real64() [2/2]

```
const real64* dng_ref_counted_block::Buffer_real64 ( ) const  [inline]
```

Return pointer to allocated memory as a const real64 ∗.

Return values

const	real64 ∗ valid for as many bytes as were allocated.

5.135.3.15 Buffer_uint16() [1/2]

```
uint16* dng_ref_counted_block::Buffer_uint16 ( )  [inline]
```

Return pointer to allocated memory as a uint16 ∗.

Return values

uint16	∗ valid for as many bytes as were allocated.

5.135.3.16 Buffer_uint16() [2/2]

```
const uint16* dng_ref_counted_block::Buffer_uint16 ( ) const  [inline]
```

Return pointer to allocated memory as a const uint16 ∗.

Return values

const	uint16 ∗ valid for as many bytes as were allocated.

5.135.3.17 Buffer_uint32() [1/2]

```
uint32* dng_ref_counted_block::Buffer_uint32 ( )  [inline]
```

Return pointer to allocated memory as a uint32 ∗.

Return values

uint32	∗ valid for as many bytes as were allocated.

5.135.3.18 Buffer_uint32() [2/2]

```
const uint32* dng_ref_counted_block::Buffer_uint32 ( ) const  [inline]
```

Return pointer to allocated memory as a uint32 ∗.

Return values

uint32	∗ valid for as many bytes as were allocated.

5.135.3.19 Buffer_uint64() [1/2]

```
uint64* dng_ref_counted_block::Buffer_uint64 ( )  [inline]
```

Return pointer to allocated memory as a uint64 ∗.

Return values

uint64	∗ valid for as many bytes as were allocated.

5.135.3.20 Buffer_uint64() [2/2]

```
const uint64* dng_ref_counted_block::Buffer_uint64 ( ) const  [inline]
```

Return pointer to allocated memory as a uint64 ∗.

Return values

uint64	∗ valid for as many bytes as were allocated.

5.135.3.21 Buffer_uint8() [1/2]

```
uint8* dng_ref_counted_block::Buffer_uint8 ( )  [inline]
```

Return pointer to allocated memory as a uint8 ∗.

Return values

uint8	∗ valid for as many bytes as were allocated.

5.135.3.22 Buffer_uint8() [2/2]

```
const uint8* dng_ref_counted_block::Buffer_uint8 ( ) const  [inline]
```

Return pointer to allocated memory as a const uint8 ∗.

Return values

const	uint8 ∗ valid for as many bytes as were allocated.

5.135.3.23 Clear()

```
void dng_ref_counted_block::Clear ( )
```

Release any allocated memory using free. Object is still valid and Allocate can be called again.

Referenced by Allocate(), operator=(), dng_hue_sat_map::SetInvalid(), and ∼dng_ref_counted_block().

5.135.3.24 LogicalSize()

```
uint32 dng_ref_counted_block::LogicalSize ( ) const  [inline]
```

Return pointer to allocated memory as a void ∗..

Return values

void	∗ valid for as many bytes as were allocated.

The documentation for this class was generated from the following files:

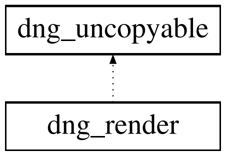
- dng_ref_counted_block.h
- dng_ref_counted_block.cpp

5.136 dng_render Class Reference

Class used to render digital negative to displayable image.

```
#include <dng_render.h>
```

Inheritance diagram for dng_render:



Public Member Functions

- dng_render (dng_host &host, const dng_negative &negative)
- void SetWhiteXY (const dng_xy_coord &white)
- const dng_xy_coord WhiteXY () const
- void SetExposure (real64 exposure)
- real64 Exposure () const
- void SetShadows (real64 shadows)
- real64 Shadows () const
- void SetToneCurve (const dng_1d_function &curve)
- const dng_1d_function & ToneCurve () const
- void SetFinalSpace (const dng_color_space &space)
- const dng_color_space & FinalSpace () const
- void SetFinalPixelType (uint32 type)
- uint32 FinalPixelType () const
- void SetMaximumSize (uint32 size)
- uint32 MaximumSize () const
- virtual dng_image ∗ Render ()

Protected Attributes

- dng_host & **fHost**
- const dng_negative & **fNegative**
- dng_xy_coord **fWhiteXY**
- real64 **fExposure**
- real64 **fShadows**
- const dng_1d_function ∗ **fToneCurve**
- const dng_color_space ∗ **fFinalSpace**
- uint32 **fFinalPixelType**
- uint32 **fMaximumSize**

5.136.1 Detailed Description

Class used to render digital negative to displayable image.

5.136.2 Constructor & Destructor Documentation

5.136.2.1 dng_render()

```
dng_render::dng_render (
             dng_host & host,
             const dng_negative & negative )
```

Construct a rendering instance that will be used to convert a given digital negative.

Parameters

host	The host to use for memory allocation, progress updates, and abort testing.
negative	The digital negative to convert to a displayable image.

References dng_camera_profile::DefaultBlackRender(), AutoPtr< T >::Get(), dng_1d_identity::Get(), AutoPtr< T >↩
::Reset(), and dng_camera_profile::ToneCurve().

5.136.3 Member Function Documentation

5.136.3.1 Exposure()

```
real64 dng_render::Exposure ( ) const  [inline]
```

Get exposure compensation.

Return values

Compensation	value in stops, positive or negative.

5.136.3.2 FinalPixelType()

```
uint32 dng_render::FinalPixelType ( ) const  [inline]
```

Get pixel type of final image data. Can be ttByte (default), ttShort, or ttFloat.

Return values

Pixel	type to use.

Referenced by Render().

5.136.3.3 FinalSpace()

```
const dng_color_space& dng_render::FinalSpace ( ) const  [inline]
```

Get final color space in which resulting image data should be represented.

Return values

Color	space to use.

Referenced by Render().

5.136.3.4 MaximumSize()

```
uint32 dng_render::MaximumSize ( ) const  [inline]
```

Get maximum dimension, in pixels, of resulting image. If the final image would have either dimension larger than this maximum, the larger of the two dimensions is set to this maximum size and the smaller dimension is adjusted to preserve the image's aspect ratio.

Return values

Maximum	allowed size.

Referenced by Render().

5.136.3.5 Render()

```
dng_image * dng_render::Render ( )  [virtual]
```

Actually render a digital negative to a displayable image. Input digital negative is passed to the constructor of this dng_render class.

Return values

The	final resulting image.

References dng_negative::AspectRatio(), dng_image::Bounds(), dng_negative::DefaultCropArea(), dng_negative::↩ DefaultFinalHeight(), dng_negative::DefaultFinalWidth(), FinalPixelType(), FinalSpace(), AutoPtr< T >::Get(), dng_↩ color_space::IsMonochrome(), dng_host::Make_dng_image(), MaximumSize(), dng_host::PerformAreaTask(), dng_↩ image::PixelType(), dng_image::Planes(), AutoPtr< T >::Release(), and AutoPtr< T >::Reset().

5.136.3.6 SetExposure()

```
void dng_render::SetExposure (
             real64 exposure )  [inline]
```

Set exposure compensation.

Parameters

exposure	Compensation value in stops, positive or negative.

5.136.3.7 SetFinalPixelType()

```
void dng_render::SetFinalPixelType (
             uint32 type )  [inline]
```

Set pixel type of final image data. Can be ttByte (default), ttShort, or ttFloat.

Parameters

type	Pixel type to use.

5.136.3.8 SetFinalSpace()

```
void dng_render::SetFinalSpace (
            const dng_color_space & space )  [inline]
```

Set final color space in which resulting image data should be represented. (See dng_color_space.h for possible values.)

Parameters

space	Color space to use.

5.136.3.9 SetMaximumSize()

```
void dng_render::SetMaximumSize (
            uint32 size )  [inline]
```

Set maximum dimension, in pixels, of resulting image. If final image would have either dimension larger than maximum, the larger of the two dimensions is set to this maximum size and the smaller dimension is adjusted to preserve aspect ratio.

Parameters

size	Maximum size to allow.

5.136.3.10 SetShadows()

```
void dng_render::SetShadows (
            real64 shadows )  [inline]
```

Set shadow clip amount.

Parameters

shadows	Shadow clip amount.

5.136.3.11 SetToneCurve()

```
void dng_render::SetToneCurve (
            const dng_1d_function & curve )  [inline]
```

Set custom tone curve for conversion.

Parameters

curve	1D function that defines tone mapping to use during conversion.

5.136.3.12 SetWhiteXY()

```
void dng_render::SetWhiteXY (
             const dng_xy_coord & white )  [inline]
```

Set the white point to be used for conversion.

Parameters

white	White point to use.

5.136.3.13 Shadows()

```
real64 dng_render::Shadows ( ) const  [inline]
```

Get shadow clip amount.

Return values

Shadow	clip amount.

5.136.3.14 ToneCurve()

```
const dng_1d_function& dng_render::ToneCurve ( ) const  [inline]
```

Get custom tone curve for conversion.

Return values

1D	function that defines tone mapping to use during conversion.

5.136.3.15 WhiteXY()

```
const dng_xy_coord dng_render::WhiteXY ( ) const  [inline]
```

Get the white point to be used for conversion.

Return values

White	point to use.

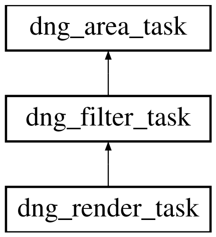
Referenced by dng_render_task::Start().

The documentation for this class was generated from the following files:

- dng_render.h
- dng_render.cpp

5.137 dng_render_task Class Reference

Inheritance diagram for dng_render_task:



Public Member Functions

- **dng_render_task** (const dng_image &srcImage, const dng_image ∗srcMask, dng_image &dstImage, const dng_negative &negative, const dng_render &params, const dng_point &srcOffset)
- virtual dng_rect SrcArea (const dng_rect &dstArea)
- virtual void Start (uint32 threadCount, const dng_rect &dstArea, const dng_point &tileSize, dng_memory_allocator ∗allocator, dng_abort_sniffer ∗sniffer)
- virtual void ProcessArea (uint32 threadIndex, dng_pixel_buffer &srcBuffer, dng_pixel_buffer &dstBuffer)

Protected Attributes

- const dng_image ∗ **fSrcMask**
- const dng_negative & **fNegative**
- const dng_render & **fParams**
- dng_point **fSrcOffset**
- dng_1d_table **fZeroOffsetRamp**
- dng_vector **fCameraWhite**
- dng_matrix **fCameraToRGB**
- AutoPtr< dng_hue_sat_map > **fHueSatMap**
- dng_1d_table **fExposureRamp**
- AutoPtr< dng_hue_sat_map > **fLookTable**
- dng_1d_table **fToneCurve**
- dng_matrix **fRGBtoFinal**
- dng_1d_table **fEncodeGamma**
- AutoPtr< dng_1d_table > **fHueSatMapEncode**
- AutoPtr< dng_1d_table > **fHueSatMapDecode**
- AutoPtr< dng_1d_table > **fLookTableEncode**
- AutoPtr< dng_1d_table > **fLookTableDecode**
- AutoPtr< dng_memory_block > **fTempBuffer** [kMaxMPThreads]
- AutoPtr< dng_memory_block > **fMaskBuffer** [kMaxMPThreads]

Additional Inherited Members

5.137.1 Member Function Documentation

5.137.1.1 ProcessArea()

```
void dng_render_task::ProcessArea (
             uint32 threadIndex,
             dng_pixel_buffer & srcBuffer,
             dng_pixel_buffer & dstBuffer )  [virtual]
```

Implements filtering operation from one buffer to another. Source and destination pixels are set up in member fields of this class. Ideally, no allocation should be done in this routine.

Parameters

threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Start method.
srcBuffer	Input area and source pixels.
dstBuffer	Output area and destination pixels.

Implements dng_filter_task.

References dng_memory_block::Buffer_real32(), dng_pixel_buffer::DirtyPixel(), and dng_image::Get().

5.137.1.2 SrcArea()

```
dng_rect dng_render_task::SrcArea (
             const dng_rect & dstArea )  [virtual]
```

Compute the source area needed for a given destination area. Default implementation assumes destination area is equal to source area for all cases.

Parameters

dstArea	Area to for which pixels will be computed.

Return values

The	source area needed as input to calculate the requested destination area.

Reimplemented from dng_filter_task.

5.137.1.3 Start()

```
void dng_render_task::Start (
            uint32 threadCount,
            const dng_rect & dstArea,
            const dng_point & tileSize,
            dng_memory_allocator * allocator,
            dng_abort_sniffer * sniffer )  [virtual]
```

Called prior to processing on specific threads. Can be used to allocate per-thread memory buffers, etc.

Parameters

threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads of dng_area_task.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.
sniffer	Sniffer to test for user cancellation and to set up progress.

Reimplemented from dng_filter_task.

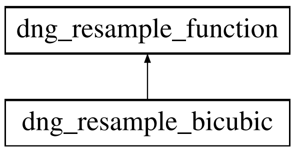
References dng_negative::CameraNeutral(), dng_negative::HasCameraNeutral(), dng_1d_table::Initialize(), dng_↩ negative::IsMonochrome(), dng_filter_task::Start(), and dng_render::WhiteXY().

The documentation for this class was generated from the following file:

- dng_render.cpp

5.138 dng_resample_bicubic Class Reference

Inheritance diagram for dng_resample_bicubic:



Public Member Functions

- virtual real64 **Extent** () const
- virtual real64 **Evaluate** (real64 x) const

Static Public Member Functions

- static const dng_resample_function & **Get** ()

The documentation for this class was generated from the following files:

- dng_resample.h
- dng_resample.cpp

5.139 dng_resample_coords Class Reference

Public Member Functions

- void **Initialize** (int32 srcOrigin, int32 dstOrigin, uint32 srcCount, uint32 dstCount, dng_memory_allocator &allocator)
- const int32 $*$ **Coords** (int32 index) const
- int32 **Pixel** (int32 index) const

Protected Attributes

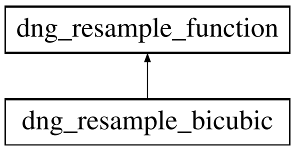
- int32 **fOrigin**
- AutoPtr< dng_memory_block > **fCoords**

The documentation for this class was generated from the following files:

- dng_resample.h
- dng_resample.cpp

5.140 dng_resample_function Class Reference

Inheritance diagram for dng_resample_function:



Public Member Functions

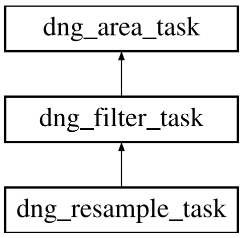
- virtual real64 **Extent** () const =0
- virtual real64 **Evaluate** (real64 x) const =0

The documentation for this class was generated from the following file:

- dng_resample.h

5.141 dng_resample_task Class Reference

Inheritance diagram for dng_resample_task:



Public Member Functions

- **dng_resample_task** (const dng_image &srcImage, dng_image &dstImage, const dng_rect &srcBounds, const dng_rect &dstBounds, const dng_resample_function &kernel)
- virtual dng_rect SrcArea (const dng_rect &dstArea)
- virtual dng_point SrcTileSize (const dng_point &dstTileSize)
- virtual void Start (uint32 threadCount, const dng_rect &dstArea, const dng_point &tileSize, dng_memory_allocator ∗allocator, dng_abort_sniffer ∗sniffer)
- virtual void ProcessArea (uint32 threadIndex, dng_pixel_buffer &srcBuffer, dng_pixel_buffer &dstBuffer)

Protected Attributes

- dng_rect **fSrcBounds**
- dng_rect **fDstBounds**
- const dng_resample_function & **fKernel**
- real64 **fRowScale**
- real64 **fColScale**
- dng_resample_coords **fRowCoords**
- dng_resample_coords **fColCoords**
- dng_resample_weights **fWeightsV**
- dng_resample_weights **fWeightsH**
- dng_point **fSrcTileSize**
- AutoPtr< dng_memory_block > **fTempBuffer** [kMaxMPThreads]

Additional Inherited Members

5.141.1 Member Function Documentation

5.141.1.1 ProcessArea()

```
void dng_resample_task::ProcessArea (
            uint32 threadIndex,
            dng_pixel_buffer & srcBuffer,
            dng_pixel_buffer & dstBuffer )  [virtual]
```

Implements filtering operation from one buffer to another. Source and destination pixels are set up in member fields of this class. Ideally, no allocation should be done in this routine.

Parameters

threadIndex	The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Start method.
srcBuffer	Input area and source pixels.
dstBuffer	Output area and destination pixels.

Implements dng_filter_task.

References dng_memory_block::Buffer_real32(), and dng_pixel_buffer::ConstPixel_real32().

5.141.1.2 SrcArea()

```
dng_rect dng_resample_task::SrcArea (
             const dng_rect & dstArea )  [virtual]
```

Compute the source area needed for a given destination area. Default implementation assumes destination area is equal to source area for all cases.

Parameters

dstArea	Area to for which pixels will be computed.

Return values

The	source area needed as input to calculate the requested destination area.

Reimplemented from dng_filter_task.

5.141.1.3 SrcTileSize()

```
dng_point dng_resample_task::SrcTileSize (
             const dng_point & dstTileSize )  [virtual]
```

Given a destination tile size, calculate input tile size. Simlar to SrcArea, and should seldom be overridden.

Parameters

dstTileSize	The destination tile size that is targeted for output.

Return values

The	source tile size needed to compute a tile of the destination size.

Reimplemented from dng_filter_task.

5.141.1.4 Start()

```
void dng_resample_task::Start (
            uint32 threadCount,
            const dng_rect & dstArea,
            const dng_point & tileSize,
            dng_memory_allocator * allocator,
            dng_abort_sniffer * sniffer )  [virtual]
```

Called prior to processing on specific threads. Can be used to allocate per-thread memory buffers, etc.

Parameters

threadCount	Total number of threads that will be used for processing. Less than or equal to MaxThreads of dng_area_task.
tileSize	Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)
allocator	dng_memory_allocator to use for allocating temporary buffers, etc.
sniffer	Sniffer to test for user cancellation and to set up progress.

Reimplemented from dng_filter_task.

References dng_memory_allocator::Allocate(), dng_filter_task::Start(), and ThrowOverflow().

The documentation for this class was generated from the following file:

- dng_resample.cpp

5.142 dng_resample_weights Class Reference

Public Member Functions

- void **Initialize** (real64 scale, const dng_resample_function &kernel, dng_memory_allocator &allocator)
- uint32 **Radius** () const
- uint32 **Width** () const
- int32 **Offset** () const
- uint32 **Step** () const
- const real32 ∗ **Weights32** (uint32 fract) const
- const int16 ∗ **Weights16** (uint32 fract) const

Protected Attributes

- uint32 **fRadius**
- uint32 **fWeightStep**
- AutoPtr< dng_memory_block > **fWeights32**
- AutoPtr< dng_memory_block > **fWeights16**

The documentation for this class was generated from the following files:

- dng_resample.h
- dng_resample.cpp

5.143 dng_resample_weights_2d Class Reference

Public Member Functions

- void **Initialize** (const dng_resample_function &kernel, dng_memory_allocator &allocator)
- uint32 **Radius** () const
- uint32 **Width** () const
- int32 **Offset** () const
- uint32 **RowStep** () const
- uint32 **ColStep** () const
- const real32 ∗ **Weights32** (dng_point fract) const
- const int16 ∗ **Weights16** (dng_point fract) const

Protected Attributes

- uint32 **fRadius**
- uint32 **fRowStep**
- uint32 **fColStep**
- AutoPtr< dng_memory_block > **fWeights32**
- AutoPtr< dng_memory_block > **fWeights16**

The documentation for this class was generated from the following files:

- dng_resample.h
- dng_resample.cpp

5.144 dng_resolution Class Reference

Image resolution.

```
#include <dng_image_writer.h>
```

Public Attributes

- dng_urational **fXResolution**
- dng_urational **fYResolution**
- uint16 **fResolutionUnit**

5.144.1 Detailed Description

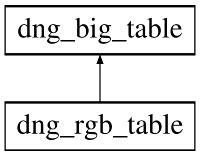
Image resolution.

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.145 dng_rgb_table Class Reference

Inheritance diagram for dng_rgb_table:



Public Types

- enum {
  kMinDivisions1D = 2, **kMaxDivisions1D** = 4096, **kMinDivisions3D** = 2, **kMaxDivisions3D** = 32,
  kMaxDivisions3D_InMemory = 128 }
- enum **primaries_enum** {
  primaries_sRGB = 0, **primaries_Adobe**, **primaries_ProPhoto**, **primaries_P3**,
  primaries_Rec2020, **primaries_count** }
- enum **gamma_enum** {
  gamma_Linear = 0, **gamma_sRGB**, **gamma_1_8**, **gamma_2_2**,
  gamma_Rec2020, **gamma_count** }
- enum **gamut_enum** { **gamut_clip** = 0, **gamut_extend**, **gamut_count** }

Public Member Functions

- **dng_rgb_table** (const dng_rgb_table &table)
- dng_rgb_table & **operator=** (const dng_rgb_table &table)
- bool **operator==** (const dng_rgb_table &table) const
- bool **operator!=** (const dng_rgb_table &table) const
- virtual bool **IsValid** () const
- void **SetInvalid** ()
- primaries_enum **Primaries** () const
- void **SetPrimaries** (primaries_enum primaries)
- gamma_enum **Gamma** () const
- void **SetGamma** (gamma_enum gamma)
- gamut_enum **Gamut** () const
- void **SetGamut** (gamut_enum gamut)
- real64 **MinAmount** () const
- real64 **MaxAmount** () const
- void **SetAmountRange** (real64 minAmount, real64 maxAmount)
- real64 **Amount** () const
- void **SetAmount** (real64 amount)
- uint32 **Dimensions** () const
- uint32 **Divisions** () const
- const uint16 ∗ **Samples** () const
- bool **Monochrome** () const
- void **Set** (uint32 dimensions, uint32 divisions, dng_ref_counted_block samples)

Protected Member Functions

- virtual void **GetStream** (dng_stream &stream)
- virtual void **PutStream** (dng_stream &stream, bool forFingerprint) const

Friends

- class **dng_rgb_table_cache**

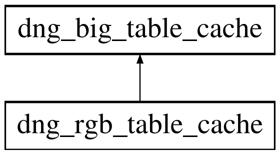
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_big_table.h
- dng_big_table.cpp

5.146 dng_rgb_table_cache Class Reference

Inheritance diagram for dng_rgb_table_cache:



Public Member Functions

- virtual void **InsertTableData** (dng_lock_std_mutex &, const dng_big_table &table)
- virtual void **EraseTableData** (dng_lock_std_mutex &, const dng_fingerprint &fingerprint)
- virtual void **ExtractTableData** (dng_lock_std_mutex &, const dng_fingerprint &fingerprint, dng_big_table &table)

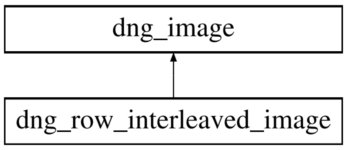
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_big_table.cpp

5.147 dng_row_interleaved_image Class Reference

Inheritance diagram for dng_row_interleaved_image:



Public Member Functions

- **dng_row_interleaved_image** (dng_image &image, uint32 factor)
- virtual void **DoGet** (dng_pixel_buffer &buffer) const
- virtual void **DoPut** (const dng_pixel_buffer &buffer)

Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_read_image.h
- dng_read_image.cpp

5.148 dng_safe_int32 Class Reference

Public Member Functions

- template<typename T >
  dng_safe_int32 (T x)
- **dng_safe_int32** (const dng_safe_uint32 &x)
- int32 **Get** () const
- void **Set_uint32** (uint32 x)
- dng_safe_int32 & **operator+=** (const dng_safe_int32 &x)
- template<typename T >
  dng_safe_int32 & **operator+=** (T x)
- dng_safe_int32 & **operator-=** (const dng_safe_int32 &x)
- template<typename T >
  dng_safe_int32 & **operator-=** (T x)

The documentation for this class was generated from the following files:

- dng_safe_arithmetic.h
- dng_safe_arithmetic.cpp

5.149 dng_safe_uint32 Class Reference

Public Member Functions

- template<typename T >
  dng_safe_uint32 (T x)
- **dng_safe_uint32** (const dng_safe_int32 &x)
- uint32 **Get** () const
- dng_safe_uint32 & **operator+=** (const dng_safe_uint32 &x)
- template<typename T >
  dng_safe_uint32 & **operator+=** (T x)
- dng_safe_uint32 & **operator ∗=** (const dng_safe_uint32 &x)
- template<typename T >
  dng_safe_uint32 & **operator ∗=** (T x)
- const dng_safe_uint32 **operator+** (const dng_safe_uint32 &x) const
- template<typename T >
  const dng_safe_uint32 **operator+** (T x) const
- const dng_safe_uint32 **operator** ∗ (const dng_safe_uint32 &x) const
- template<typename T >
  const dng_safe_uint32 **operator** ∗ (T x) const

The documentation for this class was generated from the following files:

- dng_safe_arithmetic.h
- dng_safe_arithmetic.cpp

5.150 dng_set_minimum_priority Class Reference

Convenience class for setting thread priority level to minimum.

```
#include <dng_abort_sniffer.h>
```

Public Member Functions

- **dng_set_minimum_priority** (dng_priority priority, const char ∗name)

5.150.1 Detailed Description

Convenience class for setting thread priority level to minimum.

The documentation for this class was generated from the following files:

- dng_abort_sniffer.h
- dng_abort_sniffer.cpp

5.151 dng_shared Class Reference

Public Member Functions

- virtual bool **ParseTag** (dng_stream &stream, dng_exif &exif, uint32 parentCode, bool isMainIFD, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset, int64 offsetDelta)
- virtual void **PostParse** (dng_host &host, dng_exif &exif)
- virtual bool **IsValidDNG** ()

Public Attributes

- uint64 **fExifIFD**
- uint64 **fGPSInfo**
- uint64 **fInteroperabilityIFD**
- uint64 **fKodakDCRPrivateIFD**
- uint64 **fKodakKDCPrivateIFD**
- uint32 **fXMPCount**
- uint64 **fXMPOffset**
- uint32 **fIPTC_NAA_Count**
- uint64 **fIPTC_NAA_Offset**
- uint32 **fMakerNoteCount**
- uint64 **fMakerNoteOffset**
- uint32 **fMakerNoteSafety**
- uint32 **fDNGVersion**
- uint32 **fDNGBackwardVersion**
- dng_string **fUniqueCameraModel**
- dng_string **fLocalizedCameraModel**

- dng_camera_profile_info **fCameraProfile**
- dng_std_vector< dng_camera_profile_info > **fExtraCameraProfiles**
- dng_matrix **fCameraCalibration1**
- dng_matrix **fCameraCalibration2**
- dng_string **fCameraCalibrationSignature**
- dng_vector **fAnalogBalance**
- dng_vector **fAsShotNeutral**
- dng_xy_coord **fAsShotWhiteXY**
- dng_srational **fBaselineExposure**
- dng_urational **fBaselineNoise**
- dng_urational **fBaselineSharpness**
- dng_urational **fLinearResponseLimit**
- dng_urational **fShadowScale**
- bool **fHasBaselineExposure**
- bool **fHasShadowScale**
- uint32 **fDNGPrivateDataCount**
- uint64 **fDNGPrivateDataOffset**
- dng_fingerprint **fRawImageDigest**
- dng_fingerprint **fNewRawImageDigest**
- dng_fingerprint **fRawDataUniqueID**
- dng_string **fOriginalRawFileName**
- uint32 **fOriginalRawFileDataCount**
- uint64 **fOriginalRawFileDataOffset**
- dng_fingerprint **fOriginalRawFileDigest**
- uint32 **fAsShotICCProfileCount**
- uint64 **fAsShotICCProfileOffset**
- dng_matrix **fAsShotPreProfileMatrix**
- uint32 **fCurrentICCProfileCount**
- uint64 **fCurrentICCProfileOffset**
- dng_matrix **fCurrentPreProfileMatrix**
- uint32 **fColorimetricReference**
- dng_string **fAsShotProfileName**
- dng_point **fOriginalDefaultFinalSize**
- dng_point **fOriginalBestQualityFinalSize**
- dng_urational **fOriginalDefaultCropSizeH**
- dng_urational **fOriginalDefaultCropSizeV**
- uint32 **fDepthFormat**
- dng_urational **fDepthNear**
- dng_urational **fDepthFar**
- uint32 **fDepthUnits**
- uint32 **fDepthMeasureType**

Protected Member Functions

- virtual bool **Parse_ifd0** (dng_stream &stream, dng_exif &exif, uint32 parentCode, uint32 tagCode, uint32 tag←Type, uint32 tagCount, uint64 tagOffset)
- virtual bool **Parse_ifd0_exif** (dng_stream &stream, dng_exif &exif, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)

The documentation for this class was generated from the following files:

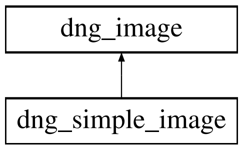
- dng_shared.h
- dng_shared.cpp

5.152 dng_simple_image Class Reference

dng_image derived class with simple Trim and Rotate functionality.

```
#include <dng_simple_image.h>
```

Inheritance diagram for dng_simple_image:



Public Member Functions

- **dng_simple_image** (const dng_rect &bounds, uint32 planes, uint32 pixelType, dng_memory_allocator &allocator)
- virtual dng_image ∗ **Clone** () const
- virtual void SetPixelType (uint32 pixelType)

  Setter for pixel type.
- virtual void Trim (const dng_rect &r)

  Trim image data outside of given bounds. Memory is not reallocated or freed.
- virtual void Rotate (const dng_orientation &orientation)

  Rotate image according to orientation.
- void GetPixelBuffer (dng_pixel_buffer &buffer)

  Get the buffer for direct processing. (Unique to dng_simple_image.)

Protected Member Functions

- virtual void **AcquireTileBuffer** (dng_tile_buffer &buffer, const dng_rect &area, bool dirty) const

Protected Attributes

- dng_pixel_buffer **fBuffer**
- AutoPtr< dng_memory_block > **fMemory**
- dng_memory_allocator & **fAllocator**

Additional Inherited Members

5.152.1 Detailed Description

dng_image derived class with simple Trim and Rotate functionality.

The documentation for this class was generated from the following files:

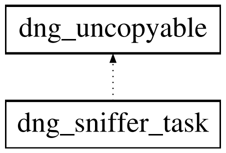
- dng_simple_image.h
- dng_simple_image.cpp

5.153 dng_sniffer_task Class Reference

Class to establish scope of a named subtask in DNG processing.

```
#include <dng_abort_sniffer.h>
```

Inheritance diagram for dng_sniffer_task:



Public Member Functions

- dng_sniffer_task (dng_abort_sniffer ∗sniffer, const char ∗name=NULL, real64 fract=0.0)
- void Sniff ()
- void UpdateProgress (real64 fract)
- void UpdateProgress (uint32 done, uint32 total)
- void Finish ()

  Signal task completed for progress purposes.

5.153.1 Detailed Description

Class to establish scope of a named subtask in DNG processing.

Instances of this class are intended to be stack allocated.

5.153.2 Constructor & Destructor Documentation

5.153.2.1 dng_sniffer_task()

```
dng_sniffer_task::dng_sniffer_task (
            dng_abort_sniffer ∗ sniffer,
            const char ∗ name = NULL,
            real64 fract = 0.0 )  [inline]
```

Inform a sniffer of a subtask in DNG processing.

Parameters

sniffer	The sniffer associated with the host on which this processing is occurring.
name	The name of this subtask as a NUL terminated string.
fract	Percentage of total processing this task is expected to take, from 0.0 to 1.0 .

References dng_abort_sniffer::StartTask().

5.153.3 Member Function Documentation

5.153.3.1 Sniff()

```
void dng_sniffer_task::Sniff ( )  [inline]
```

Check for pending user cancellation or other abort. ThrowUserCanceled will be called if one is pending.

References dng_abort_sniffer::SniffForAbort().

5.153.3.2 UpdateProgress() [1/2]

```
void dng_sniffer_task::UpdateProgress (
             real64 fract )  [inline]
```

Update progress on this subtask.

Parameters

fract	Percentage of processing completed on current task, from 0.0 to 1.0 .

References dng_abort_sniffer::UpdateProgress().

Referenced by Finish(), and UpdateProgress().

5.153.3.3 UpdateProgress() [2/2]

```
void dng_sniffer_task::UpdateProgress (
             uint32 done,
             uint32 total )  [inline]
```

Update progress on this subtask.

Parameters

done	Amount of task completed in arbitrary integer units.
total	Total size of task in same arbitrary integer units as done.

References UpdateProgress().

The documentation for this class was generated from the following file:

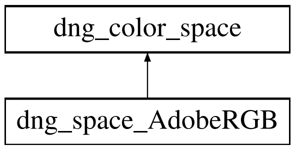
- dng_abort_sniffer.h

5.154 dng_space_AdobeRGB Class Reference

Singleton class for AdobeRGB color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for dng_space_AdobeRGB:



Public Member Functions

- virtual const dng_1d_function & GammaFunction () const

  Returns dng_function_GammaEncode_2_2.
- virtual bool ICCProfile (uint32 &size, const uint8 ∗&data) const

  Returns AdobeRGB (1998) ICC profile.

Static Public Member Functions

- static const dng_color_space & Get ()

  Static method for getting single global instance of this color space.

Additional Inherited Members

5.154.1 Detailed Description

Singleton class for AdobeRGB color space.

The documentation for this class was generated from the following files:

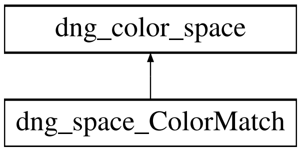
- dng_color_space.h
- dng_color_space.cpp

5.155 dng_space_ColorMatch Class Reference

Singleton class for ColorMatch color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for dng_space_ColorMatch:



Public Member Functions

- virtual const dng_1d_function & GammaFunction () const

  Returns dng_function_GammaEncode_1_8.
- virtual bool ICCProfile (uint32 &size, const uint8 ∗&data) const

  Returns ColorMatch RGB ICC profile.

Static Public Member Functions

- static const dng_color_space & Get ()

  Static method for getting single global instance of this color space.

Additional Inherited Members

5.155.1 Detailed Description

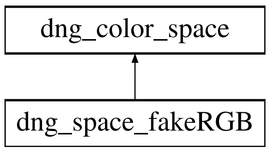
Singleton class for ColorMatch color space.

The documentation for this class was generated from the following files:

- dng_color_space.h
- dng_color_space.cpp

5.156 dng_space_fakeRGB Class Reference

Inheritance diagram for dng_space_fakeRGB:



Static Public Member Functions

- static const dng_color_space & **Get** ()

Additional Inherited Members

The documentation for this class was generated from the following files:

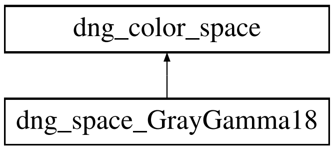
- dng_color_space.h
- dng_color_space.cpp

5.157 dng_space_GrayGamma18 Class Reference

Singleton class for gamma 1.8 grayscale color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for dng_space_GrayGamma18:



Public Member Functions

- virtual const dng_1d_function & GammaFunction () const

    Returns dng_function_GammaEncode_1_8.
- virtual bool ICCProfile (uint32 &size, const uint8 ∗&data) const

    Returns simple grayscale gamma 1.8 ICC profile.

Static Public Member Functions

- static const dng_color_space & Get ()

    Static method for getting single global instance of this color space.

Additional Inherited Members

5.157.1 Detailed Description

Singleton class for gamma 1.8 grayscale color space.

The documentation for this class was generated from the following files:

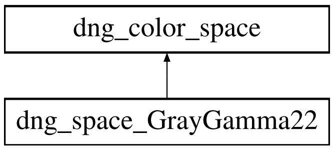
- dng_color_space.h
- dng_color_space.cpp

5.158 dng_space_GrayGamma22 Class Reference

Singleton class for gamma 2.2 grayscale color space.

`#include <dng_color_space.h>`

Inheritance diagram for dng_space_GrayGamma22:



Public Member Functions

- virtual const dng_1d_function & GammaFunction () const

    Returns dng_function_GammaEncode_2_2.
- virtual bool ICCProfile (uint32 &size, const uint8 ∗&data) const

    Returns simple grayscale gamma 2.2 ICC profile.

Static Public Member Functions

- static const dng_color_space & Get ()

    Static method for getting single global instance of this color space.

Additional Inherited Members

5.158.1 Detailed Description

Singleton class for gamma 2.2 grayscale color space.

The documentation for this class was generated from the following files:

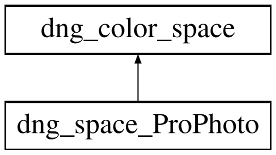
- dng_color_space.h
- dng_color_space.cpp

5.159 dng_space_ProPhoto Class Reference

Singleton class for ProPhoto RGB color space.

`#include <dng_color_space.h>`

Inheritance diagram for dng_space_ProPhoto:



Public Member Functions

- virtual const dng_1d_function & GammaFunction () const

  Returns dng_function_GammaEncode_1_8.
- virtual bool ICCProfile (uint32 &size, const uint8 ∗&data) const

  Returns ProPhoto RGB ICC profile.

Static Public Member Functions

- static const dng_color_space & Get ()

  Static method for getting single global instance of this color space.

Additional Inherited Members

5.159.1 Detailed Description

Singleton class for ProPhoto RGB color space.

The documentation for this class was generated from the following files:

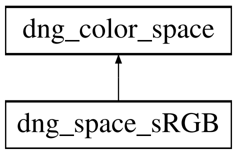
- dng_color_space.h
- dng_color_space.cpp

5.160 dng_space_sRGB Class Reference

Singleton class for sRGB color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for dng_space_sRGB:



Public Member Functions

- virtual const dng_1d_function & GammaFunction () const

  Returns dng_function_GammaEncode_sRGB.
- virtual bool ICCProfile (uint32 &size, const uint8 ∗&data) const

  Returns sRGB IEC61966-2.1 ICC profile.

Static Public Member Functions

- static const dng_color_space & Get ()

    Static method for getting single global instance of this color space.

Additional Inherited Members

5.160.1 Detailed Description

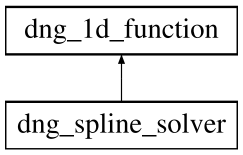
Singleton class for sRGB color space.

The documentation for this class was generated from the following files:

- dng_color_space.h
- dng_color_space.cpp

5.161 dng_spline_solver Class Reference

Inheritance diagram for dng_spline_solver:



Public Member Functions

- void **Reset** ()
- void **Add** (real64 x, real64 y)
- virtual void **Solve** ()
- virtual bool IsIdentity () const

    Returns true if this function is the map x -> y such that x == y for all x . That is if Evaluate(x) == x for all x.
- virtual real64 Evaluate (real64 x) const

Protected Attributes

- dng_std_vector< real64 > **X**
- dng_std_vector< real64 > **Y**
- dng_std_vector< real64 > **S**

5.161.1 Member Function Documentation

5.161.1.1 Evaluate()

```
real64 dng_spline_solver::Evaluate (
            real64 x ) const  [virtual]
```

Return the mapping for value x. This method must be implemented by a derived class of dng_1d_function and the derived class determines the lookup method and function used.

Parameters

x	A value between 0.0 and 1.0 (inclusive).

Return values

Mapped	value for x

Implements dng_1d_function.

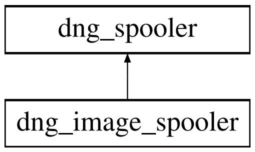
References DNG_ASSERT.

The documentation for this class was generated from the following files:

- dng_spline.h
- dng_spline.cpp

5.162 dng_spooler Class Reference

Inheritance diagram for dng_spooler:



Public Member Functions

- virtual void **Spool** (const void ∗data, uint32 count)=0

The documentation for this class was generated from the following file:

- dng_lossless_jpeg.h

5.163 dng_srational Class Reference

Public Member Functions

- **dng_srational** (int32 nn, int32 dd)
- void **Clear** ()
- bool **IsValid** () const
- bool **NotValid** () const
- bool **operator==** (const dng_srational &r) const
- bool **operator!=** (const dng_srational &r) const
- real64 **As_real64** () const
- void **Set_real64** (real64 x, int32 dd=0)
- void **ReduceByFactor** (int32 factor)

Public Attributes

- int32 **n**
- int32 **d**

The documentation for this class was generated from the following files:

- dng_rational.h
- dng_rational.cpp

5.164 dng_std_allocator< T > Class Template Reference

C++ allocator (i.e. an implementation of the Allocator concept) that throws a dng_exception with error code dng_error←_memory if it cannot allocate memory.

```
#include <dng_memory.h>
```

Public Types

- typedef T **value_type**

Public Member Functions

- T * **allocate** (size_t n)
- void **deallocate** (T *ptr, size_t)

5.164.1 Detailed Description

template<typename T>
class dng_std_allocator< T >

C++ allocator (i.e. an implementation of the Allocator concept) that throws a dng_exception with error code dng_error←_memory if it cannot allocate memory.

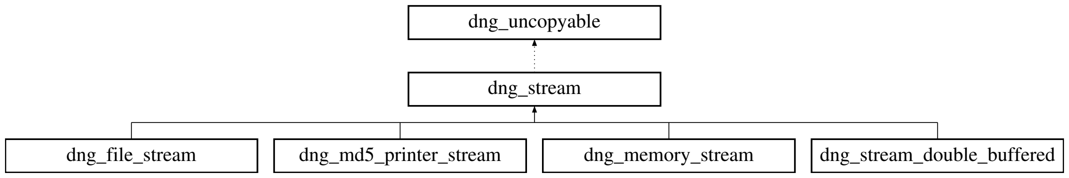
The documentation for this class was generated from the following file:

- dng_memory.h

5.165 dng_stream Class Reference

```
#include <dng_stream.h>
```

Inheritance diagram for dng_stream:



Public Types

- enum { **kSmallBufferSize** = 8 ∗ 1024, **kBigBufferSize** = 64 ∗ 1024, **kDefaultBufferSize** = kSmallBufferSize }

Public Member Functions

- dng_stream (const void ∗data, uint32 count, uint64 offsetInOriginalFile=kDNGStreamInvalidOffset)
- bool SwapBytes () const
- void SetSwapBytes (bool swapBytes)
- bool BigEndian () const
- void SetBigEndian (bool bigEndian=true)
- bool LittleEndian () const
- void SetLittleEndian (bool littleEndian=true)
- uint32 BufferSize () const

  Returns the size of the buffer used by the stream.
- void SetBufferSize (dng_memory_allocator &allocator, uint32 newBufferSize)

  Change the buffer size on the stream, if possible.
- uint64 Length ()
- uint64 Position () const
- uint64 PositionInOriginalFile () const
- uint64 OffsetInOriginalFile () const
- const void ∗ Data () const
- dng_memory_block ∗ AsMemoryBlock (dng_memory_allocator &allocator)
- void SetReadPosition (uint64 offset)

  Seek to a new position in stream for reading.
- void Skip (uint64 delta)
- bool DataInBuffer (uint32 count, uint64 offset)

  Quick check to see if data range in completely buffered.
- void Get (void ∗data, uint32 count, uint32 maxOverRead=0)
- void SetWritePosition (uint64 offset)

  Seek to a new position in stream for writing.
- void Flush ()

  Force any stored data in stream to be written to underlying storage.
- void SetLength (uint64 length)
- void Put (const void ∗data, uint32 count)

- uint8 Get_uint8 ()
- void Put_uint8 (uint8 x)
- uint16 Get_uint16 ()
- void Put_uint16 (uint16 x)
- uint32 Get_uint32 ()
- uint32 **Get_uint32_LE** ()
- void Put_uint32 (uint32 x)
- uint64 Get_uint64 ()
- void Put_uint64 (uint64 x)
- int8 Get_int8 ()
- void Put_int8 (int8 x)
- int16 Get_int16 ()
- void Put_int16 (int16 x)
- int32 Get_int32 ()
- void Put_int32 (int32 x)
- int64 Get_int64 ()
- void Put_int64 (int64 x)
- real32 Get_real32 ()
- void Put_real32 (real32 x)
- real64 Get_real64 ()
- void Put_real64 (real64 x)
- void Get_CString (char ∗data, uint32 maxLength)
- void Get_UString (char ∗data, uint32 maxLength)
- void PutZeros (uint64 count)
- void PadAlign2 ()

  Writes zeros to align the stream position to a multiple of 2.
- void PadAlign4 ()

  Writes zeros to align the stream position to a multiple of 4.
- uint32 TagValue_uint32 (uint32 tagType)
- int32 TagValue_int32 (uint32 tagType)
- dng_urational TagValue_urational (uint32 tagType)
- dng_srational TagValue_srational (uint32 tagType)
- real64 TagValue_real64 (uint32 tagType)
- dng_abort_sniffer ∗ Sniffer () const
- void SetSniffer (dng_abort_sniffer ∗sniffer)
- virtual void CopyToStream (dng_stream &dstStream, uint64 count)
- void DuplicateStream (dng_stream &dstStream)

Protected Member Functions

- **dng_stream** (dng_abort_sniffer ∗sniffer=NULL, uint32 bufferSize=kDefaultBufferSize, uint64 offsetInOriginal←File=kDNGStreamInvalidOffset)
- virtual uint64 **DoGetLength** ()
- virtual void **DoRead** (void ∗data, uint32 count, uint64 offset)
- virtual void **DoSetLength** (uint64 length)
- virtual void **DoWrite** (const void ∗data, uint32 count, uint64 offset)

5.165.1 Detailed Description

Base stream abstraction. Has support for going between stream and pointer abstraction.

5.165.2 Constructor & Destructor Documentation

5.165.2.1 dng_stream()

```
dng_stream::dng_stream (
            const void * data,
            uint32 count,
            uint64 offsetInOriginalFile = kDNGStreamInvalidOffset )
```

Construct a stream with initial data.

Parameters

data	Pointer to initial contents of stream.
count	Number of bytes data is valid for.
offsetInOriginalFile	If data came from a file originally, offset can be saved here for later use.

5.165.3 Member Function Documentation

5.165.3.1 AsMemoryBlock()

```
dng_memory_block * dng_stream::AsMemoryBlock (
            dng_memory_allocator & allocator )
```

Return the entire stream as a single memory block. This works for all streams, but requires copying the data to a new buffer.

Parameters

allocator	Allocator used to allocate memory.

References dng_memory_allocator::Allocate(), Flush(), Get(), Length(), SetReadPosition(), and ThrowProgramError().

Referenced by dng_iptc::Spool().

5.165.3.2 BigEndian()

```
bool dng_stream::BigEndian ( ) const
```

Getter for whether data in stream is big endian.

Return values

If	true, data in stream is big endian.

Referenced by LittleEndian().

5.165.3.3 CopyToStream()

```
void dng_stream::CopyToStream (
             dng_stream & dstStream,
             uint64 count )  [virtual]
```

Copy a specified number of bytes to a target stream.

Parameters

dstStream	The target stream.
count	The number of bytes to copy.

Reimplemented in dng_memory_stream.

References dng_memory_data::Buffer(), Get(), and Put().

Referenced by dng_memory_stream::CopyToStream(), and DuplicateStream().

5.165.3.4 Data()

```
const void * dng_stream::Data ( ) const
```

Return pointer to stream contents if the stream is entirely available as a single memory block, NULL otherwise.

5.165.3.5 DuplicateStream()

```
void dng_stream::DuplicateStream (
             dng_stream & dstStream )
```

Makes the target stream a copy of this stream.

Parameters

dstStream	The target stream.

References CopyToStream(), Flush(), Length(), SetLength(), SetReadPosition(), and SetWritePosition().

5.165.3.6 Get()

```
void dng_stream::Get (
             void * data,
             uint32 count,
             uint32 maxOverRead = 0 )
```

Get data from stream. Exception is thrown and no data is read if insufficient data available in stream.

Parameters

data	Buffer to put data into. Must be valid for count bytes.
count	Bytes of data to read.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References DNG_ASSERT, Flush(), Length(), and ThrowEndOfFile().

Referenced by AsMemoryBlock(), CopyToStream(), Get_real64(), Get_uint16(), Get_uint32(), Get_uint64(), Get_uint8(), and dng_iptc::Parse().

5.165.3.7 Get_CString()

```
void dng_stream::Get_CString (
             char * data,
             uint32 maxLength )
```

Get an 8-bit character string from stream and advance read position. Routine always reads until a NUL character (8-bits of zero) is read. (That is, only maxLength bytes will be returned in buffer, but the stream is always advanced until a NUL is read or EOF is reached.)

Parameters

data	Buffer in which string is returned.
maxLength	Maximum number of bytes to place in buffer.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if stream runs out before NUL is seen.

References Get_uint8().

5.165.3.8 Get_int16()

```
int16 dng_stream::Get_int16 ( ) [inline]
```

Get one 16-bit integer from stream and advance read position. Byte swap if byte swapping is turned on.

Return values

One	16-bit integer.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get_uint16().

Referenced by TagValue_int32().

5.165.3.9 Get_int32()

```
int32 dng_stream::Get_int32 ( ) [inline]
```

Get one 32-bit integer from stream and advance read position. Byte swap if byte swapping is turned on.

Return values

One	32-bit integer.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get_uint32().

Referenced by dng_area_spec::GetData(), TagValue_int32(), TagValue_real64(), TagValue_srational(), and TagValue↩
_urational().

5.165.3.10 Get_int64()

```
int64 dng_stream::Get_int64 ( ) [inline]
```

Get one 64-bit integer from stream and advance read position. Byte swap if byte swapping is turned on.

Return values

One	64-bit integer.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get_uint64().

5.165.3.11 Get_int8()

```
int8 dng_stream::Get_int8 ( )  [inline]
```

Get one 8-bit integer from stream and advance read position.

Return values

One	8-bit integer.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get_uint8().

Referenced by dng_iptc::Parse(), and TagValue_int32().

5.165.3.12 Get_real32()

```
real32 dng_stream::Get_real32 ( )
```

Get one 32-bit IEEE floating-point number from stream and advance read position. Byte swap if byte swapping is turned on.

Return values

One	32-bit IEEE floating-point number.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get_uint32().

Referenced by dng_gain_map::GetStream(), and TagValue_real64().

5.165.3.13 Get_real64()

```
real64 dng_stream::Get_real64 ( )
```

Get one 64-bit IEEE floating-point number from stream and advance read position. Byte swap if byte swapping is turned on.

Return values

One	64-bit IEEE floating-point number .

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get(), and Get_uint32().

Referenced by dng_gain_map::GetStream(), and TagValue_real64().

5.165.3.14 Get_uint16()

```
uint16 dng_stream::Get_uint16 ( )
```

Get an unsigned 16-bit integer from stream and advance read position. Byte swap if byte swapping is turned on.

Return values

One	unsigned 16-bit integer.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get().

Referenced by Get_int16(), Get_UString(), dng_info::Parse(), dng_iptc::Parse(), and TagValue_uint32().

5.165.3.15 Get_uint32()

```
uint32 dng_stream::Get_uint32 ( )
```

Get an unsigned 32-bit integer from stream and advance read position. Byte swap if byte swapping is turned on.

Return values

One	unsigned 32-bit integer.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get().

Referenced by Get_int32(), Get_real32(), Get_real64(), Get_uint64(), dng_area_spec::GetData(), dng_gain_map↵::GetStream(), dng_info::Parse(), dng_opcode_list::Parse(), TagValue_real64(), TagValue_uint32(), and TagValue_↵urational().

5.165.3.16 Get_uint64()

```
uint64 dng_stream::Get_uint64 ( )
```

Get an unsigned 64-bit integer from stream and advance read position. Byte swap if byte swapping is turned on.

Return values

One	unsigned 64-bit integer.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get(), and Get_uint32().

Referenced by Get_int64().

5.165.3.17 Get_uint8()

```
uint8 dng_stream::Get_uint8 ( ) [inline]
```

Get an unsigned 8-bit integer from stream and advance read position.

Return values

One	unsigned 8-bit integer.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get().

Referenced by Get_CString(), Get_int8(), dng_iptc::Parse(), and TagValue_uint32().

5.165.3.18 Get_UString()

```
void dng_stream::Get_UString (
            char * data,
            uint32 maxLength )
```

Get a 16-bit character string from stream and advance read position. 16-bit characters are truncated to 8-bits. Routine always reads until a NUL character (16-bits of zero) is read. (That is, only maxLength bytes will be returned in buffer, but the stream is always advanced until a NUL is read or EOF is reached.)

Parameters

data	Buffer to place string in.
maxLength	Maximum number of bytes to place in buffer.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if stream runs out before NUL is seen.

References Get_uint16().

5.165.3.19 Length()

```
uint64 dng_stream::Length ( )  [inline]
```

Getter for length of data in stream.

Return values

Length	of readable data in stream.

Referenced by AsMemoryBlock(), dng_memory_stream::CopyToStream(), DuplicateStream(), Get(), dng_info::Parse(), dng_iptc::Parse(), Put_uint8(), SetLength(), SetReadPosition(), and dng_iptc::Spool().

5.165.3.20 LittleEndian()

```
bool dng_stream::LittleEndian ( ) const  [inline]
```

Getter for whether data in stream is big endian.

Return values

If	true, data in stream is big endian.

References BigEndian().

5.165.3.21 OffsetInOriginalFile()

```
uint64 dng_stream::OffsetInOriginalFile ( ) const
```

Getter for offset in original file.

Return values

kInvalidOffset	if no offset in original file is set, offset in original file otherwise.

5.165.3.22 Position()

```
uint64 dng_stream::Position ( ) const  [inline]
```

Getter for current offset in stream.

Return values

current	offset from start of stream.

Referenced by dng_memory_stream::CopyToStream(), PadAlign2(), PadAlign4(), dng_info::Parse(), dng_iptc::Parse(), dng_opcode_list::Parse(), PositionInOriginalFile(), and Skip().

5.165.3.23 PositionInOriginalFile()

```
uint64 dng_stream::PositionInOriginalFile ( ) const
```

Getter for current position in original file, taking into account OffsetInOriginalFile stream data was taken from.

Return values

kInvalidOffset	if no offset in original file is set, sum of offset in original file and current position otherwise.

References Position().

Referenced by dng_info::Parse().

5.165.3.24 Put()

```
void dng_stream::Put (
            const void * data,
            uint32 count )
```

Write data to stream.

Parameters

data	Buffer of data to write to stream.
count	Bytes of in data.

References Flush().

Referenced by dng_memory_stream::CopyToStream(), CopyToStream(), Put_real32(), Put_real64(), Put_uint16(), Put_uint32(), Put_uint64(), Put_uint8(), dng_opcode_Unknown::PutData(), dng_iptc::Spool(), and dng_camera_↩ profile::UniqueID().

5.165.3.25 Put_int16()

```
void dng_stream::Put_int16 (
            int16 x ) [inline]
```

Put one 16-bit integer to stream and advance write position. Byte swap if byte swapping is turned on.

Parameters

x	One 16-bit integer.

References Put_uint16().

5.165.3.26 Put_int32()

```
void dng_stream::Put_int32 (
            int32 x ) [inline]
```

Put one 32-bit integer to stream and advance write position. Byte swap if byte swapping is turned on.

Parameters

x	One 32-bit integer.

References Put_uint32().

Referenced by dng_opcode_TrimBounds::PutData(), dng_area_spec::PutData(), and dng_opcode_FixBadPixelsList↩ ::PutData().

5.165.3.27 Put_int64()

```
void dng_stream::Put_int64 (
             int64 x )  [inline]
```

Put one 64-bit integer to stream and advance write position. Byte swap if byte swapping is turned on.

Parameters

x	One 64-bit integer.

References Put_uint64().

5.165.3.28 Put_int8()

```
void dng_stream::Put_int8 (
             int8 x )  [inline]
```

Put one 8-bit integer to stream and advance write position.

Parameters

x	One 8-bit integer.

References Put_uint8().

5.165.3.29 Put_real32()

```
void dng_stream::Put_real32 (
             real32 x )
```

Put one 32-bit IEEE floating-point number to stream and advance write position. Byte swap if byte swapping is turned on.

Parameters

x	One 32-bit IEEE floating-point number.

References Put(), and Put_uint32().

Referenced by dng_opcode_DeltaPerRow::PutData(), dng_opcode_DeltaPerColumn::PutData(), dng_opcode_Scale↩ PerRow::PutData(), dng_opcode_ScalePerColumn::PutData(), and dng_gain_map::PutStream().

5.165.3.30 Put_real64()

```
void dng_stream::Put_real64 (
            real64 x )
```

Put one 64-bit IEEE floating-point number to stream and advance write position. Byte swap if byte swapping is turned on.

Parameters

x	One64-bit IEEE floating-point number.

References Put(), and Put_uint32().

Referenced by dng_opcode_MapPolynomial::PutData(), dng_opcode_WarpRectilinear::PutData(), dng_opcode_↩ WarpFisheye::PutData(), dng_opcode_FixVignetteRadial::PutData(), and dng_gain_map::PutStream().

5.165.3.31 Put_uint16()

```
void dng_stream::Put_uint16 (
            uint16 x )
```

Put an unsigned 16-bit integer to stream and advance write position. Byte swap if byte swapping is turned on.

Parameters

x	One unsigned 16-bit integer.

References Put().

Referenced by Put_int16(), dng_opcode_MapTable::PutData(), and dng_iptc::Spool().

5.165.3.32 Put_uint32()

```
void dng_stream::Put_uint32 (
            uint32 x )
```

Put an unsigned 32-bit integer to stream and advance write position. Byte swap if byte swapping is turned on.

Parameters

x	One unsigned 32-bit integer.

References Put().

Referenced by dng_opcode_list::FingerprintToStream(), Put_int32(), Put_real32(), Put_real64(), Put_uint64(), dng↩
_opcode_TrimBounds::PutData(), dng_opcode_FixBadPixelsConstant::PutData(), dng_area_spec::PutData(), dng↩
_opcode_GainMap::PutData(), dng_opcode_MapTable::PutData(), dng_opcode::PutData(), dng_opcode_Map↩
Polynomial::PutData(), dng_opcode_FixBadPixelsList::PutData(), dng_opcode_Unknown::PutData(), dng_opcode↩
_DeltaPerRow::PutData(), dng_opcode_DeltaPerColumn::PutData(), dng_opcode_ScalePerRow::PutData(), dng_↩
opcode_ScalePerColumn::PutData(), dng_opcode_WarpRectilinear::PutData(), dng_opcode_WarpFisheye::PutData(),
dng_opcode_FixVignetteRadial::PutData(), and dng_gain_map::PutStream().

5.165.3.33 Put_uint64()

```
void dng_stream::Put_uint64 (
            uint64 x )
```

Put an unsigned 64-bit integer to stream and advance write position. Byte swap if byte swapping is turned on.

Parameters

x	One unsigned 64-bit integer.

References Put(), and Put_uint32().

Referenced by Put_int64().

5.165.3.34 Put_uint8()

```
void dng_stream::Put_uint8 (
            uint8 x )  [inline]
```

Put an unsigned 8-bit integer to stream and advance write position.

Parameters

x	One unsigned 8-bit integer.

References Length(), and Put().

Referenced by Put_int8(), and dng_iptc::Spool().

5.165.3.35 PutZeros()

```
void dng_stream::PutZeros (
             uint64 count )
```

Writes the specified number of zero bytes to stream.

Parameters

count	Number of zero bytes to write.

Referenced by PadAlign2(), and PadAlign4().

5.165.3.36 SetBigEndian()

```
void dng_stream::SetBigEndian (
             bool bigEndian = true )
```

Setter for whether data in stream is big endian.

Parameters

bigEndian	If true, data in stream is big endian.

Referenced by dng_info::Parse(), dng_iptc::Parse(), SetLittleEndian(), dng_opcode_list::Spool(), and dng_iptc::Spool().

5.165.3.37 SetLength()

```
void dng_stream::SetLength (
             uint64 length )
```

Set length of available data.

Parameters

length	Number of bytes of avialble data in stream.

References Flush(), and Length().

Referenced by DuplicateStream().

5.165.3.38 SetLittleEndian()

```
void dng_stream::SetLittleEndian (
            bool littleEndian = true )  [inline]
```

Setter for whether data in stream is big endian.

Parameters

littleEndian	If true, data in stream is big endian.

References SetBigEndian().

Referenced by dng_info::Parse(), and dng_camera_profile::UniqueID().

5.165.3.39 SetSniffer()

```
void dng_stream::SetSniffer (
            dng_abort_sniffer * sniffer )  [inline]
```

Putter for sniffer associated with stream.

Parameters

sniffer	The new sniffer to use (or NULL for none).

5.165.3.40 SetSwapBytes()

```
void dng_stream::SetSwapBytes (
            bool swapBytes )  [inline]
```

Setter for whether stream is swapping byte order on input/output.

Parameters

swapBytes	If true, stream will swap byte order on input or output for future reads/writes.

5.165.3.41 Skip()

```
void dng_stream::Skip (
            uint64 delta )  [inline]
```

Skip forward in stream.

Parameters

delta	Number of bytes to skip forward.

References Position(), and SetReadPosition().

5.165.3.42 Sniffer()

```
dng_abort_sniffer* dng_stream::Sniffer ( ) const  [inline]
```

Getter for sniffer associated with stream.

Return values

The	sniffer for this stream.

5.165.3.43 SwapBytes()

```
bool dng_stream::SwapBytes ( ) const  [inline]
```

Getter for whether stream is swapping byte order on input/output.

Return values

If	true, data will be swapped on input/output.

5.165.3.44 TagValue_int32()

```
int32 dng_stream::TagValue_int32 (
            uint32 tagType )
```

Get a value of size indicated by tag type from stream and advance read position. Byte swap if byte swapping is turned on and tag type is larger than a byte. Value is returned as a 32-bit integer.

Parameters

tagType	Tag type of data stored in stream.

Return values

One	32-bit integer.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get_int16(), Get_int32(), Get_int8(), and TagValue_real64().

Referenced by TagValue_real64(), and TagValue_urational().

5.165.3.45 TagValue_real64()

```
real64 dng_stream::TagValue_real64 (
             uint32 tagType )
```

Get a value of size indicated by tag type from stream and advance read position. Byte swap if byte swapping is turned on and tag type is larger than a byte. Value is returned as a 64-bit IEEE floating-point number.

Parameters

tagType	Tag type of data stored in stream.

Return values

One	64-bit IEEE floating-point number.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get_int32(), Get_real32(), Get_real64(), Get_uint32(), TagValue_int32(), and TagValue_uint32().

Referenced by TagValue_int32(), TagValue_srational(), TagValue_uint32(), and TagValue_urational().

5.165.3.46 TagValue_srational()

```
dng_srational dng_stream::TagValue_srational (
             uint32 tagType )
```

Get a value of size indicated by tag type from stream and advance read position. Byte swap if byte swapping is turned on and tag type is larger than a byte. Value is returned as a dng_srational.

Parameters

tagType	Tag type of data stored in stream.

Return values

One	dng_srational.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get_int32(), and TagValue_real64().

5.165.3.47 TagValue_uint32()

```
uint32 dng_stream::TagValue_uint32 (
            uint32 tagType )
```

Get a value of size indicated by tag type from stream and advance read position. Byte swap if byte swapping is turned on and tag type is larger than a byte. Value is returned as an unsigned 32-bit integer.

Parameters

tagType	Tag type of data stored in stream.

Return values

One	unsigned 32-bit integer.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get_uint16(), Get_uint32(), Get_uint8(), and TagValue_real64().

Referenced by TagValue_real64(), and TagValue_urational().

5.165.3.48 TagValue_urational()

```
dng_urational dng_stream::TagValue_urational (
            uint32 tagType )
```

Get a value of size indicated by tag type from stream and advance read position. Byte swap if byte swapping is turned on and tag type is larger than a byte. Value is returned as a dng_urational.

Parameters

tagType	Tag type of data stored in stream.

Return values

One	dng_urational.

Exceptions

dng_exception	with fErrorCode equal to dng_error_end_of_file if not enough data in stream.

References Get_int32(), Get_uint32(), TagValue_int32(), TagValue_real64(), and TagValue_uint32().

The documentation for this class was generated from the following files:

- dng_stream.h
- dng_stream.cpp

5.166 dng_stream_contiguous_read_hint Class Reference

Public Member Functions

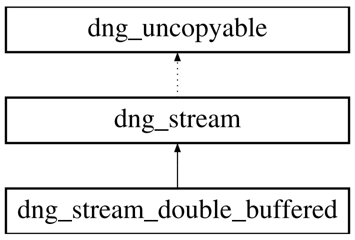
- **dng_stream_contiguous_read_hint** (dng_stream &stream, dng_memory_allocator &allocator, uint64 offset, uint64 count)

The documentation for this class was generated from the following files:

- dng_stream.h
- dng_stream.cpp

5.167 dng_stream_double_buffered Class Reference

Inheritance diagram for dng_stream_double_buffered:



Public Member Functions

- **dng_stream_double_buffered** (dng_stream &stream, uint32 bufferSize=kDefaultBufferSize)

Protected Member Functions

- virtual uint64 **DoGetLength** ()
- virtual void **DoRead** (void ∗data, uint32 count, uint64 offset)

Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_stream.h

5.168 dng_string Class Reference

Public Member Functions

- **dng_string** (const dng_string &s)
- dng_string & **operator=** (const dng_string &s)
- const char ∗ **Get** () const
- bool **IsASCII** () const
- void **Set** (const char ∗s)
- void **Set_ASCII** (const char ∗s)
- void **Set_UTF8** (const char ∗s)
- uint32 **Get_SystemEncoding** (dng_memory_data &buffer) const
- void **Set_SystemEncoding** (const char ∗s)
- bool **ValidSystemEncoding** () const
- void **Set_JIS_X208_1990** (const char ∗s)
- void **Set_UTF8_or_System** (const char ∗s)
- uint32 **Get_UTF16** (dng_memory_data &buffer) const
- void **Set_UTF16** (const uint16 ∗s)
- void **Clear** ()
- void **Truncate** (uint32 maxBytes)
- bool **TrimTrailingBlanks** ()
- bool **TrimLeadingBlanks** ()
- bool **IsEmpty** () const
- bool **NotEmpty** () const
- uint32 **Length** () const
- bool **operator==** (const dng_string &s) const
- bool **operator!=** (const dng_string &s) const
- bool **Matches** (const char ∗s, bool case_sensitive=false) const
- bool **StartsWith** (const char ∗s, bool case_sensitive=false) const
- bool **EndsWith** (const char ∗s, bool case_sensitive=false) const
- bool **Contains** (const char ∗s, bool case_sensitive=false, int32 ∗match_offset=NULL) const
- bool **Replace** (const char ∗old_string, const char ∗new_string, bool case_sensitive=true)

- void **ReplaceChars** (char oldChar, char newChar)
- bool **TrimLeading** (const char ∗s, bool case_sensitive=false)
- void **Append** (const char ∗s)
- void **SetUppercase** ()
- void **SetLowercase** ()
- void **SetLineEndings** (char ending)
- void **SetLineEndingsToNewLines** ()
- void **SetLineEndingsToReturns** ()
- void **StripLowASCII** ()
- void **ForceASCII** ()
- int32 **Compare** (const dng_string &s, bool digitsAsNumber=true) const
- void **NormalizeAsCommaSeparatedNumbers** ()

Static Public Member Functions

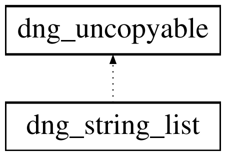
- static uint32 **DecodeUTF8** (const char ∗&s, uint32 maxBytes=6, bool ∗isValid=NULL)
- static bool **IsUTF8** (const char ∗s)
- static bool **Matches** (const char ∗t, const char ∗s, bool case_sensitive=false)

The documentation for this class was generated from the following files:

- dng_string.h
- dng_string.cpp

5.169 dng_string_list Class Reference

Inheritance diagram for dng_string_list:



Public Member Functions

- uint32 **Count** () const
- dng_string & **operator[ ]** (uint32 index)
- const dng_string & **operator[ ]** (uint32 index) const
- void **Allocate** (uint32 minSize)
- void **Insert** (uint32 index, const dng_string &s)
- void **Append** (const dng_string &s)
- bool **Contains** (const dng_string &s) const
- void **Clear** ()

The documentation for this class was generated from the following files:

- dng_string_list.h
- dng_string_list.cpp

5.170 dng_suite Struct Reference

Public Attributes

- ZeroBytesProc ∗ **ZeroBytes**
- CopyBytesProc ∗ **CopyBytes**
- SwapBytes16Proc ∗ **SwapBytes16**
- SwapBytes32Proc ∗ **SwapBytes32**
- SetArea8Proc ∗ **SetArea8**
- SetArea16Proc ∗ **SetArea16**
- SetArea32Proc ∗ **SetArea32**
- CopyArea8Proc ∗ **CopyArea8**
- CopyArea16Proc ∗ **CopyArea16**
- CopyArea32Proc ∗ **CopyArea32**
- CopyArea8_16Proc ∗ **CopyArea8_16**
- CopyArea8_S16Proc ∗ **CopyArea8_S16**
- CopyArea8_32Proc ∗ **CopyArea8_32**
- CopyArea16_S16Proc ∗ **CopyArea16_S16**
- CopyArea16_32Proc ∗ **CopyArea16_32**
- CopyArea8_R32Proc ∗ **CopyArea8_R32**
- CopyArea16_R32Proc ∗ **CopyArea16_R32**
- CopyAreaS16_R32Proc ∗ **CopyAreaS16_R32**
- CopyAreaR32_8Proc ∗ **CopyAreaR32_8**
- CopyAreaR32_16Proc ∗ **CopyAreaR32_16**
- CopyAreaR32_S16Proc ∗ **CopyAreaR32_S16**
- RepeatArea8Proc ∗ **RepeatArea8**
- RepeatArea16Proc ∗ **RepeatArea16**
- RepeatArea32Proc ∗ **RepeatArea32**
- ShiftRight16Proc ∗ **ShiftRight16**
- BilinearRow16Proc ∗ **BilinearRow16**
- BilinearRow32Proc ∗ **BilinearRow32**
- BaselineABCtoRGBProc ∗ **BaselineABCtoRGB**
- BaselineABCDtoRGBProc ∗ **BaselineABCDtoRGB**
- BaselineHueSatMapProc ∗ **BaselineHueSatMap**
- BaselineGrayToRGBProc ∗ **BaselineRGBtoGray**
- BaselineRGBtoRGBProc ∗ **BaselineRGBtoRGB**
- Baseline1DTableProc ∗ **Baseline1DTable**
- BaselineRGBToneProc ∗ **BaselineRGBTone**
- ResampleDown16Proc ∗ **ResampleDown16**
- ResampleDown32Proc ∗ **ResampleDown32**
- ResampleAcross16Proc ∗ **ResampleAcross16**
- ResampleAcross32Proc ∗ **ResampleAcross32**
- EqualBytesProc ∗ **EqualBytes**
- EqualArea8Proc ∗ **EqualArea8**
- EqualArea16Proc ∗ **EqualArea16**
- EqualArea32Proc ∗ **EqualArea32**
- VignetteMask16Proc ∗ **VignetteMask16**
- Vignette16Proc ∗ **Vignette16**
- Vignette32Proc ∗ **Vignette32**
- MapArea16Proc ∗ **MapArea16**
- BaselineMapPoly32Proc ∗ **BaselineMapPoly32**

The documentation for this struct was generated from the following file:

- dng_bottlenecks.h

5.171 dng_temperature Class Reference

Public Member Functions

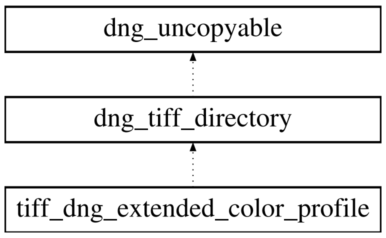
- **dng_temperature** (real64 temperature, real64 tint)
- **dng_temperature** (const dng_xy_coord &xy)
- void **SetTemperature** (real64 temperature)
- real64 **Temperature** () const
- void **SetTint** (real64 tint)
- real64 **Tint** () const
- void **Set_xy_coord** (const dng_xy_coord &xy)
- dng_xy_coord **Get_xy_coord** () const

The documentation for this class was generated from the following files:

- dng_temperature.h
- dng_temperature.cpp

5.172 dng_tiff_directory Class Reference

Inheritance diagram for dng_tiff_directory:



Public Types

- enum **OffsetsBase** { **offsetsRelativeToStream** = 0, **offsetsRelativeToExplicitBase** = 1, **offsetsRelativeToIFD** = 2 }

Public Member Functions

- void **Add** (const tiff_tag ∗tag)
- void **SetChained** (uint32 offset)
- uint32 **Size** () const
- void **Put** (dng_stream &stream, OffsetsBase offsetsBase=offsetsRelativeToStream, uint32 explicitBase=0) const

The documentation for this class was generated from the following files:

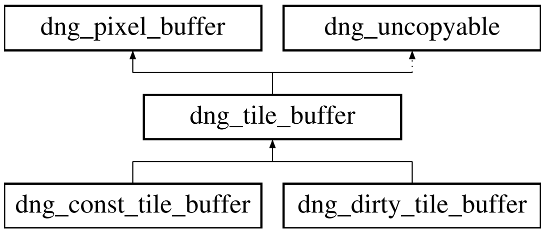
- dng_image_writer.h
- dng_image_writer.cpp

5.173 dng_tile_buffer Class Reference

Class to get resource acquisition is instantiation behavior for tile buffers. Can be dirty or constant tile access.

```
#include <dng_image.h>
```

Inheritance diagram for dng_tile_buffer:



Public Member Functions

- void **SetRefData** (void ∗refData)
- void ∗ **GetRefData** () const

Protected Member Functions

- dng_tile_buffer (const dng_image &image, const dng_rect &tile, bool dirty)

Protected Attributes

- const dng_image & **fImage**
- void ∗ **fRefData**

Additional Inherited Members

5.173.1 Detailed Description

Class to get resource acquisition is instantiation behavior for tile buffers. Can be dirty or constant tile access.

5.173.2 Constructor & Destructor Documentation

5.173.2.1 dng_tile_buffer()

```
dng_tile_buffer::dng_tile_buffer (
            const dng_image & image,
            const dng_rect & tile,
            bool dirty ) [protected]
```

Obtain a tile from an image.

Parameters

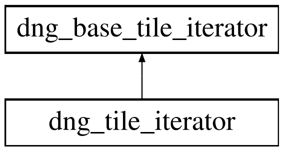
image	Image tile will come from.
tile	Rectangle denoting extent of tile.
dirty	Flag indicating whether this is read-only or read-write acesss.

The documentation for this class was generated from the following files:

- dng_image.h
- dng_image.cpp

5.174 dng_tile_iterator Class Reference

Inheritance diagram for dng_tile_iterator:



Public Member Functions

- **dng_tile_iterator** (const dng_image &image, const dng_rect &area)
- **dng_tile_iterator** (const dng_point &tileSize, const dng_rect &area)
- **dng_tile_iterator** (const dng_rect &tile, const dng_rect &area)
- virtual bool **GetOneTile** (dng_rect &tile)

Protected Attributes

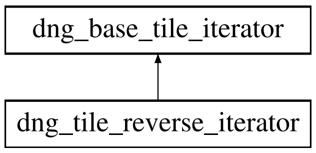
- dng_rect **fArea**
- int32 **fTileWidth**
- int32 **fTileHeight**
- int32 **fTileTop**
- int32 **fTileLeft**
- int32 **fRowLeft**
- int32 **fLeftPage**
- int32 **fRightPage**
- int32 **fTopPage**
- int32 **fBottomPage**
- int32 **fHorizontalPage**
- int32 **fVerticalPage**

The documentation for this class was generated from the following files:

- dng_tile_iterator.h
- dng_tile_iterator.cpp

5.175 dng_tile_reverse_iterator Class Reference

Inheritance diagram for dng_tile_reverse_iterator:



Public Member Functions

- **dng_tile_reverse_iterator** (const dng_image &image, const dng_rect &area)
- **dng_tile_reverse_iterator** (const dng_point &tileSize, const dng_rect &area)
- **dng_tile_reverse_iterator** (const dng_rect &tile, const dng_rect &area)
- virtual bool **GetOneTile** (dng_rect &tile)

Public Attributes

- std::vector< dng_rect > **fTiles**
- size_t **fIndex**

The documentation for this class was generated from the following files:

- dng_tile_iterator.h
- dng_tile_iterator.cpp

5.176 dng_time_zone Class Reference

Class for holding a time zone.

`#include <dng_date_time.h>`

Public Member Functions

- void **Clear** ()
- void **SetOffsetHours** (int32 offset)
- void **SetOffsetMinutes** (int32 offset)
- void **SetOffsetSeconds** (int32 offset)
- bool **IsValid** () const
- bool **NotValid** () const
- int32 **OffsetMinutes** () const
- bool **IsExactHourOffset** () const
- int32 **ExactHourOffset** () const
- dng_string **Encode_ISO_8601** () const

5.176.1 Detailed Description

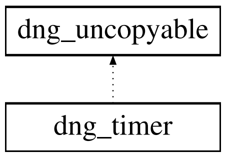
Class for holding a time zone.

The documentation for this class was generated from the following files:

- dng_date_time.h
- dng_date_time.cpp

5.177 dng_timer Class Reference

Inheritance diagram for dng_timer:



Public Member Functions

- **dng_timer** (const char ∗message)

The documentation for this class was generated from the following files:

- dng_utils.h
- dng_utils.cpp

5.178 dng_tone_curve Class Reference

Public Member Functions

- bool **operator==** (const dng_tone_curve &curve) const
- bool **operator!=** (const dng_tone_curve &curve) const
- void **SetNull** ()
- bool **IsNull** () const
- void **SetInvalid** ()
- bool **IsValid** () const
- void **Solve** (dng_spline_solver &solver) const

Public Attributes

- dng_std_vector< dng_point_real64 > **fCoord**

The documentation for this class was generated from the following files:

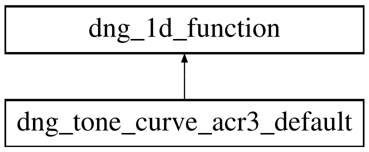
- dng_tone_curve.h
- dng_tone_curve.cpp

5.179 dng_tone_curve_acr3_default Class Reference

Default ACR3 tone curve.

```
#include <dng_render.h>
```

Inheritance diagram for dng_tone_curve_acr3_default:



Public Member Functions

- virtual real64 Evaluate (real64 x) const
  Returns output value for a given input tone.
- virtual real64 EvaluateInverse (real64 x) const
  Returns nearest input value for a given output tone.

Static Public Member Functions

- static const dng_1d_function & **Get** ()

5.179.1 Detailed Description

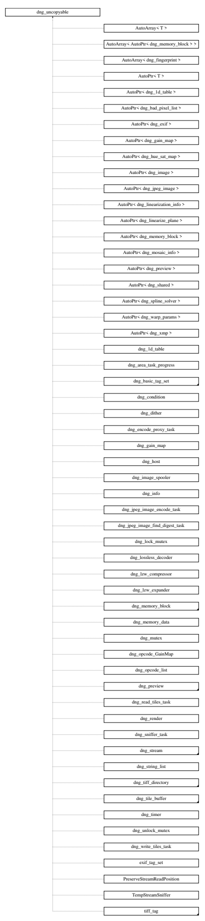
Default ACR3 tone curve.

The documentation for this class was generated from the following files:

- dng_render.h
- dng_render.cpp

5.180 dng_uncopyable Class Reference

Inheritance diagram for dng_uncopyable:

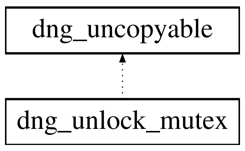


The documentation for this class was generated from the following file:

- dng_uncopyable.h

5.181 dng_unlock_mutex Class Reference

Inheritance diagram for dng_unlock_mutex:



Public Member Functions

- **dng_unlock_mutex** (dng_mutex ∗mutex)
- **dng_unlock_mutex** (dng_mutex &mutex)

The documentation for this class was generated from the following files:

- dng_mutex.h
- dng_mutex.cpp

5.182 dng_urational Class Reference

Public Member Functions

- **dng_urational** (uint32 nn, uint32 dd)
- void **Clear** ()
- bool **IsValid** () const
- bool **NotValid** () const
- bool **operator==** (const dng_urational &r) const
- bool **operator!=** (const dng_urational &r) const
- real64 **As_real64** () const
- void **Set_real64** (real64 x, uint32 dd=0)
- void **ReduceByFactor** (uint32 factor)

Public Attributes

- uint32 **n**
- uint32 **d**

The documentation for this class was generated from the following files:

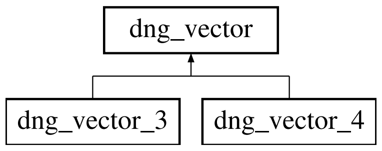
- dng_rational.h
- dng_rational.cpp

5.183 dng_vector Class Reference

Class to represent 1-dimensional vector with up to kMaxColorPlanes components.

```
#include <dng_matrix.h>
```

Inheritance diagram for dng_vector:



Public Member Functions

- **dng_vector** (uint32 count)
- **dng_vector** (const dng_vector &v)
- void **Clear** ()
- void **SetIdentity** (uint32 count)
- uint32 **Count** () const
- real64 & **operator [ ]** (uint32 index)
- const real64 & **operator [ ]** (uint32 index) const
- bool **operator==** (const dng_vector &v) const
- bool **operator!=** (const dng_vector &v) const
- bool **IsEmpty** () const
- bool **NotEmpty** () const
- real64 **MaxEntry** () const
- real64 **MinEntry** () const
- void **Scale** (real64 factor)
- void **Round** (real64 factor)
- dng_matrix **AsDiagonal** () const
- dng_matrix **AsColumn** () const

Protected Attributes

- uint32 **fCount**
- real64 **fData** [kMaxColorPlanes]

5.183.1 Detailed Description

Class to represent 1-dimensional vector with up to kMaxColorPlanes components.

The documentation for this class was generated from the following files:

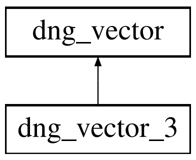
- dng_matrix.h
- dng_matrix.cpp

5.184 dng_vector_3 Class Reference

A 3-element vector.

`#include <dng_matrix.h>`

Inheritance diagram for dng_vector_3:



Public Member Functions

- **dng_vector_3** (const dng_vector &v)
- **dng_vector_3** (real64 a0, real64 a1, real64 a2)

Additional Inherited Members

5.184.1 Detailed Description

A 3-element vector.

The documentation for this class was generated from the following files:

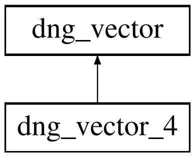
- dng_matrix.h
- dng_matrix.cpp

5.185 dng_vector_4 Class Reference

A 4-element vector.

```
#include <dng_matrix.h>
```

Inheritance diagram for dng_vector_4:



Public Member Functions

- **dng_vector_4** (const dng_vector &v)
- **dng_vector_4** (real64 a0, real64 a1, real64 a2, real64 a3)

Additional Inherited Members

5.185.1 Detailed Description

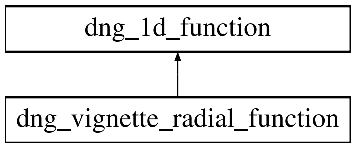
A 4-element vector.

The documentation for this class was generated from the following files:

- dng_matrix.h
- dng_matrix.cpp

5.186 dng_vignette_radial_function Class Reference

Inheritance diagram for dng_vignette_radial_function:



Public Member Functions

- **dng_vignette_radial_function** (const dng_vignette_radial_params &params)
- virtual real64 Evaluate (real64 x) const

Protected Attributes

- const dng_vignette_radial_params **fParams**

5.186.1 Member Function Documentation

5.186.1.1 Evaluate()

```
virtual real64 dng_vignette_radial_function::Evaluate (
            real64 x ) const  [inline], [virtual]
```

Return the mapping for value x. This method must be implemented by a derived class of dng_1d_function and the derived class determines the lookup method and function used.

Parameters

x	A value between 0.0 and 1.0 (inclusive).

Return values

Mapped	value for x

Implements dng_1d_function.

References DNG_REQUIRE.

The documentation for this class was generated from the following file:

- dng_lens_correction.cpp

5.187 dng_vignette_radial_params Class Reference

Radially-symmetric vignette (peripheral illuminational falloff) correction parameters.

```
#include <dng_lens_correction.h>
```

Public Member Functions

- **dng_vignette_radial_params** (const dng_std_vector< real64 > &params, const dng_point_real64 &center)
- **dng_vignette_radial_params** (const dng_vignette_radial_params &params)
- bool **IsNOP** () const
- bool **IsValid** () const
- void **Dump** () const

Public Attributes

- dng_std_vector< real64 > **fParams**
- dng_point_real64 **fCenter**

Static Public Attributes

- static const uint32 **kNumTerms** = 5

5.187.1 Detailed Description

Radially-symmetric vignette (peripheral illuminational falloff) correction parameters.

The documentation for this class was generated from the following files:

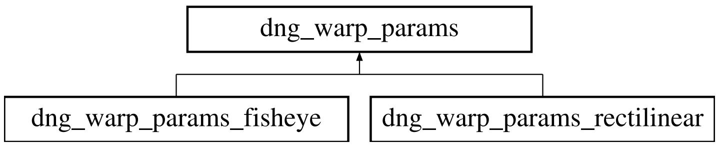
- dng_lens_correction.h
- dng_lens_correction.cpp

5.188 dng_warp_params Class Reference

Abstract base class holding common warp opcode parameters (e.g., number of planes, optical center) and common warp routines.

```
#include <dng_lens_correction.h>
```

Inheritance diagram for dng_warp_params:



Public Member Functions

- dng_warp_params ()

  Create empty (invalid) warp parameters.
- dng_warp_params (uint32 planes, const dng_point_real64 &fCenter)
- virtual bool IsNOPAll () const

  Is the entire correction a NOP for all planes?
- virtual bool IsNOP (uint32 plane) const

  Is the entire correction a NOP for the specified plane?
- virtual bool IsRadNOPAll () const

  Is the radial correction a NOP for all planes?
- virtual bool IsRadNOP (uint32 plane) const

  Is the radial correction a NOP for the specified plane?
- virtual bool IsTanNOPAll () const

  Is the tangential correction a NOP for all planes?
- virtual bool IsTanNOP (uint32 plane) const

  Is the tangential correction a NOP for the specified plane?
- virtual bool IsValid () const

  Do these warp params appear valid?
- virtual bool IsValidForNegative (const dng_negative &negative) const

  Are these warp params valid for the specified negative?
- virtual void PropagateToAllPlanes (uint32 totalPlanes)=0

  Propagate warp parameters from first plane to all other planes.
- virtual real64 Evaluate (uint32 plane, real64 r) const =0
- virtual real64 EvaluateInverse (uint32 plane, real64 r) const
- virtual real64 EvaluateRatio (uint32 plane, real64 r2) const =0
- virtual dng_point_real64 EvaluateTangential (uint32 plane, real64 r2, const dng_point_real64 &diff, const dng_point_real64 &diff2) const =0
- dng_point_real64 EvaluateTangential2 (uint32 plane, const dng_point_real64 &diff) const
- dng_point_real64 EvaluateTangential3 (uint32 plane, real64 r2, const dng_point_real64 &diff) const
- virtual real64 MaxSrcRadiusGap (real64 maxDstGap) const =0
- virtual dng_point_real64 MaxSrcTanGap (dng_point_real64 minDst, dng_point_real64 maxDst) const =0
- virtual real64 SafeMinRatio () const =0
- virtual real64 SafeMaxRatio () const =0
- virtual void Dump () const

  Debug parameters.

Public Attributes

- uint32 **fPlanes**
- dng_point_real64 **fCenter**

5.188.1 Detailed Description

Abstract base class holding common warp opcode parameters (e.g., number of planes, optical center) and common warp routines.

5.188.2 Constructor & Destructor Documentation

5.188.2.1 dng_warp_params()

```
dng_warp_params::dng_warp_params (
            uint32 planes,
            const dng_point_real64 & fCenter )
```

Create warp parameters with specified number of planes and image center.

Parameters

planes	The number of planes of parameters specified: It must be either 1 or equal to the number of planes of the image to be processed.
fCenter	The image center in relative coordinates.

References DNG_ASSERT, and kMaxColorPlanes.

5.188.3 Member Function Documentation

5.188.3.1 Evaluate()

```
virtual real64 dng_warp_params::Evaluate (
            uint32 plane,
            real64 r ) const  [pure virtual]
```

Evaluate the 1D radial warp function for the specified plane. Parameter r is the destination (i.e., corrected) normalized radius, i.e., the normalized Euclidean distance between a corrected pixel position and the optical center in the image. r lies in the range [0,1]. The returned result is non-negative.

Implemented in dng_warp_params_fisheye, and dng_warp_params_rectilinear.

Referenced by EvaluateInverse().

5.188.3.2 EvaluateInverse()

```
real64 dng_warp_params::EvaluateInverse (
            uint32 plane,
            real64 r ) const  [virtual]
```

Compute and return the inverse of Evaluate () above. The base implementation uses Newton's method to perform the inversion. Parameter r is the source (i.e., uncorrected) normalized radius, i.e., normalized Euclidean distance between a corrected pixel position and the optical center in the image. Both r and the computed result are non-negative.

References Evaluate().

5.188.3.3 EvaluateRatio()

```
virtual real64 dng_warp_params::EvaluateRatio (
             uint32 plane,
             real64 r2 ) const  [pure virtual]
```

Evaluate the 1D radial warp ratio function for the specified plane. Parameter r2 is the square of the destination (i.e., corrected) normalized radius, i.e., the square of the normalized Euclidean distance between a corrected pixel position and the optical center in the image. r2 must lie in the range [0,1]. Note that this is different than the Evaluate () function, above, in that the argument to EvaluateRatio () is the square of the radius, not the radius itself. The returned result is non-negative. Mathematically, EvaluateRatio (r $*$ r) is the same as Evaluate (r) / r.

Implemented in dng_warp_params_fisheye, and dng_warp_params_rectilinear.

5.188.3.4 EvaluateTangential()

```
virtual dng_point_real64 dng_warp_params::EvaluateTangential (
             uint32 plane,
             real64 r2,
             const dng_point_real64 & diff,
             const dng_point_real64 & diff2 ) const  [pure virtual]
```

Evaluate the 2D tangential warp for the specified plane. Parameter r2 is the square of the destination (i.e., corrected) normalized radius, i.e., the square of the normalized Euclidean distance between a corrected pixel position P and the optical center in the image. r2 must lie in the range [0,1]. diff contains the vertical and horizontal Euclidean distances (in pixels) between P and the optical center. diff2 contains the squares of the vertical and horizontal Euclidean distances (in pixels) between P and the optical center. The returned result is the tangential warp offset, measured in pixels.

Implemented in dng_warp_params_fisheye, and dng_warp_params_rectilinear.

Referenced by EvaluateTangential2(), and EvaluateTangential3().

5.188.3.5 EvaluateTangential2()

```
dng_point_real64 dng_warp_params::EvaluateTangential2 (
             uint32 plane,
             const dng_point_real64 & diff ) const
```

Evaluate the 2D tangential warp for the specified plane. diff contains the vertical and horizontal Euclidean distances (in pixels) between the destination (i.e., corrected) pixel position and the optical center in the image. The returned result is the tangential warp offset, measured in pixels.

References EvaluateTangential().

Referenced by dng_warp_params_rectilinear::MaxSrcTanGap().

5.188.3.6 EvaluateTangential3()

```
dng_point_real64 dng_warp_params::EvaluateTangential3 (
             uint32 plane,
             real64 r2,
             const dng_point_real64 & diff ) const
```

Evaluate the 2D tangential warp for the specified plane. Parameter r2 is the square of the destination (i.e., corrected) normalized radius, i.e., the square of the normalized Euclidean distance between a corrected pixel position P and the optical center in the image. r2 must lie in the range [0,1]. diff contains the vertical and horizontal Euclidean distances (in pixels) between P and the optical center. The returned result is the tangential warp offset, measured in pixels.

References EvaluateTangential().

5.188.3.7 MaxSrcRadiusGap()

```
virtual real64 dng_warp_params::MaxSrcRadiusGap (
             real64 maxDstGap ) const  [pure virtual]
```

Compute and return the maximum warped radius gap. Let D be a rectangle in a destination (corrected) image. Let rDstFar and rDstNear be the farthest and nearest points to the image center, respectively. Then the specified parameter maxDstGap is the Euclidean distance between rDstFar and rDstNear. Warp D through this warp function to a closed and bounded (generally not rectangular) region S. Let rSrcfar and rSrcNear be the farthest and nearest points to the image center, respectively. This routine returns a value that is at least (rSrcFar - rSrcNear).

Implemented in dng_warp_params_fisheye, and dng_warp_params_rectilinear.

Referenced by dng_filter_warp::SrcTileSize().

5.188.3.8 MaxSrcTanGap()

```
virtual dng_point_real64 dng_warp_params::MaxSrcTanGap (
             dng_point_real64 minDst,
             dng_point_real64 maxDst ) const  [pure virtual]
```

Compute and return the maximum warped tangential gap. minDst is the top-left pixel of the image in normalized pixel coordinates. maxDst is the bottom-right pixel of the image in normalized pixel coordinates. MaxSrcTanGap () computes the maximum absolute shift in normalized pixels in the horizontal and vertical directions that can occur as a result of the tangential warp.

Implemented in dng_warp_params_fisheye, and dng_warp_params_rectilinear.

Referenced by dng_filter_warp::SrcTileSize().

5.188.3.9 SafeMaxRatio()

```
virtual real64 dng_warp_params::SafeMaxRatio ( ) const  [pure virtual]
```

Compute and return the maximum src/dst ratio that should be used for this warp.

Implemented in dng_warp_params_fisheye, and dng_warp_params_rectilinear.

5.188.3.10 SafeMinRatio()

```
virtual real64 dng_warp_params::SafeMinRatio ( ) const  [pure virtual]
```

Compute and return the minimum src/dst ratio that should be used for this warp.

Implemented in dng_warp_params_fisheye, and dng_warp_params_rectilinear.

The documentation for this class was generated from the following files:

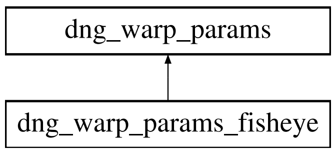
- dng_lens_correction.h
- dng_lens_correction.cpp

5.189 dng_warp_params_fisheye Class Reference

Warp parameters for fisheye camera model (radial component only). Note the restrictions described below.

```
#include <dng_lens_correction.h>
```

Inheritance diagram for dng_warp_params_fisheye:



Public Member Functions

- dng_warp_params_fisheye ()

    Create empty (invalid) fisheye warp parameters.
- dng_warp_params_fisheye (uint32 planes, const dng_vector radParams [ ], const dng_point_real64 &fCenter)
- virtual bool IsRadNOP (uint32 plane) const

    Is the radial correction a NOP for the specified plane?
- virtual bool IsTanNOP (uint32 plane) const

    Is the tangential correction a NOP for the specified plane?
- virtual bool IsValid () const

    Do these warp params appear valid?
- virtual void PropagateToAllPlanes (uint32 totalPlanes)

    Propagate warp parameters from first plane to all other planes.
- virtual real64 Evaluate (uint32 plane, real64 r) const
- virtual real64 EvaluateRatio (uint32 plane, real64 r2) const
- virtual dng_point_real64 EvaluateTangential (uint32 plane, real64 r2, const dng_point_real64 &diff, const dng_point_real64 &diff2) const
- virtual real64 MaxSrcRadiusGap (real64 maxDstGap) const
- virtual dng_point_real64 MaxSrcTanGap (dng_point_real64 minDst, dng_point_real64 maxDst) const
- virtual real64 SafeMinRatio () const
- virtual real64 SafeMaxRatio () const
- virtual void Dump () const

    Debug parameters.

Public Attributes

- dng_vector **fRadParams** [kMaxColorPlanes]

5.189.1 Detailed Description

Warp parameters for fisheye camera model (radial component only). Note the restrictions described below.

5.189.2 Constructor & Destructor Documentation

5.189.2.1 dng_warp_params_fisheye()

```
dng_warp_params_fisheye::dng_warp_params_fisheye (
             uint32 planes,
             const dng_vector radParams[ ],
             const dng_point_real64 & fCenter )
```

Create rectilinear warp parameters with the specified number of planes, radial component terms, and image center in relative coordinates.

5.189.3 Member Function Documentation

5.189.3.1 Evaluate()

```
real64 dng_warp_params_fisheye::Evaluate (
             uint32 plane,
             real64 r ) const  [virtual]
```

Evaluate the 1D radial warp function for the specified plane. Parameter r is the destination (i.e., corrected) normalized radius, i.e., the normalized Euclidean distance between a corrected pixel position and the optical center in the image. r lies in the range [0,1]. The returned result is non-negative.

Implements dng_warp_params.

Referenced by EvaluateRatio(), and MaxSrcRadiusGap().

5.189.3.2 EvaluateRatio()

```
real64 dng_warp_params_fisheye::EvaluateRatio (
             uint32 plane,
             real64 r2 ) const  [virtual]
```

Evaluate the 1D radial warp ratio function for the specified plane. Parameter r2 is the square of the destination (i.e., corrected) normalized radius, i.e., the square of the normalized Euclidean distance between a corrected pixel position and the optical center in the image. r2 must lie in the range [0,1]. Note that this is different than the Evaluate () function, above, in that the argument to EvaluateRatio () is the square of the radius, not the radius itself. The returned result is non-negative. Mathematically, EvaluateRatio (r $*$ r) is the same as Evaluate (r) / r.

Implements dng_warp_params.

References Evaluate().

5.189.3.3 EvaluateTangential()

```
dng_point_real64 dng_warp_params_fisheye::EvaluateTangential (
             uint32 plane,
             real64 r2,
             const dng_point_real64 & diff,
             const dng_point_real64 & diff2 ) const  [virtual]
```

Evaluate the 2D tangential warp for the specified plane. Parameter r2 is the square of the destination (i.e., corrected) normalized radius, i.e., the square of the normalized Euclidean distance between a corrected pixel position P and the optical center in the image. r2 must lie in the range [0,1]. diff contains the vertical and horizontal Euclidean distances (in pixels) between P and the optical center. diff2 contains the squares of the vertical and horizontal Euclidean distances (in pixels) between P and the optical center. The returned result is the tangential warp offset, measured in pixels.

Implements dng_warp_params.

References ThrowProgramError().

5.189.3.4 MaxSrcRadiusGap()

```
real64 dng_warp_params_fisheye::MaxSrcRadiusGap (
            real64 maxDstGap ) const  [virtual]
```

Compute and return the maximum warped radius gap. Let D be a rectangle in a destination (corrected) image. Let rDstFar and rDstNear be the farthest and nearest points to the image center, respectively. Then the specified parameter maxDstGap is the Euclidean distance between rDstFar and rDstNear. Warp D through this warp function to a closed and bounded (generally not rectangular) region S. Let rSrcfar and rSrcNear be the farthest and nearest points to the image center, respectively. This routine returns a value that is at least (rSrcFar - rSrcNear).

Implements dng_warp_params.

References DNG_REQUIRE, and Evaluate().

5.189.3.5 MaxSrcTanGap()

```
dng_point_real64 dng_warp_params_fisheye::MaxSrcTanGap (
            dng_point_real64 minDst,
            dng_point_real64 maxDst ) const  [virtual]
```

Compute and return the maximum warped tangential gap. minDst is the top-left pixel of the image in normalized pixel coordinates. maxDst is the bottom-right pixel of the image in normalized pixel coordinates. MaxSrcTanGap () computes the maximum absolute shift in normalized pixels in the horizontal and vertical directions that can occur as a result of the tangential warp.

Implements dng_warp_params.

5.189.3.6 SafeMaxRatio()

```
real64 dng_warp_params_fisheye::SafeMaxRatio ( ) const  [virtual]
```

Compute and return the maximum src/dst ratio that should be used for this warp.

Implements dng_warp_params.

5.189.3.7 SafeMinRatio()

```
real64 dng_warp_params_fisheye::SafeMinRatio ( ) const  [virtual]
```

Compute and return the minimum src/dst ratio that should be used for this warp.

Implements dng_warp_params.

The documentation for this class was generated from the following files:

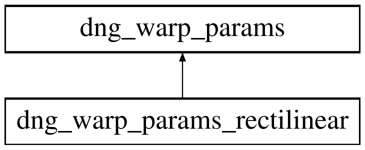
- dng_lens_correction.h
- dng_lens_correction.cpp

5.190 dng_warp_params_rectilinear Class Reference

Warp parameters for pinhole perspective rectilinear (not fisheye) camera model. Supports radial and tangential (decentering) distortion correction parameters.

```
#include <dng_lens_correction.h>
```

Inheritance diagram for dng_warp_params_rectilinear:



Public Member Functions

- dng_warp_params_rectilinear ()

  Create empty (invalid) rectilinear warp parameters.
- dng_warp_params_rectilinear (uint32 planes, const dng_vector radParams [ ], const dng_vector tanParams [ ], const dng_point_real64 &fCenter)
- virtual bool IsRadNOP (uint32 plane) const

  Is the radial correction a NOP for the specified plane?
- virtual bool IsTanNOP (uint32 plane) const

  Is the tangential correction a NOP for the specified plane?
- virtual bool IsValid () const

  Do these warp params appear valid?
- virtual void PropagateToAllPlanes (uint32 totalPlanes)

  Propagate warp parameters from first plane to all other planes.
- virtual real64 Evaluate (uint32 plane, real64 r) const
- virtual real64 EvaluateRatio (uint32 plane, real64 r2) const
- virtual dng_point_real64 EvaluateTangential (uint32 plane, real64 r2, const dng_point_real64 &diff, const dng_point_real64 &diff2) const
- virtual real64 MaxSrcRadiusGap (real64 maxDstGap) const
- virtual dng_point_real64 MaxSrcTanGap (dng_point_real64 minDst, dng_point_real64 maxDst) const
- virtual real64 SafeMinRatio () const
- virtual real64 SafeMaxRatio () const
- virtual void Dump () const

  Debug parameters.

Public Attributes

- dng_vector **fRadParams** [kMaxColorPlanes]
- dng_vector **fTanParams** [kMaxColorPlanes]

5.190.1 Detailed Description

Warp parameters for pinhole perspective rectilinear (not fisheye) camera model. Supports radial and tangential (decentering) distortion correction parameters.

Note the restrictions described below.

5.190.2 Constructor & Destructor Documentation

5.190.2.1 dng_warp_params_rectilinear()

```
dng_warp_params_rectilinear::dng_warp_params_rectilinear (
             uint32 planes,
             const dng_vector radParams[],
             const dng_vector tanParams[],
             const dng_point_real64 & fCenter )
```

Create rectilinear warp parameters with the specified number of planes, radial component terms, tangential component terms, and image center in relative coordinates.

5.190.3 Member Function Documentation

5.190.3.1 Evaluate()

```
real64 dng_warp_params_rectilinear::Evaluate (
             uint32 plane,
             real64 r ) const  [virtual]
```

Evaluate the 1D radial warp function for the specified plane. Parameter r is the destination (i.e., corrected) normalized radius, i.e., the normalized Euclidean distance between a corrected pixel position and the optical center in the image. r lies in the range [0,1]. The returned result is non-negative.

Implements dng_warp_params.

Referenced by MaxSrcRadiusGap().

5.190.3.2 EvaluateRatio()

```
real64 dng_warp_params_rectilinear::EvaluateRatio (
            uint32 plane,
            real64 r2 ) const  [virtual]
```

Evaluate the 1D radial warp ratio function for the specified plane. Parameter r2 is the square of the destination (i.e., corrected) normalized radius, i.e., the square of the normalized Euclidean distance between a corrected pixel position and the optical center in the image. r2 must lie in the range [0,1]. Note that this is different than the Evaluate () function, above, in that the argument to EvaluateRatio () is the square of the radius, not the radius itself. The returned result is non-negative. Mathematically, EvaluateRatio (r $*$ r) is the same as Evaluate (r) / r.

Implements dng_warp_params.

5.190.3.3 EvaluateTangential()

```
dng_point_real64 dng_warp_params_rectilinear::EvaluateTangential (
            uint32 plane,
            real64 r2,
            const dng_point_real64 & diff,
            const dng_point_real64 & diff2 ) const  [virtual]
```

Evaluate the 2D tangential warp for the specified plane. Parameter r2 is the square of the destination (i.e., corrected) normalized radius, i.e., the square of the normalized Euclidean distance between a corrected pixel position P and the optical center in the image. r2 must lie in the range [0,1]. diff contains the vertical and horizontal Euclidean distances (in pixels) between P and the optical center. diff2 contains the squares of the vertical and horizontal Euclidean distances (in pixels) between P and the optical center. The returned result is the tangential warp offset, measured in pixels.

Implements dng_warp_params.

5.190.3.4 MaxSrcRadiusGap()

```
real64 dng_warp_params_rectilinear::MaxSrcRadiusGap (
            real64 maxDstGap ) const  [virtual]
```

Compute and return the maximum warped radius gap. Let D be a rectangle in a destination (corrected) image. Let rDstFar and rDstNear be the farthest and nearest points to the image center, respectively. Then the specified parameter maxDstGap is the Euclidean distance between rDstFar and rDstNear. Warp D through this warp function to a closed and bounded (generally not rectangular) region S. Let rSrcfar and rSrcNear be the farthest and nearest points to the image center, respectively. This routine returns a value that is at least (rSrcFar - rSrcNear).

Implements dng_warp_params.

References Evaluate().

5.190.3.5 MaxSrcTanGap()

```
dng_point_real64 dng_warp_params_rectilinear::MaxSrcTanGap (
            dng_point_real64 minDst,
            dng_point_real64 maxDst ) const  [virtual]
```

Compute and return the maximum warped tangential gap. minDst is the top-left pixel of the image in normalized pixel coordinates. maxDst is the bottom-right pixel of the image in normalized pixel coordinates. MaxSrcTanGap () computes the maximum absolute shift in normalized pixels in the horizontal and vertical directions that can occur as a result of the tangential warp.

Implements dng_warp_params.

References dng_warp_params::EvaluateTangential2().

5.190.3.6 SafeMaxRatio()

```
real64 dng_warp_params_rectilinear::SafeMaxRatio ( ) const  [virtual]
```

Compute and return the maximum src/dst ratio that should be used for this warp.

Implements dng_warp_params.

5.190.3.7 SafeMinRatio()

```
real64 dng_warp_params_rectilinear::SafeMinRatio ( ) const  [virtual]
```

Compute and return the minimum src/dst ratio that should be used for this warp.

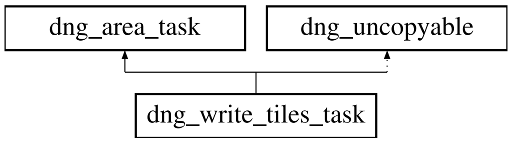
Implements dng_warp_params.

The documentation for this class was generated from the following files:

- dng_lens_correction.h
- dng_lens_correction.cpp

5.191 dng_write_tiles_task Class Reference

Inheritance diagram for dng_write_tiles_task:



Public Member Functions

- **dng_write_tiles_task** (dng_image_writer &imageWriter, dng_host &host, const dng_ifd &ifd, dng_basic_tag_set &basic, dng_stream &stream, const dng_image &image, uint32 fakeChannels, uint32 tilesDown, uint32 tiles↩Across, uint32 compressedSize, uint32 uncompressedSize)
- void Process (uint32 threadIndex, const dng_rect &tile, dng_abort_sniffer ∗sniffer)

  and progress updates.

Protected Member Functions

- void **ProcessTask** (uint32 tileIndex, AutoPtr< dng_memory_block > &compressedBuffer, AutoPtr< dng_memory_block > &uncompressedBuffer, AutoPtr< dng_memory_block > &subTileBlockBuffer, AutoPtr< dng_memory_block > &tempBuffer, uint32 &tileByteCount, dng_memory_stream &tileStream, dng_abort_sniffer ∗sniffer)
- void **WriteTask** (uint32 tileIndex, uint32 tileByteCount, dng_memory_stream &tileStream, dng_abort_sniffer ∗sniffer)

Protected Attributes

- dng_image_writer & **fImageWriter**
- dng_host & **fHost**
- const dng_ifd & **fIFD**
- dng_basic_tag_set & **fBasic**
- dng_stream & **fStream**
- const dng_image & **fImage**
- uint32 **fFakeChannels**
- uint32 **fTilesDown**
- uint32 **fTilesAcross**
- uint32 **fCompressedSize**
- uint32 **fUncompressedSize**
- std::atomic_uint **fNextTileIndex**
- dng_mutex **fMutex**
- dng_condition **fCondition**
- bool **fTaskFailed**
- uint32 **fWriteTileIndex**

Additional Inherited Members

5.191.1 Member Function Documentation

5.191.1.1 Process()

```
void dng_write_tiles_task::Process (
            uint32 threadIndex,
            const dng_rect & tile,
            dng_abort_sniffer * sniffer )  [virtual]
```

and progress updates.

Process one tile or fully partitioned area. This method is overridden by derived classes to implement the actual image processing. Note that the sniffer can be ignored if it is certain that a processing task will complete very quickly. This method should never be called directly but rather accessed via Process. There is no allocator parameter as all allocation should be done in Start.

Parameters

threadIndex	0 to threadCount - 1 index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the Start method.)
tile	Area to process.
sniffer	dng_abort_sniffer to use to check for user cancellation

Implements dng_area_task.

References dng_host::Allocate(), dng_host::Allocator(), and AutoPtr< T >::Reset().

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.192 dng_xmp Class Reference

Public Member Functions

- **dng_xmp** (dng_memory_allocator &allocator)
- **dng_xmp** (const dng_xmp &xmp)
- virtual dng_xmp ∗ **Clone** () const
- dng_memory_allocator & **Allocator** () const
- void **Parse** (dng_host &host, const void ∗buffer, uint32 count)
- dng_memory_block ∗ **Serialize** (bool asPacket=false, uint32 targetBytes=0, uint32 padBytes=4096, bool forJ↩PEG=false, bool compact=true) const
- dng_memory_block ∗ **SerializeNonCompact** () const
- void **PackageForJPEG** (AutoPtr< dng_memory_block > &stdBlock, AutoPtr< dng_memory_block > &extBlock, dng_string &extDigest) const
- void **MergeFromJPEG** (const dng_xmp &xmp)
- bool **HasMeta** () const
- void **RequireMeta** ()
- void ∗ **GetPrivateMeta** ()
- bool **Exists** (const char ∗ns, const char ∗path) const
- bool **HasNameSpace** (const char ∗ns) const
- bool **IteratePaths** (IteratePathsCallback ∗callback, void ∗callbackData, const char ∗ns=0, const char ∗path=0)
- void **Remove** (const char ∗ns, const char ∗path)
- void **RemoveProperties** (const char ∗ns)
- void **RemoveEmptyStringOrArray** (const char ∗ns, const char ∗path)
- void **RemoveEmptyStringsAndArrays** (const char ∗ns=0)
- void **Set** (const char ∗ns, const char ∗path, const char ∗text)
- bool **GetString** (const char ∗ns, const char ∗path, dng_string &s) const
- void **SetString** (const char ∗ns, const char ∗path, const dng_string &s)
- bool **GetStringList** (const char ∗ns, const char ∗path, dng_string_list &list) const
- void **SetStringList** (const char ∗ns, const char ∗path, const dng_string_list &list, bool isBag=false)
- void **SetStructField** (const char ∗ns, const char ∗path, const char ∗fieldNS, const char ∗fieldName, const dng_string &s)

- void **SetStructField** (const char ∗ns, const char ∗path, const char ∗fieldNS, const char ∗fieldName, const char ∗s)
- void **DeleteStructField** (const char ∗ns, const char ∗path, const char ∗fieldNS, const char ∗fieldName)
- bool **GetStructField** (const char ∗ns, const char ∗path, const char ∗fieldNS, const char ∗fieldName, dng_string &s) const
- void **SetAltLangDefault** (const char ∗ns, const char ∗path, const dng_string &s)
- void **SetLocalString** (const char ∗ns, const char ∗path, const dng_local_string &s)
- bool **GetAltLangDefault** (const char ∗ns, const char ∗path, dng_string &s, bool silent=false) const
- bool **GetLocalString** (const char ∗ns, const char ∗path, dng_local_string &s) const
- bool **GetBoolean** (const char ∗ns, const char ∗path, bool &x) const
- void **SetBoolean** (const char ∗ns, const char ∗path, bool x)
- bool **Get_int32** (const char ∗ns, const char ∗path, int32 &x) const
- void **Set_int32** (const char ∗ns, const char ∗path, int32 x, bool usePlus=false)
- bool **Get_uint32** (const char ∗ns, const char ∗path, uint32 &x) const
- void **Set_uint32** (const char ∗ns, const char ∗path, uint32 x)
- bool **Get_real64** (const char ∗ns, const char ∗path, real64 &x) const
- void **Set_real64** (const char ∗ns, const char ∗path, real64 x, uint32 places=6, bool trim=true, bool usePlus=false)
- bool **Get_urational** (const char ∗ns, const char ∗path, dng_urational &r) const
- void **Set_urational** (const char ∗ns, const char ∗path, const dng_urational &r)
- bool **Get_srational** (const char ∗ns, const char ∗path, dng_srational &r) const
- void **Set_srational** (const char ∗ns, const char ∗path, const dng_srational &r)
- bool **GetFingerprint** (const char ∗ns, const char ∗path, dng_fingerprint &print) const
- void **SetFingerprint** (const char ∗ns, const char ∗path, const dng_fingerprint &print, bool allowInvalid=false)
- void **SetVersion2to4** (const char ∗ns, const char ∗path, uint32 version)
- dng_fingerprint **GetIPTCDigest** () const
- void **SetIPTCDigest** (dng_fingerprint &digest)
- void **ClearIPTCDigest** ()
- void **IngestIPTC** (dng_metadata &metadata, bool xmpIsNewer=false)
- void **RebuildIPTC** (dng_metadata &metadata, dng_memory_allocator &allocator, bool padForTIFF)
- virtual void **SyncExif** (dng_exif &exif, const dng_exif ∗originalExif=NULL, bool doingUpdateFromXMP=false, bool removeFromXMP=false)
- void **ValidateStringList** (const char ∗ns, const char ∗path)
- void **ValidateMetadata** ()
- void **UpdateDateTime** (const dng_date_time_info &dt)
- void **UpdateMetadataDate** (const dng_date_time_info &dt)
- void **UpdateExifDates** (dng_exif &exif, bool removeFromXMP=false)
- bool **HasOrientation** () const
- dng_orientation **GetOrientation** () const
- void **ClearOrientation** ()
- void **SetOrientation** (const dng_orientation &orientation)
- void **SyncOrientation** (dng_negative &negative, bool xmpIsMaster)
- void **SyncOrientation** (dng_metadata &metadata, bool xmpIsMaster)
- void **ClearImageInfo** ()
- void **SetImageSize** (const dng_point &size)
- void **SetSampleInfo** (uint32 samplesPerPixel, uint32 bitsPerSample)
- void **SetPhotometricInterpretation** (uint32 pi)
- void **SetResolution** (const dng_resolution &res)
- void **ComposeArrayItemPath** (const char ∗ns, const char ∗arrayName, int32 itemNumber, dng_string &s) const
- void **ComposeStructFieldPath** (const char ∗ns, const char ∗structName, const char ∗fieldNS, const char ∗field↩ Name, dng_string &s) const
- int32 **CountArrayItems** (const char ∗ns, const char ∗path) const

- void **AppendArrayItem** (const char ∗ns, const char ∗arrayName, const char ∗itemValue, bool isBag=true, bool propIsStruct=false)
- void **DocOpsOpenXMP** (const char ∗srcMIME)
- void **DocOpsPrepareForSave** (const char ∗srcMIME, const char ∗dstMIME, bool newPath=true)
- void **DocOpsUpdateMetadata** (const char ∗srcMIME)

Static Public Member Functions

- static dng_string **EncodeFingerprint** (const dng_fingerprint &f, bool allowInvalid=false)
- static dng_fingerprint **DecodeFingerprint** (const dng_string &s)

Protected Types

- enum { **ignoreXMP** = 1, **preferXMP** = 2, **preferNonXMP** = 4, **removeXMP** = 8 }

Protected Member Functions

- bool **SyncString** (const char ∗ns, const char ∗path, dng_string &s, uint32 options=0)
- void **SyncStringList** (const char ∗ns, const char ∗path, dng_string_list &list, bool isBag=false, uint32 options=0)
- bool **SyncAltLangDefault** (const char ∗ns, const char ∗path, dng_string &s, uint32 options=0)
- void **Sync_uint32** (const char ∗ns, const char ∗path, uint32 &x, bool isDefault=false, uint32 options=0)
- void **Sync_uint32_array** (const char ∗ns, const char ∗path, uint32 ∗data, uint32 &count, uint32 maxCount, uint32 options=0)
- void **Sync_urational** (const char ∗ns, const char ∗path, dng_urational &r, uint32 options=0)
- void **Sync_srational** (const char ∗ns, const char ∗path, dng_srational &r, uint32 options=0)
- void **SyncIPTC** (dng_iptc &iptc, uint32 options)
- void **SyncFlash** (uint32 &flashState, uint32 &flashMask, uint32 options)
- void **SyncExifDate** (const char ∗ns, const char ∗path, dng_date_time_info &exifDateTime, bool canRemove↩ FromXMP, bool removeFromXMP, const dng_time_zone &fakeTimeZone)
- virtual void **SyncApproximateFocusDistance** (dng_exif &exif, const uint32 readOnly)
- virtual void **SyncLensName** (dng_exif &exif)
- virtual void **GenerateDefaultLensName** (dng_exif &exif)

Static Protected Member Functions

- static void **TrimDecimal** (char ∗s)
- static dng_string **EncodeGPSVersion** (uint32 version)
- static uint32 **DecodeGPSVersion** (const dng_string &s)
- static dng_string **EncodeGPSCoordinate** (const dng_string &ref, const dng_urational ∗coord)
- static void **DecodeGPSCoordinate** (const dng_string &s, dng_string &ref, dng_urational ∗coord)
- static dng_string **EncodeGPSDateTime** (const dng_string &dateStamp, const dng_urational ∗timeStamp)
- static void **DecodeGPSDateTime** (const dng_string &s, dng_string &dateStamp, dng_urational ∗timeStamp)

Protected Attributes

- dng_memory_allocator & **fAllocator**
- dng_xmp_sdk ∗ **fSDK**

The documentation for this class was generated from the following files:

- dng_xmp.h
- dng_xmp.cpp

5.193 dng_xmp_namespace Struct Reference

Public Attributes

- const char ∗ **fullName**
- const char ∗ **shortName**

The documentation for this struct was generated from the following file:

- dng_xmp_sdk.h

5.194 dng_xmp_private Class Reference

Public Member Functions

- **dng_xmp_private** (const dng_xmp_private &xmp)

Public Attributes

- SXMPMeta ∗ **fMeta**

The documentation for this class was generated from the following file:

- dng_xmp_sdk.cpp

5.195 dng_xmp_sdk Class Reference

Public Member Functions

- **dng_xmp_sdk** (const dng_xmp_sdk &sdk)
- bool **HasMeta** () const
- void **RequireMeta** ()
- void ∗ **GetPrivateMeta** ()
- void **Parse** (dng_host &host, const char ∗buffer, uint32 count)
- bool **Exists** (const char ∗ns, const char ∗path) const
- void **AppendArrayItem** (const char ∗ns, const char ∗arrayName, const char ∗itemValue, bool isBag=true, bool propIsStruct=false)
- int32 **CountArrayItems** (const char ∗ns, const char ∗path) const
- bool **HasNameSpace** (const char ∗ns) const
- void **Remove** (const char ∗ns, const char ∗path)
- void **RemoveProperties** (const char ∗ns)
- bool **IsEmptyString** (const char ∗ns, const char ∗path)
- bool **IsEmptyArray** (const char ∗ns, const char ∗path)
- void **ComposeArrayItemPath** (const char ∗ns, const char ∗arrayName, int32 itemNumber, dng_string &s) const
- void **ComposeStructFieldPath** (const char ∗ns, const char ∗structName, const char ∗fieldNS, const char ∗field↩
Name, dng_string &s) const
- bool **GetNamespacePrefix** (const char ∗uri, dng_string &s) const
- bool **GetString** (const char ∗ns, const char ∗path, dng_string &s) const
- void **ValidateStringList** (const char ∗ns, const char ∗path)
- bool **GetStringList** (const char ∗ns, const char ∗path, dng_string_list &list) const
- bool **GetAltLangDefault** (const char ∗ns, const char ∗path, dng_string &s, bool silent=false) const
- bool **GetLocalString** (const char ∗ns, const char ∗path, dng_local_string &s) const
- bool **GetStructField** (const char ∗ns, const char ∗path, const char ∗fieldNS, const char ∗fieldName, dng_string &s) const
- void **Set** (const char ∗ns, const char ∗path, const char ∗text)
- void **SetString** (const char ∗ns, const char ∗path, const dng_string &s)
- void **SetStringList** (const char ∗ns, const char ∗path, const dng_string_list &list, bool isBag)
- void **SetAltLangDefault** (const char ∗ns, const char ∗path, const dng_string &s)
- void **SetLocalString** (const char ∗ns, const char ∗path, const dng_local_string &s)
- void **SetStructField** (const char ∗ns, const char ∗path, const char ∗fieldNS, const char ∗fieldName, const char ∗text)
- void **DeleteStructField** (const char ∗ns, const char ∗structName, const char ∗fieldNS, const char ∗fieldName)
- dng_memory_block ∗ **Serialize** (dng_memory_allocator &allocator, bool asPacket, uint32 targetBytes, uint32 padBytes, bool forJPEG, bool compact) const
- void **PackageForJPEG** (dng_memory_allocator &allocator, AutoPtr< dng_memory_block > &stdBlock, AutoPtr< dng_memory_block > &extBlock, dng_string &extDigest) const
- void **MergeFromJPEG** (const dng_xmp_sdk ∗xmp)
- void **ReplaceXMP** (dng_xmp_sdk ∗xmp)
- bool **IteratePaths** (IteratePathsCallback ∗callback, void ∗callbackData=NULL, const char ∗startNS=0, const char ∗startingPath=0)
- void **DocOpsOpenXMP** (const char ∗srcMIME)
- void **DocOpsPrepareForSave** (const char ∗srcMIME, const char ∗dstMIME, bool newPath=true)
- void **DocOpsUpdateMetadata** (const char ∗srcMIME)

Static Public Member Functions

- static void **InitializeSDK** (dng_xmp_namespace ∗extraNamespaces=NULL, const char ∗software=NULL)
- static void **TerminateSDK** ()

The documentation for this class was generated from the following files:

- dng_xmp_sdk.h
- dng_xmp_sdk.cpp

5.196 dng_xy_coord Class Reference

Public Member Functions

- **dng_xy_coord** (real64 xx, real64 yy)
- void **Clear** ()
- bool **IsValid** () const
- bool **NotValid** () const
- bool **operator==** (const dng_xy_coord &coord) const
- bool **operator!=** (const dng_xy_coord &coord) const

Public Attributes

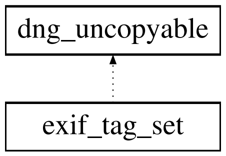
- real64 **x**
- real64 **y**

The documentation for this class was generated from the following file:

- dng_xy_coord.h

5.197 exif_tag_set Class Reference

Inheritance diagram for exif_tag_set:



Public Member Functions

- **exif_tag_set** (dng_tiff_directory &directory, const dng_exif &exif, bool makerNoteSafe=false, const void ∗makerNoteData=NULL, uint32 makerNoteLength=0, bool insideDNG=false)
- void **Locate** (uint32 offset)
- uint32 **Size** () const
- void **Put** (dng_stream &stream) const

Protected Member Functions

- void **AddLinks** (dng_tiff_directory &directory)

Protected Attributes

- dng_tiff_directory **fExifIFD**
- dng_tiff_directory **fGPSIFD**

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.198 dng_hue_sat_map::HSBModify Struct Reference

```
#include <dng_hue_sat_map.h>
```

Public Attributes

- real32 **fHueShift**
- real32 **fSatScale**
- real32 **fValScale**

5.198.1 Detailed Description

HSV delta signal. fHueShift is a delta value specified in degrees. This parameter, added to the original hue, determines the output hue. A value of 0 means no change. fSatScale and fValScale are scale factors that are applied to saturation and value components, respectively. These scale factors, multiplied by the original saturation and value, determine the output saturation and value. A scale factor of 1.0 means no change.

The documentation for this struct was generated from the following file:

- dng_hue_sat_map.h

5.199 HuffmanTable Struct Reference

Public Attributes

- uint8 **bits** [17]
- uint8 **huffval** [256]
- uint16 **mincode** [17]
- int32 **maxcode** [18]
- int16 **valptr** [17]
- int32 **numbits** [256]
- int32 **value** [256]
- uint16 **ehufco** [256]
- int8 **ehufsi** [256]

The documentation for this struct was generated from the following file:

- dng_lossless_jpeg.cpp

5.200 JpegComponentInfo Struct Reference

Public Attributes

- int16 **componentId**
- int16 **componentIndex**
- int16 **hSampFactor**
- int16 **vSampFactor**
- int16 **dcTblNo**

The documentation for this struct was generated from the following file:

- dng_lossless_jpeg.cpp

5.201 mosaic_tag_set Class Reference

Public Member Functions

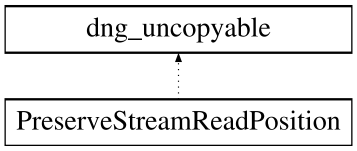
- **mosaic_tag_set** (dng_tiff_directory &directory, const dng_mosaic_info &info)

The documentation for this class was generated from the following file:

- dng_image_writer.cpp

5.202 PreserveStreamReadPosition Class Reference

Inheritance diagram for PreserveStreamReadPosition:



Public Member Functions

- **PreserveStreamReadPosition** (dng_stream &stream)

The documentation for this class was generated from the following file:

- dng_stream.h

5.203 profile_tag_set Class Reference

Public Member Functions

- **profile_tag_set** (dng_tiff_directory &directory, const dng_camera_profile &profile)

The documentation for this class was generated from the following file:

- dng_image_writer.cpp

5.204 range_tag_set Class Reference

Public Member Functions

- **range_tag_set** (dng_tiff_directory &directory, const dng_negative &negative)

The documentation for this class was generated from the following file:

- dng_image_writer.cpp

5.205 ruvt Struct Reference

Public Attributes

- real64 **r**
- real64 **u**
- real64 **v**
- real64 **t**

The documentation for this struct was generated from the following file:

- dng_temperature.cpp

5.206 SIMDTraits< SIMDType > Class Template Reference

Static Public Attributes

- static const int **kVecSizeFloat** = 1
- static const int **kVecSizeInt32** = 1

The documentation for this class was generated from the following file:

- dng_simd_type.h

5.207 SIMDTraits< AVX > Class Template Reference

Static Public Attributes

- static const int **kVecSizeFloat** = 8
- static const int **kVecSizeInt32** = 4

The documentation for this class was generated from the following file:

- dng_simd_type.h

5.208 SIMDTraits< AVX2 > Class Template Reference

Static Public Attributes

- static const int **kVecSizeFloat** = 8
- static const int **kVecSizeInt32** = 8

The documentation for this class was generated from the following file:

- dng_simd_type.h

5.209 SIMDTraits< AVX512_SKX > Class Template Reference

Static Public Attributes

- static const int **kVecSizeFloat** = 16
- static const int **kVecSizeInt32** = 16

The documentation for this class was generated from the following file:

- dng_simd_type.h

5.210 SIMDTraits< SSE2 > Class Template Reference

Static Public Attributes

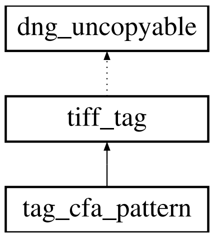
- static const int **kVecSizeFloat** = 4
- static const int **kVecSizeInt32** = 4

The documentation for this class was generated from the following file:

- dng_simd_type.h

5.211 tag_cfa_pattern Class Reference

Inheritance diagram for tag_cfa_pattern:



Public Member Functions

- **tag_cfa_pattern** (uint16 code, uint32 rows, uint32 cols, const uint8 ∗pattern)
- virtual void **Put** (dng_stream &stream) const

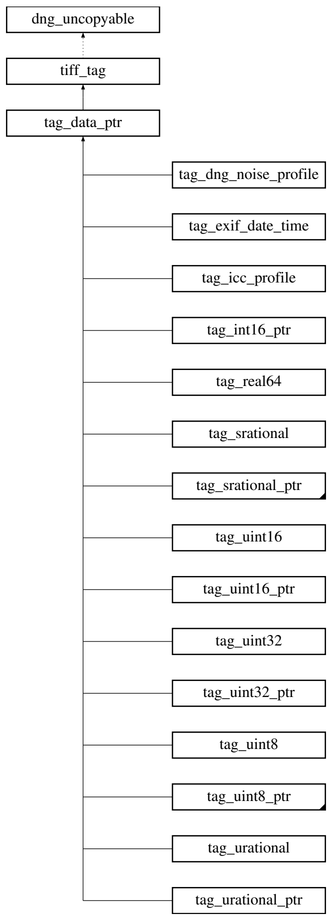
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.212 tag_data_ptr Class Reference

Inheritance diagram for tag_data_ptr:



Public Member Functions

- **tag_data_ptr** (uint16 code, uint16 type, uint32 count, const void ∗data)
- void **SetData** (const void ∗data)
- virtual void **Put** (dng_stream &stream) const

Protected Attributes

- const void ∗ **fData**

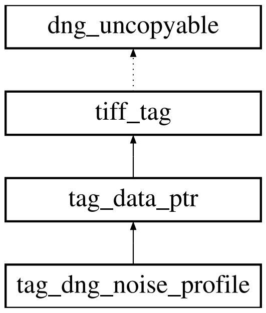
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.213 tag_dng_noise_profile Class Reference

Inheritance diagram for tag_dng_noise_profile:



Public Member Functions

- **tag_dng_noise_profile** (const dng_noise_profile &profile)

Protected Attributes

- real64 **fValues** [2 ∗kMaxColorPlanes]

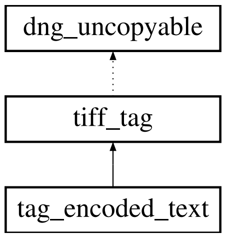
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.214 tag_encoded_text Class Reference

Inheritance diagram for tag_encoded_text:



Public Member Functions

- **tag_encoded_text** (uint16 code, const dng_string &text)
- virtual void **Put** (dng_stream &stream) const

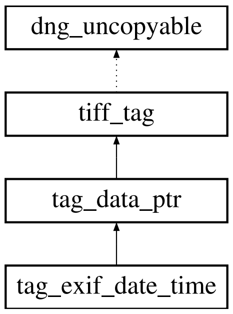
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.215 tag_exif_date_time Class Reference

Inheritance diagram for tag_exif_date_time:



Public Member Functions

- **tag_exif_date_time** (uint16 code, const dng_date_time &dt)

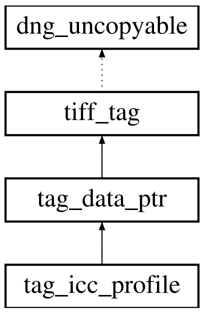
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.216 tag_icc_profile Class Reference

Inheritance diagram for tag_icc_profile:



Public Member Functions

- **tag_icc_profile** (const void ∗profileData, uint32 profileSize)

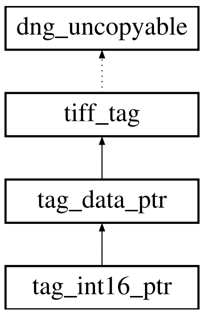
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.217 tag_int16_ptr Class Reference

Inheritance diagram for tag_int16_ptr:



Public Member Functions

- **tag_int16_ptr** (uint16 code, const int16 ∗data, uint32 count=1)

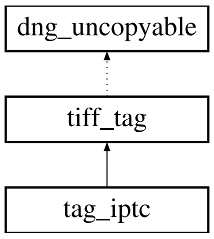
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.218 tag_iptc Class Reference

Inheritance diagram for tag_iptc:



Public Member Functions

- **tag_iptc** (const void ∗data, uint32 length)
- virtual void **Put** (dng_stream &stream) const

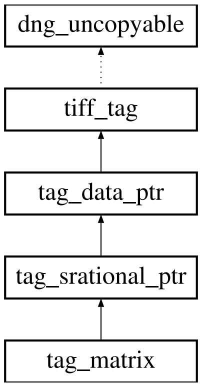
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.219 tag_matrix Class Reference

Inheritance diagram for tag_matrix:



Public Member Functions

- **tag_matrix** (uint16 code, const dng_matrix &m)

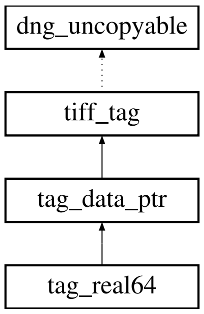
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.220 tag_real64 Class Reference

Inheritance diagram for tag_real64:



Public Member Functions

- **tag_real64** (uint16 code, real64 value=0.0)
- void **Set** (real64 value)

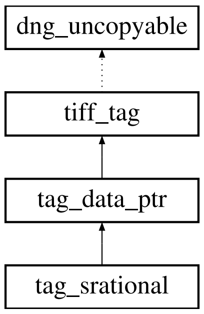
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.221 tag_srational Class Reference

Inheritance diagram for tag_srational:



Public Member Functions

- **tag_srational** (uint16 code, const dng_srational &value)

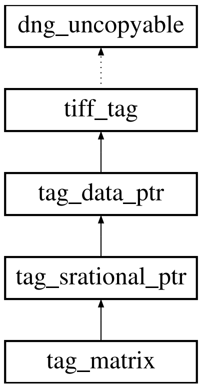
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.222 tag_srational_ptr Class Reference

Inheritance diagram for tag_srational_ptr:



Public Member Functions

- **tag_srational_ptr** (uint16 code, const dng_srational ∗data=NULL, uint32 count=1)

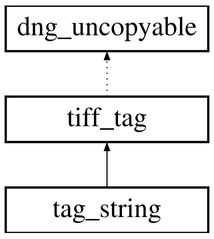
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.223 tag_string Class Reference

Inheritance diagram for tag_string:



Public Member Functions

- **tag_string** (uint16 code, const dng_string &s, bool forceASCII=true)
- virtual void **Put** (dng_stream &stream) const

Protected Attributes

- dng_string **fString**

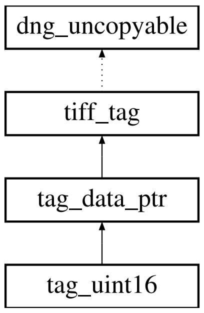
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.224 tag_uint16 Class Reference

Inheritance diagram for tag_uint16:



Public Member Functions

- **tag_uint16** (uint16 code, uint16 value=0)
- void **Set** (uint16 value)

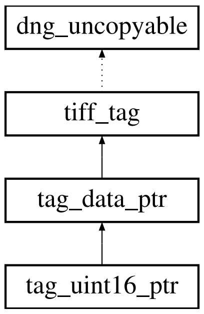
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.225 tag_uint16_ptr Class Reference

Inheritance diagram for tag_uint16_ptr:



Public Member Functions

- **tag_uint16_ptr** (uint16 code, const uint16 ∗data, uint32 count=1)

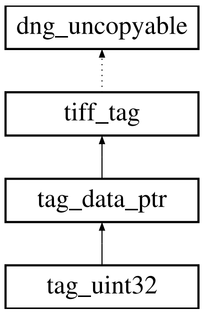
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.226 tag_uint32 Class Reference

Inheritance diagram for tag_uint32:



Public Member Functions

- **tag_uint32** (uint16 code, uint32 value=0)
- void **Set** (uint32 value)

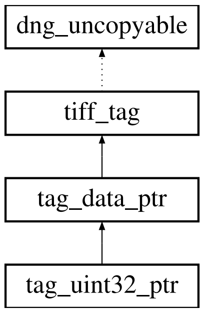
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.227 tag_uint32_ptr Class Reference

Inheritance diagram for tag_uint32_ptr:



Public Member Functions

- **tag_uint32_ptr** (uint16 code, const uint32 ∗data, uint32 count=1)

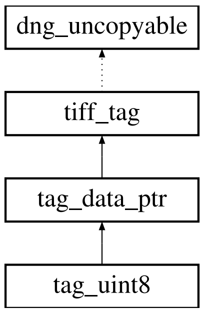
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.228 tag_uint8 Class Reference

Inheritance diagram for tag_uint8:



Public Member Functions

- **tag_uint8** (uint16 code, uint8 value=0)
- void **Set** (uint8 value)

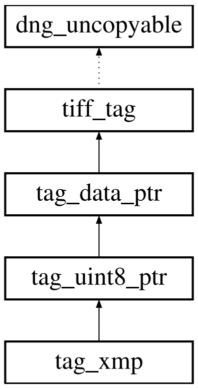
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.229 tag_uint8_ptr Class Reference

Inheritance diagram for tag_uint8_ptr:



Public Member Functions

- **tag_uint8_ptr** (uint16 code, const uint8 ∗data, uint32 count=1)

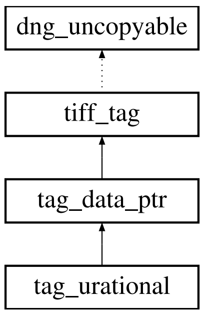
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.230 tag_urational Class Reference

Inheritance diagram for tag_urational:



Public Member Functions

- **tag_urational** (uint16 code, const dng_urational &value)

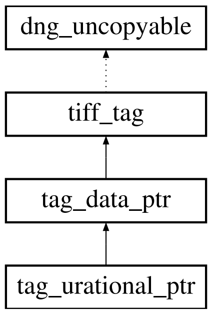
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.231 tag_urational_ptr Class Reference

Inheritance diagram for tag_urational_ptr:



Public Member Functions

- **tag_urational_ptr** (uint16 code, const dng_urational ∗data=NULL, uint32 count=1)

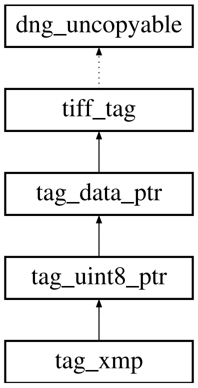
Additional Inherited Members

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.232 tag_xmp Class Reference

Inheritance diagram for tag_xmp:



Public Member Functions

- **tag_xmp** (const dng_xmp ∗xmp)

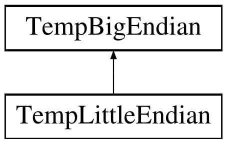
Additional Inherited Members

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.233 TempBigEndian Class Reference

Inheritance diagram for TempBigEndian:



Public Member Functions

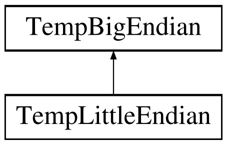
- **TempBigEndian** (dng_stream &stream, bool bigEndian=true)

The documentation for this class was generated from the following files:

- dng_stream.h
- dng_stream.cpp

5.234 TempLittleEndian Class Reference

Inheritance diagram for TempLittleEndian:



Public Member Functions

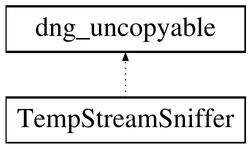
- **TempLittleEndian** (dng_stream &stream, bool littleEndian=true)

The documentation for this class was generated from the following file:

- dng_stream.h

5.235 TempStreamSniffer Class Reference

Inheritance diagram for TempStreamSniffer:



Public Member Functions

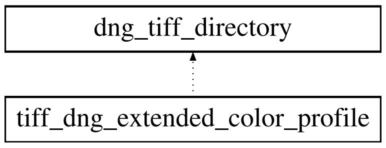
- **TempStreamSniffer** (dng_stream &stream, dng_abort_sniffer ∗sniffer)

The documentation for this class was generated from the following files:

- dng_stream.h
- dng_stream.cpp

5.236 tiff_dng_extended_color_profile Class Reference

Inheritance diagram for tiff_dng_extended_color_profile:



Public Member Functions

- **tiff_dng_extended_color_profile** (const dng_camera_profile &profile)
- void **Put** (dng_stream &stream, bool includeModelRestriction=true)

Protected Attributes

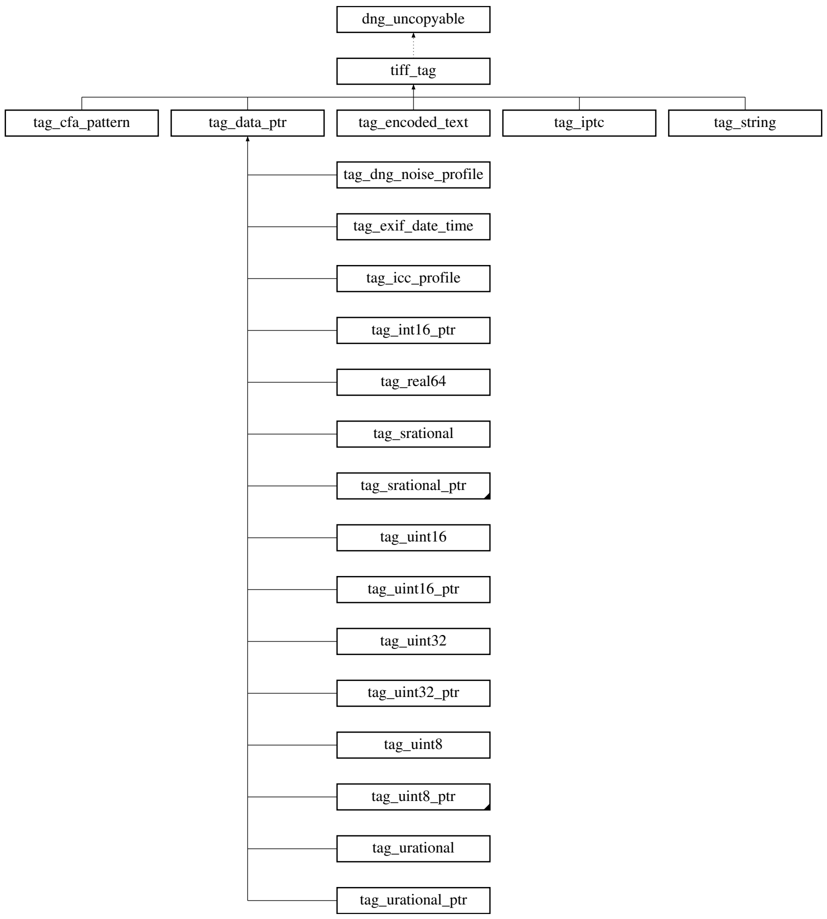
- const dng_camera_profile & **fProfile**

The documentation for this class was generated from the following files:

- dng_image_writer.h
- dng_image_writer.cpp

5.237 tiff_tag Class Reference

Inheritance diagram for tiff_tag:



Public Member Functions

- uint16 **Code** () const
- uint16 **Type** () const
- uint32 **Count** () const
- void **SetCount** (uint32 count)
- uint32 **Size** () const
- virtual void **Put** (dng_stream &stream) const =0

Protected Member Functions

- **tiff_tag** (uint16 code, uint16 type, uint32 count)

Protected Attributes

- uint16 **fCode**
- uint16 **fType**
- uint32 **fCount**

The documentation for this class was generated from the following file:

- dng_image_writer.h

5.238 UnicodeToLowASCIIEntry Struct Reference

Public Attributes

- uint32 **unicode**
- const char ∗ **ascii**

The documentation for this struct was generated from the following file:

- dng_string.cpp

6 File Documentation

6.1 dng_1d_function.h File Reference

```
#include "dng_classes.h"
#include "dng_types.h"
```

Classes

- class dng_1d_function

  A 1D floating-point function.
- class dng_1d_identity

  An identity (x -> y such that x == y for all x) mapping function.
- class dng_1d_concatenate

  A dng_1d_function that represents the composition (curry) of two other dng_1d_functions.
- class dng_1d_inverse

  A dng_1d_function that represents the inverse of another dng_1d_function.

6.1.1 Detailed Description

Classes for a 1D floating-point to floating-point function abstraction.

6.2 dng_1d_table.h File Reference

```
#include "dng_assertions.h"
#include "dng_auto_ptr.h"
#include "dng_classes.h"
#include "dng_types.h"
#include "dng_uncopyable.h"
```

Classes

- class dng_1d_table

  A 1D floating-point lookup table using linear interpolation.

6.2.1 Detailed Description

Definition of a lookup table based 1D floating-point to floating-point function abstraction using linear interpolation.

6.3 dng_abort_sniffer.h File Reference

```
#include "dng_classes.h"
#include "dng_flags.h"
#include "dng_string.h"
#include "dng_types.h"
#include "dng_uncopyable.h"
```

Classes

- class dng_set_minimum_priority

  Convenience class for setting thread priority level to minimum.
- class dng_abort_sniffer

  Class for signaling user cancellation and receiving progress updates.
- class dng_sniffer_task

  Class to establish scope of a named subtask in DNG processing.

Enumerations

- enum dng_priority {
  dng_priority_low, **dng_priority_medium**, **dng_priority_high**, **dng_priority_count**,
  dng_priority_minimum = dng_priority_low, **dng_priority_maximum** = dng_priority_high }

  Thread priority level.

6.3.1 Detailed Description

Classes supporting user cancellation and progress tracking.

6.4 dng_area_task.h File Reference

```
#include "dng_classes.h"
#include "dng_point.h"
#include "dng_string.h"
#include "dng_types.h"
#include "dng_uncopyable.h"
```

Classes

- class dng_area_task_progress
- class dng_area_task

  Abstract class for rectangular processing operations with support for partitioning across multiple processing resources and observing memory constraints.

6.4.1 Detailed Description

Class to handle partitioning a rectangular image processing operation taking into account multiple processing resources and memory constraints.

6.5 dng_assertions.h File Reference

```
#include "dng_exceptions.h"
#include "dng_flags.h"
```

Macros

- #define DNG_ASSERT(x, y)
- #define DNG_REQUIRE(condition, msg)
- #define DNG_REPORT(x) DNG_ASSERT (false, x)

6.5.1 Detailed Description

Conditionally compiled assertion check support.

6.5.2 Macro Definition Documentation

6.5.2.1 DNG_ASSERT

```
#define DNG_ASSERT(
              x,
              y )
```

Conditionally compiled macro to check an assertion and display a message if it fails and assertions are compiled in via qDNGDebug

Parameters

x	Predicate which must be true.
y	String to display if x is not true.

6.5.2.2 DNG_REPORT

```
#define DNG_REPORT(
             x ) DNG_ASSERT (false, x)
```

Macro to display an informational message

Parameters

x	String to display.

6.5.2.3 DNG_REQUIRE

```
#define DNG_REQUIRE(
             condition,
             msg )
```

Value:

```
do                                  \
      {                               \
                                      \
      if (!(condition))                  \
         {                            \
                                      \
         ThrowProgramError (msg);        \
                                      \
         }                            \
                                      \
      }                               \
   while (0)
```

Conditionally compiled macro to check an assertion, display a message, and throw an exception if it fails and assertions are compiled in via qDNGDebug

Parameters

condition	Predicate which must be true.
msg	String to display if condition is not true.

6.6 dng_auto_ptr.h File Reference

```
#include <stddef.h>
#include "dng_uncopyable.h"
```

Classes

- class AutoPtr< T >

A class intended to be used in stack scope to hold a pointer from new. The held pointer will be deleted automatically if the scope is left without calling Release on the AutoPtr first.

- class AutoArray< T >

  A class intended to be used similarly to AutoPtr but for arrays.

6.6.1 Detailed Description

Class to implement std::auto_ptr like functionality even on platforms which do not have a full Standard C++ library.

6.7 dng_bad_pixels.h File Reference

```
#include "dng_memory.h"
#include "dng_opcodes.h"
#include <vector>
```

Classes

- class dng_opcode_FixBadPixelsConstant

  An opcode to fix individual bad pixels that are marked with a constant value (e.g., 0) in a Bayer image.
- class dng_bad_pixel_list

  A list of bad pixels and rectangles (usually single rows or columns).
- class dng_opcode_FixBadPixelsList

  An opcode to fix lists of bad pixels (indicated by position) in a Bayer image.

6.7.1 Detailed Description

Opcodes to fix defective pixels, including individual pixels and regions (such as defective rows and columns).

6.8 dng_bottlenecks.h File Reference

```
#include "dng_classes.h"
#include "dng_types.h"
```

Classes

- struct dng_suite

Typedefs

- typedef void() **ZeroBytesProc**(void ∗dPtr, uint32 count)
- typedef void() **CopyBytesProc**(const void ∗sPtr, void ∗dPtr, uint32 count)
- typedef void() **SwapBytes16Proc**(uint16 ∗dPtr, uint32 count)
- typedef void() **SwapBytes32Proc**(uint32 ∗dPtr, uint32 count)
- typedef void() **SetArea8Proc**(uint8 ∗dPtr, uint8 value, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep)
- typedef void() **SetArea16Proc**(uint16 ∗dPtr, uint16 value, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep)
- typedef void() **SetArea32Proc**(uint32 ∗dPtr, uint32 value, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep)
- typedef void() **CopyArea8Proc**(const uint8 ∗sPtr, uint8 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 s↩ RowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void() **CopyArea16Proc**(const uint16 ∗sPtr, uint16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void() **CopyArea32Proc**(const uint32 ∗sPtr, uint32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void() **CopyArea8_16Proc**(const uint8 ∗sPtr, uint16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void() **CopyArea8_S16Proc**(const uint8 ∗sPtr, int16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void() **CopyArea8_32Proc**(const uint8 ∗sPtr, uint32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void() **CopyArea16_S16Proc**(const uint16 ∗sPtr, int16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void() **CopyArea16_32Proc**(const uint16 ∗sPtr, uint32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void() **CopyArea8_R32Proc**(const uint8 ∗sPtr, real32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- typedef void() **CopyArea16_R32Proc**(const uint16 ∗sPtr, real32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- typedef void() **CopyAreaS16_R32Proc**(const int16 ∗sPtr, real32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- typedef void() **CopyAreaR32_8Proc**(const real32 ∗sPtr, uint8 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- typedef void() **CopyAreaR32_16Proc**(const real32 ∗sPtr, uint16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- typedef void() **CopyAreaR32_S16Proc**(const real32 ∗sPtr, int16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- typedef void() **RepeatArea8Proc**(const uint8 ∗sPtr, uint8 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 repeatV, uint32 repeatH, uint32 phaseV, uint32 phaseH)
- typedef void() **RepeatArea16Proc**(const uint16 ∗sPtr, uint16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 repeatV, uint32 repeatH, uint32 phaseV, uint32 phaseH)
- typedef void() **RepeatArea32Proc**(const uint32 ∗sPtr, uint32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 repeatV, uint32 repeatH, uint32 phaseV, uint32 phaseH)

- typedef void() **ShiftRight16Proc**(uint16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 col←Step, int32 planeStep, uint32 shift)
- typedef void() **BilinearRow16Proc**(const uint16 ∗sPtr, uint16 ∗dPtr, uint32 cols, uint32 patPhase, uint32 pat←Count, const uint32 ∗kernCounts, const int32 ∗const ∗kernOffsets, const uint16 ∗const ∗kernWeights, uint32 sShift)
- typedef void() **BilinearRow32Proc**(const real32 ∗sPtr, real32 ∗dPtr, uint32 cols, uint32 patPhase, uint32 pat←Count, const uint32 ∗kernCounts, const int32 ∗const ∗kernOffsets, const real32 ∗const ∗kernWeights, uint32 sShift)
- typedef void() **BaselineABCtoRGBProc**(const real32 ∗sPtrA, const real32 ∗sPtrB, const real32 ∗sPtrC, real32 ∗dPtrR, real32 ∗dPtrG, real32 ∗dPtrB, uint32 count, const dng_vector &cameraWhite, const dng_matrix &cameraToRGB)
- typedef void() **BaselineABCDtoRGBProc**(const real32 ∗sPtrA, const real32 ∗sPtrB, const real32 ∗sPtrC, const real32 ∗sPtrD, real32 ∗dPtrR, real32 ∗dPtrG, real32 ∗dPtrB, uint32 count, const dng_vector &cameraWhite, const dng_matrix &cameraToRGB)
- typedef void() **BaselineHueSatMapProc**(const real32 ∗sPtrR, const real32 ∗sPtrG, const real32 ∗sPtrB, real32 ∗dPtrR, real32 ∗dPtrG, real32 ∗dPtrB, uint32 count, const dng_hue_sat_map &lut, const dng_1d_table ∗encode←Table, const dng_1d_table ∗decodeTable)
- typedef void() **BaselineGrayToRGBProc**(const real32 ∗sPtrR, const real32 ∗sPtrG, const real32 ∗sPtrB, real32 ∗dPtrG, uint32 count, const dng_matrix &matrix)
- typedef void() **BaselineRGBtoRGBProc**(const real32 ∗sPtrR, const real32 ∗sPtrG, const real32 ∗sPtrB, real32 ∗dPtrR, real32 ∗dPtrG, real32 ∗dPtrB, uint32 count, const dng_matrix &matrix)
- typedef void() **Baseline1DTableProc**(const real32 ∗sPtr, real32 ∗dPtr, uint32 count, const dng_1d_table &table)
- typedef void() **BaselineRGBToneProc**(const real32 ∗sPtrR, const real32 ∗sPtrG, const real32 ∗sPtrB, real32 ∗dPtrR, real32 ∗dPtrG, real32 ∗dPtrB, uint32 count, const dng_1d_table &table)
- typedef void() **ResampleDown16Proc**(const uint16 ∗sPtr, uint16 ∗dPtr, uint32 sCount, int32 sRowStep, const int16 ∗wPtr, uint32 wCount, uint32 pixelRange)
- typedef void() **ResampleDown32Proc**(const real32 ∗sPtr, real32 ∗dPtr, uint32 sCount, int32 sRowStep, const real32 ∗wPtr, uint32 wCount)
- typedef void() **ResampleAcross16Proc**(const uint16 ∗sPtr, uint16 ∗dPtr, uint32 dCount, const int32 ∗coord, const int16 ∗wPtr, uint32 wCount, uint32 wStep, uint32 pixelRange)
- typedef void() **ResampleAcross32Proc**(const real32 ∗sPtr, real32 ∗dPtr, uint32 dCount, const int32 ∗coord, const real32 ∗wPtr, uint32 wCount, uint32 wStep)
- typedef bool() **EqualBytesProc**(const void ∗sPtr, const void ∗dPtr, uint32 count)
- typedef bool() **EqualArea8Proc**(const uint8 ∗sPtr, const uint8 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef bool() **EqualArea16Proc**(const uint16 ∗sPtr, const uint16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef bool() **EqualArea32Proc**(const uint32 ∗sPtr, const uint32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void() **VignetteMask16Proc**(uint16 ∗mPtr, uint32 rows, uint32 cols, int32 rowStep, int64 offsetH, int64 offsetV, int64 stepH, int64 stepV, uint32 tBits, const uint16 ∗table)
- typedef void() **Vignette16Proc**(int16 ∗sPtr, const uint16 ∗mPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sPlaneStep, int32 mRowStep, uint32 mBits)
- typedef void() **Vignette32Proc**(real32 ∗sPtr, const uint16 ∗mPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sPlaneStep, int32 mRowStep, uint32 mBits, uint16 blackLevel)
- typedef void() **MapArea16Proc**(uint16 ∗dPtr, uint32 count0, uint32 count1, uint32 count2, int32 step0, int32 step1, int32 step2, const uint16 ∗map)
- typedef void() **BaselineMapPoly32Proc**(real32 ∗dPtr, const int32 rowStep, const uint32 rows, const uint32 cols, const uint32 rowPitch, const uint32 colPitch, const real32 ∗coefficients, const uint32 degree, uint16 blackLevel)

Functions

- void **DoZeroBytes** (void ∗dPtr, uint32 count)
- void **DoCopyBytes** (const void ∗sPtr, void ∗dPtr, uint32 count)
- void **DoSwapBytes16** (uint16 ∗dPtr, uint32 count)
- void **DoSwapBytes32** (uint32 ∗dPtr, uint32 count)
- void **DoSetArea8** (uint8 ∗dPtr, uint8 value, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep)
- void **DoSetArea16** (uint16 ∗dPtr, uint16 value, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep)
- void **DoSetArea32** (uint32 ∗dPtr, uint32 value, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep)
- void **DoCopyArea8** (const uint8 ∗sPtr, uint8 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea16** (const uint16 ∗sPtr, uint16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea32** (const uint32 ∗sPtr, uint32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea8_16** (const uint8 ∗sPtr, uint16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea8_S16** (const uint8 ∗sPtr, int16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea8_32** (const uint8 ∗sPtr, uint32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea16_S16** (const uint16 ∗sPtr, int16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRow←↩ Step, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea16_32** (const uint16 ∗sPtr, uint32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRow←↩ Step, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea8_R32** (const uint8 ∗sPtr, real32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRow←↩ Step, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- void **DoCopyArea16_R32** (const uint16 ∗sPtr, real32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRow←↩ Step, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- void **DoCopyAreaS16_R32** (const int16 ∗sPtr, real32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRow←↩ Step, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- void **DoCopyAreaR32_8** (const real32 ∗sPtr, uint8 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRow←↩ Step, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- void **DoCopyAreaR32_16** (const real32 ∗sPtr, uint16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRow←↩ Step, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- void **DoCopyAreaR32_S16** (const real32 ∗sPtr, int16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRow←↩ Step, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- void **DoRepeatArea8** (const uint8 ∗sPtr, uint8 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 repeatV, uint32 repeatH, uint32 phaseV, uint32 phaseH)
- void **DoRepeatArea16** (const uint16 ∗sPtr, uint16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 repeatV, uint32 repeatH, uint32 phaseV, uint32 phaseH)
- void **DoRepeatArea32** (const uint32 ∗sPtr, uint32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 repeatV, uint32 repeatH, uint32 phaseV, uint32 phaseH)
- void **DoShiftRight16** (uint16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 shift)
- void **DoBilinearRow16** (const uint16 ∗sPtr, uint16 ∗dPtr, uint32 cols, uint32 patPhase, uint32 patCount, const uint32 ∗kernCounts, const int32 ∗const ∗kernOffsets, const uint16 ∗const ∗kernWeights, uint32 sShift)
- void **DoBilinearRow32** (const real32 ∗sPtr, real32 ∗dPtr, uint32 cols, uint32 patPhase, uint32 patCount, const uint32 ∗kernCounts, const int32 ∗const ∗kernOffsets, const real32 ∗const ∗kernWeights, uint32 sShift)

- void **DoBaselineABCtoRGB** (const real32 ∗sPtrA, const real32 ∗sPtrB, const real32 ∗sPtrC, real32 ∗dPtrR, real32 ∗dPtrG, real32 ∗dPtrB, uint32 count, const dng_vector &cameraWhite, const dng_matrix &cameraToRGB)
- void **DoBaselineABCDtoRGB** (const real32 ∗sPtrA, const real32 ∗sPtrB, const real32 ∗sPtrC, const real32 ∗sPtrD, real32 ∗dPtrR, real32 ∗dPtrG, real32 ∗dPtrB, uint32 count, const dng_vector &cameraWhite, const dng_matrix &cameraToRGB)
- void **DoBaselineHueSatMap** (const real32 ∗sPtrR, const real32 ∗sPtrG, const real32 ∗sPtrB, real32 ∗dPtrR, real32 ∗dPtrG, real32 ∗dPtrB, uint32 count, const dng_hue_sat_map &lut, const dng_1d_table ∗encodeTable, const dng_1d_table ∗decodeTable)
- void **DoBaselineRGBtoGray** (const real32 ∗sPtrR, const real32 ∗sPtrG, const real32 ∗sPtrB, real32 ∗dPtrG, uint32 count, const dng_matrix &matrix)
- void **DoBaselineRGBtoRGB** (const real32 ∗sPtrR, const real32 ∗sPtrG, const real32 ∗sPtrB, real32 ∗dPtrR, real32 ∗dPtrG, real32 ∗dPtrB, uint32 count, const dng_matrix &matrix)
- void **DoBaseline1DTable** (const real32 ∗sPtr, real32 ∗dPtr, uint32 count, const dng_1d_table &table)
- void **DoBaselineRGBTone** (const real32 ∗sPtrR, const real32 ∗sPtrG, const real32 ∗sPtrB, real32 ∗dPtrR, real32 ∗dPtrG, real32 ∗dPtrB, uint32 count, const dng_1d_table &table)
- void **DoResampleDown16** (const uint16 ∗sPtr, uint16 ∗dPtr, uint32 sCount, int32 sRowStep, const int16 ∗wPtr, uint32 wCount, uint32 pixelRange)
- void **DoResampleDown32** (const real32 ∗sPtr, real32 ∗dPtr, uint32 sCount, int32 sRowStep, const real32 ∗wPtr, uint32 wCount)
- void **DoResampleAcross16** (const uint16 ∗sPtr, uint16 ∗dPtr, uint32 dCount, const int32 ∗coord, const int16 ∗wPtr, uint32 wCount, uint32 wStep, uint32 pixelRange)
- void **DoResampleAcross32** (const real32 ∗sPtr, real32 ∗dPtr, uint32 dCount, const int32 ∗coord, const real32 ∗wPtr, uint32 wCount, uint32 wStep)
- bool **DoEqualBytes** (const void ∗sPtr, const void ∗dPtr, uint32 count)
- bool **DoEqualArea8** (const uint8 ∗sPtr, const uint8 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- bool **DoEqualArea16** (const uint16 ∗sPtr, const uint16 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 s←↩ RowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- bool **DoEqualArea32** (const uint32 ∗sPtr, const uint32 ∗dPtr, uint32 rows, uint32 cols, uint32 planes, int32 s←↩ RowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoVignetteMask16** (uint16 ∗mPtr, uint32 rows, uint32 cols, int32 rowStep, int64 offsetH, int64 offsetV, int64 stepH, int64 stepV, uint32 tBits, const uint16 ∗table)
- void **DoVignette16** (int16 ∗sPtr, const uint16 ∗mPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sPlaneStep, int32 mRowStep, uint32 mBits)
- void **DoVignette32** (real32 ∗sPtr, const uint16 ∗mPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sPlaneStep, int32 mRowStep, uint32 mBits, uint16 blackLevel)
- void **DoMapArea16** (uint16 ∗dPtr, uint32 count0, uint32 count1, uint32 count2, int32 step0, int32 step1, int32 step2, const uint16 ∗map)
- void **DoBaselineMapPoly32** (real32 ∗dPtr, const int32 rowStep, const uint32 rows, const uint32 cols, const uint32 rowPitch, const uint32 colPitch, const real32 ∗coefficients, const uint32 degree, uint16 blackLevel)

Variables

- dng_suite **gDNGSuite**

6.8.1 Detailed Description

Indirection mechanism for performance-critical routines that might be replaced with hand-optimized or hardware-specific implementations.

6.9 dng_camera_profile.h File Reference

```
#include "dng_auto_ptr.h"
#include "dng_assertions.h"
#include "dng_classes.h"
#include "dng_fingerprint.h"
#include "dng_hue_sat_map.h"
#include "dng_matrix.h"
#include "dng_string.h"
#include "dng_tag_values.h"
#include "dng_tone_curve.h"
```

Classes

- class dng_camera_profile_id

  An ID for a camera profile consisting of a name and optional fingerprint.
- class dng_camera_profile

  Container for DNG camera color profile and calibration data.

Functions

- void **SplitCameraProfileName** (const dng_string &name, dng_string &baseName, int32 &version)
- void **BuildHueSatMapEncodingTable** (dng_memory_allocator &allocator, uint32 encoding, AutoPtr< dng_1d_table > &encodeTable, AutoPtr< dng_1d_table > &decodeTable, bool subSample)

Variables

- const char ∗ **kProfileName_Embedded**
- const char ∗ **kAdobeCalibrationSignature**

6.9.1 Detailed Description

Support for DNG camera color profile information. Per the DNG 1.1.0 specification, a DNG file can store up to two sets of color profile information for a camera in the DNG file from that camera. The second set is optional and when there are two sets, they represent profiles made under different illumination.

Profiling information is optionally separated into two parts. One part represents a profile for a reference camera. (Color↩
Matrix1 and ColorMatrix2 here.) The second is a per-camera calibration that takes into account unit-to-unit variation. This is designed to allow replacing the reference color matrix with one of one's own construction while maintaining any unit-specific calibration the camera manufacturer may have provided.

See Appendix 6 of the DNG 1.1.0 specification for more information.

6.10 dng_color_space.h File Reference

```
#include "dng_1d_function.h"
#include "dng_classes.h"
#include "dng_matrix.h"
#include "dng_types.h"
```

Classes

- class dng_function_GammaEncode_sRGB

  A dng_1d_function for gamma encoding in sRGB color space.
- class dng_function_GammaEncode_1_8

  A dng_1d_function for gamma encoding with 1.8 gamma.
- class dng_function_GammaEncode_2_2

  A dng_1d_function for gamma encoding with 2.2 gamma.
- class dng_color_space

  An abstract color space.
- class dng_space_sRGB

  Singleton class for sRGB color space.
- class dng_space_AdobeRGB

  Singleton class for AdobeRGB color space.
- class dng_space_ColorMatch

  Singleton class for ColorMatch color space.
- class dng_space_ProPhoto

  Singleton class for ProPhoto RGB color space.
- class dng_space_GrayGamma18

  Singleton class for gamma 1.8 grayscale color space.
- class dng_space_GrayGamma22

  Singleton class for gamma 2.2 grayscale color space.
- class dng_space_fakeRGB

6.10.1 Detailed Description

Standard gamma functions and color spaces used within the DNG SDK.

6.11 dng_color_spec.h File Reference

```
#include "dng_classes.h"
#include "dng_matrix.h"
#include "dng_types.h"
#include "dng_xy_coord.h"
```

Classes

- class dng_color_spec

Functions

- dng_matrix_3by3 MapWhiteMatrix (const dng_xy_coord &white1, const dng_xy_coord &white2)

  Compute a 3x3 matrix which maps colors from white point white1 to white point white2.

6.11.1 Detailed Description

Class for holding a specific color transform.

6.11.2 Function Documentation

6.11.2.1 MapWhiteMatrix()

```
dng_matrix_3by3 MapWhiteMatrix (
             const dng_xy_coord & white1,
             const dng_xy_coord & white2 )
```

Compute a 3x3 matrix which maps colors from white point white1 to white point white2.

Uses linearized Bradford adaptation matrix to compute a mapping from colors measured with one white point (white1) to another (white2).

6.12 dng_date_time.h File Reference

```
#include "dng_classes.h"
#include "dng_string.h"
#include "dng_types.h"
```

Classes

- class dng_date_time

  Class for holding a date/time and converting to and from relevant date/time formats.
- class dng_time_zone

  Class for holding a time zone.
- class dng_date_time_info

  Class for holding complete data/time/zone information.
- class dng_date_time_storage_info

  Store file offset from which date was read.

Enumerations

- enum dng_date_time_format { **dng_date_time_format_unknown** = 0, dng_date_time_format_exif = 1, dng_date_time_format_unix_little_endian = 2, dng_date_time_format_unix_big_endian = 3 }

  Tag to encode date represenation format.

Functions

- void CurrentDateTimeAndZone (dng_date_time_info &info)
- void DecodeUnixTime (uint32 unixTime, dng_date_time &dt)

  Convert UNIX "seconds since Jan 1, 1970" time to a dng_date_time.
- dng_time_zone LocalTimeZone (const dng_date_time &dt)

6.12.1 Detailed Description

Functions and classes for working with dates and times in DNG files.

6.12.2 Enumeration Type Documentation

6.12.2.1 dng_date_time_format

```
enum dng_date_time_format
```

Tag to encode date represenation format.

Enumerator

dng_date_time_format_exif	Date format not known.
dng_date_time_format_unix_little_endian	EXIF date string.
dng_date_time_format_unix_big_endian	32-bit UNIX time as 4-byte little endian

6.12.3 Function Documentation

6.12.3.1 CurrentDateTimeAndZone()

```
void CurrentDateTimeAndZone (
            dng_date_time_info & info )
```

Get the current date/time and timezone.

Parameters

info	Receives current data/time/zone.

Referenced by LocalTimeZone().

6.12.3.2 LocalTimeZone()

```
dng_time_zone LocalTimeZone (
             const dng_date_time & dt )
```

Return timezone of current location at a given date.

Parameters

dt	Date at which to compute timezone difference. (For example, used to determine Daylight Savings, etc.)

Return values

Time	zone for date/time dt.

References CurrentDateTimeAndZone(), and dng_date_time::IsValid().

6.13 dng_errors.h File Reference

```
#include "dng_types.h"
```

Typedefs

- typedef int32 dng_error_code

    Type for all errors used in DNG SDK. Generally held inside a dng_exception.

Enumerations

- enum {
  dng_error_none = 0, dng_error_unknown = 100000, dng_error_not_yet_implemented, dng_error_silent,
  dng_error_user_canceled, dng_error_host_insufficient, dng_error_memory, dng_error_bad_format,
  dng_error_matrix_math, dng_error_open_file, dng_error_read_file, dng_error_write_file,
  dng_error_end_of_file, dng_error_file_is_damaged, dng_error_image_too_big_dng, dng_error_image_too_big_tiff,
  dng_error_unsupported_dng, dng_error_overflow }

6.13.1 Detailed Description

Error code values.

6.13.2 Enumeration Type Documentation

6.13.2.1 anonymous enum

`anonymous enum`

Enumerator

dng_error_none	No error. Success.
dng_error_unknown	Logic or program error or other unclassifiable error.
dng_error_not_yet_implemented	Functionality requested is not yet implemented.
dng_error_silent	An error which should not be signalled to user.
dng_error_user_canceled	Processing stopped by user (or host application) request.
dng_error_host_insufficient	Necessary host functionality is not present.
dng_error_memory	Out of memory.
dng_error_bad_format	File format is not valid.
dng_error_matrix_math	Matrix has wrong shape, is badly conditioned, or similar problem.
dng_error_open_file	Could not open file.
dng_error_read_file	Error reading file.
dng_error_write_file	Error writing file.
dng_error_end_of_file	Unexpected end of file.
dng_error_file_is_damaged	File is damaged in some way.
dng_error_image_too_big_dng	Image is too big to save as DNG.
dng_error_image_too_big_tiff	Image is too big to save as TIFF.
dng_error_unsupported_dng	DNG version is unsupported.
dng_error_overflow	Arithmetic overflow.

6.14 dng_exceptions.h File Reference

```
#include "dng_errors.h"
#include "dng_flags.h"
```

Classes

- class dng_exception

    All exceptions thrown by the DNG SDK use this exception class.

Functions

- void ReportWarning (const char ∗message, const char ∗sub_message=NULL)

    Display a warning message. Note that this may just eat the message.
- void ReportError (const char ∗message, const char ∗sub_message=NULL)

    Display an error message. Note that this may just eat the message.
- void Throw_dng_error (dng_error_code err, const char ∗message=NULL, const char ∗sub_message=NULL, bool silent=false) DNG_NO_RETURN

    Throw an exception based on an arbitrary error code.
- void Fail_dng_error (dng_error_code err)

    Convenience function to throw dng_exception with error code if error_code is not dng_error_none .
- void ThrowProgramError (const char ∗sub_message=NULL)

    Convenience function to throw dng_exception with error code dng_error_unknown .
- void ThrowOverflow (const char ∗sub_message=NULL)

    Convenience function to throw dng_exception with error code dng_error_overflow.
- void ThrowNotYetImplemented (const char ∗sub_message=NULL)

    Convenience function to throw dng_exception with error code dng_error_not_yet_implemented .
- void ThrowSilentError ()

    Convenience function to throw dng_exception with error code dng_error_silent .
- void ThrowUserCanceled ()

    Convenience function to throw dng_exception with error code dng_error_user_canceled .
- void ThrowHostInsufficient (const char ∗sub_message=NULL, bool silent=false)

    Convenience function to throw dng_exception with error code dng_error_host_insufficient .
- void ThrowMemoryFull (const char ∗sub_message=NULL)

    Convenience function to throw dng_exception with error code dng_error_memory .
- void ThrowBadFormat (const char ∗sub_message=NULL)

    Convenience function to throw dng_exception with error code dng_error_bad_format .
- void ThrowMatrixMath (const char ∗sub_message=NULL)

    Convenience function to throw dng_exception with error code dng_error_matrix_math .
- void ThrowOpenFile (const char ∗sub_message=NULL, bool silent=false)

    Convenience function to throw dng_exception with error code dng_error_open_file .
- void ThrowReadFile (const char ∗sub_message=NULL)

    Convenience function to throw dng_exception with error code dng_error_read_file .
- void ThrowWriteFile (const char ∗sub_message=NULL)

    Convenience function to throw dng_exception with error code dng_error_write_file .
- void ThrowEndOfFile (const char ∗sub_message=NULL)

    Convenience function to throw dng_exception with error code dng_error_end_of_file .
- void ThrowFileIsDamaged ()

    Convenience function to throw dng_exception with error code dng_error_file_is_damaged .
- void ThrowImageTooBigDNG ()

    Convenience function to throw dng_exception with error code dng_error_image_too_big_dng .
- void ThrowImageTooBigTIFF ()

    Convenience function to throw dng_exception with error code dng_error_image_too_big_tiff .
- void ThrowUnsupportedDNG ()

    Convenience function to throw dng_exception with error code dng_error_unsupported_dng .

6.14.1 Detailed Description

C++ exception support for DNG SDK.

6.15 dng_exif.h File Reference

```
#include "dng_classes.h"
#include "dng_date_time.h"
#include "dng_fingerprint.h"
#include "dng_types.h"
#include "dng_matrix.h"
#include "dng_rational.h"
#include "dng_string.h"
#include "dng_stream.h"
#include "dng_sdk_limits.h"
```

Classes

- class dng_exif

  Container class for parsing and holding EXIF tags.

6.15.1 Detailed Description

EXIF read access support. See the EXIF specification for full description of tags.

6.16 dng_fast_module.h File Reference

6.16.1 Detailed Description

Include file to set optimization to highest level for performance-critical routines. Normal files should have otpimization set to normal level to save code size as there is less cache pollution this way.

6.17 dng_file_stream.h File Reference

```
#include "dng_stream.h"
```

Classes

- class dng_file_stream

  A stream to/from a disk file. See dng_stream for read/write interface.

6.17.1 Detailed Description

Simple, portable, file read/write support.

6.18 dng_filter_task.h File Reference

```
#include "dng_area_task.h"
#include "dng_auto_ptr.h"
#include "dng_point.h"
#include "dng_rect.h"
#include "dng_sdk_limits.h"
```

Classes

- class dng_filter_task

    Represents a task which filters an area of a source dng_image to an area of a destination dng_image.

6.18.1 Detailed Description

Specialization of dng_area_task for processing an area from one dng_image to an area of another.

6.19 dng_fingerprint.h File Reference

```
#include "dng_exceptions.h"
#include "dng_types.h"
#include "dng_stream.h"
#include <cstring>
```

Classes

- class dng_fingerprint

    Container fingerprint (MD5 only at present).
- struct dng_fingerprint_less_than

    Utility to compare fingerprints (e.g., for sorting).
- struct dng_fingerprint_hash

    Utility to hash fingerprints (e.g., for hashtables).
- class dng_md5_printer

    Class to hash binary data to a fingerprint using the MD5 Message-Digest Algorithm.
- class dng_md5_printer_stream

    A dng_stream based interface to the MD5 printing logic.

6.19.1 Detailed Description

Fingerprint (cryptographic hashing) support for generating strong hashes of image data.

6.20 dng_flags.h File Reference

```
#include "RawEnvironment.h"
```

Macros

- #define qMacOS 0
- #define **qiPhone** 0
- #define **qiPhoneSimulator** 0
- #define **qAndroid** 0
- #define qWinOS 0
- #define **qWinRT** 0
- #define **qLinux** 0
- #define **qWeb** 0
- #define **qARM** 0
- #define **qARMNeon** 0
- #define qDNGDebug 0
- #define **qCRSupportTBB** 0
- #define **qDNGIntelCompiler** 0
- #define qDNGLittleEndian !qDNGBigEndian
- #define qDNG64Bit 0
- #define qDNGThreadSafe (qMacOS || qWinOS)
- #define qDNGValidateTarget 0
- #define qDNGValidate qDNGValidateTarget
- #define qDNGPrintMessages qDNGValidate
- #define **qDNGExperimental** 1
- #define qDNGXMPFiles 1
- #define qDNGXMPDocOps (!qDNGValidateTarget)
- #define qDNGUseLibJPEG qDNGValidateTarget
- #define **qDNGAVXSupport** ((qMacOS || qWinOS) && qDNG64Bit && !qARM && 1)
- #define **qDNGSupportVC5** (1)
- #define qDNGUsingSanitizer (0)
- #define **DNG_ATTRIB_NO_SANITIZE**(type)

6.20.1 Detailed Description

Conditional compilation flags for DNG SDK.

All conditional compilation macros for the DNG SDK begin with a lowercase 'q'.

6.20.2 Macro Definition Documentation

6.20.2.1 qDNG64Bit

```
#define qDNG64Bit 0
```

1 if this target platform uses 64-bit addresses, 0 otherwise.

6.20.2.2 qDNGDebug

```
#define qDNGDebug 0
```

1 if debug code is compiled in, 0 otherwise. Enables assertions and other debug checks in exchange for slower processing.

6.20.2.3 qDNGLittleEndian

```
#define qDNGLittleEndian !qDNGBigEndian
```

1 if this target platform is little endian (e.g. x86 processors), else 0.

6.20.2.4 qDNGPrintMessages

```
#define qDNGPrintMessages qDNGValidate
```

1 if dng_show_message should use fprintf to stderr. 0 if it should use a platform specific interrupt mechanism.

6.20.2.5 qDNGThreadSafe

```
#define qDNGThreadSafe (qMacOS || qWinOS)
```

1 if target platform has thread support and threadsafe libraries, 0 otherwise.

6.20.2.6 qDNGUseLibJPEG

```
#define qDNGUseLibJPEG qDNGValidateTarget
```

1 to use open-source libjpeg for lossy jpeg processing.

6.20.2.7 qDNGUsingSanitizer

```
#define qDNGUsingSanitizer (0)
```

Set to 1 when using a Sanitizer tool.

6.20.2.8 qDNGValidate

```
#define qDNGValidate qDNGValidateTarget
```

1 if DNG validation code is enabled, 0 otherwise.

6.20.2.9 qDNGValidateTarget

```
#define qDNGValidateTarget 0
```

1 if dng_validate command line tool is being built, 0 otherwise.

6.20.2.10 qDNGXMPDocOps

```
#define qDNGXMPDocOps (!qDNGValidateTarget)
```

1 to use XMPDocOps.

6.20.2.11 qDNGXMPFiles

```
#define qDNGXMPFiles 1
```

1 to use XMPFiles.

6.20.2.12 qMacOS

```
#define qMacOS 0
```

1 if compiling for Mac OS X.

6.20.2.13 qWinOS

```
#define qWinOS 0
```

1 if compiling for Windows.

6.21 dng_gain_map.h File Reference

```
#include "dng_memory.h"
#include "dng_misc_opcodes.h"
#include "dng_tag_types.h"
#include "dng_uncopyable.h"
```

Classes

- class dng_gain_map

    Holds a discrete (i.e., sampled) 2D representation of a gain map. This is effectively an image containing scale factors.
- class dng_opcode_GainMap

    An opcode to fix 2D spatially-varying light falloff or color casts (i.e., uniformity issues). This is commonly due to shading.

6.21.1 Detailed Description

Opcode to fix 2D uniformity defects, such as shading.

6.22 dng_globals.h File Reference

```
#include "dng_flags.h"
#include "dng_types.h"
```

Variables

- bool **gDNGShowTimers**
- bool **gDNGUseFakeTimeZonesInXMP**
- uint32 **gDNGStreamBlockSize**
- uint32 **gDNGMaxStreamBufferSize**
- bool **gImagecore**
- bool **gPrintTimings**
- bool **gPrintAsserts**
- bool **gBreakOnAsserts**

6.22.1 Detailed Description

Definitions of global variables controling DNG SDK behavior.

6.23 dng_host.h File Reference

```
#include "dng_auto_ptr.h"
#include "dng_classes.h"
#include "dng_errors.h"
#include "dng_types.h"
#include "dng_uncopyable.h"
```

Classes

- class dng_host

  The main class for communication between the application and the DNG SDK. Used to customize memory allocation and other behaviors.

6.23.1 Detailed Description

Class definition for dng_host, initial point of contact and control between host application and DNG SDK.

6.24 dng_hue_sat_map.h File Reference

```
#include "dng_classes.h"
#include "dng_fingerprint.h"
#include "dng_ref_counted_block.h"
#include "dng_safe_arithmetic.h"
#include "dng_types.h"
#include <atomic>
```

Classes

- class dng_hue_sat_map

  A 3D table that maps HSV (hue, saturation, and value) floating-point input coordinates in the range [0,1] to delta signals. The table must have at least 1 sample in the hue dimension, at least 2 samples in the saturation dimension, and at least 1 sample in the value dimension. Tables are stored in value-hue-saturation order.
- struct dng_hue_sat_map::HSBModify

6.24.1 Detailed Description

Table-based color correction data structure.

6.25 dng_ifd.h File Reference

```
#include "dng_fingerprint.h"
#include "dng_negative.h"
#include "dng_rect.h"
#include "dng_shared.h"
#include "dng_stream.h"
#include "dng_string.h"
#include "dng_sdk_limits.h"
#include "dng_tag_values.h"
```

Classes

- class dng_preview_info
- class dng_ifd

  Container for a single image file directory of a digital negative.

6.25.1 Detailed Description

DNG image file directory support.

6.26 dng_image.h File Reference

```
#include "dng_assertions.h"
#include "dng_classes.h"
#include "dng_pixel_buffer.h"
#include "dng_point.h"
#include "dng_rect.h"
#include "dng_tag_types.h"
#include "dng_types.h"
#include "dng_uncopyable.h"
```

Classes

- class dng_tile_buffer

  Class to get resource acquisition is instantiation behavior for tile buffers. Can be dirty or constant tile access.

- class dng_const_tile_buffer

  Class to get resource acquisition is instantiation behavior for constant (read-only) tile buffers.

- class dng_dirty_tile_buffer

  Class to get resource acquisition is instantiation behavior for dirty (writable) tile buffers.

- class dng_image

  Base class for holding image data in DNG SDK. See dng_simple_image for derived class most often used in DNG SDK.

6.26.1 Detailed Description

Support for working with image data in DNG SDK.

6.27 dng_image_writer.h File Reference

```
#include "dng_area_task.h"
#include "dng_auto_ptr.h"
#include "dng_classes.h"
#include "dng_fingerprint.h"
#include "dng_memory.h"
#include "dng_mutex.h"
#include "dng_point.h"
#include "dng_rational.h"
#include "dng_safe_arithmetic.h"
#include "dng_sdk_limits.h"
#include "dng_string.h"
#include "dng_tag_types.h"
#include "dng_tag_values.h"
#include "dng_types.h"
#include "dng_uncopyable.h"
#include <atomic>
```

Classes

- class dng_resolution

  Image resolution.
- class tiff_tag
- class tag_data_ptr
- class tag_string
- class tag_encoded_text
- class tag_uint8
- class tag_uint8_ptr
- class tag_uint16
- class tag_int16_ptr
- class tag_uint16_ptr
- class tag_uint32
- class tag_uint32_ptr
- class tag_urational
- class tag_urational_ptr
- class tag_srational
- class tag_srational_ptr
- class tag_real64
- class tag_matrix
- class tag_icc_profile
- class tag_cfa_pattern
- class tag_exif_date_time
- class tag_iptc
- class tag_xmp
- class dng_tiff_directory
- class dng_basic_tag_set
- class exif_tag_set
- class tiff_dng_extended_color_profile
- class tag_dng_noise_profile
- class dng_image_writer

  Support for writing dng_image or dng_negative instances to a dng_stream in TIFF or DNG format.
- class dng_write_tiles_task

Enumerations

- enum **dng_metadata_subset** {
  kMetadataSubset_CopyrightOnly = 0, **kMetadataSubset_CopyrightAndContact**, **kMetadataSubset_All←↩ExceptCameraInfo**, **kMetadataSubset_All**,
  kMetadataSubset_AllExceptLocationInfo, **kMetadataSubset_AllExceptCameraAndLocation**, **KMetadata←↩Subset_AllExceptCameraRawInfo**, **KMetadataSubset_AllExceptCameraRawInfoAndLocation**,
  kMetadataSubset_Last = KMetadataSubset_AllExceptCameraRawInfoAndLocation }

6.27.1 Detailed Description

Support for writing DNG images to files.

6.28 dng_info.h File Reference

```
#include "dng_auto_ptr.h"
#include "dng_classes.h"
#include "dng_errors.h"
#include "dng_exif.h"
#include "dng_ifd.h"
#include "dng_sdk_limits.h"
#include "dng_shared.h"
#include "dng_uncopyable.h"
#include <vector>
```

Classes

- class dng_info

    Top-level structure of DNG file with access to metadata.

6.28.1 Detailed Description

Class for holding top-level information about a DNG image.

6.29 dng_iptc.h File Reference

```
#include "dng_date_time.h"
#include "dng_string.h"
#include "dng_string_list.h"
```

Classes

- class dng_iptc

    Class for reading and holding IPTC metadata associated with a DNG file.

6.29.1 Detailed Description

Support for IPTC metadata within DNG files.

6.30 dng_lens_correction.h File Reference

```
#include "dng_1d_function.h"
#include "dng_matrix.h"
#include "dng_memory.h"
#include "dng_opcodes.h"
#include "dng_pixel_buffer.h"
#include "dng_point.h"
#include "dng_resample.h"
#include "dng_sdk_limits.h"
```

Classes

- class dng_warp_params

  Abstract base class holding common warp opcode parameters (e.g., number of planes, optical center) and common warp routines.
- class dng_warp_params_rectilinear

  Warp parameters for pinhole perspective rectilinear (not fisheye) camera model. Supports radial and tangential (decentering) distortion correction parameters.
- class dng_warp_params_fisheye

  Warp parameters for fisheye camera model (radial component only). Note the restrictions described below.
- class dng_opcode_WarpRectilinear

  Warp opcode for pinhole perspective (rectilinear) camera model.
- class dng_opcode_WarpFisheye

  Warp opcode for fisheye camera model.
- class dng_vignette_radial_params

  Radially-symmetric vignette (peripheral illuminational falloff) correction parameters.
- class dng_opcode_FixVignetteRadial

  Radially-symmetric lens vignette correction opcode.

6.30.1 Detailed Description

Opcodes to fix lens aberrations such as geometric distortion, lateral chromatic aberration, and vignetting (peripheral illumination falloff).

6.31 dng_linearization_info.h File Reference

```
#include "dng_auto_ptr.h"
#include "dng_classes.h"
#include "dng_memory.h"
#include "dng_rational.h"
#include "dng_rect.h"
#include "dng_sdk_limits.h"
```

Classes

- class dng_linearization_info

  Class for managing data values related to DNG linearization.

6.31.1 Detailed Description

Support for linearization table and black level tags.

6.32 dng_lossless_jpeg.h File Reference

```
#include "dng_classes.h"
#include "dng_types.h"
```

Classes

- class dng_spooler

Functions

- void **DecodeLosslessJPEG** (dng_stream &stream, dng_spooler &spooler, uint32 minDecodedSize, uint32 maxDecodedSize, bool bug16, uint64 endOfData)
- void **EncodeLosslessJPEG** (const uint16 ∗srcData, uint32 srcRows, uint32 srcCols, uint32 srcChannels, uint32 srcBitDepth, int32 srcRowStep, int32 srcColStep, dng_stream &stream)

6.32.1 Detailed Description

Functions for encoding and decoding lossless JPEG format.

6.33 dng_matrix.h File Reference

```
#include "dng_sdk_limits.h"
#include "dng_types.h"
```

Classes

- class dng_matrix

  Class to represent 2D matrix up to kMaxColorPlanes x kMaxColorPlanes in size.
- class dng_matrix_3by3

  A 3x3 matrix.
- class dng_matrix_4by3

  A 4x3 matrix. Handy for working with 4-color cameras.
- class dng_matrix_4by4

  A 4x4 matrix. Handy for GPU APIs.
- class dng_vector

  Class to represent 1-dimensional vector with up to kMaxColorPlanes components.
- class dng_vector_3

  A 3-element vector.
- class dng_vector_4

  A 4-element vector.

Functions

- dng_matrix **operator** ∗ (const dng_matrix &A, const dng_matrix &B)
- dng_vector **operator** ∗ (const dng_matrix &A, const dng_vector &B)
- dng_matrix **operator** ∗ (real64 scale, const dng_matrix &A)
- dng_vector **operator** ∗ (real64 scale, const dng_vector &A)
- dng_matrix **operator+** (const dng_matrix &A, const dng_matrix &B)
- dng_vector **operator-** (const dng_vector &a, const dng_vector &b)
- dng_matrix **Transpose** (const dng_matrix &A)
- dng_matrix **Invert** (const dng_matrix &A)
- dng_matrix **Invert** (const dng_matrix &A, const dng_matrix &hint)
- real64 **MaxEntry** (const dng_matrix &A)
- real64 **MaxEntry** (const dng_vector &A)
- real64 **MinEntry** (const dng_matrix &A)
- real64 **MinEntry** (const dng_vector &A)
- real64 **Dot** (const dng_vector &a, const dng_vector &b)
- real64 **Distance** (const dng_vector &a, const dng_vector &b)

6.33.1 Detailed Description

Matrix and vector classes, including specialized 3x3 and 4x3 versions as well as length 3 vectors.

6.34 dng_memory_stream.h File Reference

```
#include "dng_stream.h"
```

Classes

- class dng_memory_stream

  A dng_stream which can be read from or written to memory.

6.34.1 Detailed Description

Stream abstraction to/from in-memory data.

6.35 dng_misc_opcodes.h File Reference

```
#include "dng_classes.h"
#include "dng_opcodes.h"
```

Classes

- class dng_opcode_TrimBounds

  Opcode to trim image to a specified rectangle.
- class dng_area_spec

  A class to describe an area of an image, including a pixel subrectangle, plane range, and row/column pitch (e.g., for mosaic images). Useful for specifying opcodes that only apply to specific color planes or pixel types (e.g., only one of the two green Bayer pixels).
- class dng_opcode_MapTable

  An opcode to apply a 1D function (represented as a 16-bit table) to an image area.
- class dng_opcode_MapPolynomial

  An opcode to apply a 1D function (represented as a polynomial) to an image area.
- class dng_opcode_DeltaPerRow

  An opcode to apply a delta (i.e., offset) that varies per row. Within a row, the same delta value is applied to all specified pixels.
- class dng_opcode_DeltaPerColumn

  An opcode to apply a delta (i.e., offset) that varies per column. Within a column, the same delta value is applied to all specified pixels.
- class dng_opcode_ScalePerRow

  An opcode to apply a scale factor that varies per row. Within a row, the same scale factor is applied to all specified pixels.
- class dng_opcode_ScalePerColumn

  An opcode to apply a scale factor that varies per column. Within a column, the same scale factor is applied to all specified pixels.

6.35.1 Detailed Description

Miscellaneous DNG opcodes.

6.36 dng_mosaic_info.h File Reference

```
#include "dng_classes.h"
#include "dng_rect.h"
#include "dng_sdk_limits.h"
#include "dng_types.h"
```

Classes

- class dng_mosaic_info

  Support for describing color filter array patterns and manipulating mosaic sample data.

6.36.1 Detailed Description

Support for descriptive information about color filter array patterns.

6.37 dng_negative.h File Reference

```
#include "dng_1d_function.h"
#include "dng_auto_ptr.h"
#include "dng_classes.h"
#include "dng_fingerprint.h"
#include "dng_image.h"
#include "dng_linearization_info.h"
#include "dng_matrix.h"
#include "dng_mosaic_info.h"
#include "dng_mutex.h"
#include "dng_opcode_list.h"
#include "dng_orientation.h"
#include "dng_rational.h"
#include "dng_sdk_limits.h"
#include "dng_string.h"
#include "dng_tag_types.h"
#include "dng_tag_values.h"
#include "dng_types.h"
#include "dng_utils.h"
#include "dng_xy_coord.h"
#include <vector>
```

Classes

- class dng_noise_function

  Noise model for photon and sensor read noise, assuming that they are independent random variables and spatially invariant.
- class dng_noise_profile

  Noise profile for a negative.
- class dng_metadata

  Main class for holding metadata.
- class dng_negative

  Main class for holding DNG image data and associated metadata.

Macros

- #define **qMetadataOnConst** 0
- #define **METACONST**

6.37.1 Detailed Description

Functions and classes for working with a digital negative (image data and corresponding metadata).

6.38 dng_opcode_list.h File Reference

```
#include "dng_auto_ptr.h"
#include "dng_classes.h"
#include "dng_opcodes.h"
#include "dng_uncopyable.h"
#include <vector>
```

Classes

- class dng_opcode_list

    A list of opcodes.

6.38.1 Detailed Description

List of opcodes.

6.39 dng_opcodes.h File Reference

```
#include "dng_auto_ptr.h"
#include "dng_classes.h"
#include "dng_rect.h"
#include "dng_types.h"
```

Classes

- class dng_opcode

    Virtual base class for opcode.
- class dng_opcode_Unknown

    Class to represent unknown opcodes (e.g, opcodes defined in future DNG versions).
- class dng_filter_opcode

    Class to represent a filter opcode, such as a convolution.
- class dng_inplace_opcode

    Class to represent an in-place (i.e., pointwise, per-pixel) opcode, such as a global tone curve.

Enumerations

- enum dng_opcode_id {
  dngOpcode_Private = 0, **dngOpcode_WarpRectilinear** = 1, **dngOpcode_WarpFisheye** = 2, **dngOpcode_↩ FixVignetteRadial** = 3,
  dngOpcode_FixBadPixelsConstant = 4, **dngOpcode_FixBadPixelsList** = 5, **dngOpcode_TrimBounds** = 6, **dngOpcode_MapTable** = 7,
  dngOpcode_MapPolynomial = 8, **dngOpcode_GainMap** = 9, **dngOpcode_DeltaPerRow** = 10, **dngOpcode↩ _DeltaPerColumn** = 11,
  dngOpcode_ScalePerRow = 12, **dngOpcode_ScalePerColumn** = 13 }

  List of supported opcodes (by ID).

6.39.1 Detailed Description

Base class and common data structures for opcodes (introduced in DNG 1.3).

6.40 dng_pixel_buffer.h File Reference

```
#include "dng_assertions.h"
#include "dng_rect.h"
#include "dng_safe_arithmetic.h"
#include "dng_tag_types.h"
```

Classes

- class dng_pixel_buffer

  Holds a buffer of pixel data with "pixel geometry" metadata.

Macros

- #define **qDebugPixelType** 0
- #define **ASSERT_PIXEL_TYPE**(typeVal) DNG_ASSERT (fPixelType == typeVal, "Pixel type access mismatch")

Functions

- void OptimizeOrder (const void ∗&sPtr, void ∗&dPtr, uint32 sPixelSize, uint32 dPixelSize, uint32 &count0, uint32 &count1, uint32 &count2, int32 &sStep0, int32 &sStep1, int32 &sStep2, int32 &dStep0, int32 &dStep1, int32 &dStep2)

  Compute best set of step values for a given source and destination area and stride.
- void **OptimizeOrder** (const void ∗&sPtr, uint32 sPixelSize, uint32 &count0, uint32 &count1, uint32 &count2, int32 &sStep0, int32 &sStep1, int32 &sStep2)
- void **OptimizeOrder** (void ∗&dPtr, uint32 dPixelSize, uint32 &count0, uint32 &count1, uint32 &count2, int32 &dStep0, int32 &dStep1, int32 &dStep2)

6.40.1 Detailed Description

Support for holding buffers of sample data.

6.41 dng_rational.h File Reference

```
#include "dng_types.h"
```

Classes

- class dng_srational
- class dng_urational

6.41.1 Detailed Description

Signed and unsigned rational data types.

6.42 dng_read_image.h File Reference

```
#include "dng_area_task.h"
#include "dng_auto_ptr.h"
#include "dng_classes.h"
#include "dng_image.h"
#include "dng_memory.h"
#include "dng_mutex.h"
#include "dng_types.h"
```

Classes

- class dng_row_interleaved_image
- class dng_read_image
- class dng_read_tiles_task

Functions

- bool **DecodePackBits** (dng_stream &stream, uint8 ∗dPtr, int32 dstCount)

6.42.1 Detailed Description

Support for DNG image reading.

6.43 dng_render.h File Reference

```
#include "dng_1d_function.h"
#include "dng_auto_ptr.h"
#include "dng_classes.h"
#include "dng_spline.h"
#include "dng_uncopyable.h"
#include "dng_xy_coord.h"
```

Classes

- class dng_function_zero_offset

  Curve for removing zero offset from stage3 image.
- class dng_function_exposure_ramp

  Curve for pre-exposure-compensation adjustment based on noise floor, shadows, and highlight level.
- class dng_function_exposure_tone

  Exposure compensation curve for a given compensation amount in stops using quadric for roll-off.
- class dng_tone_curve_acr3_default

  Default ACR3 tone curve.
- class dng_function_gamma_encode

  Encoding gamma curve for a given color space.
- class dng_render

  Class used to render digital negative to displayable image.

6.43.1 Detailed Description

Classes for conversion of RAW data to final image.

6.44 dng_sdk_limits.h File Reference

```
#include "dng_types.h"
```

Variables

- const uint32 kMaxDNGPreviews = 20
- const uint32 kMaxSubIFDs = kMaxDNGPreviews + 1

  The maximum number of SubIFDs that will be parsed.
- const uint32 kMaxChainedIFDs = 10

  The maximum number of chained IFDs that will be parsed.
- const uint32 kMaxSamplesPerPixel = 4

  The maximum number of samples per pixel.
- const uint32 kMaxColorPlanes = kMaxSamplesPerPixel

  Maximum number of color planes.

- const uint32 kMaxCFAPattern = 8

  The maximum size of a CFA repeating pattern.
- const uint32 kMaxBlackPattern = 8

  The maximum size of a black level repeating pattern.
- const uint32 kMaxMaskedAreas = 4

  The maximum number of masked area rectangles.
- const uint32 kMaxImageSide = 65000

  The maximum image size supported (pixels per side).
- const uint32 kMaxToneCurvePoints = 8192

  The maximum number of tone curve points supported.
- const uint32 kMaxMPThreads = 8

  Maximum number of MP threads for dng_area_task operations.
- const real64 kMaxStage3BlackLevelNormalized = 0.2

  Maximum supported value of Stage3BlackLevelNormalized.

6.44.1 Detailed Description

Collection of constants detailing maximum values used in processing in the DNG SDK.

6.44.2 Variable Documentation

6.44.2.1 kMaxDNGPreviews

```
const uint32 kMaxDNGPreviews = 20
```

The maximum number of previews (in addition to the main IFD's thumbnail) that we support embedded in a DNG.

6.45 dng_string.h File Reference

```
#include "dng_types.h"
#include "dng_memory.h"
```

Classes

- class dng_string

6.45.1 Detailed Description

Text string representation.

6.46 dng_temperature.h File Reference

```
#include "dng_classes.h"
#include "dng_types.h"
```

Classes

- class dng_temperature

6.46.1 Detailed Description

Representation of color temperature and offset (tint) using black body radiator definition.

6.47 dng_tone_curve.h File Reference

```
#include "dng_classes.h"
#include "dng_memory.h"
#include "dng_point.h"
```

Classes

- class dng_tone_curve

6.47.1 Detailed Description

Representation of 1-dimensional tone curve.

6.48 dng_xy_coord.h File Reference

```
#include "dng_classes.h"
#include "dng_types.h"
```

Classes

- class dng_xy_coord

Functions

- dng_xy_coord **operator+** (const dng_xy_coord &A, const dng_xy_coord &B)
- dng_xy_coord **operator-** (const dng_xy_coord &A, const dng_xy_coord &B)
- dng_xy_coord **operator** ∗ (real64 scale, const dng_xy_coord &A)
- real64 **operator** ∗ (const dng_xy_coord &A, const dng_xy_coord &B)
- dng_xy_coord **StdA_xy_coord** ()
- dng_xy_coord **D50_xy_coord** ()
- dng_xy_coord **D55_xy_coord** ()
- dng_xy_coord **D65_xy_coord** ()
- dng_xy_coord **D75_xy_coord** ()
- dng_xy_coord **XYZtoXY** (const dng_vector_3 &coord)
- dng_vector_3 **XYtoXYZ** (const dng_xy_coord &coord)
- dng_xy_coord **PCStoXY** ()
- dng_vector_3 **PCStoXYZ** ()

6.48.1 Detailed Description

Representation of colors in xy and XYZ coordinates.

Index