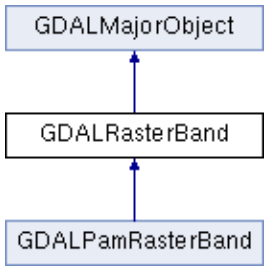
GDALRasterBand Class Reference abstract

A single raster band (or channel). More...

`#include <gdal_priv.h>`

Inheritance diagram for GDALRasterBand:



Public Member Functions

	GDALRasterBand ()
	GDALRasterBand (int bForceCachedIO) Constructor. More...
	~GDALRasterBand () override
int	**GetXSize** () Fetch XSize of raster. More...
int	**GetYSize** () Fetch YSize of raster. More...
int	**GetBand** () Fetch the band number. More...
GDALDataset *	**GetDataset** () Fetch the owning dataset handle. More...
GDALDataType	**GetRasterDataType** (void) Fetch the pixel data type for this band. More...
void	**GetBlockSize** (int *, int *) Fetch the "natural" block size of this band. More...
CPLErr	**GetActualBlockSize** (int, int, int *, int *) Fetch the actual block size for a given block offset. More...
GDALAccess	**GetAccess** () Find out if we have update permission for this band. More...
CPLErr	**RasterIO** (**GDALRWFlag**, int, int, int, int, void *, int, int, **GDALDataType**, **GSpacing**, **GSpacing**, **GDALRasterIOExtraArg** *psExtraArg) **CPL_WARN_UNUSED_RESULT** Read/write a region of image data for this band. More...
CPLErr	**ReadBlock** (int, int, void *) **CPL_WARN_UNUSED_RESULT** Read a block of image data efficiently. More...
CPLErr	**WriteBlock** (int, int, void *) **CPL_WARN_UNUSED_RESULT** Write a block of image data efficiently. More...
GDALRasterBlock *	**GetLockedBlockRef** (int nXBlockOff, int nYBlockOff, int

	bJustInitialize=FALSE) **CPL_WARN_UNUSED_RESULT** Fetch a pointer to an internally cached raster block. More...
CPLErr	**FlushBlock** (int, int, int bWriteDirtyBlock=TRUE) Flush a block out of the block cache. More...
unsigned char *	**GetIndexColorTranslationTo** (**GDALRasterBand** *poReferenceBand, unsigned char *pTranslationTable=nullptr, int *pApproximateMatching=nullptr) Compute translation table for color tables. More...
virtual **CPLErr**	**FlushCache** () Flush raster data cache. More...
virtual char **	**GetCategoryNames** () Fetch the list of category names for this raster. More...
virtual double	**GetNoDataValue** (int *pbSuccess=nullptr) Fetch the no data value for this band. More...
virtual double	**GetMinimum** (int *pbSuccess=nullptr) Fetch the minimum value for this band. More...
virtual double	**GetMaximum** (int *pbSuccess=nullptr) Fetch the maximum value for this band. More...
virtual double	**GetOffset** (int *pbSuccess=nullptr) Fetch the raster value offset. More...
virtual double	**GetScale** (int *pbSuccess=nullptr) Fetch the raster value scale. More...
virtual const char *	**GetUnitType** () Return raster unit type. More...
virtual **GDALColorInterp**	**GetColorInterpretation** () How should this band be interpreted as color? More...
virtual **GDALColorTable** *	**GetColorTable** () Fetch the color table associated with band. More...
virtual **CPLErr**	**Fill** (double dfRealValue, double dfImaginaryValue=0) Fill this band with a constant value. More...
virtual **CPLErr**	**SetCategoryNames** (char **papszNames) Set the category names for this band. More...
virtual **CPLErr**	**SetNoDataValue** (double dfNoData) Set the no data value for this band. More...
virtual **CPLErr**	**DeleteNoDataValue** () Remove the no data value for this band. More...
virtual **CPLErr**	**SetColorTable** (**GDALColorTable** *poCT) Set the raster color table. More...
virtual **CPLErr**	**SetColorInterpretation** (**GDALColorInterp** eColorInterp) Set color interpretation of a band. More...
virtual **CPLErr**	**SetOffset** (double dfNewOffset) Set scaling offset. More...
virtual **CPLErr**	**SetScale** (double dfNewScale)

	Set scaling ratio. More...
virtual **CPLErr**	**SetUnitType** (const char *pszNewValue) Set unit type. More...
virtual **CPLErr**	**GetStatistics** (int bApproxOK, int bForce, double *pdfMin, double *pdfMax, double *pdfMean, double *padfStdDev) Fetch image statistics. More...
virtual **CPLErr**	**ComputeStatistics** (int bApproxOK, double *pdfMin, double *pdfMax, double *pdfMean, double *pdfStdDev, GDALProgressFunc, void *pProgressData) Compute image statistics. More...
virtual **CPLErr**	**SetStatistics** (double dfMin, double dfMax, double dfMean, double dfStdDev) Set statistics on band. More...
virtual **CPLErr**	**ComputeRasterMinMax** (int, double *) Compute the min/max values for a band. More...
char **	**GetMetadata** (const char *pszDomain="") override Fetch metadata. More...
CPLErr	**SetMetadata** (char **papszMetadata, const char *pszDomain) override Set metadata. More...
const char *	**GetMetadataItem** (const char *pszName, const char *pszDomain) override Fetch single metadata item. More...
CPLErr	**SetMetadataItem** (const char *pszName, const char *pszValue, const char *pszDomain) override Set single metadata item. More...
virtual int	**HasArbitraryOverviews** () Check for arbitrary overviews. More...
virtual int	**GetOverviewCount** () Return the number of overview layers available. More...
virtual **GDALRasterBand** *	**GetOverview** (int) Fetch overview raster band object. More...
virtual **GDALRasterBand** *	**GetRasterSampleOverview** (**GUIntBig**) Fetch best sampling overview. More...
virtual **CPLErr**	**BuildOverviews** (const char *pszResampling, int nOverviews, int *panOverviewList, GDALProgressFunc pfnProgress, void *pProgressData) Build raster overview(s) More...
virtual **CPLErr**	**AdviseRead** (int nXOff, int nYOff, int nXSize, int nYSize, int nBufXSize, int nBufYSize, **GDALDataType** eBufType, char **papszOptions) Advise driver of upcoming read requests. More...
virtual **CPLErr**	**GetHistogram** (double dfMin, double dfMax, int nBuckets, **GUIntBig** *panHistogram, int bIncludeOutOfRange, int bApproxOK, GDALProgressFunc, void *pProgressData)

	Compute raster histogram. More...
virtual **CPLErr**	**GetDefaultHistogram** (double *pdfMin, double *pdfMax, int *pnBuckets, **GUIntBig** **ppanHistogram, int bForce, GDALProgressFunc, void *pProgressData) Fetch default raster histogram. More...
virtual **CPLErr**	**SetDefaultHistogram** (double dfMin, double dfMax, int nBuckets, **GUIntBig** *panHistogram) Set default histogram. More...
virtual **GDALRasterAttributeTable** *	**GetDefaultRAT** () Fetch default Raster Attribute Table. More...
virtual **CPLErr**	**SetDefaultRAT** (const **GDALRasterAttributeTable** *poRAT) Set default Raster Attribute Table. More...
virtual **GDALRasterBand** *	**GetMaskBand** () Return the mask band associated with the band. More...
virtual int	**GetMaskFlags** () Return the status flags of the mask band associated with the band. More...
virtual **CPLErr**	**CreateMaskBand** (int nFlagsIn) Adds a mask band to the current band. More...
virtual **CPLVirtualMem** *	**GetVirtualMemAuto** (**GDALRWFlag** eRWFlag, int *pnPixelSpace, **GIntBig** *pnLineSpace, char **papszOptions) **CPL_WARN_UNUSED_RESULT** Create a CPLVirtualMem object from a GDAL raster band object. More...
int	**GetDataCoverageStatus** (int nXOff, int nYOff, int nXSize, int nYSize, int nMaskFlagStop=0, double *pdfDataPct=nullptr) Get the coverage status of a sub-window of the raster. More...
void	**ReportError** (**CPLErr** eErrClass, **CPLErrorNum** err_no, const char *fmt,...) **CPL_PRINT_FUNC_FORMAT**(4 Emits an error related to a raster band. More...

▸ **Public Member Functions inherited from GDALMajorObject**

Static Public Member Functions

void static **GDALRasterBandH**	**ToHandle** (**GDALRasterBand** *poBand) Convert a GDALRasterBand* to a GDALRasterBandH. More...
static **GDALRasterBand** *	**FromHandle** (**GDALRasterBandH** hBand) Convert a GDALRasterBandH to a GDALRasterBand*. More...

▸ **Static Public Member Functions inherited from GDALMajorObject**

Protected Member Functions

virtual **CPLErr**	**IReadBlock** (int nBlockXOff, int nBlockYOff, void *pData)=0 Read a block of data. More...
virtual **CPLErr**	**IWriteBlock** (int nBlockXOff, int nBlockYOff, void *pData)

	Write a block of data. More...
virtual **CPLErr**	**IRasterIO** (**GDALRWFlag**, int, int, int, int, void *, int, int, **GDALDataType**, **GSpacing**, **GSpacing**, **GDALRasterIOExtraArg** *psExtraArg) **CPL_WARN_UNUSED_RESULT** Read/write a region of image data for this band. More...
virtual int	**IGetDataCoverageStatus** (int nXOff, int nYOff, int nXSize, int nYSize, int nMaskFlagStop, double *pdfDataPct) Get the coverage status of a sub-window of the raster. More...
GDALRasterBlock *	**TryGetLockedBlockRef** (int nXBlockOff, int nYBlockYOff) Try fetching block ref. More...

▸ **Protected Member Functions inherited from GDALMajorObject**

Friends

class	**GDALArrayBandBlockCache**
class	**GDALHashSetBandBlockCache**
class	**GDALRasterBlock**

Detailed Description

A single raster band (or channel).

Constructor & Destructor Documentation

GDALRasterBand::GDALRasterBand ( )

Constructor. Applications should never create GDALRasterBands directly.

GDALRasterBand::GDALRasterBand (int bForceCachedIOIn) explicit

Constructor.

Applications should never create GDALRasterBands directly.

Parameters

bForceCachedIOIn Whether cached IO should be forced.

GDALRasterBand::~GDALRasterBand ( ) override

Destructor. Applications should never destroy GDALRasterBands directly, instead destroy the **GDALDataset**.

Member Function Documentation

```
CPLErr GDALRasterBand::AdviseRead ( int           nXOff,
                                    int           nYOff,
                                    int           nXSize,
                                    int           nYSize,
                                    int           nBufXSize,
                                    int           nBufYSize,
                                    GDALDataType  eBufType,
                                    char **       papszOptions
                                  )                                  virtual
```

Advise driver of upcoming read requests.

Some GDAL drivers operate more efficiently if they know in advance what set of upcoming read requests will be made. The **AdviseRead()** method allows an application to notify the driver of the region of interest, and at what resolution the region will be read.

Many drivers just ignore the **AdviseRead()** call, but it can dramatically accelerate access via some drivers.

Depending on call paths, drivers might receive several calls to **AdviseRead()** with the same parameters.

Parameters

nXOff	The pixel offset to the top left corner of the region of the band to be accessed. This would be zero to start from the left side.
nYOff	The line offset to the top left corner of the region of the band to be accessed. This would be zero to start from the top.
nXSize	The width of the region of the band to be accessed in pixels.
nYSize	The height of the region of the band to be accessed in lines.
nBufXSize	the width of the buffer image into which the desired region is to be read, or from which it is to be written.
nBufYSize	the height of the buffer image into which the desired region is to be read, or from which it is to be written.
eBufType	the type of the pixel values in the pData data buffer. The pixel values will automatically be translated to/from the **GDALRasterBand** data type as needed.
papszOptions	a list of name=value strings with special control options. Normally this is NULL.

Returns

CE_Failure if the request is invalid and CE_None if it works or is ignored.

```
CPLErr GDALRasterBand::BuildOverviews ( const char *        pszResampling,
                                        int                 nOverviews,
                                        int *               panOverviewList,
                                        GDALProgressFunc    pfnProgress,
                                        void *              pProgressData
                                      )                                          virtual
```

Build raster overview(s)

If the operation is unsupported for the indicated dataset, then CE_Failure is returned, and CPLGetLastErrorNo() will return CPLE_NotSupported.

WARNING: It is not possible to build overviews for a single band in TIFF format, and thus this method does not work for TIFF format, or any formats that use the default overview building in TIFF format. Instead it is necessary to build overviews on the dataset as a whole using GDALDataset::BuildOverviews(). That makes this method pretty useless from a practical point of view.

Parameters

pszResampling	one of "NEAREST", "GAUSS", "CUBIC", "AVERAGE", "MODE", "AVERAGE_MAGPHASE" or "NONE" controlling the downsampling method applied.
nOverviews	number of overviews to build.
panOverviewList	the list of overview decimation factors to build.
pfnProgress	a function to call to report progress, or NULL.
pProgressData	application data to pass to the progress function.

Returns

CE_None on success or CE_Failure if the operation doesn't work.

CPLErr GDALRasterBand::ComputeRasterMinMax (int bApproxOK,
double * adfMinMax
) virtual

Compute the min/max values for a band.

If approximate is OK, then the band's **GetMinimum()**/GetMaximum() will be trusted. If it doesn't work, a subsample of blocks will be read to get an approximate min/max. If the band has a nodata value it will be excluded from the minimum and maximum.

If bApprox is FALSE, then all pixels will be read and used to compute an exact range.

This method is the same as the C function **GDALComputeRasterMinMax()**.

Parameters

bApproxOK TRUE if an approximate (faster) answer is OK, otherwise FALSE.

adfMinMax the array in which the minimum (adfMinMax[0]) and the maximum (adfMinMax[1]) are returned.

Returns

CE_None on success or CE_Failure on failure.

```
CPLErr GDALRasterBand::ComputeStatistics ( int             bApproxOK,
                                           double *        pdfMin,
                                           double *        pdfMax,
                                           double *        pdfMean,
                                           double *        pdfStdDev,
                                           GDALProgressFunc pfnProgress,
                                           void *          pProgressData
                                         )                                  virtual
```

Compute image statistics.

Returns the minimum, maximum, mean and standard deviation of all pixel values in this band. If approximate statistics are sufficient, the bApproxOK flag can be set to true in which case overviews, or a subset of image tiles may be used in computing the statistics.

Once computed, the statistics will generally be "set" back on the raster band using **SetStatistics()**.

This method is the same as the C function **GDALComputeRasterStatistics()**.

Parameters

bApproxOK	If TRUE statistics may be computed based on overviews or a subset of all tiles.
pdfMin	Location into which to load image minimum (may be NULL).
pdfMax	Location into which to load image maximum (may be NULL).-
pdfMean	Location into which to load image mean (may be NULL).
pdfStdDev	Location into which to load image standard deviation (may be NULL).
pfnProgress	a function to call to report progress, or NULL.
pProgressData	application data to pass to the progress function.

Returns

CE_None on success, or CE_Failure if an error occurs or processing is terminated by the user.

CPLErr GDALRasterBand::CreateMaskBand (int nFlagsIn) virtual

Adds a mask band to the current band.

The default implementation of the **CreateMaskBand()** method is implemented based on similar rules to the .ovr handling implemented using the GDALDefaultOverviews object. A TIFF file with the extension .msk will be created with the same basename as the original file, and it will have as many bands as the original image (or just one for GMF_PER_DATASET). The mask images will be deflate compressed tiled images with the same block size as the original image if possible. It will have INTERNAL_MASK_FLAGS_xx metadata items set at the dataset level, where xx matches the band number of a band of the main dataset. The value of those items will be the one of the nFlagsIn parameter.

Note that if you got a mask band with a previous call to **GetMaskBand()**, it might be invalidated by **CreateMaskBand()**. So you have to call **GetMaskBand()** again.

This method is the same as the C function **GDALCreateMaskBand()**.

Since

GDAL 1.5.0

Parameters

nFlagsIn 0 or combination of GMF_PER_DATASET / GMF_ALPHA.

Returns

CE_None on success or CE_Failure on an error.

See also

http://trac.osgeo.org/gdal/wiki/rfc15_nodatabitmask

GDALDataset::CreateMaskBand()

CPLErr GDALRasterBand::DeleteNoDataValue ( ) virtual

Remove the no data value for this band.

This method is the same as the C function **GDALDeleteRasterNoDataValue()**.

Returns

CE_None on success, or CE_Failure on failure. If unsupported by the driver, CE_Failure is returned by no error message will have been emitted.

Since

GDAL 2.1

Reimplemented in **GDALPamRasterBand**.

CPLErr GDALRasterBand::Fill (double dfRealValue,
double dfImaginaryValue = 0
) virtual

Fill this band with a constant value.

GDAL makes no guarantees about what values pixels in newly created files are set to, so this method can be used to clear a band to a specified "default" value. The fill value is passed in as a double but this will be converted to the underlying type before writing to the file. An optional second argument allows the imaginary component of a complex constant value to be specified.

This method is the same as the C function **GDALFillRaster()**.

Parameters

dfRealValue	Real component of fill value
dfImaginaryValue	Imaginary component of fill value, defaults to zero

Returns

CE_Failure if the write fails, otherwise CE_None

CPLErr GDALRasterBand::FlushBlock (int nXBlockOff,
int nYBlockOff,
int bWriteDirtyBlock = TRUE
)

Flush a block out of the block cache.

Parameters

nXBlockOff	block x offset
nYBlockOff	blocky offset
bWriteDirtyBlock	whether the block should be written to disk if dirty.

Returns

CE_None in case of success, an error code otherwise.

CPLErr GDALRasterBand::FlushCache (void) virtual

Flush raster data cache.

This call will recover memory used to cache data blocks for this raster band, and ensure that new requests are referred to the underlying driver.

This method is the same as the C function **GDALFlushRasterCache()**.

Returns

CE_None on success.

static GDALRasterBand* GDALRasterBand::FromHandle (GDALRasterBandH hBand) inline static

Convert a GDALRasterBandH to a GDALRasterBand*.

Since

GDAL 2.3

GDALAccess GDALRasterBand::GetAccess ( )

Find out if we have update permission for this band.

This method is the same as the C function GDALGetRasterAccess().

Returns

Either GA_Update or GA_ReadOnly.

CPLErr GDALRasterBand::GetActualBlockSize (int nXBlockOff,
int nYBlockOff,
int * pnXValid,
int * pnYValid
)

Fetch the actual block size for a given block offset.

Handles partial blocks at the edges of the raster and returns the true number of pixels

Parameters

nXBlockOff the horizontal block offset for which to calculate the number of valid pixels, with zero indicating the left most block, 1 the next block and so forth.

nYBlockOff the vertical block offset, with zero indicating the left most block, 1 the next block and so forth.

pnXValid pointer to an integer in which the number of valid pixels in the x direction will be stored

pnYValid pointer to an integer in which the number of valid pixels in the y direction will be stored

Returns

CE_None if the input parameter are valid, CE_Failure otherwise

Since

GDAL 2.2

int GDALRasterBand::GetBand ( )

Fetch the band number.

This method returns the band that this **GDALRasterBand** objects represents within its dataset. This method may return a value of 0 to indicate **GDALRasterBand** objects without an apparently relationship to a dataset, such as GDALRasterBands serving as overviews.

This method is the same as the C function **GDALGetBandNumber()**.

Returns

band number (1+) or 0 if the band number isn't known.

void GDALRasterBand::GetBlockSize (int * pnXSize, int * pnYSize)

Fetch the "natural" block size of this band.

GDAL contains a concept of the natural block size of rasters so that applications can organized data access efficiently for some file formats. The natural block size is the block size that is most efficient for accessing the format. For many formats this is simple a whole scanline in which case *pnXSize is set to **GetXSize()**, and *pnYSize is set to 1.

However, for tiled images this will typically be the tile size.

Note that the X and Y block sizes don't have to divide the image size evenly, meaning that right and bottom edge blocks may be incomplete. See **ReadBlock()** for an example of code dealing with these issues.

This method is the same as the C function **GDALGetBlockSize()**.

Parameters

pnXSize integer to put the X block size into or NULL.

pnYSize integer to put the Y block size into or NULL.

char ** GDALRasterBand::GetCategoryNames ( ) virtual

Fetch the list of category names for this raster.

The return list is a "StringList" in the sense of the CPL functions. That is a NULL terminated array of strings. Raster values without associated names will have an empty string in the returned list. The first entry in the list is for raster values of zero, and so on.

The returned stringlist should not be altered or freed by the application. It may change on the next GDAL call, so please copy it if it is needed for any period of time.

This method is the same as the C function **GDALGetRasterCategoryNames()**.

Returns

list of names, or NULL if none.

Reimplemented in **GDALPamRasterBand**.

GDALColorInterp GDALRasterBand::GetColorInterpretation ( ) virtual

How should this band be interpreted as color?

GCI_Undefined is returned when the format doesn't know anything about the color interpretation.

This method is the same as the C function **GDALGetRasterColorInterpretation()**.

Returns

color interpretation value for band.

Reimplemented in **GDALPamRasterBand**.

GDALColorTable * GDALRasterBand::GetColorTable ( ) virtual

Fetch the color table associated with band.

If there is no associated color table, the return result is NULL. The returned color table remains owned by the **GDALRasterBand**, and can't be depended on for long, nor should it ever be modified by the caller.

This method is the same as the C function **GDALGetRasterColorTable()**.

Returns

internal color table, or NULL.

Reimplemented in **GDALPamRasterBand**.

```
int GDALRasterBand::GetDataCoverageStatus ( int        nXOff,
                                            int        nYOff,
                                            int        nXSize,
                                            int        nYSize,
                                            int        nMaskFlagStop = 0,
                                            double *   pdfDataPct = nullptr
                                          )
```

Get the coverage status of a sub-window of the raster.

Returns whether a sub-window of the raster contains only data, only empty blocks or a mix of both. This function can be used to determine quickly if it is worth issuing RasterIO / ReadBlock requests in datasets that may be sparse.

Empty blocks are blocks that contain only pixels whose value is the nodata value when it is set, or whose value is 0 when the nodata value is not set.

The query is done in an efficient way without reading the actual pixel values. If not possible, or not implemented at all by the driver, GDAL_DATA_COVERAGE_STATUS_UNIMPLEMENTED | GDAL_DATA_COVERAGE_STATUS_DATA will be returned.

The values that can be returned by the function are the following, potentially combined with the binary or operator :

- GDAL_DATA_COVERAGE_STATUS_UNIMPLEMENTED : the driver does not implement **GetDataCoverageStatus()**. This flag should be returned together with GDAL_DATA_COVERAGE_STATUS_DATA.
- GDAL_DATA_COVERAGE_STATUS_DATA: There is (potentially) data in the queried window.
- GDAL_DATA_COVERAGE_STATUS_EMPTY: There is nodata in the queried window. This is typically identified by the concept of missing block in formats that supports it.

Note that GDAL_DATA_COVERAGE_STATUS_DATA might have false positives and should be interpreted more as hint of potential presence of data. For example if a GeoTIFF file is created with blocks filled with zeroes (or set to the nodata value), instead of using the missing block mechanism, GDAL_DATA_COVERAGE_STATUS_DATA will be returned. On the contrary, GDAL_DATA_COVERAGE_STATUS_EMPTY should have no false positives.

The nMaskFlagStop should be generally set to 0. It can be set to a binary-or'ed mask of the above mentioned values to enable a quick exiting of the function as soon as the computed mask matches the nMaskFlagStop. For example, you can issue a request on the whole raster with nMaskFlagStop = GDAL_DATA_COVERAGE_STATUS_EMPTY. As soon as one missing block is encountered, the function will exit, so that you can potentially refine the requested area to find which particular region(s) have missing blocks.

See also

GDALGetDataCoverageStatus()

Parameters

nXOff	The pixel offset to the top left corner of the region of the band to be queried. This would be zero to start from the left side.

nYOff	The line offset to the top left corner of the region of the band to be queried. This would be zero to start from the top.
nXSize	The width of the region of the band to be queried in pixels.
nYSize	The height of the region of the band to be queried in lines.
nMaskFlagStop	0, or a binary-or'ed mask of possible values GDAL_DATA_COVERAGE_STATUS_UNIMPLEMENTED, GDAL_DATA_COVERAGE_STATUS_DATA and GDAL_DATA_COVERAGE_STATUS_EMPTY. As soon as the computation of the coverage matches the mask, the computation will be stopped. *pdfDataPct will not be valid in that case.
pdfDataPct	Optional output parameter whose pointed value will be set to the (approximate) percentage in [0,100] of pixels in the queried sub-window that have valid values. The implementation might not always be able to compute it, in which case it will be set to a negative value.

Returns

a binary-or'ed combination of possible values GDAL_DATA_COVERAGE_STATUS_UNIMPLEMENTED, GDAL_DATA_COVERAGE_STATUS_DATA and GDAL_DATA_COVERAGE_STATUS_EMPTY

Note

Added in GDAL 2.2

GDALDataset * GDALRasterBand::GetDataset ( )

Fetch the owning dataset handle.

Note that some GDALRasterBands are not considered to be a part of a dataset, such as overviews or other "freestanding" bands.

This method is the same as the C function **GDALGetBandDataset()**.

Returns

the pointer to the **GDALDataset** to which this band belongs, or NULL if this cannot be determined.

```
CPLErr GDALRasterBand::GetDefaultHistogram ( double *        pdfMin,
                                             double *        pdfMax,
                                             int *           pnBuckets,
                                             GUIntBig **     ppanHistogram,
                                             int             bForce,
                                             GDALProgressFunc pfnProgress,
                                             void *          pProgressData
                                           )                                  virtual
```

Fetch default raster histogram.

The default method in **GDALRasterBand** will compute a default histogram. This method is overridden by derived classes (such as **GDALPamRasterBand**, VRTDataset, HFADataset...) that may be able to fetch efficiently an already stored histogram.

This method is the same as the C functions **GDALGetDefaultHistogram()** and **GDALGetDefaultHistogramEx()**.

Parameters

pdfMin	pointer to double value that will contain the lower bound of the histogram.
pdfMax	pointer to double value that will contain the upper bound of the histogram.
pnBuckets	pointer to int value that will contain the number of buckets in *ppanHistogram.
ppanHistogram	pointer to array into which the histogram totals are placed. To be freed with VSIFree
bForce	TRUE to force the computation. If FALSE and no default histogram is available, the method will return CE_Warning
pfnProgress	function to report progress to completion.
pProgressData	application data to pass to pfnProgress.

Returns

CE_None on success, CE_Failure if something goes wrong, or CE_Warning if no default histogram is available.

Reimplemented in **GDALPamRasterBand**.

GDALRasterAttributeTable * GDALRasterBand::GetDefaultRAT ( ) virtual

Fetch default Raster Attribute Table.

A RAT will be returned if there is a default one associated with the band, otherwise NULL is returned. The returned RAT is owned by the band and should not be deleted by the application.

This method is the same as the C function **GDALGetDefaultRAT()**.

Returns

NULL, or a pointer to an internal RAT owned by the band.

Reimplemented in **GDALPamRasterBand**.

```
CPLErr GDALRasterBand::GetHistogram ( double          dfMin,
                                      double          dfMax,
                                      int             nBuckets,
                                      GUIntBig *      panHistogram,
                                      int             bIncludeOutOfRange,
                                      int             bApproxOK,
                                      GDALProgressFunc pfnProgress,
                                      void *          pProgressData
                                    )                                    virtual
```

Compute raster histogram.

Note that the bucket size is (dfMax-dfMin) / nBuckets.

For example to compute a simple 256 entry histogram of eight bit data, the following would be suitable. The unusual bounds are to ensure that bucket boundaries don't fall right on integer values causing possible errors due to rounding after scaling.

```
GUIntBig anHistogram[256];

poBand->GetHistogram( -0.5, 255.5, 256, anHistogram, FALSE, FALSE,
                      GDALDummyProgress, nullptr );
```

Note that setting bApproxOK will generally result in a subsampling of the file, and will utilize overviews if available. It should generally produce a representative histogram for the data that is suitable for use in generating histogram based luts for instance. Generally bApproxOK is much faster than an exactly computed histogram.

This method is the same as the C functions **GDALGetRasterHistogram()** and **GDALGetRasterHistogramEx()**.

Parameters

dfMin	the lower bound of the histogram.
dfMax	the upper bound of the histogram.
nBuckets	the number of buckets in panHistogram.
panHistogram	array into which the histogram totals are placed.
bIncludeOutOfRange	if TRUE values below the histogram range will mapped into panHistogram[0], and values above will be mapped into panHistogram[nBuckets-1] otherwise out of range values are discarded.
bApproxOK	TRUE if an approximate, or incomplete histogram OK.
pfnProgress	function to report progress to completion.
pProgressData	application data to pass to pfnProgress.

Returns

CE_None on success, or CE_Failure if something goes wrong.

Reimplemented in **GDALPamRasterBand**.

```
unsigned char *
GDALRasterBand::GetIndexColorTranslationTo ( GDALRasterBand *  poReferenceBand,
                                             unsigned char *   pTranslationTable = nullptr,
                                             int *             pApproximateMatching = nullptr
                                           )
```

Compute translation table for color tables.

When the raster band has a palette index, it may be useful to compute the "translation" of this palette to the palette of another band. The translation tries to do exact matching first, and then approximate matching if no exact matching is possible. This method returns a table such that table[i] = j where i is an index of the 'this' rasterband and j the corresponding index for the reference rasterband.

This method is thought as internal to GDAL and is used for drivers like RPFTOC.

The implementation only supports 1-byte palette rasterbands.

Parameters

poReferenceBand	the raster band
pTranslationTable	an already allocated translation table (at least 256 bytes), or NULL to let the method allocate it
pApproximateMatching	a pointer to a flag that is set if the matching is approximate. May be NULL.

Returns

a translation table if the two bands are palette index and that they do not match or NULL in other cases. The table must be freed with CPLFree if NULL was passed for pTranslationTable.

```
GDALRasterBlock * GDALRasterBand::GetLockedBlockRef ( int  nXBlockOff,
                                                      int  nYBlockOff,
                                                      int  bJustInitialize = FALSE
                                                    )
```

Fetch a pointer to an internally cached raster block.

This method will returned the requested block (locked) if it is already in the block cache for the layer. If not, the block will be read from the driver, and placed in the layer block cached, then returned. If an error occurs reading the block from the driver, a NULL value will be returned.

If a non-NULL value is returned, then a lock for the block will have been acquired on behalf of the caller. It is absolutely imperative that the caller release this lock (with **GDALRasterBlock::DropLock()**) or else severe problems may result.

Note that calling **GetLockedBlockRef()** on a previously uncached band will enable caching.

Parameters

nXBlockOff	the horizontal block offset, with zero indicating the left most block, 1 the next block and so forth.
nYBlockOff	the vertical block offset, with zero indicating the top most block, 1 the next block and so forth.
bJustInitialize	If TRUE the block will be allocated and initialized, but not actually read from the source. This is useful when it will just be completely set and written back.

Returns

pointer to the block object, or NULL on failure.

GDALRasterBand * GDALRasterBand::GetMaskBand ( ) virtual

Return the mask band associated with the band.

The **GDALRasterBand** class includes a default implementation of **GetMaskBand()** that returns one of four default implementations :

- If a corresponding .msk file exists it will be used for the mask band.
- If the dataset has a NODATA_VALUES metadata item, an instance of the new GDALNoDataValuesMaskBand class will be returned. **GetMaskFlags()** will return GMF_NODATA | GMF_PER_DATASET.

  Since
  GDAL 1.6.0

- If the band has a nodata value set, an instance of the new GDALNodataMaskRasterBand class will be returned. **GetMaskFlags()** will return GMF_NODATA.
- If there is no nodata value, but the dataset has an alpha band that seems to apply to this band (specific rules yet to be determined) and that is of type GDT_Byte then that alpha band will be returned, and the flags GMF_PER_DATASET and GMF_ALPHA will be returned in the flags.
- If neither of the above apply, an instance of the new GDALAllValidRasterBand class will be returned that has 255 values for all pixels. The null flags will return GMF_ALL_VALID.

Note that the **GetMaskBand()** should always return a **GDALRasterBand** mask, even if it is only an all 255 mask with the flags indicating GMF_ALL_VALID.

For an external .msk file to be recognized by GDAL, it must be a valid GDAL dataset, with the same name as the main dataset and suffixed with .msk, with either one band (in the GMF_PER_DATASET case), or as many bands as the main dataset. It must have INTERNAL_MASK_FLAGS_xx metadata items set at the dataset level, where xx matches the band number of a band of the main dataset. The value of those items is a combination of the flags GMF_ALL_VALID, GMF_PER_DATASET, GMF_ALPHA and GMF_NODATA. If a metadata item is missing for a band, then the other rules explained above will be used to generate a on-the-fly mask band.

See also
CreateMaskBand() for the characteristics of .msk files created by GDAL.

This method is the same as the C function **GDALGetMaskBand()**.

Returns
a valid mask band.

Since
GDAL 1.5.0

See also
http://trac.osgeo.org/gdal/wiki/rfc15_nodatabitmask

int GDALRasterBand::GetMaskFlags ( ) virtual

Return the status flags of the mask band associated with the band.

The **GetMaskFlags()** method returns an bitwise OR-ed set of status flags with the following available definitions that may be extended in the future:

- **GMF_ALL_VALID(0x01)**: There are no invalid pixels, all mask values will be 255. When used this will normally be the only flag set.
- **GMF_PER_DATASET(0x02)**: The mask band is shared between all bands on the dataset.
- **GMF_ALPHA(0x04)**: The mask band is actually an alpha band and may have values other than 0 and 255.
- **GMF_NODATA(0x08)**: Indicates the mask is actually being generated from nodata values. (mutually exclusive of GMF_ALPHA)

The **GDALRasterBand** class includes a default implementation of **GetMaskBand()** that returns one of four default implementations :

- If a corresponding .msk file exists it will be used for the mask band.
- If the dataset has a NODATA_VALUES metadata item, an instance of the new GDALNoDataValuesMaskBand class will be returned. **GetMaskFlags()** will return GMF_NODATA | GMF_PER_DATASET.

  Since
  GDAL 1.6.0

- If the band has a nodata value set, an instance of the new GDALNodataMaskRasterBand class will be returned. **GetMaskFlags()** will return GMF_NODATA.
- If there is no nodata value, but the dataset has an alpha band that seems to apply to this band (specific rules yet to be determined) and that is of type GDT_Byte then that alpha band will be returned, and the flags GMF_PER_DATASET and GMF_ALPHA will be returned in the flags.
- If neither of the above apply, an instance of the new GDALAllValidRasterBand class will be returned that has 255 values for all pixels. The null flags will return GMF_ALL_VALID.

For an external .msk file to be recognized by GDAL, it must be a valid GDAL dataset, with the same name as the main dataset and suffixed with .msk, with either one band (in the GMF_PER_DATASET case), or as many bands as the main dataset. It must have INTERNAL_MASK_FLAGS_xx metadata items set at the dataset level, where xx matches the band number of a band of the main dataset. The value of those items is a combination of the flags GMF_ALL_VALID, GMF_PER_DATASET, GMF_ALPHA and GMF_NODATA. If a metadata item is missing for a band, then the other rules explained above will be used to generate a on-the-fly mask band.

See also
CreateMaskBand() for the characteristics of .msk files created by GDAL.

This method is the same as the C function **GDALGetMaskFlags()**.

Since
GDAL 1.5.0

Returns
a valid mask band.

See also

http://trac.osgeo.org/gdal/wiki/rfc15_nodatabitmask

double GDALRasterBand::GetMaximum (int * pbSuccess = nullptr) virtual

Fetch the maximum value for this band.

For file formats that don't know this intrinsically, the maximum supported value for the data type will generally be returned.

This method is the same as the C function **GDALGetRasterMaximum()**.

Parameters

pbSuccess pointer to a boolean to use to indicate if the returned value is a tight maximum or not. May be NULL (default).

Returns

the maximum raster value (excluding no data pixels)

char** GDALRasterBand::GetMetadata (const char * pszDomain = "") override virtual

Fetch metadata.

The returned string list is owned by the object, and may change at any time. It is formatted as a "Name=value" list with the last pointer value being NULL. Use the CPL StringList functions such as **CSLFetchNameValue()** to manipulate it.

Note that relatively few formats return any metadata at this time.

This method does the same thing as the C function **GDALGetMetadata()**.

Parameters

pszDomain the domain of interest. Use "" or NULL for the default domain.

Returns

NULL or a string list.

Reimplemented from **GDALMajorObject**.

const char* GDALRasterBand::GetMetadataItem (const char * pszName, const char * pszDomain) override virtual

Fetch single metadata item.

The C function **GDALGetMetadataItem()** does the same thing as this method.

Parameters

pszName the key for the metadata item to fetch.

pszDomain the domain to fetch for, use NULL for the default domain.

Returns

NULL on failure to find the key, or a pointer to an internal copy of the value string on success.

Reimplemented from **GDALMajorObject**.

double GDALRasterBand::GetMinimum (int * pbSuccess = nullptr) virtual

Fetch the minimum value for this band.

For file formats that don't know this intrinsically, the minimum supported value for the data type will generally be returned.

This method is the same as the C function **GDALGetRasterMinimum()**.

Parameters

pbSuccess pointer to a boolean to use to indicate if the returned value is a tight minimum or not. May be NULL (default).

Returns

the minimum raster value (excluding no data pixels)

double GDALRasterBand::GetNoDataValue (int * pbSuccess = nullptr) virtual

Fetch the no data value for this band.

If there is no out of data value, an out of range value will generally be returned. The no data value for a band is generally a special marker value used to mark pixels that are not valid data. Such pixels should generally not be displayed, nor contribute to analysis operations.

The no data value returned is 'raw', meaning that it has no offset and scale applied.

This method is the same as the C function **GDALGetRasterNoDataValue()**.

Parameters

pbSuccess pointer to a boolean to use to indicate if a value is actually associated with this layer. May be NULL (default).

Returns

the nodata value for this band.

Reimplemented in **GDALPamRasterBand**.

double GDALRasterBand::GetOffset (int * pbSuccess = nullptr) virtual

Fetch the raster value offset.

This value (in combination with the **GetScale()** value) can be used to transform raw pixel values into the units returned by **GetUnitType()**. For example this might be used to store elevations in GUInt16 bands with a precision of 0.1, and starting from -100.

Units value = (raw value * scale) + offset

Note that applying scale and offset is of the responsibility of the user, and is not done by methods such as **RasterIO()** or **ReadBlock()**.

For file formats that don't know this intrinsically a value of zero is returned.

This method is the same as the C function **GDALGetRasterOffset()**.

Parameters

pbSuccess pointer to a boolean to use to indicate if the returned value is meaningful or not. May be NULL (default).

Returns

the raster offset.

Reimplemented in **GDALPamRasterBand**.

GDALRasterBand * GDALRasterBand::GetOverview (int i) virtual

Fetch overview raster band object.

This method is the same as the C function GDALGetOverview().

Parameters

i overview index between 0 and GetOverviewCount()-1.

Returns

overview GDALRasterBand.

int GDALRasterBand::GetOverviewCount ( ) virtual

Return the number of overview layers available.

This method is the same as the C function GDALGetOverviewCount().

Returns

overview count, zero if none.

GDALDataType GDALRasterBand::GetRasterDataType (void)

Fetch the pixel data type for this band.

This method is the same as the C function GDALGetRasterDataType().

Returns

the data type of pixels for this band.

GDALRasterBand * GDALRasterBand::GetRasterSampleOverview (GUIntBig nDesiredSamples) virtual

Fetch best sampling overview.

Returns the most reduced overview of the given band that still satisfies the desired number of samples. This function can be used with zero as the number of desired samples to fetch the most reduced overview. The same band as was passed in will be returned if it has not overviews, or if none of the overviews have enough samples.

This method is the same as the C functions GDALGetRasterSampleOverview() and GDALGetRasterSampleOverviewEx().

Parameters

nDesiredSamples the returned band will have at least this many pixels.

Returns

optimal overview or the band itself.

double GDALRasterBand::GetScale (int * pbSuccess = nullptr) virtual

Fetch the raster value scale.

This value (in combination with the **GetOffset()** value) can be used to transform raw pixel values into the units returned by **GetUnitType()**. For example this might be used to store elevations in GUInt16 bands with a precision of 0.1, and starting from -100.

Units value = (raw value * scale) + offset

Note that applying scale and offset is of the responsibility of the user, and is not done by methods such as **RasterIO()** or **ReadBlock()**.

For file formats that don't know this intrinsically a value of one is returned.

This method is the same as the C function **GDALGetRasterScale()**.

Parameters

pbSuccess pointer to a boolean to use to indicate if the returned value is meaningful or not. May be NULL (default).

Returns

the raster scale.

Reimplemented in **GDALPamRasterBand**.

```
CPLErr GDALRasterBand::GetStatistics ( int        bApproxOK,
                                       int        bForce,
                                       double *   pdfMin,
                                       double *   pdfMax,
                                       double *   pdfMean,
                                       double *   pdfStdDev
                                     )                          virtual
```

Fetch image statistics.

Returns the minimum, maximum, mean and standard deviation of all pixel values in this band. If approximate statistics are sufficient, the bApproxOK flag can be set to true in which case overviews, or a subset of image tiles may be used in computing the statistics.

If bForce is FALSE results will only be returned if it can be done quickly (i.e. without scanning the data). If bForce is FALSE and results cannot be returned efficiently, the method will return CE_Warning but no warning will have been issued. This is a non-standard use of the CE_Warning return value to indicate "nothing done".

Note that file formats using PAM (Persistent Auxiliary Metadata) services will generally cache statistics in the .pam file allowing fast fetch after the first request.

This method is the same as the C function **GDALGetRasterStatistics()**.

Parameters

bApproxOK	If TRUE statistics may be computed based on overviews or a subset of all tiles.
bForce	If FALSE statistics will only be returned if it can be done without rescanning the image.
pdfMin	Location into which to load image minimum (may be NULL).
pdfMax	Location into which to load image maximum (may be NULL).-
pdfMean	Location into which to load image mean (may be NULL).
pdfStdDev	Location into which to load image standard deviation (may be NULL).

Returns

CE_None on success, CE_Warning if no values returned, CE_Failure if an error occurs.

const char * GDALRasterBand::GetUnitType ( ) virtual

Return raster unit type.

Return a name for the units of this raster's values. For instance, it might be "m" for an elevation model in meters, or "ft" for feet. If no units are available, a value of "" will be returned. The returned string should not be modified, nor freed by the calling application.

This method is the same as the C function **GDALGetRasterUnitType()**.

Returns

unit name string.

Reimplemented in **GDALPamRasterBand**.

```
CPLVirtualMem * GDALRasterBand::GetVirtualMemAuto ( GDALRWFlag  eRWFlag,
                                                    int *       pnPixelSpace,
                                                    GIntBig *   pnLineSpace,
                                                    char **     papszOptions
                                                  )                                virtual
```

Create a CPLVirtualMem object from a GDAL raster band object.

Only supported on Linux and Unix systems with mmap() for now.

This method allows creating a virtual memory object for a **GDALRasterBand**, that exposes the whole image data as a virtual array.

The default implementation relies on **GDALRasterBandGetVirtualMem()**, but specialized implementation, such as for raw files, may also directly use mechanisms of the operating system to create a view of the underlying file into virtual memory (**CPLVirtualMemFileMapNew()**)

At the time of writing, the GeoTIFF driver and "raw" drivers (EHdr, ...) offer a specialized implementation with direct file mapping, provided that some requirements are met :

- for all drivers, the dataset must be backed by a "real" file in the file system, and the byte ordering of multi-byte datatypes (Int16, etc.) must match the native ordering of the CPU.
- in addition, for the GeoTIFF driver, the GeoTIFF file must be uncompressed, scanline oriented (i.e. not tiled). Strips must be organized in the file in sequential order, and be equally spaced (which is generally the case). Only power-of-two bit depths are supported (8 for GDT_Bye, 16 for GDT_Int16/GDT_UInt16, 32 for GDT_Float32 and 64 for GDT_Float64)

The pointer returned remains valid until **CPLVirtualMemFree()** is called. **CPLVirtualMemFree()** must be called before the raster band object is destroyed.

If p is such a pointer and base_type the type matching **GDALGetRasterDataType()**, the element of image coordinates (x, y) can be accessed with *(base_type*) ((GByte*)p + x * *pnPixelSpace + y * *pnLineSpace)

This method is the same as the C **GDALGetVirtualMemAuto()** function.

Parameters

eRWFlag	Either GF_Read to read the band, or GF_Write to read/write the band.
pnPixelSpace	Output parameter giving the byte offset from the start of one pixel value in the buffer to the start of the next pixel value within a scanline.
pnLineSpace	Output parameter giving the byte offset from the start of one scanline in the buffer to the start of the next.
papszOptions	NULL terminated list of options. If a specialized implementation exists, defining USE_DEFAULT_IMPLEMENTATION=YES will cause the default implementation to be used. On the contrary, starting with GDAL 2.2, defining USE_DEFAULT_IMPLEMENTATION=NO will prevent the default implementation from being used (thus only allowing efficient implementations to be used). When requiring or falling back to the default implementation, the following options are available : CACHE_SIZE (in bytes, defaults to 40 MB), PAGE_SIZE_HINT (in bytes), SINGLE_THREAD ("FALSE" / "TRUE", defaults to FALSE)

Returns

a virtual memory object that must be unreferenced by **CPLVirtualMemFree()**, or NULL in case of failure.

Since

GDAL 1.11

int GDALRasterBand::GetXSize ( )

Fetch XSize of raster.

This method is the same as the C function **GDALGetRasterBandXSize()**.

Returns

the width in pixels of this band.

int GDALRasterBand::GetYSize ( )

Fetch YSize of raster.

This method is the same as the C function **GDALGetRasterBandYSize()**.

Returns

the height in pixels of this band.

int GDALRasterBand::HasArbitraryOverviews ( ) virtual

Check for arbitrary overviews.

This returns TRUE if the underlying datastore can compute arbitrary overviews efficiently, such as is the case with OGDI over a network. Datastores with arbitrary overviews don't generally have any fixed overviews, but the **RasterIO()** method can be used in downsampling mode to get overview data efficiently.

This method is the same as the C function **GDALHasArbitraryOverviews()**,

Returns

TRUE if arbitrary overviews available (efficiently), otherwise FALSE.

```
int GDALRasterBand::IGetDataCoverageStatus ( int        nXOff,
                                             int        nYOff,
                                             int        nXSize,
                                             int        nYSize,
                                             int        nMaskFlagStop,
                                             double *   pdfDataPct
                                           )
```

protected virtual

Get the coverage status of a sub-window of the raster.

Returns whether a sub-window of the raster contains only data, only empty blocks or a mix of both. This function can be used to determine quickly if it is worth issuing RasterIO / ReadBlock requests in datasets that may be sparse.

Empty blocks are blocks that contain only pixels whose value is the nodata value when it is set, or whose value is 0 when the nodata value is not set.

The query is done in an efficient way without reading the actual pixel values. If not possible, or not implemented at all by the driver, GDAL_DATA_COVERAGE_STATUS_UNIMPLEMENTED | GDAL_DATA_COVERAGE_STATUS_DATA will be returned.

The values that can be returned by the function are the following, potentially combined with the binary or operator :

- GDAL_DATA_COVERAGE_STATUS_UNIMPLEMENTED : the driver does not implement **GetDataCoverageStatus()**. This flag should be returned together with GDAL_DATA_COVERAGE_STATUS_DATA.
- GDAL_DATA_COVERAGE_STATUS_DATA: There is (potentially) data in the queried window.
- GDAL_DATA_COVERAGE_STATUS_EMPTY: There is nodata in the queried window. This is typically identified by the concept of missing block in formats that supports it.

Note that GDAL_DATA_COVERAGE_STATUS_DATA might have false positives and should be interpreted more as hint of potential presence of data. For example if a GeoTIFF file is created with blocks filled with zeroes (or set to the nodata value), instead of using the missing block mechanism, GDAL_DATA_COVERAGE_STATUS_DATA will be returned. On the contrary, GDAL_DATA_COVERAGE_STATUS_EMPTY should have no false positives.

The nMaskFlagStop should be generally set to 0. It can be set to a binary-or'ed mask of the above mentioned values to enable a quick exiting of the function as soon as the computed mask matches the nMaskFlagStop. For example, you can issue a request on the whole raster with nMaskFlagStop = GDAL_DATA_COVERAGE_STATUS_EMPTY. As soon as one missing block is encountered, the function will exit, so that you can potentially refine the requested area to find which particular region(s) have missing blocks.

See also

GDALGetDataCoverageStatus()

Parameters

nXOff	The pixel offset to the top left corner of the region of the band to be queried. This would be zero to start from the left side.

nYOff The line offset to the top left corner of the region of the band to be queried. This would be zero to start from the top.

nXSize The width of the region of the band to be queried in pixels.

nYSize The height of the region of the band to be queried in lines.

nMaskFlagStop 0, or a binary-or'ed mask of possible values GDAL_DATA_COVERAGE_STATUS_UNIMPLEMENTED, GDAL_DATA_COVERAGE_STATUS_DATA and GDAL_DATA_COVERAGE_STATUS_EMPTY. As soon as the computation of the coverage matches the mask, the computation will be stopped. *pdfDataPct will not be valid in that case.

pdfDataPct Optional output parameter whose pointed value will be set to the (approximate) percentage in [0,100] of pixels in the queried sub-window that have valid values. The implementation might not always be able to compute it, in which case it will be set to a negative value.

Returns

a binary-or'ed combination of possible values GDAL_DATA_COVERAGE_STATUS_UNIMPLEMENTED, GDAL_DATA_COVERAGE_STATUS_DATA and GDAL_DATA_COVERAGE_STATUS_EMPTY

Note

Added in GDAL 2.2

```
CPLErr GDALRasterBand::IRasterIO ( GDALRWFlag          eRWFlag,
                                   int                 nXOff,
                                   int                 nYOff,
                                   int                 nXSize,
                                   int                 nYSize,
                                   void *              pData,
                                   int                 nBufXSize,
                                   int                 nBufYSize,
                                   GDALDataType        eBufType,
                                   GSpacing            nPixelSpace,
                                   GSpacing            nLineSpace,
                                   GDALRasterIOExtraArg * psExtraArg
                                 )
```

protected virtual

Read/write a region of image data for this band.

This method allows reading a region of a **GDALRasterBand** into a buffer, or writing data from a buffer into a region of a **GDALRasterBand**. It automatically takes care of data type translation if the data type (eBufType) of the buffer is different than that of the **GDALRasterBand**. The method also takes care of image decimation / replication if the buffer size (nBufXSize x nBufYSize) is different than the size of the region being accessed (nXSize x nYSize).

The nPixelSpace and nLineSpace parameters allow reading into or writing from unusually organized buffers. This is primarily used for buffers containing more than one bands raster data in interleaved format.

Some formats may efficiently implement decimation into a buffer by reading from lower resolution overview images.

For highest performance full resolution data access, read and write on "block boundaries" as returned by **GetBlockSize()**, or use the **ReadBlock()** and **WriteBlock()** methods.

This method is the same as the C **GDALRasterIO()** or **GDALRasterIOEx()** functions.

Parameters

eRWFlag	Either GF_Read to read a region of data, or GF_Write to write a region of data.
nXOff	The pixel offset to the top left corner of the region of the band to be accessed. This would be zero to start from the left side.
nYOff	The line offset to the top left corner of the region of the band to be accessed. This would be zero to start from the top.
nXSize	The width of the region of the band to be accessed in pixels.
nYSize	The height of the region of the band to be accessed in lines.
pData	The buffer into which the data should be read, or from which it should be written. This buffer must contain at least nBufXSize * nBufYSize words of type eBufType. It is organized in left to right, top to bottom pixel order. Spacing is controlled by the nPixelSpace, and nLineSpace parameters.
nBufXSize	the width of the buffer image into which the desired region is to be read, or from

which it is to be written.

nBufYSize the height of the buffer image into which the desired region is to be read, or from which it is to be written.

eBufType the type of the pixel values in the pData data buffer. The pixel values will automatically be translated to/from the **GDALRasterBand** data type as needed.

nPixelSpace The byte offset from the start of one pixel value in pData to the start of the next pixel value within a scanline. If defaulted (0) the size of the datatype eBufType is used.

nLineSpace The byte offset from the start of one scanline in pData to the start of the next. If defaulted (0) the size of the datatype eBufType * nBufXSize is used.

psExtraArg (new in GDAL 2.0) pointer to a **GDALRasterIOExtraArg** structure with additional arguments to specify resampling and progress callback, or NULL for default behaviour. The GDAL_RASTERIO_RESAMPLING configuration option can also be defined to override the default resampling to one of BILINEAR, CUBIC, CUBICSPLINE, LANCZOS, AVERAGE or MODE.

Returns

CE_Failure if the access fails, otherwise CE_None.

GDALRasterBand::IReadBlock (**int** **nBlockXOff,**
int **nBlockYOff,**
void * **pData**
)

protected pure virtual

Read a block of data.

Default internal implementation ... to be overridden by subclasses that support reading.

Parameters

nBlockXOff Block X Offset

nBlockYOff Block Y Offset

pData Pixel buffer into which to place read data.

Returns

CE_None on success or CE_Failure on an error.

CPLErr GDALRasterBand::IWriteBlock (int nBlockXOff,
int nBlockYOff,
void * pData
)

protected virtual

Write a block of data.

Default internal implementation ... to be overridden by subclasses that support writing.

Parameters

nBlockXOff Block X Offset

nBlockYOff Block Y Offset

pData Pixel buffer to write

Returns

CE_None on success or CE_Failure on an error.

```
CPLErr GDALRasterBand::RasterIO ( GDALRWFlag          eRWFlag,
                                  int                 nXOff,
                                  int                 nYOff,
                                  int                 nXSize,
                                  int                 nYSize,
                                  void *              pData,
                                  int                 nBufXSize,
                                  int                 nBufYSize,
                                  GDALDataType        eBufType,
                                  GSpacing            nPixelSpace,
                                  GSpacing            nLineSpace,
                                  GDALRasterIOExtraArg * psExtraArg
                                )
```

Read/write a region of image data for this band.

This method allows reading a region of a **GDALRasterBand** into a buffer, or writing data from a buffer into a region of a **GDALRasterBand**. It automatically takes care of data type translation if the data type (eBufType) of the buffer is different than that of the **GDALRasterBand**. The method also takes care of image decimation / replication if the buffer size (nBufXSize x nBufYSize) is different than the size of the region being accessed (nXSize x nYSize).

The nPixelSpace and nLineSpace parameters allow reading into or writing from unusually organized buffers. This is primarily used for buffers containing more than one bands raster data in interleaved format.

Some formats may efficiently implement decimation into a buffer by reading from lower resolution overview images.

For highest performance full resolution data access, read and write on "block boundaries" as returned by **GetBlockSize()**, or use the **ReadBlock()** and **WriteBlock()** methods.

This method is the same as the C **GDALRasterIO()** or **GDALRasterIOEx()** functions.

Parameters

	eRWFlag	Either GF_Read to read a region of data, or GF_Write to write a region of data.
	nXOff	The pixel offset to the top left corner of the region of the band to be accessed. This would be zero to start from the left side.
	nYOff	The line offset to the top left corner of the region of the band to be accessed. This would be zero to start from the top.
	nXSize	The width of the region of the band to be accessed in pixels.
	nYSize	The height of the region of the band to be accessed in lines.
`[in,out]`	**pData**	The buffer into which the data should be read, or from which it should be written. This buffer must contain at least nBufXSize * nBufYSize words of type eBufType. It is organized in left to right, top to bottom pixel order. Spacing is controlled by the nPixelSpace, and nLineSpace parameters.

	nBufXSize	the width of the buffer image into which the desired region is to be read, or from which it is to be written.
	nBufYSize	the height of the buffer image into which the desired region is to be read, or from which it is to be written.
	eBufType	the type of the pixel values in the pData data buffer. The pixel values will automatically be translated to/from the **GDALRasterBand** data type as needed.
	nPixelSpace	The byte offset from the start of one pixel value in pData to the start of the next pixel value within a scanline. If defaulted (0) the size of the datatype eBufType is used.
	nLineSpace	The byte offset from the start of one scanline in pData to the start of the next. If defaulted (0) the size of the datatype eBufType * nBufXSize is used.
[in]	**psExtraArg**	(new in GDAL 2.0) pointer to a **GDALRasterIOExtraArg** structure with additional arguments to specify resampling and progress callback, or NULL for default behaviour. The GDAL_RASTERIO_RESAMPLING configuration option can also be defined to override the default resampling to one of BILINEAR, CUBIC, CUBICSPLINE, LANCZOS, AVERAGE or MODE.

Returns

CE_Failure if the access fails, otherwise CE_None.

CPLErr GDALRasterBand::ReadBlock (int nXBlockOff,
int nYBlockOff,
void * pImage
)

Read a block of image data efficiently.

This method accesses a "natural" block from the raster band without resampling, or data type conversion. For a more generalized, but potentially less efficient access use **RasterIO()**.

This method is the same as the C **GDALReadBlock()** function.

See the **GetLockedBlockRef()** method for a way of accessing internally cached block oriented data without an extra copy into an application buffer.

Parameters

nXBlockOff the horizontal block offset, with zero indicating the left most block, 1 the next block and so forth.

nYBlockOff the vertical block offset, with zero indicating the top most block, 1 the next block and so forth.

pImage the buffer into which the data will be read. The buffer must be large enough to hold GetBlockXSize()*GetBlockYSize() words of type **GetRasterDataType()**.

Returns

CE_None on success or CE_Failure on an error.

The following code would efficiently compute a histogram of eight bit raster data. Note that the final block may be partial ... data beyond the edge of the underlying raster band in these edge blocks is of an undermined value.

```
CPLErr GetHistogram( GDALRasterBand *poBand, GUIntBig *panHistogram )

{
    memset( panHistogram, 0, sizeof(GUIntBig) * 256 );

    CPLAssert( poBand->GetRasterDataType() == GDT_Byte );

    int nXBlockSize, nYBlockSize;

    poBand->GetBlockSize( &nXBlockSize, &nYBlockSize );
    int nXBlocks = (poBand->GetXSize() + nXBlockSize - 1) / nXBlockSize;
    int nYBlocks = (poBand->GetYSize() + nYBlockSize - 1) / nYBlockSize;

    GByte *pabyData = (GByte *) CPLMalloc(nXBlockSize * nYBlockSize);

    for( int iYBlock = 0; iYBlock < nYBlocks; iYBlock++ )
    {
        for( int iXBlock = 0; iXBlock < nXBlocks; iXBlock++ )
        {
            int        nXValid, nYValid;

            poBand->ReadBlock( iXBlock, iYBlock, pabyData );
```

```
            // Compute the portion of the block that is valid
            // for partial edge blocks.
            poBand->GetActualBlockSize(iXBlock, iYBlock, &nXValid, &nYValid)

            // Collect the histogram counts.
            for( int iY = 0; iY < nYValid; iY++ )
            {
                for( int iX = 0; iX < nXValid; iX++ )
                {
                    panHistogram[pabyData[iX + iY * nXBlockSize]] += 1;
                }
            }
        }
    }
}
```

void GDALRasterBand::ReportError (CPLErr eErrClass,
CPLErrorNum err_no,
const char * fmt,
...
)

Emits an error related to a raster band.

This function is a wrapper for regular **CPLError()**. The only difference with **CPLError()** is that it prepends the error message with the dataset name and the band number.

Parameters

eErrClass one of CE_Warning, CE_Failure or CE_Fatal.

err_no the error number (CPLE_*) from **cpl_error.h**.

fmt a printf() style format string. Any additional arguments will be treated as arguments to fill in this format in a manner similar to printf().

Since

GDAL 1.9.0

CPLErr GDALRasterBand::SetCategoryNames (char ** papszNames) virtual

Set the category names for this band.

See the GetCategoryNames() method for more on the interpretation of category names.

This method is the same as the C function GDALSetRasterCategoryNames().

Parameters

papszNames the NULL terminated StringList of category names. May be NULL to just clear the existing list.

Returns

CE_None on success of CE_Failure on failure. If unsupported by the driver CE_Failure is returned, but no error message is reported.

Reimplemented in GDALPamRasterBand.

CPLErr GDALRasterBand::SetColorInterpretation (GDALColorInterp eColorInterp) virtual

Set color interpretation of a band.

This method is the same as the C function GDALSetRasterColorInterpretation().

Parameters

eColorInterp the new color interpretation to apply to this band.

Returns

CE_None on success or CE_Failure if method is unsupported by format.

Reimplemented in GDALPamRasterBand.

CPLErr GDALRasterBand::SetColorTable (GDALColorTable * poCT) virtual

Set the raster color table.

The driver will make a copy of all desired data in the colortable. It remains owned by the caller after the call.

This method is the same as the C function GDALSetRasterColorTable().

Parameters

poCT the color table to apply. This may be NULL to clear the color table (where supported).

Returns

CE_None on success, or CE_Failure on failure. If the action is unsupported by the driver, a value of CE_Failure is returned, but no error is issued.

Reimplemented in GDALPamRasterBand.

CPLErr GDALRasterBand::SetDefaultHistogram (double dfMin,
double dfMax,
int nBuckets,
GUIntBig * panHistogram
) virtual

Set default histogram.

This method is the same as the C function **GDALSetDefaultHistogram()** and **GDALSetDefaultHistogramEx()**

Reimplemented in **GDALPamRasterBand**.

CPLErr GDALRasterBand::SetDefaultRAT (const GDALRasterAttributeTable * poRAT) virtual

Set default Raster Attribute Table.

Associates a default RAT with the band. If not implemented for the format a CPLE_NotSupported error will be issued. If successful a copy of the RAT is made, the original remains owned by the caller.

This method is the same as the C function **GDALSetDefaultRAT()**.

Parameters

poRAT the RAT to assign to the band.

Returns

CE_None on success or CE_Failure if unsupported or otherwise failing.

Reimplemented in **GDALPamRasterBand**.

GDALRasterBand::SetMetadata (char ** papszMetadata,
const char * pszDomain
)

override virtual

Set metadata.

CAUTION: depending on the format, older values of the updated information might still be found in the file in a "ghost" state, even if no longer accessible through the GDAL API. This is for example the case of the GTiff format (this is not a exhaustive list)

The C function **GDALSetMetadata()** does the same thing as this method.

Parameters

papszMetadata the metadata in name=value string list format to apply.

pszDomain the domain of interest. Use "" or NULL for the default domain.

Returns

CE_None on success, CE_Failure on failure and CE_Warning if the metadata has been accepted, but is likely not maintained persistently by the underlying object between sessions.

Reimplemented from **GDALMajorObject**.

GDALRasterBand::SetMetadataItem (const char * pszName,
const char * pszValue,
const char * pszDomain
)

override virtual

Set single metadata item.

CAUTION: depending on the format, older values of the updated information might still be found in the file in a "ghost" state, even if no longer accessible through the GDAL API. This is for example the case of the GTiff format (this is not a exhaustive list)

The C function **GDALSetMetadataItem()** does the same thing as this method.

Parameters

pszName the key for the metadata item to fetch.

pszValue the value to assign to the key.

pszDomain the domain to set within, use NULL for the default domain.

Returns

CE_None on success, or an error code on failure.

Reimplemented from **GDALMajorObject**.

CPLErr GDALRasterBand::SetNoDataValue (double dfNoData) virtual

Set the no data value for this band.

Depending on drivers, changing the no data value may or may not have an effect on the pixel values of a raster that has just been created. It is thus advised to explicitly called **Fill()** if the intent is to initialize the raster to the nodata value. In ay case, changing an existing no data value, when one already exists and the dataset exists or has been initialized, has no effect on the pixel whose value matched the previous nodata value.

To clear the nodata value, use **DeleteNoDataValue()**.

This method is the same as the C function **GDALSetRasterNoDataValue()**.

Parameters

dfNoData the value to set.

Returns

CE_None on success, or CE_Failure on failure. If unsupported by the driver, CE_Failure is returned by no error message will have been emitted.

Reimplemented in **GDALPamRasterBand**.

CPLErr GDALRasterBand::SetOffset (double dfNewOffset) virtual

Set scaling offset.

Very few formats implement this method. When not implemented it will issue a CPLE_NotSupported error and return CE_Failure.

This method is the same as the C function **GDALSetRasterOffset()**.

Parameters

dfNewOffset the new offset.

Returns

CE_None or success or CE_Failure on failure.

Reimplemented in **GDALPamRasterBand**.

CPLErr GDALRasterBand::SetScale (double dfNewScale) virtual

Set scaling ratio.

Very few formats implement this method. When not implemented it will issue a CPLE_NotSupported error and return CE_Failure.

This method is the same as the C function GDALSetRasterScale().

Parameters

dfNewScale the new scale.

Returns

CE_None or success or CE_Failure on failure.

Reimplemented in GDALPamRasterBand.

CPLErr GDALRasterBand::SetStatistics (double dfMin, double dfMax, double dfMean, double dfStdDev) virtual

Set statistics on band.

This method can be used to store min/max/mean/standard deviation statistics on a raster band.

The default implementation stores them as metadata, and will only work on formats that can save arbitrary metadata. This method cannot detect whether metadata will be properly saved and so may return CE_None even if the statistics will never be saved.

This method is the same as the C function GDALSetRasterStatistics().

Parameters

dfMin minimum pixel value.

dfMax maximum pixel value.

dfMean mean (average) of all pixel values.

dfStdDev Standard deviation of all pixel values.

Returns

CE_None on success or CE_Failure on failure.

CPLErr GDALRasterBand::SetUnitType (const char * pszNewValue) virtual

Set unit type.

Set the unit type for a raster band. Values should be one of "" (the default indicating it is unknown), "m" indicating meters, or "ft" indicating feet, though other nonstandard values are allowed.

This method is the same as the C function GDALSetRasterUnitType().

Parameters

pszNewValue the new unit type value.

Returns

CE_None on success or CE_Failure if not successful, or unsupported.

Reimplemented in GDALPamRasterBand.

void static GDALRasterBandH GDALRasterBand::ToHandle (GDALRasterBand * poBand) inline static

Convert a GDALRasterBand* to a GDALRasterBandH.

Since

GDAL 2.3

GDALRasterBlock * GDALRasterBand::TryGetLockedBlockRef (int nXBlockOff, int nYBlockOff) protected

Try fetching block ref.

This method will returned the requested block (locked) if it is already in the block cache for the layer. If not, nullptr is returned.

If a non-NULL value is returned, then a lock for the block will have been acquired on behalf of the caller. It is absolutely imperative that the caller release this lock (with GDALRasterBlock::DropLock()) or else severe problems may result.

Parameters

nXBlockOff the horizontal block offset, with zero indicating the left most block, 1 the next block and so forth.

nYBlockOff the vertical block offset, with zero indicating the top most block, 1 the next block and so forth.

Returns

NULL if block not available, or locked block pointer.

```
CPLErr GDALRasterBand::WriteBlock ( int      nXBlockOff,
                                    int      nYBlockOff,
                                    void *   pImage
                                  )
```

Write a block of image data efficiently.

This method accesses a "natural" block from the raster band without resampling, or data type conversion. For a more generalized, but potentially less efficient access use **RasterIO()**.

This method is the same as the C **GDALWriteBlock()** function.

See **ReadBlock()** for an example of block oriented data access.

Parameters

nXBlockOff the horizontal block offset, with zero indicating the left most block, 1 the next block and so forth.

nYBlockOff the vertical block offset, with zero indicating the left most block, 1 the next block and so forth.

pImage the buffer from which the data will be written. The buffer must be large enough to hold GetBlockXSize()*GetBlockYSize() words of type **GetRasterDataType()**.

Returns

CE_None on success or CE_Failure on an error.

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

- **gdal_priv.h**
- gdalrasterband.cpp
- rasterio.cpp
