Remap Reference Manual

0

Generated by Doxygen 1.3.9.1

Wed Apr 19 14:36:57 2006

Contents

Chapter 1

Remap Directory Hierarchy

1.1 Remap Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 2

Remap Hierarchical Index

2.1 Remap Class Hierarchy

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

Chapter 3

Remap Class Index

3.1 Remap Class List

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

Chapter 4

Remap File Index

4.1 Remap File List

Here is a list of all files with brief descriptions:

Chapter 5

Remap Directory Documentation

5.1 src/Pixel/ Directory Reference

Files

- file **debug.h**
- file **Pixel.h**

5.2 src/VFiles/seviri_latlon/ Directory Reference

Files

- file **geostat.h**

5.3 src/ Directory Reference

Directories

- directory**Pixel**
- directory**VFiles**

Files

- file **allocation.hpp**
- file **common.h**
- file **debug.h**
- file **filetypes.h**
- file **grid.h**
- file **hdf_utils.h**
- file **parse_argument.h**
- file **reproject.h**
- file **tokenize.h**

5.4 src/VFiles/ Directory Reference

Directories

- directory**seviri_latlon**

Files

- file **debug.h**
- file **normalize_cal_factors.h**
- file **VFile.h**
- file **VFiles.h**
- file **VHdf.h**
- file **VHdf_Seviri.h**
- file **VIIR.h**
- file **VModis.h**
- file **VModis_interpol.h**
- file **VModis_latlon_resolve.h**
- file **VXRIT_SEVIRI.h**

Chapter 6

Remap Class Documentation

6.1 grid_type_ Struct Reference

`#include <grid.h>`

Public Attributes

- char **file** [STRING_MAXLEN+1]

  field used as the file source or target (for the grid contents) by functions **load_grid**(p. 47) *and* **save_grid**(p. 48)

- char **input_dataset** [STRING_MAXLEN+1]

  input dataset of the grid, in the input file (relevant for source and target grids)

- char **output_dataset** [STRING_MAXLEN+1]

  output dataset of the grid, in the output file (relevant for source and target grids), should be equal to input_ dataset by default

- int **ichannel**

  only relevant for 3-dimensional input data (stacks of 2-dimensional planes), e.g. Modis. The first plane is numbered 1. Should be set to 0 when unused.

- int **rank**

  rank of the grid buffers (should always be 2 for the time being)

- int **nrows**

  number of rows of the grid buffers (common to all the buffers lat, lon, tim, data, distance_from_ref, time_from_ref)

- int **ncols**

  number of columns of the grid (common to all the buffers lat, lon tim, data, distance_from_ref, time_from_ref)

- **coord_type** * **lat**

  *buffer for latitudes (nrows*ncols elements)*

- **coord_type * lon**

  *buffer for longitudes (nrows*ncols elements)*

- **time_type * tim**

  *buffer for times of acquisition (nrows*ncols elements)*

- **data_type * data**

  *buffer for measures, currently in 16-bit unsigned integer counts (nrows*ncols elements)*

- **distance_type * distance_from_ref**

  *buffer for distances between pixels in the target (reference) grid and their nearest neighbour in the source grid (nrows*ncols elements)*

- **time_type * time_from_ref**

  *buffer for time differences between pixels in the target (reference) grid and their nearest neighbour in the source grid (nrows*ncols elements)*

- float64 **slope**

  calibration scale factor (slope) to apply to a measure count to convert it into a physical value: phys_val = slope(count - offset)*

- float64 **offset**

  calibration offset factor to apply to a measure count to convert it into a physical value: phys_val = slope(count - offset)*

- bool **is_target**
- int * **src_irows**
- int * **src_icols**

6.1.1 Detailed Description

Grids are the main data structures handled by the software. They should have been designed as a class instead of a simple struct with functions to handle it, but by lack of time the software had to be delivered as is. A rewritten code with a grid class instead of a struct should be much clearer and easier to maintain, but will need a substantial amount of time to reimplement (and of course redocument !), that is not available today.

Grids are abstracts for reprojection. Their main purpose is to handle 2-dimensional buffers of geolocated data (measures along with their latitudes, longitudes and times of acquisition). The reprojection algorithm remaps a grid of data (from one instrument product) into another one. For convenience, origin and target of the data (files and datasets) are also maintained in the structure (although this is not a very clever design, I have to confess, I hope I'll have a chance to change this if more time is given to this project)

6.1.2 Member Data Documentation

6.1.2.1 data_type* grid_type_::data

buffer for measures, currently in 16-bit unsigned integer counts (nrows*ncols elements)

6.1.2.2 distance_type* grid_type_::distance_from_ref

buffer for distances between pixels in the target (reference) grid and their nearest neighbour in the source grid (nrows*ncols elements)

6.1.2.3 char grid_type_::file[STRING_MAXLEN+1]

field used as the file source or target (for the grid contents) by functions **load_grid**(p. 47) and **save_grid**(p. 48)

6.1.2.4 int grid_type_::ichannel

only relevant for 3-dimensional input data (stacks of 2-dimensional planes), e.g. Modis. The first plane is numbered 1. Should be set to 0 when unused.

6.1.2.5 char grid_type_::input_dataset[STRING_MAXLEN+1]

input dataset of the grid, in the input file (relevant for source and target grids)

6.1.2.6 bool grid_type_::is_target

specifies if a grid is a source (is_target == false) or a target (is_target == true) currently not used (intended as a future optimization of the reprojection code, in order to reuse precomputed source rows and cols instead of recomputing them again and again)

6.1.2.7 coord_type* grid_type_::lat

buffer for latitudes (nrows*ncols elements)

6.1.2.8 coord_type* grid_type_::lon

buffer for longitudes (nrows*ncols elements)

6.1.2.9 int grid_type_::ncols

number of columns of the grid (common to all the buffers lat, lon tim, data, distance_from_ref, time_from_ref)

6.1.2.10 int grid_type_::nrows

number of rows of the grid buffers (common to all the buffers lat, lon, tim, data, distance_from_-ref, time_from_ref)

6.1.2.11 float64 grid_type_::offset

calibration offset factor to apply to a measure count to convert it into a physical value: phys_val = slope*(count - offset)

6.1.2.12 char grid_type_::output_dataset[STRING_MAXLEN+1]

output dataset of the grid, in the output file (relevant for source and target grids), should be equal to input_dataset by default

6.1.2.13 int grid_type_::rank

rank of the grid buffers (should always be 2 for the time being)

6.1.2.14 float64 grid_type_::slope

calibration scale factor (slope) to apply to a measure count to convert it into a physical value: phys_val = slope*(count - offset)

6.1.2.15 int* grid_type_::src_icols

buffer to store, for each pixel of the target grid, its nearest neighbour's column in the source grid (nrows*ncols elements); valid only if **grid_type::is_target**(p. 15) == true; currently not used (intended as a future optimization of the reprojection code, in order to reuse precomputed source rows and cols instead of recomputing them again and again)

6.1.2.16 int* grid_type_::src_irows

buffer to store, for each pixel of the target grid, its nearest neighbour's row in the source grid (nrows*ncols elements); valid only if **grid_type::is_target**(p. 15) == true; currently not used (intended as a future optimization of the reprojection code, in order to reuse precomputed source rows and cols instead of recomputing them again and again)

6.1.2.17 time_type* grid_type_::tim

buffer for times of acquisition (nrows*ncols elements)

6.1.2.18 time_type* grid_type_::time_from_ref

buffer for time differences between pixels in the target (reference) grid and their nearest neighbour in the source grid (nrows*ncols elements)

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

- src/**grid.h**

6.2 Nearer_from< Pixel_type > Class Template Reference

`#include <Pixel.h>`

Public Member Functions

- **Nearer_from** (const Pixel_type &**this_pixel**)
- bool **operator()** (const Pixel_type &pixel1, const Pixel_type &pixel2) const

Private Attributes

- const Pixel_type **this_pixel**

template<typename Pixel_type> class Nearer_from< Pixel_type >

6.2.1 Constructor & Destructor Documentation

6.2.1.1 template<typename Pixel_type> Nearer_from< Pixel_type >::Nearer_from (const Pixel_type & *this_pixel*) [inline]

6.2.2 Member Function Documentation

6.2.2.1 template<typename Pixel_type> bool Nearer_from< Pixel_type >::operator() (const Pixel_type & *pixel1*, const Pixel_type & *pixel2*) const [inline]

6.2.3 Member Data Documentation

6.2.3.1 template<typename Pixel_type> const Pixel_type Nearer_from< Pixel_type >::this_pixel [private]

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

- src/Pixel/**Pixel.h**

6.3 Pixel_base< T, V > Class Template Reference

`#include <Pixel.h>`

Public Types

- typedef **Pixel_base**< T, V > **Pixel_type**
- typedef T **coord_type**
- typedef T **distance_type**
- typedef V **value_type**
- enum **unit_type** { **RADIANS**, **DEGREES** }

Public Member Functions

- **Pixel_base** ()
- **Pixel_base** (const **coord_type** lat, const **coord_type** lon, const **value_type** &val, const **unit_type** unit=RADIANS)
- bool **operator**< (const **Pixel_type** &other) const
- bool **operator**== (const **Pixel_type** &other) const
- **coord_type lat** () const
- **coord_type lon** () const
- **value_type val** () const
- **distance_type distance** (const **Pixel_type** &other) const
- void **get_neighbours** (std::vector< **Pixel_type** > &neighbours, std::multiset< **Pixel_type** > &pixels, **distance_type resolution**, bool sorted=false) const

Static Public Member Functions

- **coord_type get_resolution** ()
- void **set_resolution** (**distance_type** new_resolution)

Static Public Attributes

- const **coord_type DEFAULT_RESOLUTION** = 1.
- const **coord_type R_EARTH** = 6371.005076
- const **distance_type DEG2RAD** = M_PI/180.
- const **distance_type RAD2DEG** = 180.*M_1_PI

Private Attributes

- **coord_type lat_**
- **coord_type lon_**
- **value_type val_**
- int **ilat**
- int **ilon**

Static Private Attributes

- **distance_type resolution** = **DEFAULT_RESOLUTION**

Friends

- std::ostream & **operator**<< (std::ostream &os, const **Pixel_type** &pixel)

template<typename T, typename V> class Pixel_base< T, V >

6.3.1 Member Typedef Documentation

6.3.1.1 template<typename T, typename V> typedef T Pixel_base< T, V >::coord_type

6.3.1.2 template<typename T, typename V> typedef T Pixel_base< T, V >::distance_type

6.3.1.3 template<typename T, typename V> typedef Pixel_base<T, V> Pixel_base< T, V >::Pixel_type

6.3.1.4 template<typename T, typename V> typedef V Pixel_base< T, V >::value_type

6.3.2 Member Enumeration Documentation

6.3.2.1 template<typename T, typename V> enum Pixel_base::unit_type

Enumeration values:

RADIANS

DEGREES

6.3.3 Constructor & Destructor Documentation

6.3.3.1 template<typename T, typename V> Pixel_base< T, V >::Pixel_base () [inline]

6.3.3.2 template<typename T, typename V> Pixel_base< T, V >::Pixel_base (const coord_type *lat*, const coord_type *lon*, const value_type & *val*, const unit_type *unit* = RADIANS) [inline]

6.3.4 Member Function Documentation

6.3.4.1 template<typename T, typename V> distance_type Pixel_base< T, V >::distance (const Pixel_type & *other*) const [inline]

6.3.4.2 template<typename T, typename V> void Pixel_base< T, V >::get_neighbours (std::vector< Pixel_type > & *neighbours*, std::multiset< Pixel_type > & *pixels*, distance_type *resolution*, bool *sorted* = false) const [inline]

6.3.4.3 template<typename T, typename V> coord_type Pixel_base< T, V >::get_resolution () [inline, static]

6.3.4.4 template<typename T, typename V> coord_type Pixel_base< T, V >::lat () const [inline]

6.3.4.5 template<typename T, typename V> coord_type Pixel_base< T, V >::lon () const [inline]

6.3.4.6 template<typename T, typename V> bool Pixel_base< T, V >::operator< (const Pixel_type & *other*) const [inline]

6.3.4.7 template<typename T, typename V> bool Pixel_base< T, V >::operator== (const Pixel_type & *other*) const [inline]

6.3.4.8 template<typename T, typename V> void Pixel_base< T, V >::set_resolution (distance_type *new_resolution*) [inline, static]

6.3.4.9 template<typename T, typename V> value_type Pixel_base< T, V >::val () const [inline]

6.3.5 Friends And Related Function Documentation

6.3.5.1 template<typename T, typename V> std::ostream& operator<< (std::ostream & *os*, const Pixel_type & *pixel*) [friend]

6.3.6 Member Data Documentation

6.3.6.1 template<typename T, typename V> const Pixel_base< T, V >::coord_type Pixel_base< T, V >::DEFAULT_RESOLUTION = 1. [static]

6.3.6.2 template<typename T, typename V> const Pixel_base< T, V >::distance_type Pixel_base< T, V >::DEG2RAD = M_PI/180. [static]

6.3.6.3 template<typename T, typename V> int Pixel_base< T, V >::ilat [private]

6.3.6.4 template<typename T, typename V> int Pixel_base< T, V >::ilon [private]

6.3.6.5 template<typename T, typename V> coord_type Pixel_base< T, V

- src/Pixel/**Pixel.h**

6.4 PRODUCT Struct Reference

`#include <VModis_interpol.h>`

Public Attributes

- struct {
  int **Nl**
  int **Np**
  } **sds**

- struct {
  PROJTYPE type
  } **projparam**

- int **Nl_geo**
- int **Np_geo**
- float **rowoffset**
- float **rowstep**
- float **coloffset**
- float **colstep**

6.4.1 Member Data Documentation

6.4.1.1 float PRODUCT::coloffset

6.4.1.2 float PRODUCT::colstep

6.4.1.3 int PRODUCT::Nl

6.4.1.4 int PRODUCT::Nl_geo

6.4.1.5 int PRODUCT::Np

6.4.1.6 int PRODUCT::Np_geo

6.4.1.7 struct { ... } PRODUCT::projparam

6.4.1.8 float PRODUCT::rowoffset

6.4.1.9 float PRODUCT::rowstep

6.4.1.10 struct { ... } PRODUCT::sds

6.4.1.11 PROJTYPE PRODUCT::type

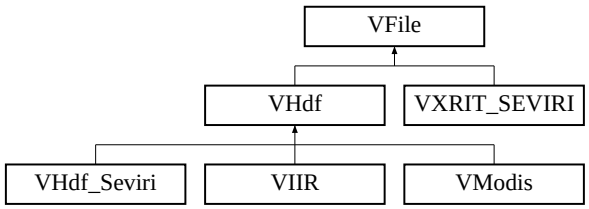
The documentation for this struct was generated from the following file:

- src/VFiles/**VModis_interpol.h**

6.5 VFile Class Reference

`#include <VFile.h>`

Inheritance diagram for VFile::



Public Member Functions

- **VFile** (const char *filename, const char *dataset, int ichannel=0, const char *sds_time=NULL, const char *sds_latitude="Latitude", const char *sds_longitude="Longitude")
- virtual std::string **filename** () const
- virtual std::string **dataset** () const
- virtual ~**VFile** ()
- virtual void **get_calibration** (double &slope, double &offset) const =0
- virtual **coord_type** * **lat** () const =0
- virtual **coord_type** * **lon** () const =0
- virtual **time_type** * **time** () const =0
- virtual **data_type** * **data** () const =0
- virtual std::vector< int > **dimensions** () const
- virtual int **rank** () const
- virtual int **dimension** (int idim) const

Protected Attributes

- char **latlon_filename_** [STRING_MAXLEN+1]
- char **time_filename_** [STRING_MAXLEN+1]
- char **data_filename_** [STRING_MAXLEN+1]
- char **sds_lat_** [STRING_MAXLEN+1]
- char **sds_lon_** [STRING_MAXLEN+1]
- char **sds_time_** [STRING_MAXLEN+1]
- char **sds_data_** [STRING_MAXLEN+1]
- int **ichannel_**
- std::vector< int > **dimensions_**

6.5.1 Constructor & Destructor Documentation

6.5.1.1 VFile::VFile (const char * *filename*, const char * *dataset*, int *ichannel* = 0, const char * *sds_time* = NULL, const char * *sds_latitude* = "Latitude", const char * *sds_longitude* = "Longitude") [inline]

6.5.1.2 virtual VFile::~VFile () [inline, virtual]

6.5.2 Member Function Documentation

6.5.2.1 virtual data_type* VFile::data () const [pure virtual]

Implemented in **VHdf** (p. 27), **VModis** (p. 31), and **VXRIT_SEVIRI** (p. 34).

6.5.2.2 virtual std::string VFile::dataset () const [inline, virtual]

6.5.2.3 virtual int VFile::dimension (int *idim*) const [inline, virtual]

6.5.2.4 virtual std::vector<int> VFile::dimensions () const [inline, virtual]

6.5.2.5 virtual std::string VFile::filename () const [inline, virtual]

6.5.2.6 virtual void VFile::get_calibration (double & *slope*, double & *offset*) const [pure virtual]

Implemented in **VHdf** (p. 27), **VHdf_Seviri** (p. 29), **VIIR** (p. 30), **VModis** (p. 32), and **VXRIT_SEVIRI** (p. 34).

6.5.2.7 virtual coord_type* VFile::lat () const [pure virtual]

Implemented in **VHdf** (p. 27), **VModis** (p. 32), and **VXRIT_SEVIRI** (p. 34).

6.5.2.8 virtual coord_type* VFile::lon () const [pure virtual]

Implemented in **VHdf** (p. 27), **VModis** (p. 32), and **VXRIT_SEVIRI** (p. 34).

6.5.2.9 virtual int VFile::rank () const [inline, virtual]

6.5.2.10 virtual time_type* VFile::time () const [pure virtual]

Implemented in **VHdf** (p. 28), **VModis** (p. 32), and **VXRIT_SEVIRI** (p. 34).

6.5.3 Member Data Documentation

6.5.3.1 char VFile::data_filename_[STRING_MAXLEN+1] [protected]

6.5.3.2 std::vector<int> VFile::dimensions_ [protected]

6.5.3.3 int VFile::ichannel_ [protected]

6.5.3.4 char VFile::latlon_filename_[STRING_MAXLEN+1] [protected]

6.5.3.5 char VFile::sds_data_[STRING_MAXLEN+1] [protected]

6.5.3.6 char VFile::sds_lat_[STRING_MAXLEN+1] [protected]

6.5.3.7 char VFile::sds_lon_[STRING_MAXLEN+1] [protected]

6.5.3.8 char VFile::sds_time_[STRING_MAXLEN+1] [protected]

6.5.3.9 char VFile::time_filename_[STRING_MAXLEN+1] [protected]

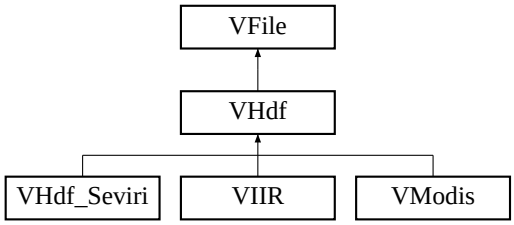
The documentation for this class was generated from the following file:

- src/VFiles/**VFile.h**

6.6 VHdf Class Reference

`#include <VHdf.h>`

Inheritance diagram for VHdf::



Public Member Functions

- **VHdf** (const char *filename, const char *dataset, int ichannel=0, const char *sds_time=NULL, const char *sds_latitude="Latitude", const char *sds_longitude="Longitude")
- int **ichannel** ()
- virtual void **get_calibration** (double &slope, double &offset) const
- **coord_type** * **lat** () const
- **coord_type** * **lon** () const
- **time_type** * **time** () const
- **data_type** * **data** () const
- virtual ~**VHdf** ()

Protected Member Functions

- virtual void **read_data_2D** (const char *dataset)
- virtual void **read_data_3D** (const char *dataset, int ichannel)
- virtual void **read_lat_lon_time** ()=0
- virtual void **destroy** ()

Protected Attributes

- float32 * **lat_**
- float32 * **lon_**
- **time_type** * **time_**
- uint16 * **data_**

6.6.1 Constructor & Destructor Documentation

6.6.1.1 VHdf::VHdf (const char * *filename*, const char * *dataset*, int *ichannel* = 0, const char * *sds_time* = NULL, const char * *sds_latitude* = "Latitude", const char * *sds_longitude* = "Longitude")

6.6.1.2 virtual VHdf::~VHdf () [virtual]

6.6.2 Member Function Documentation

6.6.2.1 data_type* VHdf::data () const [inline, virtual]

Implements **VFile** (p. 24).

Reimplemented in **VModis** (p. 31).

6.6.2.2 virtual void VHdf::destroy () [protected, virtual]

Reimplemented in **VModis** (p. 32).

6.6.2.3 virtual void VHdf::get_calibration (double & *slope*, double & *offset*) const [virtual]

Implements **VFile** (p. 24).

Reimplemented in **VHdf_Seviri** (p. 29), **VIIR** (p. 30), and **VModis** (p. 32).

6.6.2.4 int VHdf::ichannel () [inline]

Reimplemented in **VModis** (p. 32).

6.6.2.5 coord_type* VHdf::lat () const [inline, virtual]

Implements **VFile** (p. 24).

Reimplemented in **VModis** (p. 32).

6.6.2.6 coord_type* VHdf::lon () const [inline, virtual]

Implements **VFile** (p. 24).

Reimplemented in **VModis** (p. 32).

6.6.2.7 virtual void VHdf::read_data_2D (const char * *dataset*) [protected, virtual]

6.6.2.8 virtual void VHdf::read_data_3D (const char * *dataset*, int *ichannel*) [protected, virtual]

6.6.2.9 virtual void VHdf::read_lat_lon_time () [protected, pure virtual]

Implemented in **VHdf_Seviri** (p. 29), **VIIR** (p. 30), and **VModis** (p. 32).

6.6.2.10 time_type* VHdf::time () const [inline, virtual]

Implements **VFile** (p. 24).

Reimplemented in **VModis** (p. 32).

6.6.3 Member Data Documentation

6.6.3.1 uint16* VHdf::data_ [protected]

6.6.3.2 float32* VHdf::lat_ [protected]

6.6.3.3 float32* VHdf::lon_ [protected]

6.6.3.4 time_type* VHdf::time_ [protected]

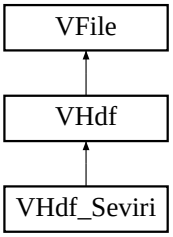
The documentation for this class was generated from the following file:

- src/VFiles/**VHdf.h**

6.7 VHdf_Seviri Class Reference

`#include <VHdf_Seviri.h>`

Inheritance diagram for VHdf_Seviri::



Public Member Functions

- **VHdf_Seviri** (const char *filename, const char *dataset)
- void **get_calibration** (double &slope, double &offset) const

Protected Member Functions

- void **read_lat_lon_time** ()

6.7.1 Constructor & Destructor Documentation

6.7.1.1 VHdf_Seviri::VHdf_Seviri (const char * *filename*, const char * *dataset*)

6.7.2 Member Function Documentation

6.7.2.1 void VHdf_Seviri::get_calibration (double & *slope*, double & *offset*) const [virtual]

Reimplemented from **VHdf** (p. 27).

6.7.2.2 void VHdf_Seviri::read_lat_lon_time () [protected, virtual]

Implements **VHdf** (p. 27).

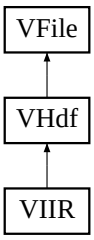
The documentation for this class was generated from the following file:

- src/VFiles/**VHdf_Seviri.h**

6.8 VIIR Class Reference

`#include <VIIR.h>`

Inheritance diagram for VIIR::



Public Member Functions

- **VIIR** (const char *filename, const char *dataset, int ichannel=0)
- void **get_calibration** (double &slope, double &offset) const

Protected Member Functions

- void **read_lat_lon_time** ()

6.8.1 Constructor & Destructor Documentation

6.8.1.1 VIIR::VIIR (const char * *filename*, const char * *dataset*, int *ichannel* = 0)

6.8.2 Member Function Documentation

6.8.2.1 void VIIR::get_calibration (double & *slope*, double & *offset*) const [virtual]

Reimplemented from **VHdf** (p. 27).

6.8.2.2 void VIIR::read_lat_lon_time () [protected, virtual]

Implements **VHdf** (p. 27).

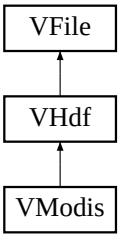
The documentation for this class was generated from the following file:

- src/VFiles/**VIIR.h**

6.9 VModis Class Reference

`#include <VModis.h>`

Inheritance diagram for VModis::



Public Member Functions

- **VModis** (const char *filename, const char *dataset, int ichannel=0)
- int **ichannel** ()
- void **get_calibration** (double &slope, double &offset) const
- **coord_type** * **lat** () const
- **coord_type** * **lon** () const
- **time_type** * **time** () const
- **data_type** * **data** () const
- ~**VModis** ()

Protected Member Functions

- void **read_lat_lon_time** ()
- void **destroy** ()

Private Member Functions

- void **interpol** (int sample_nrows, int sample_ncols, const **coord_type** *sample_lat, const **coord_type** *sample_lon, int full_nrows, int full_ncols, **coord_type** *full_lat, **coord_type** *full_lon)

6.9.1 Constructor & Destructor Documentation

6.9.1.1 VModis::VModis (const char * *filename*, const char * *dataset*, int *ichannel* = 0)

6.9.1.2 VModis::~VModis ()

6.9.2 Member Function Documentation

6.9.2.1 data_type* VModis::data () const [inline, virtual]

Reimplemented from **VHdf** (p. 27).

6.9.2.2 void VModis::destroy () [protected, virtual]

Reimplemented from **VHdf** (p. 27).

6.9.2.3 void VModis::get_calibration (double & *slope*, double & *offset*) const [virtual]

Reimplemented from **VHdf** (p. 27).

6.9.2.4 int VModis::ichannel () [inline]

Reimplemented from **VHdf** (p. 27).

6.9.2.5 void VModis::interpol (int *sample_nrows*, int *sample_ncols*, const coord_type * *sample_lat*, const coord_type * *sample_lon*, int *full_nrows*, int *full_ncols*, coord_type * *full_lat*, coord_type * *full_lon*) [private]

6.9.2.6 coord_type* VModis::lat () const [inline, virtual]

Reimplemented from **VHdf** (p. 27).

6.9.2.7 coord_type* VModis::lon () const [inline, virtual]

Reimplemented from **VHdf** (p. 27).

6.9.2.8 void VModis::read_lat_lon_time () [protected, virtual]

Implements **VHdf** (p. 27).

6.9.2.9 time_type* VModis::time () const [inline, virtual]

Reimplemented from **VHdf** (p. 28).

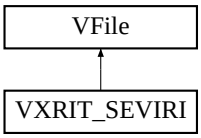
The documentation for this class was generated from the following file:

- src/VFiles/**VModis.h**

6.10 VXRIT_SEVIRI Class Reference

`#include <VXRIT_SEVIRI.h>`

Inheritance diagram for VXRIT_SEVIRI::



Public Member Functions

- **VXRIT_SEVIRI** (const char *filename, const char *dataset)
- void **get_calibration** (double &slope, double &offset) const
- **coord_type** * **lat** () const
- **coord_type** * **lon** () const
- **time_type** * **time** () const
- **data_type** * **data** () const
- ~**VXRIT_SEVIRI** ()

Private Member Functions

- void **read_seviri_latlon** (const char *filename, size_t nbuffer, float *buffer)
- void **allocate** (size_t n)
- void **destroy** ()

Private Attributes

- float * **lat_**
- float * **lon_**
- **time_type** * **time_**
- uint16_t * **data_**
- MSG_Prologue **prologue_**

Static Private Attributes

- const int **SEVIRI_NROWS** = 3712
- const int **SEVIRI_NCOLS** = 3712

6.10.1 Constructor & Destructor Documentation

6.10.1.1 VXRIT_SEVIRI::VXRIT_SEVIRI (const char * *filename*, const char * *dataset*)

6.10.1.2 VXRIT_SEVIRI::~VXRIT_SEVIRI ()

6.10.2 Member Function Documentation

6.10.2.1 void VXRIT_SEVIRI::allocate (size_t *n*) [private]

6.10.2.2 data_type* VXRIT_SEVIRI::data () const [inline, virtual]

Implements **VFile** (p. 24).

6.10.2.3 void VXRIT_SEVIRI::destroy () [private]

6.10.2.4 void VXRIT_SEVIRI::get_calibration (double & *slope*, double & *offset*) const [virtual]

Implements **VFile** (p. 24).

6.10.2.5 coord_type* VXRIT_SEVIRI::lat () const [inline, virtual]

Implements **VFile** (p. 24).

6.10.2.6 coord_type* VXRIT_SEVIRI::lon () const [inline, virtual]

Implements **VFile** (p. 24).

6.10.2.7 void VXRIT_SEVIRI::read_seviri_latlon (const char * *filename*, size_t *nbuffer*, float * *buffer*) [private]

6.10.2.8 time_type* VXRIT_SEVIRI::time () const [inline, virtual]

Implements **VFile** (p. 24).

6.10.3 Member Data Documentation

6.10.3.1 uint16_t* VXRIT_SEVIRI::data_ [private]

6.10.3.2 float* VXRIT_SEVIRI::lat_ [private]

6.10.3.3 float* VXRIT_SEVIRI::lon_ [private]

6.10.3.4 MSG_Prologue VXRIT_SEVIRI::prologue_ [private]

6.10.3.5 const int VXRIT_SEVIRI::SEVIRI_NCOLS = 3712 [static, private]

6.10.3.6 const int VXRIT_SEVIRI::SEVIRI_NROWS = 3712 [static, private]

6.10.3.7 time_type* VXRIT_SEVIRI::time_ [private]

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

- src/VFiles/**VXRIT_SEVIRI.h**

Chapter 7

Remap File Documentation

7.1 src/allocation.hpp File Reference

`#include <new>`

`#include <cassert>`

Defines

- #define **PDEBUG**

Functions

- template<typename data_type> void **allocate** (**data_type** **&data, int nrows, int ncols)
- template<typename data_type> void **deallocate** (**data_type** **&data)

7.1.1 Define Documentation

7.1.1.1 #define PDEBUG

7.1.2 Function Documentation

7.1.2.1 template<typename data_type> void allocate (data_type **& *data*, int *nrows*, int *ncols*)

7.1.2.2 template<typename data_type> void deallocate (data_type **& *data*)

7.2 src/common.h File Reference

```
#include <iostream>
#include <cassert>
#include <cstring>
#include <cstdio>
#include <cstdlib>
#include <stdint.h>
#include <ctime>
#include "Hdf.hpp"
#include "debug.h"
```

Defines

- #define **STRING_MAXLEN** 255
- #define **APPNAME** "remap"
- #define **OPT_VERBOSE** 0x00000001
- #define **DATA_TYPE_MIN** 0
- #define **DATA_TYPE_MAX** 65535
- #define **HDF_COORD_TYPE** DFNT_FLOAT32
- #define **HDF_DISTANCE_TYPE** DFNT_FLOAT32
- #define **HDF_TIME_TYPE** DFNT_FLOAT64
- #define **HDF_DATA_TYPE** DFNT_UINT16
- #define **DATA_TYPE** uint16_t
- #define **FLOAT_TYPE** float

Typedefs

- typedef FLOAT_TYPE **float_type**
- typedef **float_type distance_type**
- typedef **float_type coord_type**
- typedef float64 **time_type**
- typedef DATA_TYPE **data_type**

Variables

- const float **FLOAT_NAN** = 0./0.
- const double **DOUBLE_NAN** = 0./0.
- const **distance_type DEFAULT_DISTANCE_FILL_VALUE** = **FLOAT_NAN**
- const **coord_type DEFAULT_COORD_FILL_VALUE** = **FLOAT_NAN**
- const **time_type DEFAULT_TIME_FILL_VALUE** = **DOUBLE_NAN**
- const **data_type DEFAULT_DATA_FILL_VALUE** = DATA_TYPE_MAX
- bool **g_verbose**

7.2.1 Define Documentation

7.2.1.1 #define APPNAME "remap"

7.2.1.2 #define DATA_TYPE uint16_t

7.2.1.3 #define DATA_TYPE_MAX 65535

7.2.1.4 #define DATA_TYPE_MIN 0

7.2.1.5 #define FLOAT_TYPE float

7.2.1.6 #define HDF_COORD_TYPE DFNT_FLOAT32

7.2.1.7 #define HDF_DATA_TYPE DFNT_UINT16

7.2.1.8 #define HDF_DISTANCE_TYPE DFNT_FLOAT32

7.2.1.9 #define HDF_TIME_TYPE DFNT_FLOAT64

7.2.1.10 #define OPT_VERBOSE 0x00000001

7.2.1.11 #define STRING_MAXLEN 255

7.2.2 Typedef Documentation

7.2.2.1 typedef float_type coord_type

7.2.2.2 typedef DATA_TYPE data_type

7.2.2.3 typedef float_type distance_type

7.2.2.4 typedef FLOAT_TYPE float_type

7.2.2.5 typedef float64 time_type

7.2.3 Variable Documentation

7.2.3.1 const coord_type DEFAULT_COORD_FILL_VALUE = FLOAT_NAN

7.2.3.2 const data_type DEFAULT_DATA_FILL_VALUE = DATA_TYPE_MAX

7.2.3.3 const distance_type DEFAULT_DISTANCE_FILL_VALUE = FLOAT_NAN

7.2.3.4 const time_type DEFAULT_TIME_FILL_VALUE = DOUBLE_NAN

7.2.3.5 const double DOUBLE_NAN = 0./0.

7.2.3.6 const float FLOAT_NAN = 0./0.

7.2.3.7 bool g_verbose

7.3 src/debug.h File Reference

```
#include <iostream>
#include <cassert>
```

Defines

- #define **Debug**(code)
- #define **PDEBUG**
- #define **Bench**(code) do { code } while (0)

7.3.1 Define Documentation

7.3.1.1 #define Bench(code) do { code } while (0)

7.3.1.2 #define Debug(code)

7.3.1.3 #define PDEBUG

7.4 src/Pixel/debug.h File Reference

```
#include <iostream>
#include <cassert>
#include <ctime>
```

Defines

- #define **Debug**(code)
- #define **Bench**(code) do { code } while (0)

7.4.1 Define Documentation

7.4.1.1 #define Bench(code) do { code } while (0)

7.4.1.2 #define Debug(code)

7.5 src/VFiles/debug.h File Reference

```
#include <iostream>
#include <cassert>
```

Defines

- #define **Debug**(code)
- #define **PDEBUG**
- #define **Bench**(code) do { code } while (0)

7.5.1 Define Documentation

7.5.1.1 #define Bench(code) do { code } while (0)

7.5.1.2 #define Debug(code)

7.5.1.3 #define PDEBUG

7.6 src/filetypes.h File Reference

Enumerations

- enum **filetype_type** {
  FILETYPE_MODIS_AQUA_1KM, **FILETYPE_MODIS_TERRA_1KM**, **FILETYPE_CAL_IIR_L1**, **FILETYPE_HDF_SEVIRI**,
  FILETYPE_XRIT_SEVIRI, **FILETYPE_UNKNOWN** }

Functions

- char * **filetype_to_cstr** (**filetype_type** filetype)
- **filetype_type get_filetype** (const char *filename)
- void **print_supported_filetypes** ()

7.6.1 Enumeration Type Documentation

7.6.1.1 enum filetype_type

Enum type for filetypes (files identifiers). Once a file is resolved into a filetype (or file identifier), the latter will be used to instantiate a specific file object able to handle its contents

Enumeration values:

FILETYPE_MODIS_AQUA_1KM enum code identifying MYD021KM files (Modis over Aqua products)

FILETYPE_MODIS_TERRA_1KM enum code identifying MOD021KM files (Modis over Terra products)

FILETYPE_CAL_IIR_L1 enum code identifying CAL_IIR_L1 files (IIR over CALIPSO products)

FILETYPE_HDF_SEVIRI enum code identifying SEVIRI mosaic files (SEVIRI hdf products, from Icare)

FILETYPE_XRIT_SEVIRI enum code identifying SEVIRI original files (SEVIRI XRIT products)

FILETYPE_UNKNOWN enum code for not known or not supported products (should always be the last field of the enum)

7.6.2 Function Documentation

7.6.2.1 char* filetype_to_cstr (filetype_type *filetype*)

resolves a filetype into a static C-style (char *) string (one of the enum codes from filetype_type).

cautious: as the returned value is a pointer to a static zone of memory, the function should never be called more than once in the same expression, nor in a multithread application (in theses cases its behaviour is undefined)

Parameters:

filetype one of the filetypes owned by the filetype_type enumeration

Returns:
a pointer adressing a static array of chars containing a human readable description of the filetype

7.6.2.2 filetype_type get_filetype (const char * *filename*)

resolves a filename into one of the filetypes from the filetype_type enumeration

Parameters:
filename the name of one of the software-supported products

Returns:
one of the filetypes from the filetype_type enumeration, or FILETYPE_UNKNOWN is the argument is invalid or not supported

7.6.2.3 void print_supported_filetypes ()

an helper function that prints every supported types of files in the application

it simply calls filetype_to_cstr for each field of the filetype_type enumeration (except FILETYPE_UNKNOWN) and prints the result on the standard error

See also:
filetype_type(p. 43)
filetype_to_cstr(filetype_type filetype)(p. 43)
usage()

7.7 src/grid.h File Reference

```
#include <ctime>
#include "Hdf.hpp"
#include "common.h"
#include "VFiles.h"
```

Classes

- struct **grid_type_**

Typedefs

- typedef **grid_type_** **grid_type**

Functions

- void **create_grid** (**grid_type** *grid, int nrows=0, int ncols=0, float64 slope=1., float64 offset=0., **data_type** data_fill_value=**DEFAULT_DATA_FILL_VALUE**, **coord_type** coord_fill_value=**DEFAULT_COORD_FILL_VALUE**, **distance_type** distance_fill_value=**DEFAULT_DISTANCE_FILL_VALUE**, **time_type** time_fill_value=**DEFAULT_TIME_FILL_VALUE**)
- void **reset_grid** (**grid_type** *grid, **data_type** data_fill_value, **coord_type** coord_fill_value, **distance_type** distance_fill_value, **time_type** time_fill_value)
- void **reset_grid** (**grid_type** *grid, **data_type** data_fill_value, **distance_type** distance_fill_value, **time_type** time_fill_value)
- void **destroy_grid** (**grid_type** *grid)
- int **load_grid** (**grid_type** *grid, **data_type** data_fill_value=**DEFAULT_DATA_FILL_VALUE**, **coord_type** coord_fill_value=**DEFAULT_COORD_FILL_VALUE**, **distance_type** distance_fill_value=**DEFAULT_DISTANCE_FILL_VALUE**, **time_type** time_fill_value=**DEFAULT_TIME_FILL_VALUE**)
- int **save_grid** (**grid_type** *grid, bool record_delta, bool record_latlontime)
- void **copy_grid** (**grid_type** *target_grid, **grid_type** *src_grid)
- void **copy_grid_footprint** (**grid_type** *target_grid, **grid_type** *src_grid)
- bool **have_same_grid_footprint** (**grid_type** *grid1, **grid_type** *grid2)

7.7.1 Typedef Documentation

7.7.1.1 typedef struct grid_type_ grid_type

Grids are the main data structures handled by the software. They should have been designed as a class instead of a simple struct with functions to handle it, but by lack of time the software had to be delivered as is. A rewritten code with a grid class instead of a struct should be much clearer and easier to maintain, but will need a substantial amount of time to reimplement (and of course redocument !), that is not available today.

Grids are abstracts for reprojection. Their main purpose is to handle 2-dimensional buffers of geolocated data (measures along with their latitudes, longitudes and times of acquisition). The reprojection algorithm remaps a grid of data (from one instrument product) into another one. For

convenience, origin and target of the data (files and datasets) are also maintained in the structure (although this is not a very clever design, I have to confess, I hope I'll have a chance to change this if more time is given to this project)

7.7.2 Function Documentation

7.7.2.1 void copy_grid (grid_type * *target_grid*, grid_type * *src_grid*)

intended to copy src_grid into target_grid (does correct allocation and copies) this function is not used in the current implementation

7.7.2.2 void copy_grid_footprint (grid_type * *target_grid*, grid_type * *src_grid*)

intended to copy the footprint of a grid (the footprint includes lightweight data, namely scalar fields, but not buffers) this function is not used in the current implementation

7.7.2.3 void create_grid (grid_type * *grid*, int *nrows* = 0, int *ncols* = 0, float64 *slope* = 1., float64 *offset* = 0., data_type *data_fill_value* = DEFAULT_DATA_FILL_VALUE, coord_type *coord_fill_value* = DEFAULT_COORD_FILL_VALUE, distance_type *distance_fill_value* = DEFAULT_DISTANCE_FILL_VALUE, time_type *time_fill_value* = DEFAULT_TIME_FILL_VALUE)

the grid constructor

Parameters:

grid the address of a grid structure to initialize. All fields will be allocated (for field pointers) and initialized with the values of the subsequent parameters.

nrows the number of rows of the grid (common number of rows of all the buffers in the grid)

ncols the number of columns of the grid (common number of columns of all the buffers in the grid)

slope the initialization value for the slope field of the grid

offset the initialization value for the offset field of the grid

data_fill_value the initialization value for the **grid_type::data**(p. 14) buffer of the grid (after allocation of nrows*ncols elements)

coord_fill_value the initialization value the **grid_type::lat**(p. 15) and **grid_type::lon**(p. 15) buffers of the grid (after allocation of nrows*ncols elements)

distance_fill_value the initialization value for the **grid_type::distance_from_ref**(p. 15) buffer of the grid (after allocation of nrows*ncols elements)

time_fill_value the initialization value for the **grid_type::tim**(p. 16) and **grid_type::time_from_ref**(p. 16) buffers of the grid (after allocation of nrows*ncols elements)

See also:

reset_grid(grid_type *, data_type, coord_type, distance_type, time_type)(p. 48)

cautious: create_grid currently does not initialize the **grid_type::is_target**(p. 15) field (this one had to be added to meet some user's new requirements, only after the interface of **create_grid**(p. 46) was designed. So for the time being, the **grid_type::is_target**(p. 15) field must be

set directly before **create_grid**(p. 46) is called, in order to allow **create_grid**(p. 46) to allocate and initialize correctly the **grid_type::src_irows**(p. 16) and **grid_type::src_icols**(p. 16) fields. This should be fixed in a future release (this will imply to break the interface of **create_grid**(p. 46), in order to pass the initialization value of **grid_type::is_target**(p. 15) as an argument).

7.7.2.4 void destroy_grid (grid_type * *grid*)

the destructor of a grid

it resets all the scalars fields and deallocates (and nullifies) all the pointer fields of a grid structure

Parameters:

grid a pointer to the grid to destroy

See also:

create_grid(p. 46)
load_grid(p. 47)

7.7.2.5 bool have_same_grid_footprint (grid_type * *grid1*, grid_type * *grid2*)

intended to compare two grids footprints (the footprint includes lightweight data, namely scalar fields, but not buffers) this function is not used in the current implementation

7.7.2.6 int load_grid (grid_type * *grid*, data_type *data_fill_value* = DEFAULT_DATA_FILL_VALUE, coord_type *coord_fill_value* = DEFAULT_COORD_FILL_VALUE, distance_type *distance_fill_value* = DEFAULT_DISTANCE_FILL_VALUE, time_type *time_fill_value* = DEFAULT_TIME_FILL_VALUE)

loads data from the dataset **grid_type::input_dataset**(p. 15) of the file **grid_type::file**(p. 15) into a grid. The function calls **create_grid**(p. 46), so don't do it yourself. Grids allocated and filled by load_grid must be destroyed by the caller with **destroy_grid**(p. 47)

Parameters:

grid a pointer to the **grid_type**(p. 45) structure to allocate and fill with the file data

data_fill_value the new initialization value of the **grid_type::data**(p. 14) buffer elements

coord_fill_value the new initialization value of the **grid_type::lat**(p. 15), **grid_type::lon**(p. 15) buffer elements

distance_fill_value the new initialization value of the **grid_type::distance_from_ref**(p. 15) buffer elements

time_fill_value the new initialization value of the **grid_type::tim**(p. 16) and **grid_type::time_from_ref**(p. 16) buffer_elements

See also:

create_grid(p. 46) (called by this one)
destroy_grid(p. 47) (to call when the grid is not used anymore)

7.7.2.7 void reset_grid (grid_type * *grid*, data_type *data_fill_value*, distance_type *distance_fill_value*, time_type *time_fill_value*)

resets all the fields of a grid, except **grid_type::lat**(p. 15), **grid_type::lon**(p. 15), **grid_type::tim**(p. 16)

It is used for the target grid whose geolocation fields never change. On the other hand, **grid_type::data**(p. 14) (measures), **grid_type::distance_from_ref**(p. 15) and **grid_type::time_from_ref**(p. 16) buffers must be reset with each new source grid (as source will be remapped into target). This function is essentially called from the main loop in the ::main function.

Parameters:

grid the pointer to the grid to reset

data_fill_value the new initialization value of the **grid_type::data**(p. 14) buffer elements

distance_fill_value the new initialization value of the **grid_type::distance_from_ref**(p. 15) buffer elements

time_fill_value the new initialization value of the **grid_type::time_from_ref**(p. 16) buffer_elements

7.7.2.8 void reset_grid (grid_type * *grid*, data_type *data_fill_value*, coord_type *coord_fill_value*, distance_type *distance_fill_value*, time_type *time_fill_value*)

resets all the fields of a grid. It is essentially called by **create_grid**(p. 46) after allocations of pointers have been done.

Parameters:

grid the pointer to the grid to reset

data_fill_value the new initialization value of the **grid_type::data**(p. 14) buffer elements

coord_fill_value the new initialization value of the **grid_type::lat**(p. 15), **grid_type::lon**(p. 15) buffer elements

distance_fill_value the new initialization value of the **grid_type::distance_from_ref**(p. 15) buffer elements

time_fill_value the new initialization value of the **grid_type::tim**(p. 16) and **grid_type::time_from_ref**(p. 16) buffer_elements

See also:

create_grid(p. 46)

7.7.2.9 int save_grid (grid_type * *grid*, bool *record_delta*, bool *record_latlontime*)

saves **grid_type::data**(p. 14) buffer (with its slope and offset) into the **grid_type::output_dataset**(p. 16) of the file **grid_type::file**(p. 15) recorded in the grid structure.

Parameters:

grid a pointer to the grid to save into a file

record_delta if true, saves **grid_type::distance_from_ref**(p. 15) and **grid_type::time_from_ref**(p. 16) (namely 'delta' buffers) along with the **grid_type::data**(p. 14) buffer (this parameter may be controled by a user option, see ::main)

record_latlontime if true, saves **grid_type::lat**(p. 15), **grid_type::lon**(p. 15) and **grid_type::tim**(p. 16) buffers along with the **grid_type::data**(p. 14) buffer (this is currently only used to save latitudes, longitudes and times of acquisition of the reference grid, see ::main)

7.8 src/hdf_utils.h File Reference

```
#include "Hdf.hpp"
```

Functions

- int **hdf_create_empty_file** (const char *file)
- int **hdf_add_empty_sds** (const char *file, const char *sds_name, int32 sds_type, int32 rank, int32 *dimensions, const float64 cal=1., const float64 offset=0., const float64 cal_err=0., const float64 off_err=0.)

7.8.1 Function Documentation

7.8.1.1 int hdf_add_empty_sds (const char * *file*, const char * *sds_name*, int32 *sds_type*, int32 *rank*, int32 * *dimensions*, const float64 *cal* = 1., const float64 *offset* = 0., const float64 *cal_err* = 0., const float64 *off_err* = 0.)

creates a new dataset in an existing hdf file (will fail if a dataset with the same name already exists)

Parameters:

file the hdf file to update

sds_name the name of the new dataset (sds: scientific dataset) to add

sds_type the hdf type code of the new dataset, e.g. DFNT_UINT16, DFNT_FLOAT32... (see HDF documentation for valid type codes)

rank the rank (number of dimensions) of the new dataset to create

dimensions a pointer to the array of dimensions of the new dataset

cal the calibration factor linked to the new dataset (also named slope); relation between calibrated and uncalibrated data is: cal_data = cal*(uncal_data - offset)

offset the offset factor linked to the new dataset; relation between calibrated and uncalibrated data is: cal_data = cal*(uncal_data - offset)

cal_err error on the calibration factor (unused for the time being)

off_err error on the offset (unused for the time being)

Returns:

0 on success, -1 on failure

7.8.1.2 int hdf_create_empty_file (const char * *file*)

creates a new (empty) hdf file

Parameters:

file the name of the file to create

Returns:

0 on success, -1 on failure

7.9 src/parse_argument.h File Reference

```
#include "grid.h"
```

Functions

- int **parse_argument** (int iarg, char *argv[], **grid_type** *grid)

7.9.1 Function Documentation

7.9.1.1 int parse_argument (int *iarg*, char * *argv*[], grid_type * *grid*)

parses a user argument in order to extract some useful values for a grid_type structure. Expected argument must match the regular expression ^([^[@/]+)(\[([0-9]+)\])?(/[^@]+)?@(.*)$ which means in clear one of the four alternatives :

- input_dataset@input_file
- input_dataset[ichannel]@input_file
- input_dataset/output_dataset@input_file
- input_dataset[ichannel]/output_dataset@input_file

(with : ichannel an integer, input_dataset a char *, output_dataset a char *, and input_file a char *)

Parameters:

iarg the index of the argument to parse (between 1 and argc - 1)

argv the list of the command arguments, from ::main

grid a pointer to a grid_type structure to update with infos found in the i-th argument

Returns:

0 on success, -1 on failure; the current implementation actually exits in a brutal way when a invalid argument is found, simply sending an error message on the standard error as a side effect. Indeed, an invalid argument was seen as a fatal error in the application context (no processing can be done), so it was judged useless to continue.

The fields of grid_type updated by parse_argument include:

- **grid_type::input_dataset**(p. 15)
- **grid_type::output_dataset**(p. 16) (set to **grid_type::input_dataset**(p. 15) if not found in the regexp)
- **grid_type::file**(p. 15)
- **grid_type::ichannel**(p. 15) (set to 0 if not found in the regexp)

7.10 src/Pixel/Pixel.h File Reference

```
#include <cassert>
#include <set>
#include <algorithm>
#include <utility>
#include <functional>
#include <cmath>
#include <vector>
#include <iostream>
#include "debug.h"
```

Classes

- class **Nearer_from< Pixel_type >**
- class **Pixel_base< T, V >**

Defines

- #define **BENCH**

Functions

- template<typename T> T **sqr** (T x)

7.10.1 Define Documentation

7.10.1.1 #define BENCH

7.10.2 Function Documentation

7.10.2.1 template<typename T> T sqr (T *x*) [inline]

7.11 src/reproject.h File Reference

```
#include "common.h"
#include "grid.h"
```

Functions

- void **reproject** (**grid_type** *src_grid, **grid_type** *target_grid, **distance_type** distance, **time_type** max_dtime)

7.11.1 Function Documentation

7.11.1.1 void reproject (grid_type * *src_grid*, grid_type * *target_grid*, distance_type *distance*, time_type *max_dtime*)

reprojects src_grid over target_grid

The current implementation uses a *"nearest neighbour"* policy, an optional argument might be added to choose the reprojection policy (with this one by default)

Source and target grids must meet some requirements :

- their **grid_type::lat**(p. 15), **grid_type::lon**(p. 15) and **grid_type::tim**(p. 16) fields must be filled
- **grid_type::lat**(p. 15) must contain values in *degrees* between MIN_LAT and MAX_LAT
- **grid_type::lon**(p. 15) must contain values in *degrees* between MIN_LON and MAX_LON
- **grid_type::tim**(p. 16) must contain values in **time_type**(p. 39) (TAI93, Temps Atomique International 1993), e.g. a number of seconds since the 1st Jan 1993, 0h00 UTC

Parameters:

src_grid the grid whose content is to reproject

target_grid the grid over which reprojection is to be done

distance the maximal search distance for the nearest neighbour

max_dtime the maximal time difference for the nearest neighbour

reproject(p. 53) updates the following fields of target_grid :

- **grid_type::data**(p. 14)
- **grid_type::distance_from_ref**(p. 15)
- **grid_type::time_from_ref**(p. 16)
- **grid_type::src_irows**(p. 16) (currently not used)
- **grid_type::src_icols**(p. 16) (currently not used)

7.12 src/tokenize.h File Reference

Typedefs

- typedef enum **etok_type_** **etok_type**

Enumerations

- enum **etok_type_** {
  ETOK_SUCCESS, **ETOK_REGCOMP**, **ETOK_MEMORY**, **ETOK_NOMATCH**,
  ETOK_ESPACE }

Functions

- **etok_type tokenize** (const char *pattern, const char *a_string, size_t *ntokens, char ***ptokens)
- void **free_tokens** (size_t ntokens, char **tokens)

7.12.1 Typedef Documentation

7.12.1.1 typedef enum etok_type_ etok_type

7.12.2 Enumeration Type Documentation

7.12.2.1 enum etok_type_

Enumeration values:

ETOK_SUCCESS

ETOK_REGCOMP

ETOK_MEMORY

ETOK_NOMATCH

ETOK_ESPACE

7.12.3 Function Documentation

7.12.3.1 void free_tokens (size_t *ntokens*, char ** *tokens*)

7.12.3.2 etok_type tokenize (const char * *pattern*, const char * *a_string*, size_t * *ntokens*, char *** *ptokens*)

7.13 src/VFiles/normalize_cal_factors.h File Reference

Functions

- void **normalize_seviri_factors** (int met8_channel, double &slope, double &offset)
- void **normalize_seviri_factors** (const char *dataset, double &slope, double &offset)

7.13.1 Function Documentation

7.13.1.1 void normalize_seviri_factors (const char * *dataset*, double & *slope*, double & *offset*)

7.13.1.2 void normalize_seviri_factors (int *met8_channel*, double & *slope*, double & *offset*)

7.14 src/VFiles/seviri_latlon/geostat.h File Reference

Typedefs

- typedef enum **channel_enum channel_t**
- typedef enum **geostat_err_enum geostat_err_t**

Enumerations

- enum **channel_enum** { **GEOSTAT_CHNL_IR** = 1, **GEOSTAT_CHNL_VIS** = 2, **GEOSTAT_CHNL_WV** = 3 }
- enum **geostat_err_enum** { **GEOSTAT_ERR_OK** = 0, **GEOSTAT_ERR_OUT_OF_RANGE** = -1, **GEOSTAT_ERR_BAD_INIT** = -2, **GEOSTAT_ERR_BAD_INPUT_COORD** = -3 }

Functions

- **geostat_err_t geostat_init** (const char *satellite)
- **geostat_err_t geostat_latlon_to_lincol_double** (double latitude, double longitude, double *line, double *column)
- **geostat_err_t geostat_lincol_to_latlon_double** (double line, double column, double *latitude, double *longitude)
- **geostat_err_t geostat_latlon_to_lincol** (const double latitude, const double longitude, unsigned long *line, unsigned long *column)
- **geostat_err_t geostat_lincol_to_latlon** (const unsigned long line, const unsigned long column, double *latitude, double *longitude)
- **geostat_err_t geostat_get_azimut_and_zenithal** (const double latitude, const double longitude, double *azimut, double *zenithal)

7.14.1 Typedef Documentation

7.14.1.1 typedef enum channel_enum channel_t

7.14.1.2 typedef enum geostat_err_enum geostat_err_t

7.14.2 Enumeration Type Documentation

7.14.2.1 enum channel_enum

Enumeration values:

GEOSTAT_CHNL_IR

GEOSTAT_CHNL_VIS

GEOSTAT_CHNL_WV

7.14.2.2 enum geostat_err_enum

Enumeration values:

GEOSTAT_ERR_OK

GEOSTAT_ERR_OUT_OF_RANGE
GEOSTAT_ERR_BAD_INIT
GEOSTAT_ERR_BAD_INPUT_COORD

7.14.3 Function Documentation

7.14.3.1 geostat_err_t geostat_get_azimut_and_zenithal (const double *latitude*, const double *longitude*, double * *azimut*, double * *zenithal*)

7.14.3.2 geostat_err_t geostat_init (const char * *satellite*)

7.14.3.3 geostat_err_t geostat_latlon_to_lincol (const double *latitude*, const double *longitude*, unsigned long * *line*, unsigned long * *column*)

7.14.3.4 geostat_err_t geostat_latlon_to_lincol_double (double *latitude*, double *longitude*, double * *line*, double * *column*)

7.14.3.5 geostat_err_t geostat_lincol_to_latlon (const unsigned long *line*, const unsigned long *column*, double * *latitude*, double * *longitude*)

7.14.3.6 geostat_err_t geostat_lincol_to_latlon_double (double *line*, double *column*, double * *latitude*, double * *longitude*)

7.15 src/VFiles/VFile.h File Reference

```
#include <cassert>
#include <stdint.h>
#include <cstring>
#include <vector>
#include "common.h"
#include "debug.h"
```

Classes

- class **VFile**

Defines

- #define **DEBUG**

7.15.1 Define Documentation

7.15.1.1 #define DEBUG

7.16 src/VFiles/VFiles.h File Reference

```
#include "VFile.h"
#include "VHdf.h"
#include "VIIR.h"
#include "VHdf_Seviri.h"
#include "VXRIT_SEVIRI.h"
#include "VModis.h"
```

7.17 src/VFiles/VHdf.h File Reference

```
#include <cstring>
#include "VFile.h"
#include "Hdf.hpp"
```

Classes

- class **VHdf**

7.18 src/VFiles/VHdf_Seviri.h File Reference

```
#include "VHdf.h"
```

Classes

- class **VHdf_Seviri**

7.19 src/VFiles/VIIR.h File Reference

```
#include "VHdf.h"
```

Classes

- class **VIIR**

7.20 src/VFiles/VModis.h File Reference

```
#include <cstring>
#include "VHdf.h"
#include "Hdf.hpp"
```

Classes

- class **VModis**

7.21 src/VFiles/VModis_interpol.h File Reference

```
#include "common.h"
```

Classes

- struct **PRODUCT**

Enumerations

- enum **PROJTYPE** { **INTERP_SWATH** }
- enum **INTERP_OUTPUT** { **INTERP_BOUNDS**, **INTERP_CENTERS** }

Functions

- void **get_step_and_offset** (int n1, int n2, float *offset, float *step, int rowsperscan, int georowsamplingstep, unsigned char crosstrack)
- void **resample_row_geolocation** (**INTERP_OUTPUT** output, double *oldlon, double *oldlat, **PRODUCT** *product, **coord_type** *newlon, **coord_type** *newlat)
- void **get_virtual_row** (const **coord_type** *lon1, const **coord_type** *lat1, const **coord_type** *lon2, const **coord_type** *lat2, double t, int ncols, double *lon3, double *lat3)
- void **get_georows** (double fractrow, **PRODUCT** *input, int iscan, int *georow1, int *georow2, double *t)

7.21.1 Enumeration Type Documentation

7.21.1.1 enum INTERP_OUTPUT

Enumeration values:

INTERP_BOUNDS

INTERP_CENTERS

7.21.1.2 enum PROJTYPE

Enumeration values:

INTERP_SWATH

7.21.2 Function Documentation

7.21.2.1 void get_georows (double *fractrow*, PRODUCT * *input*, int *iscan*, int * *georow1*, int * *georow2*, double * *t*)

7.21.2.2 void get_step_and_offset (int *n1*, int *n2*, float * *offset*, float * *step*, int *rowsperscan*, int *georowsamplingstep*, unsigned char *crosstrack*)

7.21.2.3 void get_virtual_row (const coord_type * *lon1*, const coord_type * *lat1*, const coord_type * *lon2*, const coord_type * *lat2*, double *t*, int *ncols*, double * *lon3*, double * *lat3*)

7.21.2.4 void resample_row_geolocation (INTERP_OUTPUT *output*, double * *oldlon*, double * *oldlat*, PRODUCT * *product*, coord_type * *newlon*, coord_type * *newlat*)

7.22 src/VFiles/VModis_latlon_resolve.h File Reference

Functions

- char * **VModis_latlon_resolve** (char *modis_file, char *modis_latlon_path=NULL)

7.22.1 Function Documentation

7.22.1.1 char* VModis_latlon_resolve (char * *modis_file*, char * *modis_latlon_path* = NULL)

7.23 src/VFiles/VXRIT_SEVIRI.h File Reference

```
#include <vector>
#include <stdint.h>
#include "VFile.h"
#include "libxrit.h"
```

Classes

- class **VXRIT_SEVIRI**

Index