# ODV API for C++

Reiner Schlitzer

Version 2.0
March 24, 2015

**OCEAN DATA VIEW APPLICATION PROGRAMMING INTERFACE (ODV4API) LICENSE AGREEMENT**

By downloading or using this Software, you agree to be bound by the following legal agreement between you and the Alfred-Wegener-Institute for Polar and Marine Research (AWI). If you do not agree to the terms of this Agreement, do not download or use the Software.

1. SCIENTIFIC USE AND TEACHING

The ODV4API is allowed to be used free of charge for non-commercial, non-military research and teaching purposes only. If you use the software for your scientific work, you must reference Ocean Data View in your publications as follows:

Schlitzer, R., Ocean Data View Application Programming Interface (ODV4API), http://odv.awi.de, 2015.

2. COMMERCIAL AND MILITARY USE

For the use of the ODV4API or any of its components for commercial or military applications and products, a special, written software license is needed. Please contact the address below for further information.

3. REDISTRIBUTION

Redistribution of the ODV4API software on CD-ROM, DVD, or other electronic media or the Internet is not permitted without the prior written consent of the AWI. Please contact the address below for further information.

4. WARRANTY DISCLAIMER

THE ODV4API SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE SOFTWARE IS WITH YOU. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT WILL AWI, ITS CONTRIBUTORS OR ANY ODV COPYRIGHT HOLDER BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY DIRECT, INDIRECT, GENERAL, SPECIAL, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL DAMAGES HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES, A FAILURE OF THE SOFTWARE TO OPERATE WITH ANY OTHER SOFTWARE OR BUSINESS INTERRUPTION).

# Table of Contents

# ODV Application Programming Interface for C++

**Author:**

Reiner Schlitzer



## Introduction

The ODV API for C++ provides a set of classes that can be used in your own C++ applications to open existing *Ocean Data View* data collections and access metadata and data of arbitrary stations in the collection. This opens the way for custom data usage strategies not already covered by the *Ocean Data View* software itself.

Compiled versions of the ODV API are provided for Windows, Mac OS X and Linux systems as 64 and 32 bit libraries. The API package includes detailed documentation of all provided classes and their functions. The package also includes example C++ code showing how to use the API.

### Main Classes

The ODV API provides three fundamental classes that every application requires to successfully open an existing ODV data collection and read its data: (1) *ODVCollection* , (2) *ODVVariable* , and (3) *ODVStation* .

(1) An *ODVCollection* object represents an ODV data collection on a local or network-attached storage device. This class provides functions for opening and closing the collection, inquiring the number and kind of metadata and data variables maintained by the collection and inquiring the number of stations in the collection.

(2) Metadata and data variables are described by *ODVVariable* objects. *ODVVariables* have name and unit labels as well as variable and value types. *ODVVariables* can hold numeric or string data.

(3) *ODVStation* objects can hold the metadata and data of one station in the collection. This class provides functions for reading the metadata or data of a given station and for providing access to the numeric or string data of arbitrary metadata and data variables for the given station. The *readData()* function can be called repeatedly to load the data of arbitrary stations into the *ODVStation* object (thereby overwriting any previously loaded station data).

If you need the data of two or more stations at the same time, for instance for calculating geostrophic flows, you create two or more *ODVStation* objects and load the required station data into the respective objects.

The *ODVCollectionInventory* and *ODVCruiseInfo* classes are not necessarily required for basic metadata and data access, but provide convenient and fast access to basic metadata, such as longitudes, latitudes, dates and times. These metadata are crucial for quickly scanning over large data collections and filtering station subsets in particular space and time domains.

### Installation

Installation information for your platform and descriptions of the content of the ODV API package can be found in the *install_odv4api_c++_....txt* file shipped with the ODV API.

## Example Code

Two examples, *simplecollectioninfo* and *exportspreadsheet* , are provided to show how to use the ODV API. The source code of the examples as well as template *Qt .pro* project files are found in directories *examples* and *examples* , respectively. Compiled executables of the examples are in the *examples* directory.

The *exportspreadsheet* application opens an ODV collection, reads station by station and writes the station metadata and data to a spreadsheet text file.

The function *exportAsTextTable()* in file *exportspreadsheet.cpp* shows a typical use of the ODV API for reading data from a collection. First, an ODVCollection object is created and the ODVCollection::open() function is called. Then an ODVStation object is created using a pointer to the collection object as argument. The metadata and data of a particular station are then read from the collection using the ODVStation::readData() function. Pointers to the collection's data and metadata variables are obtained via the ODVCollection::var() or ODVCollection::metaVar() functions. Finally, numeric or string data for metadata and data variables in the given station are obtained with them ODVStation::value() and ODVStation::stringValue() functions. The ODVStation::data() function is called to obtain a pointer to the array of data values of a particular variable for all the samples of the given station. A pointer to the quality flag data of a variable is obtained with the ODVStation::qfData() function. The ODVCollectionInventory class is useful for quickly retrieving position and date/time information for all or some stations in the collection.

## HowTo Build Your Own Application

Before developing your own C++ ODV API application you must install the 64 or 32 bit version of Qt 5 (http://www.qt.io) on your development system. In your C++ code you must include the Qt headers as well as the ODV header files contained in directory *include* . You must link against the ODV API and *Qt5Core.lib* .

As a first step you must prepare a project file (extension *.pro* ) that will be used by the Qt `qmake` utility to create a Makefile or Visual Studio or XCode project file, which then can be used to build your application. Example *.pro* project files can be found in *examples* . Use these project files as templates for your own application. Normally only few changes are needed for adaptation. Simply assign your own source files to the variable `SOURCES` and your own header files to the variable `HEADERS` . If you need additional libraries add them to the variable `LIBS` . Note that `"USEDLL"` must always be defined when using the ODV API. `qmake` project files allow many options and settings, please see the Qt qmake manual for details.

Your application executable will have the same name as the project file (without extension). See the comments at the top of the example project files for the `qmake` calling syntax.

# Hierarchical Index

## Class Hierarchy

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

# Class Index

## Class List

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

# File Index

## File List

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

# Class Documentation

## ODV Class Reference

Declarations for global use.
`#include "odv.h"`

### Public Types

- enum Status { NoErr =0, UserAbort, EOD, CollReadOnly, UnknownFileTypeErr, NoSuchDirErr, DirCreateErr, FileOpenErr, FileReadErr, FileWriteErr, FileErr, CollOpenErr, CollCreateErr, CollDelErr, CollCopyErr, CollFormatUnsupported, CollReadErr, CollWriteErr, CollNotSorted, ImportErr, ImportProblems, AnalyzeFileErr, MacroErr, VarNotFound, PSPreambleNotFound, StatIDOutOfRange, InvalidStationData, OutOfMemory, BadParameter, NotImplemented }
- enum AccessMode { NoAccess =0, ReadOnly =1, ReadWrite =3 }
- enum DateForm { IsoDate, mmddyyyyDate, mmmddyyyyDate, ddmonthyyyyDate, ddmmmyyyyDate }

### Static Public Attributes

- static const qint8 missINT8
- static const quint8 missUINT8
- static const qint16 missINT16
- static const quint16 missUINT16
- static const qint32 missINT32
- static const quint32 missUINT32
- static const float missFLOAT
- static const double missDOUBLE
- static const qint8 largeINT8
- static const quint8 largeUINT8
- static const qint16 largeINT16
- static const quint16 largeUINT16
- static const qint32 largeINT32
- static const quint32 largeUINT32
- static const float largeFLOAT
- static const double largeDOUBLE
- static const qint32 **STREAMVERSION**

---

### Detailed Description

Declarations for global use.

The ODV class contains declarations and definitions used throughout ODV. This includes the return status of functions and constants.

---

### Member Enumeration Documentation

#### enum ODV::AccessMode

File and collection access modes

**Enumerator**

***NoAccess*** No access.

***ReadOnly*** Read-only access.

***ReadWrite*** Read-write access.

**enum ODV::DateForm**

Date text formats supported by ODV.

**Enumerator**

***IsoDate*** Date format according to ISO 8601.

Example: `2006-02-23` for Feb/23/2006

***mmddyyyyDate*** Date in one pass without separators.

Example: `02232006` for Feb/23/2006

***mmmddyyyyDate*** Date with abbreviated english month name in front.

Example: `Feb 23 2006` for Feb/23/2006

***ddmonthyyyyDate*** Date with full english month name.

Example: `23 February 2006` for Feb/23/2006.

***ddmmmyyyyDate*** Date with abbreviated english month name.

Example: `23 Feb 2006` for Feb/23/2006

**enum ODV::Status**

Status values returned by many functions.

**Enumerator**

***NoErr*** OK

***UserAbort*** Aborted by user

***EOD*** End of data reached

***CollReadOnly*** Collection not writable

***UnknownFileTypeErr*** Unknown file type

***NoSuchDirErr*** Directory does not exist

***DirCreateErr*** Could not create directory

***FileOpenErr*** Could not open file

***FileReadErr*** Error on read in file

***FileWriteErr*** Error on write to file

***FileErr*** A general error on file operation occured

***CollOpenErr*** Could not open collection

***CollCreateErr*** Could not create collection

***CollDelErr*** Could not delete collection

***CollCopyErr*** Could not copy collection

***CollFormatUnsupported*** The collection format is not supported

***CollReadErr*** Error on read in collection

***CollWriteErr*** Error on write to collection

***CollNotSorted*** Could not sort & condense collection

***ImportErr*** Error on import

***ImportProblems*** Import was done but there were problems. Probably not all data could be imported.

***AnalyzeFileErr*** Error on analyzing file

***MacroErr*** Syntax error in macro

***VarNotFound*** A variable could not be identified
***PSPreambleNotFound*** Postscript preamble file not found.
***StatIDOutOfRange*** ID of requested station is out of range.
***InvalidStationData*** Data of station is invalid or unreasonable.
***OutOfMemory*** Machine is out of memory.
***BadParameter*** A bad parameter value was supplied.
***NotImplemented*** The requested functionality is not implemented.

---

## Member Data Documentation

**const double ODV::largeDOUBLE[static]**

Large positive value for double data type.

**const float ODV::largeFLOAT[static]**

Large positive value for float data type.

**const qint16 ODV::largeINT16[static]**

Large positive value for qint16 data type.

**const qint32 ODV::largeINT32[static]**

Large positive value for qint32 data type.

**const qint8 ODV::largeINT8[static]**

Large value constants for different data types. Large positive value for qint8 data type.

**const quint16 ODV::largeUINT16[static]**

Large positive value for quint16 data type.

**const quint32 ODV::largeUINT32[static]**

Large positive value for quint32 data type.

**const quint8 ODV::largeUINT8[static]**

Large positive value for quint8 data type.

**const double ODV::missDOUBLE[static]**

Missing value for double data type.

**const float ODV::missFLOAT[static]**

Missing value for float data type.

**const qint16 ODV::missINT16[static]**

Missing value for qint16 data type.

**const qint32 ODV::missINT32[static]**

Missing value for qint32 data type.

**const qint8 ODV::missINT8[static]**

Missing value constants for different data types. Missing value for qint8 data type.

**const quint16 ODV::missUINT16[static]**

Missing value for quint16 data type.

**const quint32 ODV::missUINT32[static]**

Missing value for quint32 data type.

**const quint8 ODV::missUINT8[static]**

Missing value for quint8 data type.

# ODVCollection Class Reference

Class for handling `ODVCF6` (*.odv), `ODVCF5` (*.odv) and `GENERIC` (*.var) data collections.
`#include "collection/odvcollection.h"`

## Public Types

- enum DataField { GeneralField =0, Ocean, Atmosphere, Land, IceSheet, SeaIce, Sediment }
- enum DataType { GeneralType =0, Profiles, Trajectories, TimeSeries }
- enum StateFlag { ColUndefined =0, ColDefined =1, ColDeleteWhenClosed =2, ColEmptyNetCDF =16, ColVarsRead =32, ColOpen =64 }

## Public Member Functions

- **Q_DECLARE_FLAGS** (State, StateFlag)
- ODVCollection (const QString &collectionName, const QString &userDir=QString())
  *Creates a new ODVCollection object.*
- ODV::AccessMode accessMode ()
- quint32 appendHistoryString (quint32 accession, const QString &string)
  *Appends the string string as a new history record of the station with accession number accession .*
- int appendHistoryStrings (quint32 accession, const QStringList &stringList)
  *Appends all strings in stringList as new history records of the station with accession number accession .*
- int appendMetaVar (ODVVariable *var)
  *Adds meta variable var to the list of meta variables.*
- virtual int appendVar (ODVVariable *var, int varID=ODV::missINT32)
  *Adds basic variable var to the list of collection variables.*
- QString baseDir () const
- int basicVarCount () const
  *Returns the number of basic variables in this collection.*
- bool changePassword (const QString &newPassWord, const QString &oldPassWord=QString())
  *Changes the password of the collection from oldPassWord to newPassWord .*
- void close ()
  *Closes the collection and deletes all temporary files.*
- ODVCollectionInventory * collectionInventory ()
- ODV::AccessMode currentAccessMode ()
- QList< ODVVariable * > extendedMetaVars ()
- QString extension () const
- QString filePath () const
- void generalProperties (DataField &dField, DataType &dType) const
- QStringList historyStrings (quint32 accession) const
  *Retrieves all history strings for the station with accession number accession .*
- int instanceID () const
- QString inventoryFilePath () const
- bool isOpen () const
- bool isPasswordProtected ()
- QDateTime lastModified () const
- ODV::Status loadCollectionFile ()
  *Loads the .odv or .var collection file and creates the meta and data variables.*
- ODVVariable * metaVar (int varID) const
- ODVVariable * metaVar (ODVVariable::VarType varType) const

*Returns pointer to meta variable with type varType , or `NULL` if no such variable is found.*

- int metaVarCount () const
- int metaVarID (ODVVariable::VarType varType) const
- QList< ODVVariable * > metaVarPtrList () const
- QString name () const
- ODV::Status open (ODV::AccessMode requestedAccessMode, const QString &password=QString())
  *Opens the collection in access mode requestedAccessMode using password password .*
- QString pathName () const
- ODVVariable * primaryVar () const
- int primaryVarID () const
- QString rootDir () const
- QString settingsFilePath () const
- qint64 sizeOfDataFile () const
- int stationCount () const
- State state () const
- int totalVarCount () const
- int userCount ()
  *Returns the number of applications currently using this collection.*
- ODVVariable * var (int varID) const
- ODVVariable * var (ODVVariable::VarType varType) const
- int varID (ODVVariable::VarType varType) const
- QList< int > varIDList (bool includeBasicVars=true, bool includeDerivedVars=true) const
- QBitArray varsWithErrorValues (const QVector< int > &stationIDs)
- QBitArray varsWithInfoStrings (const QVector< int > &stationIDs)
- QList< ODVVariable * > varPtrList (bool includeBasicVars=true, bool includeDerivedVars=true) const

## Static Public Member Functions

- static DataField dataFieldID (const QString &fieldName)
- static DataType dataTypeID (const QString &typeName)
- static bool isInUse (const QString &filePath)

## Protected Member Functions

- const PCollection * **dFunc** () const
- PCollection * **dFunc** ()
- void setCollection (PCollection *pColl)
  *This routine is not for public use.*
- bool transferCollection (const QString &fileName)
  *This routine is not for public use.*

---

## Detailed Description

Class for handling `ODVCF6` (*.odv), `ODVCF5` (*.odv) and `GENERIC` (*.var) data collections.

ODVCollection provides a wide range of functions for retrieving and setting collection properties.

Variables: Meta variables and basic collection variables have 0-based variable IDs (*varID* ).

Variables can be added but not deleted. Presently this class does not allow to add variables permanently.

Reading of data from an ODV collection is performed via ODVStation objects.

---

## Member Enumeration Documentation

### enum ODVCollection::DataField

This property describes the field to which the collection data belongs.

**Enumerator**

***GeneralField*** General data field (default)

***Ocean*** Oceanographic data

***Atmosphere*** Atmospheric data

***Land*** Data is related to land

***IceSheet*** Data on ice sheets

***SeaIce*** Sea ice data

***Sediment*** Sediment data

### enum ODVCollection::DataType

This property describes the type of data in the collection.

**Enumerator**

***GeneralType*** General data type (default)

***Profiles*** Profile data

***Trajectories*** Data collected from a moving platform reapeated over time

***TimeSeries*** Data at a fixed location reapeated over time

### enum ODVCollection::StateFlag

This flag describes the current state of the collection

**Enumerator**

***ColUndefined*** Collection object is not initialized, format unknown.

***ColDefined*** Collection associated with collection file name, format is set, basic initialization is done but no files are checked or read. This state is set if the collection object was constructed successfully.

***ColDeleteWhenClosed*** This flag is currently not used.

***ColEmptyNetCDF*** This flag applies to NetCDF collections only.

***ColVarsRead*** Collection file was read successfully but no data files are opened yet. This flag is set when loadCollectionFile() was called successfully.

***ColOpen*** Collection has been opened successfully (via open()) and is ready to work with. If this flag is set, ODVCollection::ColVarsRead and ODVCollection::ColDefined are also set.

---

## Constructor & Destructor Documentation

### ODVCollection::ODVCollection (const QString & *collectionName*, const QString & *userDir* = `QString()`)

Creates a new ODVCollection object.

The full path name to the collection file must be supplied in *collectionName* . The format of the collection is inferred from the *collectionName* extension and must be one of ".odv" or ".var".

The collection specified by *collectionName* must exist.

The *userDir* parameter is for internal usage only.

## Member Function Documentation

**ODV::AccessMode ODVCollection::accessMode ()**

**Returns:**
The access mode for the collection granted during the `open()` call.

**See also:**
currentAccessMode()

**quint32 ODVCollection::appendHistoryString (quint32 *accession*, const QString & *string*)**

Appends the string *string* as a new history record of the station with accession number *accession* .

**Returns:**
The string index for *string* in the history string pool, or `ODV::missUINT32` if the append failed.

**int ODVCollection::appendHistoryStrings (quint32 *accession*, const QStringList & *stringList*)**

Appends all strings in *stringList* as new history records of the station with accession number *accession* .

**Returns:**
The number of history records appended.

**int ODVCollection::appendMetaVar (ODVVariable * *var*)**

Adds meta variable *var* to the list of meta variables.

**Returns:**
The ID of the new meta variable.
The variable ID of meta variables is equal to their 0-based index.

**int ODVCollection::appendVar (ODVVariable * *var*, int *varID* = `ODV::missINT32`)`[virtual]`**

Adds basic variable *var* to the list of collection variables.

**Returns:**
The ID of the new variable.
Assigns variable ID *varID* , or the next available ID, if *varID*= `ODV::missINT32` . The var ID of basic collection variables must be equal to their 0-based index.

**QString ODVCollection::baseDir () const**

**Returns:**
The name of the collection's base directory, i.e. the directory that contains the collection's metadata and data files.

**int ODVCollection::basicVarCount () const**

Returns the number of basic variables in this collection.

Basic variables are those data variables which together with their values are really saved in the collection files, i.e. they do not include derived variables.

**bool ODVCollection::changePassword (const QString & *newPassWord*, const QString & *oldPassWord* = `QString()`)**

Changes the password of the collection from *oldPassWord* to *newPassWord* .

**Note:**

By default newly created ODV collections have no password. To establish password protection for a previously unprotected collection simply use `changePassword(newPassWord)` . Also note that password protection is only available for `ODVCF6` collections.

**Returns:**

`true` if successful or `false` if the password was not changed, for instance, because *oldPassWord* was wrong or the collection is not `ODVCF6` .

**void ODVCollection::close ()**

Closes the collection and deletes all temporary files.

**ODVCollectionInventory * ODVCollection::collectionInventory ()**

**Returns:**

The pointer to the collection inventory.

**ODV::AccessMode ODVCollection::currentAccessMode ()**

**Returns:**

The current access mode for the collection.

**Note:**

The current access mode may be less than the access mode granted during the open() call. For instance, `ODV::ReadWrite` access is downgraded to `ODV::ReadOnly` if more than one application are using this collection at the same time.

**See also:**

accessMode()

**ODVCollection::DataField ODVCollection::dataFieldID (const QString & *fieldName*)`[static]`**

**Returns:**

The ODVCollection::DataField ID for data field name *fieldName* , or ODVCollection::GeneralField if *fieldName* is invalid.

**ODVCollection::DataType ODVCollection::dataTypeID (const QString & *typeName*)`[static]`**

**Returns:**

The ODVCollection::DataType ID for data type name *typeName* , or ODVCollection::GeneralType if *typeName* is invalid.

**QList< ODVVariable * > ODVCollection::extendedMetaVars ()**

**Returns:**

A list of pointers to the collection's extended meta variables.

Extended meta variables are those with types outside the [METACRUISE,METAPRIMVARMAX] range.

**QString ODVCollection::extension () const**

**Returns:**

The extension of the collection file including the leading dot.

**QString ODVCollection::filePath () const**

**Returns:**

The collection's full file path including extension.

**void ODVCollection::generalProperties (ODVCollection::DataField & *dField*, ODVCollection::DataType & *dType*) const**

**Returns:**

The general properties of this collection in *dField* and *dType* .

**QStringList ODVCollection::historyStrings (quint32 *accession*) const**

Retrieves all history strings for the station with accession number *accession* .

**int ODVCollection::instanceID () const**

**Returns:**

The unique instance ID of the collection session.

**QString ODVCollection::inventoryFilePath () const**

**Returns:**

The path to the inventory file.

**bool ODVCollection::isInUse (const QString & *filePath*)`[static]`**

**Returns:**

`true` if the collection *filePath* is in use or `false` otherwise.

*filePath* must be an absolute file path to the collection (including extension).

This function always returns `false` if *filePath* is not a `ODVCF5` or `ODVCF6` collection.

**bool ODVCollection::isOpen () const**

**Returns:**
`true` if the collection is open and `false` otherwise.

**bool ODVCollection::isPasswordProtected ()**

**Returns:**
`true` if the collection is password protected or `false` otherwise.

**QDateTime ODVCollection::lastModified () const**

**Returns:**
The date/time of collection's last change.

**ODV::Status ODVCollection::loadCollectionFile ()**

Loads the .odv or .var collection file and creates the meta and data variables.

**Note:**
This function does not open the collection; no data can be accessed yet.

**Returns:**

- ODV::NoErr if successful
- ODV::CollFormatUnsupported if file format is unsupported
- ODV::FileOpenErr if collection file not found or incomplete
- ODV::CollReadErr if parameters in collection file were not found

**See also:**
open()

**ODVVariable * ODVCollection::metaVar (int *varID*) const**

**Returns:**
Pointer to meta variable with variable ID *varID* , or `NULL` , if *varID* is not found.

**ODVVariable * ODVCollection::metaVar (ODVVariable::VarType *varType*) const**

Returns pointer to meta variable with type *varType* , or `NULL` if no such variable is found.

**int ODVCollection::metaVarCount () const**

**Returns:**
The number of meta variables.

**int ODVCollection::metaVarID (ODVVariable::VarType *varType*) const**

**Returns:**

The meta variable ID of meta variable with VarType *varType* , or `ODV::missINT32` , if no such meta variable is found.

### QList< ODVVariable * > ODVCollection::metaVarPtrList () const

**Returns:**

The list of pointers to meta variables.

### QString ODVCollection::name () const

**Returns:**

The collection name.

**See also:**

filePath()

### ODV::Status ODVCollection::open (ODV::AccessMode *requestedAccessMode*, const QString & *password* = `QString()`)

Opens the collection in access mode *requestedAccessMode* using password *password* .

Checks the supported access modes and automatically switches to `ODV::ReadOnly` if `ODV::ReadWrite` access is requested but not supported. Leave *password* empty if the collection is not password protected.

**Returns:**

- `ODV::NoErr` if collection was successfully opened,
- `ODV::CollReadOnly` if collection was successfully opened but is read only,
- `ODV::CollOpenErr` on failure.

**See also:**

close(), isOpen(), currentAccessMode(), isPasswordProtected()

### QString ODVCollection::pathName () const

**Returns:**

The full pathname of the collection excluding the extension.

**See also:**

filePath(), name()

### ODVVariable * ODVCollection::primaryVar () const

**Returns:**

A pointer to the primary variable of the collection.

### int ODVCollection::primaryVarID () const

**Returns:**

The variable ID of the collection's primary variable.

**QString ODVCollection::rootDir () const**

**Returns:**
The pathname of the collection's root directory (i.e., the directory containing the .odv or .var collection file.

**void ODVCollection::setCollection (PCollection * *pColl*)[protected]**

This routine is not for public use.

**QString ODVCollection::settingsFilePath () const**

**Returns:**
The full pathname of the collection's settings file.

**qint64 ODVCollection::sizeOfDataFile () const**

**Returns:**
The size in bytes of the collection's data file, or `0` , if no data file exists.

**ODVCollection::State ODVCollection::state () const**

**Returns:**
Collection state.
This is an Qt QFlags object which constitutes an "OR" combination of ODVCollection::StateFlag.

**int ODVCollection::stationCount () const**

**Returns:**
The number of stations in this collection.

**int ODVCollection::totalVarCount () const**

**Returns:**
The total number of data variables.
This is the number of basic variables but excludes the meta variables.

**bool ODVCollection::transferCollection (const QString & *fileName*)[protected]**

This routine is not for public use.

**int ODVCollection::userCount ()**

Returns the number of applications currently using this collection.

**ODVVariable * ODVCollection::var (int *varID*) const**

**Returns:**
A pointer to the data variable with variable ID *varID* or `NULL` , if *varID* is not found.

**ODVVariable * ODVCollection::var (ODVVariable::VarType *varType*) const**

**Returns:**

A pointer to the first data variable with variable type *varType* , or `NULL` if no such variable is found.

**int ODVCollection::varID (ODVVariable::VarType *varType*) const**

**Returns:**

The variable ID of the first data variable with type *varType* , or `VID_NONE` if no such variable is found.

**QList< int > ODVCollection::varIDList (bool *includeBasicVars* = `true`, bool *includeDerivedVars* = `true`) const**

**Returns:**

The list of data variable IDs.

Use the default (`true` ) for the parameters. Derived variables are not supported by the API yet.

**QList< ODVVariable * > ODVCollection::varPtrList (bool *includeBasicVars* = `true`, bool *includeDerivedVars* = `true`) const**

**Returns:**

A list of pointers to the data variables.

Please use the default values (`true` , `true` ) for the input parameters *includeBasicVars* *includeDerivedVars* . Note that in the API derived variables are presently not supported.

**QBitArray ODVCollection::varsWithErrorValues (const QVector< int > & *stationIDs*)**

**Returns:**

A `QBitArray` of length *nBasicVars* indicating for every basic data variable whether at least one of the stations in *stationIDs* contains data error values.

**QBitArray ODVCollection::varsWithInfoStrings (const QVector< int > & *stationIDs*)**

**Returns:**

A `QBitArray` of length *nBasicVars* indicating for every basic data variable whether at least one of the stations in *stationIDs* contains data info strings.

# ODVCollectionInventory Class Reference

The collection inventory holds information for all cruises of a collection as well as cruise IDs, position, date/time, sample count and data availability for all stations of the collection.

`#include "collection/odvcollectioninventory.h"`

## Public Member Functions

- ODVCollectionInventory (ODVCollection *col)
  *Creates an new inventory object for collection col .*
- virtual ~ODVCollectionInventory ()
  *Deletes the inventory object.*
- const quint32 * accessionNumberData () const
- quint32 accessionNumber (qint32 statID) const
  *Returns the accession number of station statID .*
- const QBitArray * availabilityData () const
- int cruiseCount () const
- qint32 cruiseID (qint32 statID) const
- const qint32 * cruiseIdData () const
- bool cruiseIDs (qint32 *crIDs, int nStats, int *statIDs) const
  *Returns at crIDs the cruise IDs of the nStats stations in statIDs .*
- const ODVCruiseInfo * cruiseInfo (const QString &cruiseName) const
- QStringList cruiseNames () const
- qint32 dataCount (qint32 varID) const
- const qint16 * dayTimeData () const
  *A pointer to the day-time data (in units of 0.1 min since mid-night) of all stations in the collection.*
- double decimalYear (qint32 statID) const
- bool decimalYears (double *decYears, int nStats, int *statIDs) const
- const qint32 * gregorianDayData () const
- qint32 load ()
  *Loads the collection inventory of the parent collection.*
- double latitude (qint32 statID) const
- bool latitudes (double *lats, int nStats, int *statIDs) const
  *Returns at lats the decimal latitudes of the nStats stations in statIDs .*
- double longitude (qint32 statID) const
- bool longitudes (double *lons, int nStats, int *statIDs) const
  *Returns at lons the longitudes of the nStats stations in statIDs .*
- const ODVMapDomain * nativeMapDomain () const
- qint32 sampleCount (qint32 statID=-1) const
- int sampleCount (int nStats, int *statIDs) const
- const qint32 * sampleCountData () const
  *Returns a pointer to the sample count data of all stations in the collection.*
- qint32 stationIDFromAccessionNumber (quint32 accNum) const
- const ODVCruiseInfo * summaryCruiseInfo () const

## Protected Member Functions

- const PCollectionInventory * **dFunc** () const
- PCollectionInventory * **dFunc** ()

## Detailed Description

The collection inventory holds information for all cruises of a collection as well as cruise IDs, position, date/time, sample count and data availability for all stations of the collection.

The collection inventory provides fast access to various metadata of the stations in the collection without requiring reading individual station metadata from the collection files.

Detailed per-cruise information is available via the ODVCruiseInfo object returned by cruiseInfo(). A summary of all cruises is provided by summaryCruiseInfo().

Each cruise is assigned a unique ID in the range [0 <= ID < nCruise]. Assignment is in the order of appearance of the cruises in the collection and does not imply lexical ordering.

---

## Constructor & Destructor Documentation

### ODVCollectionInventory::ODVCollectionInventory (ODVCollection * *col*)

Creates an new inventory object for collection *col* .

### ODVCollectionInventory::~ODVCollectionInventory ()[virtual]

Deletes the inventory object.

---

## Member Function Documentation

### quint32 ODVCollectionInventory::accessionNumber (qint32 *statID*) const

Returns the accession number of station *statID* .

Every station in a collection has a unique and un-modifiable accession number that is assigned when a station is added to the collection (normally during import) and preserved during Sort and Condense operations. This is in contrast to station IDs, which change when the order of stations in the collection changes.

**Note:**

Collection formats `ODVGENERIC` and `ODVCF5` do not maintain accession numbers. ODV::missUINT32 is returned in these cases.

### const quint32 * ODVCollectionInventory::accessionNumberData () const

**Returns:**

The array of accession numbers for all station IDs in the collection.

Every station in a collection has a unique and un-modifiable accession number that is assigned when a station is added to the collection (normally during import) and preserved during Sort and Condense operations. This is in contrast to station IDs, which change when the order of stations in the collection changes.

### const QBitArray * ODVCollectionInventory::availabilityData () const

**Returns:**

A pointer to a bit array indicating whether a given station contains data for a particular basic data variable.

Station *statID* (0-based station ID) contains data for data variable *varID* (0-based data variable ID) if the bit at index *statID*nBV+varID* is set (*nBV* is the number of basic data variables).

**int ODVCollectionInventory::cruiseCount () const**

**Returns:**
The number of cruises in this ODVCollectionInventory.

**qint32 ODVCollectionInventory::cruiseID (qint32 *statID*) const**

**Returns:**
Cruise ID of station with station ID *statID* , or -1 if *statID* is out of range.

**const qint32 * ODVCollectionInventory::cruiseIdData () const**

**Returns:**
A pointer to the cruise IDs of all stations in the collection.
Index values range from `0` to `ODVCollection::stationCount()-1` .

**bool ODVCollectionInventory::cruiseIDs (qint32 * *crIDs*, int *nStats*, int * *statIDs*) const**

Returns at *crIDs* the cruise IDs of the *nStats* stations in *statIDs* .

**Note:**
The memory at *crIDs* must be allocated outside this function and must be sufficient to hold *nStats* `qint32` values for all the entries in *statIDs* .

**Returns:**
`true` if successful, or `false` otherwise.

**const ODVCruiseInfo * ODVCollectionInventory::cruiseInfo (const QString & *cruiseName*) const**

**Returns:**
A pointer to the ODVCruiseInfo object of cruise *cruiseName* , or `NULL` , if no such cruise exists.

**QStringList ODVCollectionInventory::cruiseNames () const**

**Returns:**
A list of all cruise names in the collection.

**qint32 ODVCollectionInventory::dataCount (qint32 *varID*) const**

**Returns:**
The total number of non-ODV::missDOUBLE samples for variable ID *varID* of all cruises in this ODVCollectionInventory.

**const qint16 * ODVCollectionInventory::dayTimeData () const**

A pointer to the day-time data (in units of 0.1 min since mid-night) of all stations in the collection.

Example: A value of 5121 represents 08:32:06 am.

A value of `ODV::missINT16` indicates that day-time is not available or incomplete.

Hour, minute and second day-time values can be obtained from a `dayTimeData()` value *dt* as: `daytimeFromFractionalDay` (dt/14400.,hour,min,sec).

**double ODVCollectionInventory::decimalYear (qint32 *statID*) const**

**Returns:**

The date/time of station *statID* as decimal years.

Stations without date and time information return ODV::missDOUBLE. If daytime is missing or incomplete `00:00h` is assumed.

The retrieved decimal year can be converted to date & time by dateFromDecimalYear() and getDayOfYear().

**bool ODVCollectionInventory::decimalYears (double * *decYears*, int *nStats*, int * *statIDs*) const**

**Returns:**

At *decYears* the date/time of the *nStats* stations in *statIDs* as decimal years.

Stations without date and time information return ODV::missDOUBLE. If daytime is missing or incomplete `00:00h` is assumed.

**Note:**

The memory at *decYears* must be allocated outside this function and must be sufficient to hold *nStats* `double` values for all the entries in *statIDs* .

**Returns:**

`true` if successful, or `false` otherwise.

The retrieved decimal years can be converted to date & time by methods like dateFromDecimalYear() and getDayOfYear().

**const qint32 * ODVCollectionInventory::gregorianDayData () const**

**Returns:**

A pointer to the date data (in Gregorian days) of all stations in the collection.

A value of `ODV::missINT32` indicates that date information is not available or incomplete.

The retrieved gregorian days can be converted to decimal years by decimalYearFromGregorianDay() and then be converted to date & time by other methods in odvdate.h. An ISO date string can be created directly with isoDateFromGregorianDay().

**double ODVCollectionInventory::latitude (qint32 *statID*) const**

**Returns:**

The decimal latitude of station *statID* .

**bool ODVCollectionInventory::latitudes (double * *lats*, int *nStats*, int * *statIDs*) const**

Returns at *lats* the decimal latitudes of the *nStats* stations in *statIDs* .

**Note:**

The memory at *lats* must be allocated outside this function and must be sufficient to hold *nStats* `double` values for all the entries in *statIDs* .

**Returns:**

`true` if successful, or `false` otherwise.

### qint32 ODVCollectionInventory::load ()

Loads the collection inventory of the parent collection.

**Note:**

The inventory is automatically loaded when the collection is opened so there is no need to call this method explicitly.

### double ODVCollectionInventory::longitude (qint32 *statID*) const

**Returns:**

The decimal longitude of station *statID* .

### bool ODVCollectionInventory::longitudes (double * *lons*, int *nStats*, int * *statIDs*) const

Returns at *lons* the longitudes of the *nStats* stations in *statIDs* .

**Note:**

The memory at *lons* must be allocated outside this function and must be sufficient to hold *nStats* `double` values for all the entries in *statIDs* .

**Returns:**

`true` if successful, or `false` otherwise.

### const ODVMapDomain * ODVCollectionInventory::nativeMapDomain () const

**Returns:**

A pointer to the native map domain covered by this collection.

### qint32 ODVCollectionInventory::sampleCount (qint32 *statID* = -1) const

**Returns:**

The number of samples of station *statID* , or the total number of samples of all cruises in this ODVCollectionInventory if *statID* is -1 on entry (the default).

### int ODVCollectionInventory::sampleCount (int *nStats*, int * *statIDs*) const

**Returns:**

The total number of samples of the *nStats* stations *statIDs* .

### const qint32 * ODVCollectionInventory::sampleCountData () const

Returns a pointer to the sample count data of all stations in the collection.

The value at index *statID* (0-based station ID) is the number of samples of station *statID* .

**qint32 ODVCollectionInventory::stationIDFromAccessionNumber (quint32 *accNum*) const**

**Returns:**

The 0-based station ID of the station with accession number *accNum* , or `-1` if no such station exists or the cruise inventory does not have accession number data.

**const ODVCruiseInfo * ODVCollectionInventory::summaryCruiseInfo () const**

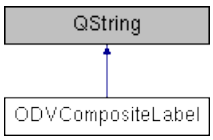
**Returns:**

The pointer to the summary ODVCruiseInfo containing information about all cruises in the collection together.

# ODVCompositeLabel Class Reference

`#include "collection/odvcompositelabel.h"`

Inheritance diagram for ODVCompositeLabel:




## Public Member Functions

- **ODVCompositeLabel** (const QChar *unicode, int size)
- **ODVCompositeLabel** (const QChar *unicode)
- **ODVCompositeLabel** (QChar ch)
- **ODVCompositeLabel** (int size, QChar ch)
- **ODVCompositeLabel** (const QLatin1String &str)
- **ODVCompositeLabel** (const QString &other)
- **ODVCompositeLabel** (const char *str)
- **ODVCompositeLabel** (const QByteArray &ba)
- ODVCompositeLabel (const QString &nameLabel, const QString &unitsLabel)
  *This constructor builds a full variable label using the supplied nameLabel and unitsLabel . Sets this string with the built label.*
- ODVCompositeLabel & compositeLabel (const ODVVariable *var, int primaryVarID=-1, bool withQualifiers=true)
  *Sets this composite label to a fully qualified composite label for the supplied variable var and returns a reference to it.*
- ODVCompositeLabel & fullVariableLabel (const QString &nameLabel, const QString &unitsLabel)
  *Builds a full variable label using the supplied nameLabel and unitsLabel . Sets this string with the built label and returns a reference to it.*
- QString nameLabel (bool allowSubstitution=true) const
  *Returns the name part from compositeLabel .*
- ODVCompositeLabel & qfCompositeLabel (const ODVQualityFlagSet &qfSet)
  *Sets this composite label to the quality flag label of the supplied quality flag set qfSet and returns a reference to it.*
- ODVQualityFlagSet::QFSetID qfSetID () const
- QString qualifierLabel () const
- QString unitLabel () const
- bool valueProperties (ODVVariable::ValueType &vt, int &nBytes, int &nDigits)
  *Extracts ValueType valtyp , byte length nBytes and number of significant digits nDigits from this composite label.*

---

## Detailed Description

ODVCompositeLabel is a QString representing a variable label consisting of the variable's name, units, quality flag and possibly more qualifiers specifying the type of the data values and whether the variable is a meta or a primary data variable.

Member functions exist to compose, extract or manipulate these properties.

The syntax of variable labels is explained in detail in Appendix 16.3 *"Generic ODV Spreadsheet Format"* paragraph 16.3.2 *"Column Labels"* of *"ODV User's Guide"* .

All constructors work similar to the corresponding QString constructors.

## Constructor & Destructor Documentation

### ODVCompositeLabel::ODVCompositeLabel (const QString & *nameLabel*, const QString & *unitsLabel*)

This constructor builds a full variable label using the supplied *nameLabel* and *unitsLabel* . Sets this string with the built label.

**See also:**

fullVariableLabel()

---

## Member Function Documentation

### ODVCompositeLabel & ODVCompositeLabel::compositeLabel (const ODVVariable * *var*, int *primaryVarID* = `-1`, bool *withQualifiers* = `true`)

Sets this composite label to a fully qualified composite label for the supplied variable *var* and returns a reference to it.

Appropriate qualifiers are appended to the label if it is a variable of known type, the data type or the data byte length differ from the default values and *withQualifiers* is `true` .

The `":PRIMARYVAR"` suffix is appended to the label if the *primaryVarID* parameter is provided and the variable ID of *var* matches *primaryVarID* .

**See also:**

nameLabel(), unitLabel(), qfSetID(), valueProperties(), qfCompositeLabel()

### ODVCompositeLabel & ODVCompositeLabel::fullVariableLabel (const QString & *nameLabel*, const QString & *unitsLabel*)

Builds a full variable label using the supplied *nameLabel* and *unitsLabel* . Sets this string with the built label and returns a reference to it.

**See also:**

ODVVariable::fullLabel()

### QString ODVCompositeLabel::nameLabel (bool *allowSubstitution* = `true`) const

Returns the name part from *compositeLabel* .

Any units within square or curved brackets are excluded (last occurrence).

If *allowSubstitution* is `true` (the default) and *compositeLabel* starts with a known label the respective substitute is returned.

**Note:**

Any ';' is replaced by a ',' (ODV variable labels may not contain ';').

**See also:**

unitLabel()

**ODVCompositeLabel & ODVCompositeLabel::qfCompositeLabel (const ODVQualityFlagSet & *qfSet*)**

Sets this composite label to the quality flag label of the supplied quality flag set *qfSet* and returns a reference to it.

**See also:**

qfSetID(), compositeLabel()

**ODVQualityFlagSet::QFSetID ODVCompositeLabel::qfSetID () const**

**Returns:**

The Quality flag schema ID from this composite label or `ODVQualityFlagSet::ODV` , if no valid schema name is found.

**See also:**

qfCompositeLabel()

**QString ODVCompositeLabel::qualifierLabel () const**

**Returns:**

The qualifier part from this composite label.

**See also:**

nameLabel(), unitLabel()

**QString ODVCompositeLabel::unitLabel () const**

**Returns:**

The units part from this composite label or an empty string, if no square or curved brackets are found.

The units in this composite label are assumed within square or curved brackets (last occurrence).

**Note:**

Any ';' is replaced by a ',' (ODV variable labels may not contain ';').

**See also:**

nameLabel()

**bool ODVCompositeLabel::valueProperties (ODVVariable::ValueType & *valtyp*, int & *nBytes*, int & *nDigits*)**

Extracts ValueType *valtyp* , byte length *nBytes* and number of significant digits *nDigits* from this composite label.

**Returns:**

`true` if successful or `false` if no :

string or recognized variable label is found.

# ODVCruiseInfo Class Reference

Holds summary information for one cruise of a collection.

`#include "collection/odvcruiseinfo.h"`

## Public Member Functions

- virtual QString availabilityString () const
- int cruiseID () const
- int dataCount (int varID) const
- QString dateString (bool endDate, ODV::DateForm dateForm=ODV::ddmmmyyyyDate) const
- QString latitudeRangeString () const
- QString longitudeRangeString () const
- int maxGregorianDay () const
- int maxStationID () const
- int minGregorianDay () const
- int minStationID () const
- const ODVMapDomain * nativeMapDomain () const
- int sampleCount () const
- int stationCount () const
- int variableCount () const

## Static Public Attributes

- static const QString missString =QString("unavailable")
  *String for unavailable values.*

## Protected Member Functions

- ODVCruiseInfo (ODVCollectionInventory *colInv)
  *Creates an empty ODVCruiseInfo object.*
- ODVCruiseInfo (ODVCollectionInventory *colInv, ODVStation *stat, int cruiseID)
  *Creates an ODVCruiseInfo object with ID cruiseID using the station information in stat .*
- ODVCruiseInfo (ODVCollectionInventory *colInv, QDataStream &in)
  *Creates an ODVCruiseInfo object and reads its data from in .*
- const PCruiseInfo * **dFunc** () const
- PCruiseInfo * **dFunc** ()

---

## Detailed Description

Holds summary information for one cruise of a collection.

Each cruise is assigned a unique ID in the range [0 <= ID < nCruise]. Assignment is in the order of appearance of the cruises in the collection and does not imply lexical ordering.

Note that the cruise name is not part of the ODVCruiseInfo class.

---

## Constructor & Destructor Documentation

### ODVCruiseInfo::ODVCruiseInfo (ODVCollectionInventory * *colInv*)[protected]

Creates an empty ODVCruiseInfo object.

**Note:**

The constructor is protected because cruise info objects are only created by ODVCollectionInventory.

**ODVCruiseInfo::ODVCruiseInfo (ODVCollectionInventory * *collInv*, ODVStation * *stat*, int *cruiseID*)[protected]**

Creates an ODVCruiseInfo object with ID *cruiseID* using the station information in *stat* .

**Note:**

The constructor is protected because cruise info objects are only created by ODVCollectionInventory.

**ODVCruiseInfo::ODVCruiseInfo (ODVCollectionInventory * *collInv*, QDataStream & *in*)[protected]**

Creates an ODVCruiseInfo object and reads its data from *in* .

**Note:**

The constructor is protected because cruise info objects are only created by ODVCollectionInventory.

---

## Member Function Documentation

**QString ODVCruiseInfo::availabilityString () const[virtual]**

**Returns:**

The data availability indicators of all basic variables for this ODVCruiseInfo as string.

Data availability indicators are single digit numbers from 0 to 9, with, for instance, 9 indicating that more than 90 % of the samples containing data for the particular variable. There is a '.' separator every five variables and a '|' separator every ten variables. A '-' indicates that there are no data for the given variable in this cruise.

**int ODVCruiseInfo::cruiseID () const**

**Returns:**

The ID of the cruise kept by this object (0-based index in inventory).

**int ODVCruiseInfo::dataCount (int *varID*) const**

**Returns:**

The number of non-ODV::missDOUBLE samples for basic variable ID *varID* in this cruise.

The possible varIDs range from 0 to variableCount() - 1.

**QString ODVCruiseInfo::dateString (bool *endDate*, ODV::DateForm *dateForm* = ODV::ddmmmyyyyDate) const**

**Returns:**

The start (*endDate* = `false` ) or end (*endDate* = `true` ) date of the cruise as string. If the date should be unavailable, missString is returned.

The string format can be chosen via *dateForm* , the default is as in `"30 Sep 2007"` for example.

**QString ODVCruiseInfo::latitudeRangeString () const**

**Returns:**
The latitudinal range of the cruise.

**QString ODVCruiseInfo::longitudeRangeString () const**

**Returns:**
The longitudinal range of the cruise.

**int ODVCruiseInfo::maxGregorianDay () const**

**Returns:**
The maximal gregorian day of this cruise.

**See also:**
minGregorianDay()

**int ODVCruiseInfo::maxStationID () const**

**Returns:**
The maximal station ID (0-based index).

Note that between minimal and maximal station ID may be station IDs which are not part of the cruise.

**See also:**
minStationID()

**int ODVCruiseInfo::minGregorianDay () const**

**Returns:**
The minimal gregorian day of this cruise.

**See also:**
maxGregorianDay()

**int ODVCruiseInfo::minStationID () const**

**Returns:**
The minimal station ID (0-based index).

Note that between minimal and maximal station ID may be station IDs which are not part of the cruise.

**See also:**
maxStationID()

**const ODVMapDomain * ODVCruiseInfo::nativeMapDomain () const**

**Returns:**
A pointer to the native map domain of this cruise.

### int ODVCruiseInfo::sampleCount () const

**Returns:**
The total number of samples in this cruise.

### int ODVCruiseInfo::stationCount () const

**Returns:**
The number of stations in this cruise.

### int ODVCruiseInfo::variableCount () const

**Returns:**
The number of basic variables for which counts are kept.

**See also:**
dataCount()

---

## Member Data Documentation

### const QString ODVCruiseInfo::missString =QString("unavailable")[static]

String for unavailable values.

# ODVMapDomain Class Reference

The ODVMapDomain class holds bounding map domain information and provides functions to append individual lon/lat points or other ODVMapDomain objects.

```
#include "geography/odvmapdomain.h"
```

## Public Member Functions

- ODVMapDomain ()
  *Creates an empty ODVMapDomain object.*
- ODVMapDomain (double lonmin, double lonmax, double latmin, double latmax)
  *Creates an ODVMapDomain object with specified longitude/latitude bounds.*
- ODVMapDomain (const ODVMapDomain &otherDomain)
  *Creates a copy of ODVMapDomain otherDomain .*
- ~ODVMapDomain ()
  *Deletes the ODVMapDomain object.*
- void append (double lon, double lat)
  *Appends point lon / lat to the domain.*
- void append (int n, double *lon, double *lat)
  *Appends n points in lon / lat to the domain.*
- void append (int nLon, double *lon, int nLat, double *lat)
  *Appends a grid consisting of nLon points in lon and nLat points lat to the domain.*
- void append (double lonmin, double lonmax, double latmin, double latmax)
  *Appends the domain specified by the input parameters.*
- void append (const ODVMapDomain &otherDomain)
  *Appends ODVMapDomain otherDomain .*
- double centerLatitude () const
- double centerLongitude () const
- void clear ()
- bool domain (double &lonmin, double &lonmax, double &latmin, double &latmax, bool autoEnlarge=true) const
  *Retrieves the ODVMapDomain's domain.*
- void domainValues (double &xlVal, double &XlVal, double &xrVal, double &XrVal, double &ybVal, double &ytVal) const
  *Retrieves the internal domain values.*
- double eastLongitude () const
- bool intersects (const ODVMapDomain *otherDomain) const
- bool isEmpty () const
- bool isInsideOf (ODVMapDomain *otherDomain) const
- double latitudeRange () const
- QString latitudeRangeString (int nDigits=1) const
- double longitudeRange () const
- QString longitudeRangeString (int nDigits=1) const
- double northLatitude () const
- void setDomain (double lonmin, double lonmax, double latmin, double latmax)
  *Sets the domain to the specified longitude/latitude bounds.*
- void setDomainValues (double xlVal, double XlVal, double xrVal, double XrVal, double ybVal, double ytVal)
  *Sets the ODVMapDomain's internal domain parameters to the specified values.*
- double southLatitude () const
- double westLongitude () const

- ODVMapDomain & operator= (const ODVMapDomain &otherDomain)
  *Assigns otherDomain to this map domain and returns a reference to this ODVMapDomain.*

## Protected Member Functions

- const PMapDomain * **dFunc** () const
- PMapDomain * **dFunc** ()

---

## Detailed Description

The ODVMapDomain class holds bounding map domain information and provides functions to append individual lon/lat points or other ODVMapDomain objects.

---

## Constructor & Destructor Documentation

### ODVMapDomain::ODVMapDomain ()

Creates an empty ODVMapDomain object.

### ODVMapDomain::ODVMapDomain (double *lonmin*, double *lonmax*, double *latmin*, double *latmax*)

Creates an ODVMapDomain object with specified longitude/latitude bounds.

### ODVMapDomain::ODVMapDomain (const ODVMapDomain & *otherDomain*)

Creates a copy of ODVMapDomain *otherDomain* .

### ODVMapDomain::~ODVMapDomain ()

Deletes the ODVMapDomain object.

---

## Member Function Documentation

### void ODVMapDomain::append (double *lon*, double *lat*)

Appends point *lon* / *lat* to the domain.

The point is not used if either *lon* or *lat* is equal to `ODV::missDOUBLE` .

### void ODVMapDomain::append (int *n*, double * *lon*, double * *lat*)

Appends *n* points in *lon* / *lat* to the domain.

Points with either *lon* or *lat* equal to `ODV::missDOUBLE` are not used.

### void ODVMapDomain::append (int *nLon*, double * *lon*, int *nLat*, double * *lat*)

Appends a grid consisting of *nLon* points in *lon* and *nLat* points *lat* to the domain.

**void ODVMapDomain::append (double *lonmin*, double *lonmax*, double *latmin*, double *latmax*)**

Appends the domain specified by the input parameters.

**void ODVMapDomain::append (const ODVMapDomain & *otherDomain*)**

Appends ODVMapDomain *otherDomain* .

**double ODVMapDomain::centerLatitude () const**

**Returns:**
The latitude of the map domain center.

**double ODVMapDomain::centerLongitude () const**

**Returns:**
The longitude of the map domain center.

**void ODVMapDomain::clear ()**

Resets the ODVMapDomain object to an empty (invalid) state.

**bool ODVMapDomain::domain (double & *lonmin*, double & *lonmax*, double & *latmin*, double & *latmax*, bool *autoEnlarge* = `true`) const**

Retrieves the ODVMapDomain's domain.

The domain is automatically enlarged if *autoEnlarge* is `true` on entry. This ensures that none of the added *lon* / *lat* points will lie on the domain boundaries.

**Returns:**
`true` if successful and `false` if the object is empty.

**void ODVMapDomain::domainValues (double & *xlVal*, double & *XlVal*, double & *xrVal*, double & *XrVal*, double & *ybVal*, double & *ytVal*) const**

Retrieves the internal domain values.

**double ODVMapDomain::eastLongitude () const**

**Returns:**
The eastern border longitude of the map domain.

**bool ODVMapDomain::intersects (const ODVMapDomain * *otherDomain*) const**

**Returns:**
`true` if this ODVMapDomain intersects *otherDomain* , or `false` otherwise.

**bool ODVMapDomain::isEmpty () const**

**Returns:**
`true` if this ODVMapDomain is empty.

**bool ODVMapDomain::isInsideOf (ODVMapDomain * *otherDomain*) const**

**Returns:**
`true` if this ODVMapDomain is inside of *otherDomain* , or `false` otherwise.

**double ODVMapDomain::latitudeRange () const**

**Returns:**
The latitudinal range of the map domain.

**QString ODVMapDomain::latitudeRangeString (int *nDigits* = 1) const**

**Returns:**
The latitudinal range as a string.
The latitude values are rounded to *nDigits* significant digits before producing the text representation.

**double ODVMapDomain::longitudeRange () const**

**Returns:**
The longitudinal range of the map domain.

**QString ODVMapDomain::longitudeRangeString (int *nDigits* = 1) const**

**Returns:**
The longitudinal range as a string.
The longitude values are rounded to *nDigits* significant digits before producing the text representation.

**double ODVMapDomain::northLatitude () const**

**Returns:**
The northern border latitude of the map domain.

**ODVMapDomain & ODVMapDomain::operator= (const ODVMapDomain & *otherDomain*)**

Assigns *otherDomain* to this map domain and returns a reference to this ODVMapDomain.

**void ODVMapDomain::setDomain (double *lonmin*, double *lonmax*, double *latmin*, double *latmax*)**

Sets the domain to the specified longitude/latitude bounds.

**void ODVMapDomain::setDomainValues (double *xlVal*, double *XlVal*, double *xrVal*, double *XrVal*, double *ybVal*, double *ytVal*)**

Sets the ODVMapDomain's internal domain parameters to the specified values.

**double ODVMapDomain::southLatitude () const**

**Returns:**

The southern border latitude of the map domain.

### double ODVMapDomain::westLongitude () const

**Returns:**

The western border longitude of the map domain.

# ODVQualityFlagSet Class Reference

Represents an ODV Quality Flag Schema. See appendix of *"ODV User's Guide"* , section *"Quality Flag Schemes"* for further details on various schemes.

```
#include "collection/odvqualityflagset.h"
```

## Public Types

- enum QFSetID { ODV =0, GTSPP =1, ARGO =2, SEADATANET =3, ESEAS =4, WOD =5, WODSTATION =6, WOCEBOTTLE =7, WOCECTD =8, WOCESAMPLE =9, QARTOD =10, BODC =11, PANGAEA =12, SMHI =13, OCEANSITES =14, IODE =15 }

## Public Member Functions

- ODVQualityFlagSet (QFSetID setID=ODV)
  *Creates a ODVQualityFlagSet of type setID .*
- ODVQualityFlagSet (const ODVQualityFlagSet &other)
  *Creates a ODVQualityFlagSet from other .*
- char badQfVal () const
- char defaultQfVal (bool isDrvd=false) const
- QStringList descriptions () const
- char genericQfVal (char qfVal) const
- char goodQfVal () const
- int indexOf (char qfVal) const
- char mapTo (char qfVal, QFSetID targetSetID) const
  *Maps the quality flag value qfVal of this set to the corresponding quality flag value of the target set targetSetID .*
- char mapTo (char qfVal, const ODVQualityFlagSet &targetSet) const
  *Maps the quality flag value qfVal of this set to the corresponding quality flag value of the target set targetSet .*
- char missQfVal () const
- QString qfString (int i) const
- QString qfString (char qfVal) const
- char qfVal (int i) const
- int safeIndexOf (char qfVal) const
- QFSetID setID () const
- int size () const
- const QString & text () const
- QStringList values () const
- ODVQualityFlagSet & operator= (QFSetID setID)
  *Assignment operator: Sets ODVQualityFlagSet to type setID .*
- ODVQualityFlagSet & operator= (const ODVQualityFlagSet &other)
  *Assignment operator: Sets ODVQualityFlagSet to type of other .*
- bool operator== (QFSetID setID) const
- bool operator== (const ODVQualityFlagSet &other) const

## Protected Member Functions

- const PQualityFlagSet * **dFunc** () const
- PQualityFlagSet * **dFunc** ()

## Detailed Description

Represents an ODV Quality Flag Schema. See appendix of *"ODV User's Guide"* , section *"Quality Flag Schemes"* for further details on various schemes.

ODVQualityFlagSet contains a general text description of the quality flag set and a list of quality flags. The default constructor creates the default ODVQualityFlagSet::ODV quality flag set.

Quality flags can be mapped between different schemes using the mapTo() functions.

---

## Member Enumeration Documentation

**enum ODVQualityFlagSet::QFSetID**

List of available quality flag set identifiers for quality flag sets.

**Enumerator**

***ODV*** Ocean Data View quality flags (default)

***GTSPP*** GTSPP quality flags.

***ARGO*** ARGO quality flags.

***SEADATANET*** SEADATANET quality flags.

***ESEAS*** ESEAS quality flags.

***WOD*** World Ocean Database observed level quality codes.

***WODSTATION*** World Ocean Database entire station quality flags.

***WOCEBOTTLE*** WOCE bottle data quality flags.

***WOCECTD*** WOCE CTD data quality flags.

***WOCESAMPLE*** WOCE quality flags for the water bottle itself.

***QARTOD*** QARTOD quality flags.

***BODC*** British Oceanographic Data Centre quality flags.

***PANGAEA*** PANGAEA quality flags.

***SMHI*** Swedish Meteorological and Hydrographic Institute quality codes.

***OCEANSITES*** OceanSITES quality flags.

***IODE*** IODE quality flags.

---

## Constructor & Destructor Documentation

**ODVQualityFlagSet::ODVQualityFlagSet (QFSetID *setID* = ODV)**

Creates a ODVQualityFlagSet of type *setID* .

**ODVQualityFlagSet::ODVQualityFlagSet (const ODVQualityFlagSet & *other*)**

Creates a ODVQualityFlagSet from *other* .

---

## Member Function Documentation

**char ODVQualityFlagSet::badQfVal () const**

**Returns:**
The bad quality flag of this set.

**char ODVQualityFlagSet::defaultQfVal (bool *isDrvd* = `false`) const**

**Returns:**
The default quality flag for derived variables if *isDrvd* is `true` or the default quality flag for basic variables if *isDrvd* is `false` of this set.

**QStringList ODVQualityFlagSet::descriptions () const**

**Returns:**
The list of flag descriptions.

**char ODVQualityFlagSet::genericQfVal (char *qfVal*) const**

**Returns:**
The corresponding ODV generic quality flag value for quality flag value *qfVal* .

**char ODVQualityFlagSet::goodQfVal () const**

**Returns:**
The good quality flag of this set.

**int ODVQualityFlagSet::indexOf (char *qfVal*) const**

**Returns:**
0-based index in the set for quality flag value *qfVal* , or `-1` if *qfVal* is not in set.

**char ODVQualityFlagSet::mapTo (char *qfVal*, QFSetID *targetSetID*) const**

Maps the quality flag value *qfVal* of this set to the corresponding quality flag value of the target set *targetSetID* . If *qfVal* is an invalid member of this quality flag set, the default quality flag of this set is mapped.

**char ODVQualityFlagSet::mapTo (char *qfVal*, const ODVQualityFlagSet & *targetSet*) const**

Maps the quality flag value *qfVal* of this set to the corresponding quality flag value of the target set *targetSet* . If *qfVal* is an invalid member of this quality flag set, the default quality flag of this set is mapped.

**char ODVQualityFlagSet::missQfVal () const**

**Returns:**
The quality flag for missing values.

**ODVQualityFlagSet & ODVQualityFlagSet::operator= (QFSetID *setID*)**

Assignment operator: Sets ODVQualityFlagSet to type *setID* .

**ODVQualityFlagSet & ODVQualityFlagSet::operator= (const ODVQualityFlagSet & *other*)**

Assignment operator: Sets ODVQualityFlagSet to type of *other* .

**bool ODVQualityFlagSet::operator== (QFSetID *setID*) const**

**Returns:**

`true` if quality flags set represents the quality flag set *setID* .

**bool ODVQualityFlagSet::operator== (const ODVQualityFlagSet & *other*) const**

**Returns:**

`true` if quality flags sets are equal. They are considered equal if they have the same set ID, i.e. describe the same set.

**QString ODVQualityFlagSet::qfString (int *i*) const**

**Returns:**

The description for quality flag value at 0-based index *i* or the default quality flag description if *i* is invalid.

**QString ODVQualityFlagSet::qfString (char *qfVal*) const**

**Returns:**

The description for native quality flag value *qfVal* or the default quality flag description if *qfVal* is invalid.

**char ODVQualityFlagSet::qfVal (int *i*) const**

**Returns:**

The native quality flag value for 0-based index *i* in set or the default quality flag if *i* is invalid.

**int ODVQualityFlagSet::safeIndexOf (char *qfVal*) const**

**Returns:**

0-based index in set for quality flag value *qfVal* , or the index of the default quality flag if *qfVal* is invalid.

**ODVQualityFlagSet::QFSetID ODVQualityFlagSet::setID () const**

**Returns:**

The quality flag set ID.

**int ODVQualityFlagSet::size () const**

**Returns:**

The number of flags in set.

**const QString & ODVQualityFlagSet::text () const**

**Returns:**

The title resp. description of the quality flag set.

### QStringList ODVQualityFlagSet::values () const

**Returns:**

The list of flag values.

# ODVSampleFilter Class Reference

The ODVSampleFilter object maintains quality, range (numeric variables) and wildcard (text variables) sample filter specifications for data variables.

```
#include "collection/odvsamplefilter.h"
```

## Public Member Functions

- ODVSampleFilter ()
  *Creates a default sample filter object.*
- ODVSampleFilter (const ODVSampleFilter &other)
  *Copy constructor for sample filter.*
- ~ODVSampleFilter ()
  *Deletes the sample filter object.*
- void clear ()
  *Clears this filter by removing all filter specifications.*
- int clearQualityFilter (int varID)
  *Clears any existing quality filter for data variable varID .*
- int clearRangeFilter (int varID)
  *Clears any existing range filter for data variable varID .*
- int clearWildCardFilter (int varID)
  *Clears any existing wildcard filter for data variable varID .*
- bool isQualityFiltering (int varID=-1) const
  *Checks whether a quality filter is defined for variable with 0-based variable ID varID , or, if varID is `-1` (the default) whether there is any variable with a quality filter.*
- bool isRangeFiltering (int varID=-1) const
  *Checks whether a numeric range filter is defined for variable with 0-based variable ID varID , or, if varID is `-1` (the default) whether there is any variable with a numeric range filter.*
- bool isWildCardFiltering (int varID=-1) const
  *Checks whether a wildcard text filter is defined for variable with 0-based variable ID varID , or, if varID is `-1` (the default) whether there is any variable with a wildcard text filter.*
- QBitArray qualityFilter (int varID) const
  *Returns the quality filter for data variable varID , or a QBitArray of length 128 with all bits set to 1, if no quality filter exists for variable varID .*
- QList< int > qualityFilterVarIDs () const
  *Returns a list of variable IDs of those data variables with active quality filters.*
- QPair< double, double > rangeFilter (int varID) const
  *Returns the range filter for data variable varID , or a `ODV::missDOUBLE` , `ODV::largeDOUBLE` range, if no range filter exists for variable varID .*
- int rangeFilterCount () const
  *Returns the number of data variables with active range filters.*
- QList< int > rangeFilterVarIDs () const
  *Returns a list of variable IDs of those data variables with active range filters.*
- bool setQualityFilter (int varID, const QBitArray &ba)
  *Sets the quality filter for data variable varID to ba .*
- bool setRangeFilter (int varID, double minVal, double maxVal)
  *Sets the range filter for data variable varID to range between minVal and maxVal .*
- bool setWildCardFilter (int varID, const QStringList &wcf)

*Sets the wildcard filter for data variable varID to wcf .*

- QString statusString () const
  *Returns the ODV status-bar string of this sample filter.*
- QStringList wildCardFilter (int varID) const
  *Returns the wildcard filter for data variable varID , or an empty QStringList, if no wildcard filter exists for variable varID .*
- QList< int > wildCardFilterVarIDs () const
  *Returns a list of variable IDs of those data variables with active wildcard filters.*
- ODVSampleFilter & operator= (const ODVSampleFilter &other)
  *Assigns other to this sample filter and returns a reference to this sample filter.*
- bool operator== (const ODVSampleFilter &other) const
  *Compares this object with other .*

## Static Public Member Functions

- static bool matchesWildCardFilter (const char *text, const QStringList &wcf)
  *Checks whether text text matches any of the wildcard text filters in wcf .*

## Protected Member Functions

- const PSampleFilter * **dFunc** () const
- PSampleFilter * **dFunc** ()

---

## Detailed Description

The ODVSampleFilter object maintains quality, range (numeric variables) and wildcard (text variables) sample filter specifications for data variables.

The quality, range and wildcard filter specifications are held in three QHash containers *qfFil* , *rangeFil* , and *wildCardFil* . All these containers use the variable ID of the variable for which a filter is specified as key.

The type of values in the QHash containers depends on the filter type:

1. quality filter

The value is a QBitArray of size 128. A quality flag value *qf* is acceptable if testBit(qf) is true. Note that ODV treats quality flag values as char, e.g., flag '1' is represented as ASCII code value 49. '1' is an acceptable quality flag if testBit(49) is true. As a consequence of limiting the size of the QBitArray to 128 ODV only supports quality flag sets strictly limited to the ASCII character set (0-127).

2. range filter

The value is a QPair of double values representing the acceptable lower and upper bounds as *first* and *second* .

3. wildcard filter

The value is a QStringList containing the wildcard specifications to be statisfied by a text data value.

Variables that do not have an entry in the *qfFil* container are not quality filtered. Variables that do not have an entry in the *rangeFil* or *wildCardFil* container are not used for sample filtering.

A default ODVSampleFilter with empty containers accepts all data.

---

## Constructor & Destructor Documentation

### ODVSampleFilter::ODVSampleFilter ()

Creates a default sample filter object.

### ODVSampleFilter::ODVSampleFilter (const ODVSampleFilter & *other*)

Copy constructor for sample filter.

### ODVSampleFilter::~ODVSampleFilter ()

Deletes the sample filter object.

---

## Member Function Documentation

### void ODVSampleFilter::clear ()

Clears this filter by removing all filter specifications.

### int ODVSampleFilter::clearQualityFilter (int *varID*)

Clears any existing quality filter for data variable *varID* .

**Returns:**

`1` if a quality filter for data variable *varID* existed before this call, or `0` otherwise.

### int ODVSampleFilter::clearRangeFilter (int *varID*)

Clears any existing range filter for data variable *varID* .

**Returns:**

`1` if a range filter for data variable *varID* existed before this call, or `0` otherwise.

### int ODVSampleFilter::clearWildCardFilter (int *varID*)

Clears any existing wildcard filter for data variable *varID* .

**Returns:**

`1` if a wildcard filter for data variable *varID* existed before this call, or `0` otherwise.

### bool ODVSampleFilter::isQualityFiltering (int *varID* = `-1`) const

Checks whether a quality filter is defined for variable with 0-based variable ID *varID* , or, if *varID* is `-1` (the default) whether there is any variable with a quality filter.

**Returns:**

`true` if a suitable range filter is found, or, `false` otherwise.

### bool ODVSampleFilter::isRangeFiltering (int *varID* = `-1`) const

Checks whether a numeric range filter is defined for variable with 0-based variable ID *varID* , or, if *varID* is `-1` (the default) whether there is any variable with a numeric range filter.

**Returns:**

`true` if a suitable range filter is found, or, `false` otherwise.

### bool ODVSampleFilter::isWildCardFiltering (int *varID* = `-1`) const

Checks whether a wildcard text filter is defined for variable with 0-based variable ID *varID* , or, if *varID* is `-1` (the default) whether there is any variable with a wildcard text filter.

**Returns:**

`true` if a suitable range filter is found, or, `false` otherwise.

### bool ODVSampleFilter::matchesWildCardFilter (const char * *text*, const QStringList & *wcf*)`[static]`

Checks whether text *text* matches any of the wildcard text filters in *wcf* .

**Returns:**

`true` if a match is found, or, `false` otherwise.

### ODVSampleFilter & ODVSampleFilter::operator= (const ODVSampleFilter & *other*)

Assigns *other* to this sample filter and returns a reference to this sample filter.

### bool ODVSampleFilter::operator== (const ODVSampleFilter & *other*) const

Compares this object with *other* .

**Returns:**

`true` if the two objects are identical, or `false` if they differ.

### QBitArray ODVSampleFilter::qualityFilter (int *varID*) const

Returns the quality filter for data variable *varID* , or a QBitArray of length 128 with all bits set to 1, if no quality filter exists for variable *varID* .

### QList< int > ODVSampleFilter::qualityFilterVarIDs () const

Returns a list of variable IDs of those data variables with active quality filters.

### QPair< double, double > ODVSampleFilter::rangeFilter (int *varID*) const

Returns the range filter for data variable *varID* , or a `ODV::missDOUBLE` , `ODV::largeDOUBLE` range, if no range filter exists for variable *varID* .

### int ODVSampleFilter::rangeFilterCount () const

Returns the number of data variables with active range filters.

### QList< int > ODVSampleFilter::rangeFilterVarIDs () const

Returns a list of variable IDs of those data variables with active range filters.

### bool ODVSampleFilter::setQualityFilter (int *varID*, const QBitArray & *ba*)

Sets the quality filter for data variable *varID* to *ba* .

**Note:**

*ba* must be of length 128. Acceptable flag values must have their bit values set to 1, unacceptable flag values have their bit set to 0.

**Returns:**

`true` if the quality filter for data variable *varID* changed, or, `false` otherwise.

### bool ODVSampleFilter::setRangeFilter (int *varID*, double *minVal*, double *maxVal*)

Sets the range filter for data variable *varID* to range between *minVal* and *maxVal* .

**Returns:**

`true` if the range filter for data variable *varID* changed, or, `false` otherwise.

### bool ODVSampleFilter::setWildCardFilter (int *varID*, const QStringList & *wcf*)

Sets the wildcard filter for data variable *varID* to *wcf* .

**Returns:**

`true` if the wildcard filter for data variable *varID* changed, or, `false` otherwise.

### QString ODVSampleFilter::statusString () const

Returns the ODV status-bar string of this sample filter.

### QStringList ODVSampleFilter::wildCardFilter (int *varID*) const

Returns the wildcard filter for data variable *varID* , or an empty QStringList, if no wildcard filter exists for variable *varID* .

### QList< int > ODVSampleFilter::wildCardFilterVarIDs () const

Returns a list of variable IDs of those data variables with active wildcard filters.

# ODVStation Class Reference

Class for maintaining metadata and data of one station.
`#include "collection/odvstation.h"`

## Public Types

- enum MetaVarIndex { **MetaCruiseIndex** =0, **MetaStationIndex** =1, **MetaTypeIndex** =2, **MetaLongitudeIndex** =3, **MetaLatitudeIndex** =4, **MetaYearIndex** =5, **MetaMonthIndex** =6, **MetaDayIndex** =7, **MetaHourIndex** =8, **MetaMinuteIndex** =9, **MetaSecondIndex** =10, **MetaAccessionIndex** =11 }

## Public Member Functions

- ODVStation (ODVCollection *col, int valueCountHint=10000)
  *Creates a new ODVStation object for collection col .*
- quint32 accessionNumber () const
  *Retrieves the accession number of the station.*
- void applySampleFilter (int varID, double *doubleData)
  *Applies the sample quality, range and wildcard filters of this station for variable varID to the data at doubleData .*
- void clear ()
  *Clears the current station data (if any) and prepares the station's metadata and data arrays to receive a new station.*
- bool containsDataErrors (int varID) const
- bool containsDataInfos (int varID) const
- virtual double * data (ODVVariable *var, bool filtered=false)
  *Returns a pointer to the double data values of variable var , or `NULL` if var is NULL, meta, derived or non-numeric, or there are no samples.*
- int dataCount (int varID) const
- quint32 dataTotalByteSize () const
- double * errorData (ODVVariable *var)
  *Returns a pointer to the double data error values of variable var , or `NULL` if var is NULL, meta, derived or non-numeric, or there are no error values.*
- double errorValue (ODVVariable *var, int sampleID)
- QString errorStringValue (ODVVariable *var, int sampleID, int decimalCount=-1)
- QStringList historyStrings () const
  *Retrieves all history strings for this station.*
- QString identifierHeaderString (bool withID=false, bool withBrackets=false)
- QString identifierString (bool withID=false, bool withBrackets=false)
- QString infoStringValue (ODVVariable *var, int sampleID)
- QStringList infoStringValues (ODVVariable *var)
- double metaDecimalDay (char *qFlag=NULL) const
- QString metaFullName ()
- double metaLatitude (char *qFlag=NULL) const
- double metaLongitude (char *qFlag=NULL) const
- QString metaName ()
- QString metaStringDate (ODV::DateForm dateForm=ODV::mmmddyyyyDate, char *qFlagYear=NULL, char *qFlagMonth=NULL, char *qFlagDay=NULL) const
- QString metaStringIsoDateTime () const
- QString metaStringPosition (int decimalCount=-1, char *qFlagLon=NULL, char *qFlagLat=NULL) const
- QString metaStringPrimVarRange (char *qFlagMin=NULL, char *qFlagMax=NULL)
- QString metaStringStatType (char *qFlag=NULL)

- QString metaStringTime (char *qFlagHour=NULL, char *qFlagMinute=NULL, char *qFlagSecond=NULL) const
- QString metaStringValue (MetaVarIndex mvIndex, char *qFlag=NULL, int decimalCount=-1) const
- QString metaStringValue (ODVVariable::VarType metaVarType, char *qFlag=NULL, int decimalCount=-1)
- QString metaStringValue (ODVVariable *var, char *qFlag=NULL, int decimalCount=-1)
- double metaValue (ODVVariable *var, char *qFlag=NULL)
- double metaValue (ODVVariable::VarType metaVarType, char *qFlag=NULL)
- double metaValue (MetaVarIndex mvIndex, char *qFlag=NULL) const
- char * qfData (ODVVariable *var)
- ODV::Status readData (int statID)
  *Reads metadata and data of station with station ID statID (0-based index) from the collection files.*
- ODV::Status readMetaData (int statID)
  *Reads the metadata of station with station ID statID (0-based index) from the collection files.*
- int sampleCount () const
- ODVSampleFilter * sampleFilter ()
- int stationID () const
- int stationLabelToInt (int dfltVal=ODV::missINT32)
  *Tries to read an integer station number from the first 10 characters of the Station label.*
- QString stringValue (ODVVariable *var, int sampleID, char *qFlag=NULL, int decimalCount=-1, bool filtered=false)
- const char * textData (ODVVariable *var)
- const char * textValue (ODVVariable *var, int sampleID=0, char *qFlag=NULL, bool qFiltered=false)
  *Returns a pointer to the text value of variable var for sample sampleID (0-based index) of this station or `NULL`, if the variable is not of value type `ODVVariable::TEXT` or sampleID is out of range.*
- double value (ODVVariable *var, int sampleID, char *qFlag=NULL, bool filtered=false)

## Static Public Member Functions

- static const char * stationTypeFromSampleCount (const int sampleCount)

## Protected Member Functions

- const PStation * **dFunc** () const
- PStation * **dFunc** ()

---

## Detailed Description

Class for maintaining metadata and data of one station.

The ODVStation object maintains a ODVSampleFilter sample filter that can be used to filter the data using numerical value ranges, text wildcard expressions or sets of acceptable or unacceptable quality flag values.

Reading the metadata and/or data of one station is done using functions readMetaData() and readData(). The metadata of the station can be accessed using a large number of meta<...>() functions. The (filtered) data of a collection variable are accessed via value() and stringValue().

---

## Member Enumeration Documentation

### enum ODVStation::MetaVarIndex

Fixed IDs of mandatory meta variables allowing fast access to values.

The values should be self-explanatory, for further details see *"ODV User's Guide"* , section 3 *"ODV Collections"* , paragraph *"Meta-variables"* .

## Constructor & Destructor Documentation

### ODVStation::ODVStation (ODVCollection * *col*, int *valueCountHint* = 10000)

Creates a new ODVStation object for collection *col* .

Also allocates and prepares sufficient memory to handle *valueCountHint* data values. The size of the ODVStation memory grows automatically if a station requires more storage. The default value for *valueCountHint* (10000) should be sufficient for most cases. You can provide your own *valueCountHint* estimate (typical number of samples times number of data variables) to save memory or minimize the number of automatic re-allocations.

## Member Function Documentation

### quint32 ODVStation::accessionNumber () const

Retrieves the accession number of the station.

**Returns:**

The accession number of the station, or `ODV::missUINT32` if the station does not have one.

### void ODVStation::applySampleFilter (int *varID*, double * *doubleData*)

Applies the sample quality, range and wildcard filters of this station for variable *varID* to the data at *doubleData* .

### void ODVStation::clear ()

Clears the current station data (if any) and prepares the station's metadata and data arrays to receive a new station.

### bool ODVStation::containsDataErrors (int *varID*) const

**Returns:**

`true` if this station contains data error values for data variable *varID* , or `false` otherwise.

### bool ODVStation::containsDataInfos (int *varID*) const

**Returns:**

`true` if this station contains data info values for data variable *varID* , or `false` otherwise.

### double * ODVStation::data (ODVVariable * *var*, bool *filtered* = false)[virtual]

Returns a pointer to the double data values of variable *var* , or `NULL` if *var* is NULL, meta, derived or non-numeric, or there are no samples.

The data are filtered using the station's sample filter if `filtered` is `true` . Values not passing the filter are set to ODV::missDOUBLE.

**int ODVStation::dataCount (int *varID*) const**

**Returns:**
The number of non-miss values of variable with ID *varID* as recorded in the station metadata.

**quint32 ODVStation::dataTotalByteSize () const**

**Returns:**
The total number of bytes in the file data record, including quality flags.

**double * ODVStation::errorData (ODVVariable * *var*)**

Returns a pointer to the double data error values of variable *var* , or `NULL` if *var* is NULL, meta, derived or non-numeric, or there are no error values.

**QString ODVStation::errorStringValue (ODVVariable * *var*, int *sampleID*, int *decimalCount* = `-1`)**

**Returns:**
The text representation of the error value of variable *var* for sample *sampleID* (0-based index) of this station, or an empty string, if the error value is missing.

The error value is rounded to *decimalCount* significant digits before producing the text representation. The variable's digit count value (see ODVVariable::decimalCount()) is used, if *decimalCount* is less than zero on entry.

**double ODVStation::errorValue (ODVVariable * *var*, int *sampleID*)**

**Returns:**
The error value of variable *var* for sample *sampleID* (0-based index) of this station.

`ODV::missDOUBLE` is returned if no error value exists for this sample or the sample ID is out of range.

**Warning:**
Always returns `ODV::missDOUBLE` for meta variables.

**QStringList ODVStation::historyStrings () const**

Retrieves all history strings for this station.

**Returns:**
The list of strings.

**QString ODVStation::identifierHeaderString (bool *withID* = `false`, bool *withBrackets* = `false`)**

**Returns:**
The description of the station identifier string contents.

**See also:**
metaFullName(), identifierString()

**QString ODVStation::identifierString (bool *withID* = `false`, bool *withBrackets* = `false`)**

**Returns:**

A string that best identifies the station.

The string is the metaFullName(), or, if meta variables ODVVariable::METALOCALCDIID and ODVVariable::METAEDMOCODE exist, the concatenation of local CDI ID and EDMO code.

The identifier is prepended with a suitable station number if *withID* is `true` and enclosed in brackets if *withBrackets* is `true`.

**Note:**

ODVCF6 collections use the unique accession number for station number and [] as brackets. Other collections use the sequential station ID for station number and <> as brackets.

Example ODVCF6 identifier with ID and brackets:

```
[67: A20_316N151_3 55 (B)]
```

**See also:**

identifierHeaderString(), metaFullName()

**QString ODVStation::infoStringValue (ODVVariable * *var*, int *sampleID*)**

**Returns:**

The data info string value of variable *var* for sample *sampleID* (0-based index) of this station, or an empty string, if the data info string is missing.

**QStringList ODVStation::infoStringValues (ODVVariable * *var*)**

**Returns:**

The data info string values of variable *var* of this station, or an empty string list, if the variable does not contain data info strings.

**double ODVStation::metaDecimalDay (char * *qFlag* = `NULL`) const**

**Returns:**

The metadata date/time of the station in decimal Gregorian Days.

If *qFlag* is non-zero returns `'1'` in *qFlag*.

**QString ODVStation::metaFullName ()**

**Returns:**

The full name of the station consisting of cruise, station label and station type.

**See also:**

metaName()

**double ODVStation::metaLatitude (char * *qFlag* = `NULL`) const**

**Returns:**

The latitude of the station.

If *qFlag* is non-zero, also returns the associated quality flag in *qFlag*.

**double ODVStation::metaLongitude (char * *qFlag* = `NULL`) const**

**Returns:**

The longitude of the station.

If *qFlag* is non-zero, also returns the associated quality flag in *qFlag* .

**QString ODVStation::metaName ()**

**Returns:**

The name of the station consisting of the station label and station type (in parantheses).

**See also:**

metaFullName()

**QString ODVStation::metaStringDate (ODV::DateForm *dateForm* = `ODV::mmmddyyyyDate`, char * *qFlagYear* = `NULL`, char * *qFlagMonth* = `NULL`, char * *qFlagDay* = `NULL`) const**

**Returns:**

The metadata date in *dateForm* format.

If any one of the pointers *qFlagYear* , *qFlagMonth* , or *qFlagDay* is non-NULL on entry also returns the respective quality flags.

**QString ODVStation::metaStringIsoDateTime () const**

**Returns:**

The metadata date and time in ISO date format.

Example: `2006-02-23T10:23:06` for Feb/23/2006 10:23:06.

**QString ODVStation::metaStringPosition (int *decimalCount* = `-1`, char * *qFlagLon* = `NULL`, char * *qFlagLat* = `NULL`) const**

**Returns:**

The metadata *lon/lat* position as formatted string "*lon* / *lat*" .

*lon* and *lat* are rounded to *decimalCount* digits (if *decimalCount* is supplied and greater than or equal to zero), or to the decimal count property of the longitude/latitude meta variables otherwise.

If *qFlagLon* or *qFlagLat* are non-NULL on entry also returns the respective quality flag.

**QString ODVStation::metaStringPrimVarRange (char * *qFlagMin* = `NULL`, char * *qFlagMax* = `NULL`)**

**Returns:**

The range of observed primary variable values as formatted string "[ *min* , *max* ]".

If *qFlagMin* or *qFlagMax* are non-zero on entry also returns the respective quality flag.

**QString ODVStation::metaStringStatType (char * *qFlag* = `NULL`)**

**Returns:**

The station type as string.

If *qFlag* is non-zero on entry also returns the associated quality flag.

**QString ODVStation::metaStringTime (char * *qFlagHour* = NULL, char * *qFlagMinute* = NULL, char * *qFlagSecond* = NULL) const**

**Returns:**

The metadata time as "hh:mm:ss.sss".

If there is no value for minutes only the hours are returned, if there is no value for seconds only hours and minutes are returned.

If any one of the pointers *qFlagHour* , *qFlagMinute* , or *qFlagSecond* is non-null on entry also returns the respective quality flag.

**QString ODVStation::metaStringValue (MetaVarIndex *mvIndex*, char * *qFlag* = NULL, int *decimalCount* = -1) const**

**Returns:**

The text representation of meta variable with index *mvIndex* for this station, or an empty string if the value is missing.

Using a MetaVarIndex parameter to specify the meta variable is the most efficent way of accessing the meta variable values. This method is only available for mandatory meta variables.

If *qFlag* is non-zero, also returns the associated quality flag in *qFlag* .

If the meta variable is numeric it is rounded to *decimalCount* significant digits before producing the text representation (*decimalCount* > -1 ). If *decimalCount* is -1 on entry, the variable's ODVVariable::decimalCount() value is used. If *decimalCount* is -2 on entry, the number of significant digits is derived from variable's value type.

**QString ODVStation::metaStringValue (ODVVariable::VarType *metaVarType*, char * *qFlag* = NULL, int *decimalCount* = -1)**

**Returns:**

The text representation of meta variable *metaVarType* for this station, or an empty string if the value is missing.

If *qFlag* is non-zero, also returns the associated quality flag in *qFlag* .

If the meta variable is numeric it is rounded to *decimalCount* significant digits before producing the text representation (*decimalCount* > -1 ). If *decimalCount* is -1 on entry, the variable's ODVVariable::decimalCount() value is used. If *decimalCount* is -2 on entry, the number of significant digits is derived from variable's value type.

**Warning:**

This function must not be called for *metaVarType* == ODVVariable::METABASIC or any derived meta variable, such as ODVVariable::METADAYOFYEAR or ODVVariable::METATIME.

**QString ODVStation::metaStringValue (ODVVariable * *var*, char * *qFlag* = NULL, int *decimalCount* = -1)**

**Returns:**

The text representation of meta variable *var* for this station, or an empty string if the value is missing.

If *qFlag* is non-null, also returns the associated quality flag in *qFlag* .

If the meta variable is numeric it is rounded to *decimalCount* significant digits before producing the text representation (*decimalCount* > -1 ). If *decimalCount* is -1 on entry, the variable's

ODVVariable::decimalCount() value is used. If *decimalCount* is -2 on entry, the number of significant digits is derived from variable's value type.

**double ODVStation::metaValue (ODVVariable * *var*, char * *qFlag* = NULL)**

**Returns:**

The numeric value of meta variable *var* for this station, or ODV::missDOUBLE if the variable does not exist, the value is missing or the variable is a text variable.

If *qFlag* is non-null, also returns the associated quality flag in *qFlag* .

**double ODVStation::metaValue (ODVVariable::VarType *metaVarType*, char * *qFlag* = NULL)**

**Returns:**

The numeric value of meta variable of type *metaVarType* for this station, or ODV::missDOUBLE if the variable does not exist, the value is missing or the variable is a text variable.

If *qFlag* is non-null, also returns the associated quality flag in *qFlag* .

**Warning:**

This function should not be called for *metaVarType* = ODVVariable::METABASIC.

**double ODVStation::metaValue (MetaVarIndex *mvIndex*, char * *qFlag* = NULL) const**

**Returns:**

The numeric value of meta variable with index *mvIndex* for this station, or ODV::missDOUBLE if the value is missing or the variable is a text variable.

Using a MetaVarIndex parameter to specify the meta variable is the most efficent way of accessing the meta variable values. This method is only available for mandatory meta variables.

If *qFlag* is non-null, also returns the associated quality flag in *qFlag* .

**char * ODVStation::qfData (ODVVariable * *var*)**

**Returns:**

A pointer to the quality flags of meta or data variable *var* , or NULL if no quality flags exist.

**Note:**

This function must not be called for derived meta variables, i.e. ODVVariable::METATIME and ODVVariable::METADAYOFYEAR.

**ODV::Status ODVStation::readData (int *statID*)**

Reads metadata and data of station with station ID *statID* (0-based index) from the collection files.

**This function must be called before any meta or data variable values or quality flags can be accessed.**

**Returns:**

- ODV::NoErr if data was successfully read,
- ODV::StatIDOutOfRange if *statID* is not in collection or
- another error status in case of failure.

**ODV::Status ODVStation::readMetaData (int *statID*)**

Reads the metadata of station with station ID *statID* (0-based index) from the collection files.

**Returns:**

- ODV::NoErr if meta data was successfully read,
- ODV::StatIDOutOfRange if *statID* is not in collection or
- another error status in case of failure.

**int ODVStation::sampleCount () const**

**Returns:**

Number of samples for this station.

**ODVSampleFilter * ODVStation::sampleFilter ()**

**Returns:**

A pointer to the station's sample filter.

**int ODVStation::stationID () const**

**Returns:**

The 0-based station ID of the currently loaded station. `-1` is returned if no station is currently loaded.

**Note:**

The station ID member variable is set when the metadata are read, thus a valid returned station ID implies that the respective metadata have been loaded. However, the station data may not have been loaded yet.

**See also:**

readMetaData(), readData()

**int ODVStation::stationLabelToInt (int *dfltVal* = `ODV::missINT32`)**

Tries to read an integer station number from the first 10 characters of the Station label.

**Returns:**

The retrieved station number, if successful, or *dfltVal* otherwise.

**const char * ODVStation::stationTypeFromSampleCount (const int *sampleCount*)`[static]`**

**Returns:**

The station type based on the station's *sampleCount* .

`"B"` is returned for sample counts of up to 250, while `"C"` is returned otherwise.

**QString ODVStation::stringValue (ODVVariable * *var*, int *sampleID*, char * *qFlag* = `NULL`, int *decimalCount* = `-1`, bool *filtered* = `false`)**

**Returns:**

The text representation of variable *var* value for sample *sampleID* (0-based index) of this station, or an empty string, if the value is missing.

An empty string is also returned if *filtered* is true and the sample does not pass the filter.

If *qFlag* is non-NULL, also returns in *qFlag* the associated quality flag, or the default quality flag, if no flags are available.

If the variable is numeric the value is rounded to *decimalCount* significant digits before producing the text representation. The variable's digit count value (see ODVVariable::decimalCount()) is used, if *decimalCount* is less than zero on entry.

**Warning:**

Always returns an empty string for numeric meta variables.

**const char * ODVStation::textData (ODVVariable * *var*)**

**Returns:**

A pointer to the text data of variable *var* , or `NULL` if the variable is not of value type `ODVVariable::TEXT` .

**const char * ODVStation::textValue (ODVVariable * *var*, int *sampleID* = `0`, char * *qFlag* = `NULL`, bool *qFiltered* = `false`)**

Returns a pointer to the text value of variable *var* for sample *sampleID* (0-based index) of this station or `NULL` , if the variable is not of value type `ODVVariable::TEXT` or *sampleID* is out of range.

If *qFlag* is non-zero, also returns in *qFlag* the associated quality flag, or the default quality flag, if no flags are available. Returns `NULL` if *qFiltered* is `true` and the value's quality flag is unacceptable.

**double ODVStation::value (ODVVariable * *var*, int *sampleID*, char * *qFlag* = `NULL`, bool *filtered* = `false`)**

**Returns:**

The filtered or unfiltered value of variable *var* for sample *sampleID* (0-based index) of this station.

`ODV::missDOUBLE` is returned if no value exists for this sample or the sample ID is out of range. `ODV::missDOUBLE` is also returned if *filtered* is true and the sample does not pass the filter.

If *qFlag* is non-zero, also returns in *qFlag* the associated quality flag, or the default (value exists) or miss (value does not exist) quality flag, if no flags are available.

**Warning:**

Always returns `ODV::missDOUBLE` for meta variables.

# ODVVariable Class Reference

Represents a collection variable.
`#include "collection/odvvariable.h"`

## Public Types

- enum ValueType { **FLOAT** =1, **DOUBLE** =2, **INTEGER** =5, **SHORT** =4, **SIGNED_BYTE** =8, **UNSIGNED_INTEGER** =7, **UNSIGNED_SHORT** =6, **BYTE** =3, **TEXT** =10, **INDEXED_TEXT** =12, **UNICODETEXT** =15, **UNKNOWN** =32768 }
- enum VarType { **BASIC** =-1, **DERIVED** =999, **METADRVD** =1000, **METATIME** =1001, **METADAYOFYEAR** =1002, **METABASIC** =2000, **METACRUISE**, **METASTATION**, **METATYPE**, **METALONGITUDE**, **METALATITUDE**, **METADAY**, **METAMINUTE**, **METAMONTH**, **METAHOUR**, **METAYEAR**, **METASECOND**, **METAACCESSIONNUMBER**, **METAPRIMVARMIN**, **METAPRIMVARMAX**, **METABOTDEPTH**, **METALOCALCDIID**, **METAEDMOCODE**, **METASENSORDEPTH**, **METADURATION**, **METAEXTRACRUISEINFO**, **METAORIGCRUISE**, **METAORIGSTATION**, **METAINVESTIGATOR**, **METAGTSPPDATATYPE**, **METACOMMENTSLINK**, **METACRUISEREPORTLINK**, **METAREFERENCE**, **METASDNINSTRUMENTINFO** }

## *The enumeration values represent the type of the variable.* Public Member Functions

- ODVVariable (QString varName="", QString varUnits="", VarType varType=BASIC, int decimals=2, ValueType valType=FLOAT, int valBytes=4, ODVQualityFlagSet::QFSetID qfSetID=ODVQualityFlagSet::ODV)
  *Creates and initializes an ODVVariable.*
- ODVVariable (QString varName, QString varUnits, int decimals=2, ODVQualityFlagSet::QFSetID qfSetID=ODVQualityFlagSet::ODV, VarType varType=BASIC, ValueType valType=FLOAT, int valBytes=4)
  *Creates and initializes an ODVVariable.*
- ODVVariable (const ODVVariable &var)
  *Copy constructor.*
- virtual ~ODVVariable ()
  *Deletes all resources used by the variable.*
- char badQfVal ()
- QString commentLabel () const
- bool compareFullName (const QString &otherName, const QString &otherUnit, Qt::CaseSensitivity cs=Qt::CaseInsensitive) const
  *Compares the variable's raw name and unit strings with otherName and otherUnit .*
- bool compareName (const QString &otherName, Qt::CaseSensitivity cs=Qt::CaseInsensitive) const
  *Compares variable's raw name string with otherName .*
- bool compareUnit (const QString &otherUnit, Qt::CaseSensitivity cs=Qt::CaseInsensitive) const
  *Compares variable's raw unit string with otherUnit .*
- virtual ODVVariable * createClone () const
- int decimalCount () const
- char defaultQfVal ()
- char defaultQfVal (double dVal)
- int errorVarID () const
- virtual QString fullLabel (bool doClean=false, bool prependID=false) const
  *Returns the full label of the variable, consisting of variable name and units.*
- char goodQfVal ()
- virtual bool isDerived () const

- virtual bool isLatitude () const
- virtual bool isLongitude () const
- bool isMandatoryMetaVar () const
- bool isMeta () const
- bool isNumeric () const
- double maxVal () const
  *Returns the current maximal value for this variable.*
- double minVal () const
  *Returns the current minimal value for this variable.*
- char missQfVal ()
- QString nameLabel (bool doClean=false) const
- int nativeDecimalCount () const
- char qfGenericValue (char qfVal) const
- const ODVQualityFlagSet & qfSet () const
- void range (double &min, double &max) const
- double relativeRange () const
- void setComment (const QString &cmt)
  *Sets the comment member of this variable to cmt .*
- void setDecimalCount (int decimals)
  *Sets the number of decimal places (digits following the decimal point) for this variable to decimals .*
- void setErrorVarID (int errID)
  *Sets the error variable ID of this variable to errID (0-based; -1: no error variable)*
- void setMaxVal (double max)
  *Sets the current maximal value for this variable to max .*
- void setMinVal (double min)
  *Sets the current minimal value for this variable to min .*
- void setName (const QString &nameLabel)
  *Sets the name label of the variable to nameLabel .*
- virtual void setProperties (const ODVVariable &var)
  *Applies the properties of var to this ODVVariable object.*
- void setQFSet (ODVQualityFlagSet::QFSetID qfSetID)
  *Sets the quality flag scheme of this variable to qfSetID .*
- void setRange (double min, double max)
  *Sets the current value range for this variable to [min , max ].*
- void setType (VarType type)
  *Sets the variable type of this variable to type .*
- void setUnits (const QString &unitLabel)
  *Sets the unit label of the variable to unitLabel .*
- void setValueType (ValueType vt, unsigned int byteLengths=21)
  *Sets the value type of the variable to vt .*
- void setVarID (int varID)
  *Sets the variable ID of this variable to varID .*
- QString stringValue (double d, int decimals)
- VarType type () const
- QString unitLabel (bool doClean=false) const
- void updateRange (double min, double max, bool doAutoScale=true)
  *Updates the current range of the variable values to [min , max ] range.*
- unsigned int valueByteSize () const
- ValueType valueType () const

- int varID () const

## Static Public Member Functions

- static unsigned int valueByteSize (ValueType valueType, unsigned int nBytes=21)

## Protected Member Functions

- const PVariable * **dFunc** () const
- PVariable * **dFunc** ()

---

## Detailed Description

Represents a collection variable.

There are meta and collection variables. Meta variable values are constant for a single station, collection variables have values for each sample of a station. The variable contains information about its name, unit, value type and size, the quality flag set associated with it. The variable object is used as parameter in ODVStation calls to retrieve the data values for this variable for a certain station.

---

## Member Enumeration Documentation

**enum ODVVariable::ValueType**

Variable value types

**enum ODVVariable::VarType**

The enumeration values represent the type of the variable.

All collection variables have the type BASIC.

The META... values should be self-explanatory, for further details see *"ODV User's Guide"* , section 3 *"ODV Collections"* , paragraph *"Meta-variables"* .

---

## Constructor & Destructor Documentation

**ODVVariable::ODVVariable (QString *varName* = "", QString *varUnits* = "", VarType *varType* = BASIC, int *decimals* = 2, ValueType *valType* = FLOAT, int *valBytes* = 4, ODVQualityFlagSet::QFSetID *qfSetID* = ODVQualityFlagSet::ODV)**

Creates and initializes an ODVVariable.

Value bytes are set according to value type *valType* , only variables with value type `ODVVariable::TEXT` use the *valBytes* argument.

**ODVVariable::ODVVariable (QString *varName*, QString *varUnits*, int *decimals* = 2, ODVQualityFlagSet::QFSetID *qfSetID* = ODVQualityFlagSet::ODV, VarType *varType* = BASIC, ValueType *valType* = FLOAT, int *valBytes* = 4)**

Creates and initializes an ODVVariable.

Value bytes are set according to value type *valType* , only variables with value type `ODVVariable::TEXT` use the *valBytes* argument.

### ODVVariable::ODVVariable (const ODVVariable & *var*)

Copy constructor.

### ODVVariable::~ODVVariable ()[virtual]

Deletes all resources used by the variable.

---

## Member Function Documentation

### char ODVVariable::badQfVal ()

**Returns:**
The bad quality flag value of this variable.

**See also:**
defaultQfVal(), goodQfVal(), missQfVal()

### QString ODVVariable::commentLabel () const

**Returns:**
The comment for the variable.

### bool ODVVariable::compareFullName (const QString & *otherName*, const QString & *otherUnit*, Qt::CaseSensitivity *cs* = Qt::CaseInsensitive) const

Compares the variable's raw name and unit strings with *otherName* and *otherUnit* .

**Parameters:**

| | | |
|---|---|---|
| in | *otherName* | name label to compare with |
| in | *otherUnit* | unit label to compare with |
| in | *cs* | case-sensitivity of the comparison |

**Returns:**
`true` if the strings are equal and `false` otherwise.

**See also:**
ODVVariable::compareName(), ODVVariable::compareUnit()

### bool ODVVariable::compareName (const QString & *otherName*, Qt::CaseSensitivity *cs* = Qt::CaseInsensitive) const

Compares variable's raw name string with *otherName* .

**Parameters:**

| | | |
|---|---|---|
| in | *otherName* | name label to compare with |
| in | *cs* | case-sensitivity of the comparison |

**Returns:**
`true` if the strings are equal and `false` otherwise.

**See also:**

ODVVariable::compareFullName(), ODVVariable::compareUnit()

### bool ODVVariable::compareUnit (const QString & *otherUnit*, Qt::CaseSensitivity *cs* = `Qt::CaseInsensitive`) const

Compares variable's raw unit string with *otherUnit* .

**Parameters:**

| | | |
|---|---|---|
| in | *otherUnit* | unit label to compare with |
| in | *cs* | case-sensitivity of the comparison |

**Returns:**

`true` if the strings are equal and `false` otherwise.

**See also:**

ODVVariable::compareFullName(), ODVVariable::compareName()

### ODVVariable * ODVVariable::createClone () const`[virtual]`

**Returns:**

A new cloned ODVVariable object with the same properties.

### int ODVVariable::decimalCount () const

**Returns:**

The number of digits following the decimal point to be used in formatted output for this variable.

### char ODVVariable::defaultQfVal ()

**Returns:**

The default quality flag value of this variable.

**See also:**

badQfVal(), goodQfVal(), missQfVal()

### char ODVVariable::defaultQfVal (double *dVal*)

**Returns:**

The missing value quality flag of this variable if *dVal* is equal to `ODV::missDOUBLE` , or otherwise the default quality flag value of this variable.

**See also:**

badQfVal(), goodQfVal(), missQfVal()

### int ODVVariable::errorVarID () const

**Returns:**

The (0-based) variable ID of the error variable for this variable, or `-1` if variable has no error variable.

**QString ODVVariable::fullLabel (bool *doClean* = `false`, bool *prependID* = `false`) const`[virtual]`**

Returns the full label of the variable, consisting of variable name and units.

~-style control sequences are removed, if *doClean* is `true` . If *prependID* is `true` the label is prepended with the 1-based variable ID.

**See also:**

nameLabel(), unitLabel()

**char ODVVariable::goodQfVal ()**

**Returns:**

The good quality flag value of this variable.

**See also:**

badQfVal(), defaultQfVal(), missQfVal()

**virtual bool ODVVariable::isDerived () const`[inline]`, `[virtual]`**

**Returns:**

Always false.

**bool ODVVariable::isLatitude () const`[virtual]`**

**Returns:**

`true` if this is a `Latitude` variable, and `false` otherwise.

Essentially checks whether the variable's *name* contains the string "latitude" (case insensitive check).

**bool ODVVariable::isLongitude () const`[virtual]`**

**Returns:**

`true` if this is a `Longitude` variable, and `false` otherwise.

Essentially checks whether the variable's *name* contains the string "longitude" (case insensitive check).

**bool ODVVariable::isMandatoryMetaVar () const**

**Returns:**

`true` if this variable is a mandatory meta variable and `false` otherwise.

Currently only some of the meta-variables are considered mandatory. See paragraph *"Meta-variables"* in chapter 3 of *"ODV User's Guide"* for further details.

**bool ODVVariable::isMeta () const**

**Returns:**

`true` if this variable is a meta variable.

**bool ODVVariable::isNumeric () const**

**Returns:**

`true` if this variable has one of the numeric value types.

**See also:**

ODVVariable::ValueType

### double ODVVariable::maxVal () const

Returns the current maximal value for this variable.

**See also:**

setMaxVal(), minVal()

### double ODVVariable::minVal () const

Returns the current minimal value for this variable.

**See also:**

setMinVal(), maxVal()

### char ODVVariable::missQfVal ()

**Returns:**

The missing value quality flag value of this variable.

**Note:**

Some quality flag schemes such as ODVQualityFlagSet::ODV do not have a special flag for missing values. The default quality flag is returned if this variable uses one of these schemes.

**See also:**

badQfVal(), defaultQfVal(), goodQfVal()

### QString ODVVariable::nameLabel (bool *doClean* = `false`) const

**Returns:**

The name label of the variable.

~-style control sequences are removed, if *doClean* is `true` .

**See also:**

fullLabel(), unitLabel()

### int ODVVariable::nativeDecimalCount () const

**Returns:**

The default number of decimal places (digits following the decimal point) for the variable's value type.

**See also:**

decimalCount()

### char ODVVariable::qfGenericValue (char *qfVal*) const

**Returns:**

The corresponding `ODVQualityFlagSet::ODV` quality flag value for quality flag *qfVal* in the variable's quality flag set.

**const ODVQualityFlagSet & ODVVariable::qfSet () const**

**Returns:**

A reference to the variable's quality flag set.

**void ODVVariable::range (double & *min*, double & *max*) const**

**Returns:**

The current minimal and maximal values of the variable in *min* & *max* .

If min/max values are not known `ODV::missDOUBLE` and `ODV::largeDOUBLE` are returned.

**double ODVVariable::relativeRange () const**

**Returns:**

The current relative value range of the variable.

The relative range is defined as: (maxVal-minVal)/max(abs(maxVal),abs(minVal)). `ODV::missDOUBLE` is returned if min/max values are not known.

**void ODVVariable::setComment (const QString & *cmt*)**

Sets the comment member of this variable to *cmt* .

**void ODVVariable::setDecimalCount (int *decimals*)**

Sets the number of decimal places (digits following the decimal point) for this variable to *decimals* .

**void ODVVariable::setErrorVarID (int *errID*)**

Sets the error variable ID of this variable to *errID* (0-based; -1: no error variable)

**void ODVVariable::setMaxVal (double *max*)**

Sets the current maximal value for this variable to *max* .

**See also:**

setRange()

**void ODVVariable::setMinVal (double *min*)**

Sets the current minimal value for this variable to *min* .

**See also:**

setRange()

**void ODVVariable::setName (const QString & *nameLabel*)**

Sets the name label of the variable to *nameLabel* .

**void ODVVariable::setProperties (const ODVVariable & *var*)[virtual]**

Applies the properties of *var* to this ODVVariable object.

**void ODVVariable::setQFSet (ODVQualityFlagSet::QFSetID *qfSetID*)**

Sets the quality flag scheme of this variable to *qfSetID* .

**void ODVVariable::setRange (double *min*, double *max*)**

Sets the current value range for this variable to [*min* , *max* ].

**See also:**

setMinVal(), setMaxVal()

**void ODVVariable::setType (VarType *type*)**

Sets the variable type of this variable to *type* .

**Warning:**

It is dangerous to change the type of an existing variable! Use this function only if you know what your are doing.

**void ODVVariable::setUnits (const QString & *unitLabel*)**

Sets the unit label of the variable to *unitLabel* .

**void ODVVariable::setValueType (ValueType *vt*, unsigned int *byteLengths* = 21)**

Sets the value type of the variable to *vt* .

*byteLengths* is only used for value type `ODVVariable::TEXT` . For `ODVVariable::TEXT` variables add 1 Byte for the terminating zero character.

**void ODVVariable::setVarID (int *varID*)**

Sets the variable ID of this variable to *varID* .

**Warning:**

The user is responsible for providing a valid *varID* . It is not checked whether a variable with *varID* already exists!

**See also:**

varID()

**QString ODVVariable::stringValue (double *d*, int *decimals*)**

**Returns:**

The text representation of the value *d* or an empty string, if *d* is `ODV::missDOUBLE` .

The value is rounded to *decimals* decimal places before producing the text representation. If *decimals* is -1 on entry, the variable's decimalCount() value is used. If *decimals* is -2 on entry, the number of significant digits is derived from variable's value type.

**ODVVariable::VarType ODVVariable::type () const**

**Returns:**

The type of the variable.

**QString ODVVariable::unitLabel (bool *doClean* = `false`) const**

**Returns:**

The units label of the variable.

~-style control sequences are removed, if *doClean* is true.

**See also:**

fullLabel(), nameLabel()

**void ODVVariable::updateRange (double *min*, double *max*, bool *doAutoScale* = `true`)**

Updates the current range of the variable values to [*min* , *max* ] range.

Autoscales the range if *doAutoScale* is true.

**unsigned int ODVVariable::valueByteSize (ValueType *valueType*, unsigned int *nBytes* = 21)`[static]`**

**Returns:**

The size of data values in bytes for the given *valueType* , or *nBytes* , if *valueType* is ODVVariable::TEXT, or `0` if type is unknown

**unsigned int ODVVariable::valueByteSize () const**

**Returns:**

The number of bytes per raw value of this variable.

**ODVVariable::ValueType ODVVariable::valueType () const**

**Returns:**

The value type of the variable.

**int ODVVariable::varID () const**

**Returns:**

The variable ID of the variable.

**See also:**

setVarID()

# File Documentation

## macros.h File Reference

This file declares some helpful macros to deal with certain enumerations.
`#include <QMetaEnum>`

### Macros

- #define ENUM_OBJ(o, e) (o::staticMetaObject.enumerator(o::staticMetaObject.indexOfEnumerator(#e)))
- #define ENUM_NAME(o, e, v) (ENUM_OBJ(o,e).valueToKey((v)))

---

### Detailed Description

This file declares some helpful macros to deal with certain enumerations.

---

### Macro Definition Documentation

**#define ENUM_NAME( o, e, v) (ENUM_OBJ(o,e).valueToKey((v)))**

This macro returns the name of an enumeration object member *v* . The class name *o* and the enumeration object name *e* must be specified too.

**See also:**

ENUM_OBJ()

**#define ENUM_OBJ( o, e) (o::staticMetaObject.enumerator(o::staticMetaObject.indexOfEnumerator(#e)))**

This macro returns the QMetaEnum object for class *o* and enumeration object *e* . For more information on QMetaEnums see Qt documentation.

# odv.h File Reference

This file provides globally used declarations.

```
#include <QObject>
#include "tools/declspec.h"
```

## Classes

- class ODV

## *Declarations for global use.* Macros

- #define VID_NONE ODV::missINT32

---

## Detailed Description

This file provides globally used declarations.

---

## Macro Definition Documentation

**#define VID_NONE ODV::missINT32**

Definition for a not assigned variable ID

# odvdate.h File Reference

This file declares global functions which are needed for dealing with date and time.

```
#include <QString>
#include "tools/declspec.h"
#include "tools/odv.h"
```

## Functions

- DECLSPEC void dateFromDecimalYear (double decYear, short &sYear, short &sMonth, short &sDay, short &sHH, short &sMM, double &dSS)
  *Converts a decimal time decYear (in decimal years) to a Gregorian date consisting of year sYear , month sMonth , day sDay and daytime sHH , sMM and dSS .*
- DECLSPEC void dateFromJulianDay (double julDay, int &year, int &month, int &day, int &hour, int &min, double &sec, bool isChronological=true)
  *Converts a Chronological Julian Day julDay to a Gregorian date consisting of year , month , day and daytime hour , min and sec .*
- DECLSPEC QString dateString (ODV::DateForm dateForm, short sYear, short sMonth, short sDay)
- DECLSPEC QString dateString (ODV::DateForm dateForm, double dYear, double dMonth, double dDay)
- DECLSPEC void daytimeFromFractionalDay (double fracDay, int &hour, int &min, double &sec)
  *Retrieves daytime hour , min , and sec from fractional day fracDay .*
- DECLSPEC double decimalDay (short sHH, short sMM, double dSS=0.)
- DECLSPEC double decimalYear (short sYear, short sMonth, short sDay, short sHH, short sMM, double dSS=0.)
- DECLSPEC double decimalYearFromGregorianDay (int gregDay)
- DECLSPEC double decimalYearFromGregorianDay (double dGregDay)
- DECLSPEC void getDateAndTime (char *szDT)
  *Retrieves the current date and time in szDT .*
- DECLSPEC int getDayOfYear (short sYear, short sMonth, short sDay)
- DECLSPEC int getDayOfYear (double decYear)
- DECLSPEC void gregorianDate (int days, short &sYear, short &sMonth, short &sDay)
  *Given the Gregorian days calculates the associated date.*
- DECLSPEC void gregorianDateInYear (int year, int dayOfYear, short &sMonth, short &sDay)
  *Given the year year and the day in this year dayOfYear calculates and returns the month sMonth and day sDay .*
- DECLSPEC int gregorianDay (int year, int month, int day)
- DECLSPEC int gregorianDayOfWeek (int year, int month, int day)
  *Returns the day of the week (Gregorian calender) given a date on the Gregorian calender.*
- DECLSPEC int gregorianDayOfYear (int year, int month, int day)
- DECLSPEC int gregorianDaysInMonth (int year, int month)
- DECLSPEC int gregorianDaysInYear (int year)
- DECLSPEC bool isGregorianLeapYear (int year)
- DECLSPEC QString isoDateFromGregorianDay (double gd)
- DECLSPEC int julianDay (int year, int month, int day)
- DECLSPEC QString timeString (double dHH, double dMM, double dSS)
- DECLSPEC bool validateDate (short &sYear, short &sMonth, short &sDay, short &sHH, short &sMM, double &sec)
- DECLSPEC bool validateDate (int &year, int &month, int &day, int &hour, int &min, double &sec)
- DECLSPEC bool validateTime (short &sHH, short &sMM, double &sec, short &dayShift)
- DECLSPEC bool validateTime (int &hour, int &min, double &sec, int &dayShift)

## Detailed Description

This file declares global functions which are needed for dealing with date and time.

---

## Function Documentation

### DECLSPEC void dateFromDecimalYear (double *decYear*, short & *sYear*, short & *sMonth*, short & *sDay*, short & *sHH*, short & *sMM*, double & *dSS*)

Converts a decimal time *decYear* (in decimal years) to a Gregorian date consisting of year *sYear* , month *sMonth* , day *sDay* and daytime *sHH* , *sMM* and *dSS* .

**Returns:**

The date & time values in the parameter references.

### DECLSPEC void dateFromJulianDay (double *julDay*, int & *year*, int & *month*, int & *day*, int & *hour*, int & *min*, double & *sec*, bool *isChronological*)

Converts a Chronological Julian Day *julDay* to a Gregorian date consisting of *year* , *month* , *day* and daytime *hour* , *min* and *sec* .

*julDay* is considered to be a Chronological Julian Date if *isChronological* is true (the default). Otherwise, *julDay* is considered to be an Astronomical Julian Date.

For the definition of CJD see here.

The chronological Julian date in the GMT timezone is the number of days and fraction of a day which have elapsed since midnight GMT at the start of -4712-01-01 in the proleptic Julian Calendar. For example, for 17:13 GMT on 2007-01-19 CE the corresponding chronological Julian date is 2454120.7176.

The chronological Julian date at a particular timezone is the number of days and fraction of a day which have elapsed since midnight in that timezone at the start of -4712-01-01 in the proleptic Julian Calendar. For example, for 01:13 Beijing standard time on 2007-01-20 CE the corresponding chronological Julian date is 2454121.0509.

**Returns:**

The date & time values in the parameter references.

### DECLSPEC QString dateString (ODV::DateForm *dateForm*, short *sYear*, short *sMonth*, short *sDay*)

**Returns:**

The date string given by *sYear* , *sMonth* , and *sDay* in *dateForm* format.

### DECLSPEC QString dateString (ODV::DateForm *dateForm*, double *dYear*, double *dMonth*, double *dDay*)

**Returns:**

The date string given by *dYear* , *dMonth* , and *dDay* in *dateForm* format.

**DECLSPEC void daytimeFromFractionalDay (double *fracDay*, int & *hour*, int & *min*, double & *sec*)**

Retrieves daytime *hour* , *min* , and *sec* from fractional day *fracDay* .

*fracDay* must be between 0 and 1.

**DECLSPEC double decimalDay (short *sHH*, short *sMM*, double *dSS*)**

**Returns:**

Daytime (in decimal days) for a given time.

**DECLSPEC double decimalYear (short *sYear*, short *sMonth*, short *sDay*, short *sHH*, short *sMM*, double *dSS*)**

**Returns:**

Time (in decimal years) for a given calendar date & time.

**DECLSPEC double decimalYearFromGregorianDay (int *gregDay*)**

**Returns:**

Time (in decimal years) for a given Gregorian day *gregDay* .

**DECLSPEC double decimalYearFromGregorianDay (double *dGregDay*)**

**Returns:**

Time (in decimal years) for a given Gregorian day *dGregDay* including daytime as fractional part.

**DECLSPEC void getDateAndTime (char * *szDT*)**

Retrieves the current date and time in *szDT* .

*szDT* must be at least 26 characters long.

**DECLSPEC int getDayOfYear (short *sYear*, short *sMonth*, short *sDay*)**

**Returns:**

Gregorian day of the year for a Gregorian calendar date.

**DECLSPEC int getDayOfYear (double *decYear*)**

**Returns:**

Gregorian day of the year for a decimal time [years] *decYear* .

**DECLSPEC void gregorianDate (int *days*, short & *sYear*, short & *sMonth*, short & *sDay*)**

Given the Gregorian *days* calculates the associated date.

**DECLSPEC void gregorianDateInYear (int *year*, int *dayOfYear*, short & *sMonth*, short & *sDay*)**

Given the year *year* and the day in this year *dayOfYear* calculates and returns the month *sMonth* and day *sDay* .

**DECLSPEC int gregorianDay (int *year*, int *month*, int *day*)**

**Returns:**
The Gregorian days for a date on the Gregorian calender.

**DECLSPEC int gregorianDayOfWeek (int *year*, int *month*, int *day*)**

Returns the day of the week (Gregorian calender) given a date on the Gregorian calender.

**Returns:**
0=Monday, ... 5=Saturday, 6=Sunday
Reference: C. Zeller, Kalender-Formeln, Acta Mathematica, 9 (1887) 131-136

**DECLSPEC int gregorianDayOfYear (int *year*, int *month*, int *day*)**

**Returns:**
The day of the year (Gregorian calender) given a date on the Gregorian calender.

**DECLSPEC int gregorianDaysInMonth (int *year*, int *month*)**

**Returns:**
The last day of *month* for the Gregorian calendar.

**DECLSPEC int gregorianDaysInYear (int *year*)**

**Returns:**
The number of days in *year* .

**DECLSPEC bool isGregorianLeapYear (int *year*)**

**Returns:**
`true` if *year* is a leap year according to the Gregorian calendar, or `false` otherwise.

**DECLSPEC QString isoDateFromGregorianDay (double *dGregDay*)**

**Returns:**
The ISO date/time string for fractional Gregorian day *dGregDay* .

**DECLSPEC int julianDay (int *year*, int *month*, int *day*)**

**Returns:**
Julian day for a given date on the Gregorian calender.

**DECLSPEC QString timeString (double *dHH*, double *dMM*, double *dSS*)**

**Returns:**

The time as string.

If hour *dHH* is invalid an empty string is returned, if minute *dMM* is invalid only the hours are returned in string, if second *dSS* is invalid it is omitted.

**DECLSPEC bool validateDate (short & *year*, short & *month*, short & *day*, short & *hour*, short & *minute*, double & *sec*)**

Ensures that the specified date values are valid, and make modifications if necessary.

**Returns:**

`true` if modifications were made and `false` otherwise.

**DECLSPEC bool validateDate (int & *year*, int & *month*, int & *day*, int & *hour*, int & *minute*, double & *sec*)**

Overloaded method.

**See also:**

validateDate(short&,short&,short&,short&,short&,double&)

**DECLSPEC bool validateTime (short & *hour*, short & *minute*, double & *sec*, short & *dayShift*)**

Ensures that the specified time values are valid, and make modifications if necessary. If a day shift arises from the modifications it is returned in *dayShift* .

**Returns:**

`true` if modifications were made and `false` otherwise.

**DECLSPEC bool validateTime (int & *hour*, int & *minute*, double & *sec*, int & *dayShift*)**

Overloaded method.

**See also:**

validateTime(short&,short&,double&,short&)

# Index