ODV API for Java

Reiner Schlitzer

Version 2.0
March 25, 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

ODV4 Application Programming Interface for Java

Author:

Reiner Schlitzer

Introduction

The Java ODV API provides a set of classes that can be used in your own Java 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 Java ODV API are provided for Windows, Mac OS X and Linux systems. The ODV API for Java is delivered as package `"de.awi.odv"`, meaning that the fully qualified names of all classes have the prefix `"de.awi.odv"`. Detailed documentation of all required classes and their functions is provided in this document. The package also includes example Java code showing how to use the API.

The ODV API for Java supports all ODV collection formats, including the ODVCF6 format introduced with ODV 4.6.0.

Main Classes

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

(1) An de.awi.odv.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 de.awi.odv.ODVVariable objects. *ODVVariables* have name and unit labels, as well as variable and value types. *ODVVariables* can hold numeric or string data.

(3) de.awi.odv.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 de.awi.odv.ODVCollectionInventory and de.awi.odv.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

For an installation guide and descriptions of the content of the ODV4 API package please see the *install_odv4api_java.txt* file shipped with the API.

Example Code

The supplied *ODV4example* application reads an ODV collection and writes its contents to a spreadsheet text file. The *exportAsSpreadsheet()* function of the example contains all required code for opening and reading a collection. First an *ODVCollection* object must be created and de.awi.odv.ODVCollection.open() must be called to open the collection. Then a de.awi.odv.ODVStation object is created with the collection object as argument. The metadata and data of a station can then be read using the de.awi.odv.ODVStation.readData() member function. *ODVVariable* objects can be obtained from the collection object via the de.awi.odv.ODVCollection.var() or de.awi.odv.ODVCollection.metaVar() functions. Numeric or string values of a given variable for specific samples are obtained with de.awi.odv.ODVStation.value(ODVVariable, int) or de.awi.odv.ODVStation.stringValue(ODVVariable, int). The de.awi.odv.ODVStation.data(ODVVariable) function is called to obtain arrays of numeric data for a given variable and station. Quality flag data can be obtained using de.awi.odv.ODVStation.qfData().

String data are handled by the API as de.awi.odv.QString objects. de.awi.odv.QString has a constructor taking a Java string as parameter. To obtain a Java string from a QString use de.awi.odv.QString.toLocal8Bit().data().

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:

Class Documentation

de.awi.odv.ODV.AccessMode Enum Reference

File and collection access modes.

Public Member Functions

- final int swigValue ()

Static Public Member Functions

- static AccessMode swigToEnum (int swigValue)

Public Attributes

- NoAccess =(0)
- ReadOnly =(1)
- ReadWrite =(3)

Detailed Description

File and collection access modes.

You may apply `int swigValue()` to the AccessMode enum object to obtain the corresponding integer value.

Member Function Documentation

AccessMode ODV.AccessMode.swigToEnum (int *swigValue*)`[static]`

Returns:

The AccessMode enum value corresponding to the supplied integer *swigValue* .

final int ODV.AccessMode.swigValue ()

Returns:

The integer value corresponding to the AccessMode enum value.

Member Data Documentation

ODV.AccessMode.NoAccess =(0)

No access.

ODV.AccessMode.ReadOnly =(1)

Read-only access.

ODV.AccessMode.ReadWrite =(3)

Read-write access.

de.awi.odv.ODVCollection.DataField Enum Reference

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

Public Member Functions

- final int swigValue ()

Static Public Member Functions

- static DataField swigToEnum (int swigValue)

Public Attributes

- GeneralField =(0)
- Ocean
- Atmosphere
- Land
- IceSheet
- SeaIce

Detailed Description

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

You may apply `int swigValue()` to the DataField enum object to obtain the corresponding integer value.

Note:

There is an additional value in this enum class available which is unfortunately not documented correctly due to technical reasons:
ODVCollection.DataField.Sediment : Sediment data

Member Function Documentation

DataField ODVCollection.DataField.swigToEnum (int *swigValue*)[static]

Returns:

The DataField enum value corresponding to the supplied integer *swigValue* .

int ODVCollection.DataField.swigValue ()

Returns:

The integer value corresponding to the DataField enum value.

Member Data Documentation

ODVCollection.DataField.Atmosphere

Atmospheric data

ODVCollection.DataField.GeneralField =(0)
General data field (default)

ODVCollection.DataField.IceSheet
Data on ice sheets

ODVCollection.DataField.Land
Data is related to land

ODVCollection.DataField.Ocean
Oceanographic data

ODVCollection.DataField.SeaIce
Sea ice data

de.awi.odv.ODVCollection.DataType Enum Reference

This property describes the type of data in the collection.

Public Member Functions

- final int swigValue ()

Static Public Member Functions

- static DataType swigToEnum (int swigValue)

Public Attributes

- GeneralType =(0)
- Profiles
- Trajectories

Detailed Description

This property describes the type of data in the collection.

You may apply `int swigValue()` to the DataType enum object to obtain the corresponding integer value.

Note:

There is an additional value in this enum class available which is unfortunately not documented correctly due to technical reasons:
ODVCollection.DataType.TimeSeries : Data at a fixed location reapeated over time

---

Member Function Documentation

DataType ODVCollection.DataType.swigToEnum (int *swigValue*)[static]

Returns:

The DataType enum value corresponding to the supplied integer *swigValue* .

final int ODVCollection.DataType.swigValue ()

Returns:

The integer value corresponding to the DataType enum value.

Member Data Documentation

ODVCollection.DataType.GeneralType =(0)

General data type (default)

ODVCollection.DataType.Profiles

Profile data

ODVCollection.DataType.Trajectories

Data collected from a moving platform reapeated over time

de.awi.odv.ODV.DateForm Enum Reference

Date text formats which are supported by ODV.

Public Member Functions

- final int swigValue ()

Static Public Member Functions

- static DateForm swigToEnum (int swigValue)

Public Attributes

- IsoDate
- mmddyyyyDate
- mmmddyyyyDate
- ddmonthyyyyDate

Detailed Description

Date text formats which are supported by ODV.

You may apply `int swigValue()` to the DateForm enum object to obtain the corresponding integer value.

Note:

There is an additional value in this enum class available which is unfortunately not documented correctly due to technical reasons:
ODV.DateForm.ddmmmyyyyDate : Date with abbreviated english month name.
Example: `23 Feb 2006` for Feb/23/2006.

Member Function Documentation

DateForm ODV.DateForm.swigToEnum (int *swigValue*)[static]

Returns:

The DateForm enum value corresponding to the supplied integer *swigValue* .

final int ODV.DateForm.swigValue ()

Returns:

The integer value corresponding to the DateForm enum value.

Member Data Documentation

ODV.DateForm.ddmonthyyyyDate

Date with full english month name.

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

ODV.DateForm.IsoDate

Date format according to ISO 8601.

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

ODV.DateForm.mmddyyyyDate

Date in one pass without separators.

Example: `02232006` for Feb/23/2006

ODV.DateForm.mmmddyyyyDate

Date with abbreviated english month name in front.

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

de.awi.odv.ODVStation.MetaVarIndex Enum Reference

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

Public Member Functions

- final int swigValue ()

Static Public Member Functions

- static MetaVarIndex swigToEnum (int swigValue)

Public Attributes

- MetaCruiseIndex =(0)
- MetaStationIndex =(1)
- MetaTypeIndex =(2)
- MetaLongitudeIndex =(3)
- MetaLatitudeIndex =(4)
- MetaYearIndex =(5)
- MetaMonthIndex =(6)
- MetaDayIndex =(7)
- MetaHourIndex =(8)
- MetaMinuteIndex =(9)
- MetaSecondIndex =(10)
- MetaAccessionIndex =(11)

Detailed Description

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"* .

You may apply `int swigValue()` to the MetaVarIndex enum object to obtain the corresponding integer value.

Member Function Documentation

MetaVarIndex ODVStation.MetaVarIndex.swigToEnum (int *swigValue*)`[static]`

Returns:

The MetaVarIndex enum value corresponding to the supplied integer *swigValue* .

final int ODVStation.MetaVarIndex.swigValue ()

Returns:

The integer value corresponding to the MetaVarIndex enum value.

Member Data Documentation

ODVStation.MetaVarIndex.MetaAccessionIndex =(11)

ODVStation.MetaVarIndex.MetaCruiseIndex =(0)

ODVStation.MetaVarIndex.MetaDayIndex =(7)

ODVStation.MetaVarIndex.MetaHourIndex =(8)

ODVStation.MetaVarIndex.MetaLatitudeIndex =(4)

ODVStation.MetaVarIndex.MetaLongitudeIndex =(3)

ODVStation.MetaVarIndex.MetaMinuteIndex =(9)

ODVStation.MetaVarIndex.MetaMonthIndex =(6)

ODVStation.MetaVarIndex.MetaSecondIndex =(10)

ODVStation.MetaVarIndex.MetaStationIndex =(1)

ODVStation.MetaVarIndex.MetaTypeIndex =(2)

ODVStation.MetaVarIndex.MetaYearIndex =(5)

de.awi.odv.ODV Class Reference

This class provides globally used declarations.

Classes

- enum AccessMode
- *File and collection access modes.* enum DateForm
- *Date text formats which are supported by ODV.* enum Status

Returned error status of functions. Not all of them apply to the API. Static Public Member Functions

- static byte getMissINT8 ()
- static short getMissUINT8 ()
- static short getMissINT16 ()
- static int getMissUINT16 ()
- static int getMissINT32 ()
- static long getMissUINT32 ()
- static float getMissFLOAT ()
- static double getMissDOUBLE ()
- static byte getLargeINT8 ()
- static short getLargeUINT8 ()
- static short getLargeINT16 ()
- static int getLargeUINT16 ()
- static int getLargeINT32 ()
- static long getLargeUINT32 ()
- static float getLargeFLOAT ()
- static double getLargeDOUBLE ()

Detailed Description

This class provides globally used declarations.

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

Member Function Documentation

public static double ODV.getLargeDOUBLE ()[static]

Returns:

Large positive value for double data type.

public static float ODV.getLargeFLOAT ()[static]

Returns:

Large positive value for float data type.

public static short ODV.getLargeINT16 ()[static]

Returns:
Large positive value for 16-bit signed short data type.

public static int ODV.getLargeINT32 ()[static]

Returns:
Large positive value for 32-bit signed int data type.

public static byte ODV.getLargeINT8 ()[static]

Returns:
Large positive value for 8-bit signed char data type.

public static int ODV.getLargeUINT16 ()[static]

Returns:
Large positive value for 16-bit unsigned short data type.

public static long ODV.getLargeUINT32 ()[static]

Returns:
Large positive value for 32-bit unsigned int data type.

public static short ODV.getLargeUINT8 ()[static]

Returns:
Large positive value for 8-bit unsigned char data type.

public static double ODV.getMissDOUBLE ()[static]

Returns:
Missing value for double data type.

public static float ODV.getMissFLOAT ()[static]

Returns:
Missing value for float data type.

public static short ODV.getMissINT16 ()[static]

Returns:
Missing value for 16-bit signed short data type.

public static int ODV.getMissINT32 ()[static]

Returns:

Missing value for 32-bit signed int data type.

public static byte ODV.getMissINT8 ()[static]

Returns:

Missing value for 8-bit signed char data type.

public static int ODV.getMissUINT16 ()[static]

Returns:

Missing value for 16-bit unsigned short data type.

public static long ODV.getMissUINT32 ()[static]

Returns:

Missing value for 32-bit unsigned int data type.

public static short ODV.getMissUINT8 ()[static]

Returns:

Missing value for 8-bit unsigned char data type.

de.awi.odv.ODVCollection Class Reference

Provides read access to the data of an ODV collection and maintains lists of meta and collection variables.

Classes

- enum DataField
- *This property describes the field to which the collection data belongs.* enum DataType
- *This property describes the type of data in the collection.* enum StateFlag

This flag describes the current state of the collection. Public Member Functions

- ODVCollection (QString collectionName)
  Create a new collection object.
- ODV.AccessMode accessMode ()
- long appendHistoryString (long accession, QString string)
  Appends the string string as a new history record for the station with accession number accession .
- int appendHistoryStrings (long accession, QStringList stringList)
  Appends all strings in stringList as new history records for the station with accession number accession .
- int appendMetaVar (ODVVariable var)
  Adds meta variable var to the list of meta variables.
- int appendVar (ODVVariable var, int varID)
  Adds basic variable var to the list of collection variables.
- int appendVar (ODVVariable var)
- QString baseDir ()
- int basicVarCount ()
- boolean changePassword (QString newPassWord, QString oldPassWord)
  Changes the password of the collection from oldPassWord to newPassWord .
- boolean changePassword (QString newPassWord)
  Set the password of the collection to newPassWord .
- void close ()
- ODVCollectionInventory collectionInventory ()
- ODV.AccessMode currentAccessMode ()
- ODVVariablePtrList extendedMetaVars ()
- QString extension ()
- QString filePath ()
- void generalProperties (int[] dField, int[] dType)
- QStringList historyStrings (long accession)
  Retrieves all history strings for the station with accession number accession .
- int instanceID ()
- boolean isOpen ()
- boolean isPasswordProtected ()
- ODV.Status loadCollectionFile ()
  Loads the collection file, i.e. variable definitions and collection properties are read and set.
- ODVVariable metaVar (int varID)
- ODVVariable metaVar (ODVVariable.VarType varType)
- int metaVarCount ()
- int metaVarID (ODVVariable.VarType varType)
- ODVVariablePtrList metaVarPtrList ()
- QString name ()
- ODV.Status open (ODV.AccessMode requestedAccessMode, QString password)

- ODV.Status open (ODV.AccessMode requestedAccessMode)
- QString pathName ()
- ODVVariable primaryVar ()
- int primaryVarID ()
- QString rootDir ()
- QString settingsFilePath ()
- long sizeOfDataFile ()
- int stationCount ()
- ODVCollection_stateflag state ()
- int totalVarCount ()
- ODVVariable var (int varID)
- ODVVariable var (ODVVariable.VarType varType)
- int varID (ODVVariable.VarType varType)
- QIntList varIDList ()
- ODVVariablePtrList varPtrList ()

Static Public Member Functions

- static ODVCollection.DataField dataFieldID (QString fieldName)
- static ODVCollection.DataType dataTypeID (QString typeName)
- static boolean isInUse (QString filePath)

Detailed Description

Provides read access to the data of an ODV collection and maintains lists of meta and collection variables.

Three different collection formats are supported:

- `CF6` (*.odv) is the standard format
- `CF5` (*.odv) was the former standard format
- `GENERIC` (*.var) is the format used in ODV3

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

Variables can be added but not deleted. So far this class allows not to add variables permanently.

Reading of data can not be done directly through this interface, instead an ODVStation object needs to be created with the opened collection and this interface has to be used.

Constructor & Destructor Documentation

public ODVCollection.ODVCollection (QString *collectionName*)

Create a new collection object.

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

The collection specified by *collectionName* must exist.

Member Function Documentation

public ODV.AccessMode ODVCollection.accessMode ()

Returns:

The access mode for the collection granted during the open() call.

See also:

currentAccessMode()

public long ODVCollection.appendHistoryString (long *accession*, QString *string*)

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

Returns:

The string index for *string* in the history string pool, or ODV.getMissUINT32() if the append failed.

public int ODVCollection.appendHistoryStrings (long *accession*, QStringList *stringList*)

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

Returns:

The number of history records appended.

public 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.

public int ODVCollection.appendVar (ODVVariable *var*, int *varID*)

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.getMissINT32(). The var ID of basic collection variables must be equal to their 0-based index.

public int ODVCollection.appendVar (ODVVariable *var*)

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

Returns:

The ID of the new variable which is the next available ID.

public QString ODVCollection.baseDir ()

Returns:

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

public int ODVCollection.basicVarCount ()

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.

public boolean ODVCollection.changePassword (QString *newPassWord*, QString *oldPassWord*)

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(QString 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` .

public boolean ODVCollection.changePassword (QString *newPassWord*)

Set the password of the collection to *newPassWord* .

Note:

By default newly created ODV collections have no password. Password protection is only available for `ODVCF6` collections.

Returns:

`true` if successful or `false` if the password was not set, for instance, because the collection is not `ODVCF6` or a password was already set.

public void ODVCollection.close ()

Closes the collection and deletes all temporary files.

public ODVCollectionInventory ODVCollection.collectionInventory ()

Returns:

The collection inventory.

public 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.AccessMode.ReadWrite` access is downgraded to `ODV.AccessMode.ReadOnly` if more than one application are using this collection at the same time.

See also:

accessMode()

public static ODVCollection.DataField ODVCollection.dataFieldID (QString *fieldName*)[static]

Returns:

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

public static ODVCollection.DataType ODVCollection.dataTypeID (QString *typeName*)[static]

Returns:

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

public ODVVariablePtrList ODVCollection.extendedMetaVars ()

Returns:

A list with the collection's extended meta variables.

Extended meta variables are those with a ODVVariable.VarType outside the [METACRUISE,METAPRIMVARMAX] range.

public QString ODVCollection.extension ()

Returns:

The extension of the collection file including the leading dot.

public QString ODVCollection.filePath ()

Returns:

The collection's full file path including extension.

public void ODVCollection.generalProperties (int[] *dField*, int[] *dType*)

Returns:

The general properties of this collection in the first element of arrays *dField* and *dType* .

The returned values are the integer value of the respective enumeration value ODVCollection.DataField and ODVCollection.DataType. In order to get the correct result types use the following:

```
int dField[] = new int[1];
int dType[] = new int[1];
collection.generalProperties(dField, dType);
ODVCollection.DataField datafield = ODVCollection.DataField.swigToEnum(dField[0]);
```

ODVCollection.DataType datatype = ODVCollection.DataType.swigToEnum(dType[0]);

public QStringList ODVCollection.historyStrings (long *accession*)

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

public int ODVCollection.instanceID ()

Returns:

The unique instance ID of the collection session.

public static boolean ODVCollection.isInUse (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.

public boolean ODVCollection.isOpen ()

Returns:

`true` if collection is open and `false` otherwise.

public boolean ODVCollection.isPasswordProtected ()

Returns:

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

public ODV.Status ODVCollection.loadCollectionFile ()

Loads the collection file, i.e. variable definitions and collection properties are read and set.

Note:

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

Returns:

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

See also:

open()

public ODVVariable ODVCollection.metaVar (int *varID*)

Returns:

Meta variable with variable ID *varID* , or `null` if *varID* is not found.

public ODVVariable ODVCollection.metaVar (ODVVariable.VarType *varType*)

Returns:

Meta variable with type *varType* , or `null` if no such variable is found.

public int ODVCollection.metaVarCount ()

Returns:

The number of meta variables.

public int ODVCollection.metaVarID (ODVVariable.VarType *varType*)

Returns:

Meta variable ID of meta variable with VarType *varType* , or the value returned by ODV.getMissINT32(), if no such meta variable is found.

public ODVVariablePtrList ODVCollection.metaVarPtrList ()

Returns:

List of meta variables.

public QString ODVCollection.name ()

Returns:

Collection name.

See also:

filePath()

public ODV.Status ODVCollection.open (ODV.AccessMode *requestedAccessMode*, QString *password*)

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

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

Returns:

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

See also:

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

public ODV.Status ODVCollection.open (ODV.AccessMode *requestedAccessMode*)

Opens the collection in access mode *requestedAccessMode* .

Checks the supported access modes and automatically switches to `ODV.AccessMode.ReadOnly` if `ODV.AccessMode.ReadWrite` access is requested but not supported.

Returns:

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

See also:

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

public QString ODVCollection.pathName ()

Returns:

The full pathname of the collection excluding the extension.

See also:
filePath(), name()

public ODVVariable ODVCollection.primaryVar ()

Returns:
The primary variable of the collection.

public int ODVCollection.primaryVarID ()

Returns:
The variable ID of the collection's primary variable.

public QString ODVCollection.rootDir ()

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

public QString ODVCollection.settingsFilePath ()

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

public long ODVCollection.sizeOfDataFile ()

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

public ODVCollection_stateflag ODVCollection.state ()

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

Usage example: If you want to test if a certain flag, e.g. ODVCollection.StateFlag.ColOpen is set, do as follows:

```
ODVCollection_stateflag collState = collection.state();
boolean collOpenFlagIsSet = !collState.QFlags_and(ODVCollection.StateFlag.ColOpen).QFlags_noflagset();
```

public int ODVCollection.stationCount ()

Returns:
Number of stations in this collection.

public int ODVCollection.totalVarCount ()

Returns:
Total number of data variables.

This is the number of basic variables but excludes the meta variables.

public ODVVariable ODVCollection.var (int *varID*)

Returns:

Collection variable with ID *varID* or `null`, if *varID* is not found.

public ODVVariable ODVCollection.var (ODVVariable.VarType *varType*)

Returns:

The first collection variable with variable type *varType*, or `null` if no such variable is found.

public int ODVCollection.varID (ODVVariable.VarType *varType*)

Returns:

The variable ID of the first collection variable with type *varType*, or the value returned by ODV.getMissINT32() if no such variable is found.

public QIntList ODVCollection.varIDList ()

Returns:

The list of collection variable IDs.

public ODVVariablePtrList ODVCollection.varPtrList ()

Returns:

The list of collection variables.

de.awi.odv.ODVCollection_stateflag Class Reference

Objects of this class keep a combination of ODVCollection.StateFlag's. They describe the current state of a collection.

Public Member Functions

- ODVCollection_stateflag (ODVCollection_stateflag other)
- ODVCollection_stateflag (ODVCollection.StateFlag flag)
- int QFlags_intcast ()
- boolean QFlags_noflagset ()
- ODVCollection_stateflag QFlags_and (int mask)
- ODVCollection_stateflag QFlags_and (ODVCollection.StateFlag mask)
- ODVCollection_stateflag QFlags_and_assign (int mask)
- ODVCollection_stateflag QFlags_assign (ODVCollection_stateflag other)
- ODVCollection_stateflag QFlags_or (ODVCollection_stateflag other)
- ODVCollection_stateflag QFlags_or (ODVCollection.StateFlag other)
- ODVCollection_stateflag QFlags_or_assign (ODVCollection_stateflag other)
- ODVCollection_stateflag QFlags_or_assign (ODVCollection.StateFlag other)
- ODVCollection_stateflag QFlags_negate ()

Detailed Description

Objects of this class keep a combination of ODVCollection.StateFlag's. They describe the current state of a collection.

Flags can be assigned to them and the objects can be tested against flags.

This class is a subset and a manifestation of the Qt `QFlags<T>` template class with `T = ODVCollection.StateFlag` . See Qt documentation for further details on `QFlags` .

Constructor & Destructor Documentation

ODVCollection_stateflag.ODVCollection_stateflag (ODVCollection_stateflag *other*)

Constructs a copy of *other* .

ODVCollection_stateflag.ODVCollection_stateflag (ODVCollection.StateFlag *flag*)

Constructs an object storing the given *flag* .

Member Function Documentation

ODVCollection_stateflag ODVCollection_stateflag.QFlags_and (int *mask*)

Returns:

an ODVCollection_stateflag object containing the result of the bitwise AND operation on this object and *mask* .

ODVCollection_stateflag ODVCollection_stateflag.QFlags_and (ODVCollection.StateFlag *mask*)

Returns:

an ODVCollection_stateflag object containing the result of the bitwise AND operation on this object and *mask* .

ODVCollection_stateflag ODVCollection_stateflag.QFlags_and_assign (int *mask*)

This is an overloaded function.

Performs a bitwise AND operation with *mask* and stores the result in this object. Returns this object.

ODVCollection_stateflag ODVCollection_stateflag.QFlags_assign (ODVCollection_stateflag *other*)

Assigns *other* to this object and returns this object.

int ODVCollection_stateflag.QFlags_intcast ()

Returns:

The value stored in the object as an integer.

ODVCollection_stateflag ODVCollection_stateflag.QFlags_negate ()

Returns:

An ODVCollection_stateflag object that contains the bitwise negation of this object.

boolean ODVCollection_stateflag.QFlags_noflagset ()

Returns:

`true` if no flag is set (i.e., if the value stored by the object is 0); otherwise returns `false` .

ODVCollection_stateflag ODVCollection_stateflag.QFlags_or (ODVCollection_stateflag *other*)

Returns:

An ODVCollection_stateflag object containing the result of the bitwise OR operation on this object and *other* .

ODVCollection_stateflag ODVCollection_stateflag.QFlags_or (ODVCollection.StateFlag *other*)

This is an overloaded function.

Returns:

An ODVCollection_stateflag object containing the result of the bitwise OR operation on this object and *other* .

See also:

QFlags_or(ODVCollection_stateflag)

ODVCollection_stateflag ODVCollection_stateflag.QFlags_or_assign (ODVCollection_stateflag *other*)

Performs a bitwise OR operation with *other* and stores the result in this object. Returns this object.

ODVCollection_stateflag ODVCollection_stateflag.QFlags_or_assign (ODVCollection.StateFlag *other*)

This is an overloaded function.

Performs a bitwise OR operation with *other* and stores the result in this object. Returns this object.

See also:

QFlags_or_assign(ODVCollection_stateflag)

de.awi.odv.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.

Public Member Functions

- ODVCollectionInventory (ODVCollection col)
  Creates an new inventory object for collection col .
- SWIGTYPE_p_unsigned_int accessionNumberData ()
- long accessionNumber (int statID)
- int cruiseCount ()
- int cruiseID (int statID)
- SWIGTYPE_p_int cruiseIdData ()
- ODVCruiseInfo cruiseInfo (QString cruiseName)
- QStringList cruiseNames ()
- int dataCount (int varID)
- SWIGTYPE_p_short dayTimeData ()
  An array to the day-time data (in units of 0.1 min since mid-night) of all stations in the collection.
- double decimalYear (int statID)
- boolean decimalYears (SWIGTYPE_p_double decYears, int nStats, SWIGTYPE_p_int statIDs)
  Returns at decYears the date/time of the nStats stations in statIDs as decimal years.
- SWIGTYPE_p_int gregorianDayData ()
  Returns an array of the date data (in Gregorian days) of all stations in the collection.
- double latitude (int statID)
- boolean latitudes (SWIGTYPE_p_double lats, int nStats, SWIGTYPE_p_int statIDs)
  Returns at lats the decimal latitudes of the nStats stations in statIDs .
- double longitude (int statID)
- boolean longitudes (SWIGTYPE_p_double lons, int nStats, SWIGTYPE_p_int statIDs)
  Returns at lons the longitudes of the nStats stations in statIDs .
- ODVMapDomain nativeMapDomain ()
- int sampleCount (int statID)
- int sampleCount ()
- int sampleCount (int nStats, SWIGTYPE_p_int statIDs)
- SWIGTYPE_p_int sampleCountData ()
  Returns an array of the sample count data of all stations in the collection.
- int stationIDFromAccessionNumber (long accNum)
- ODVCruiseInfo summaryCruiseInfo ()

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

public ODVCollectionInventory.ODVCollectionInventory (ODVCollection *col*)

Creates an new inventory object for collection *col* .

Member Function Documentation

public long ODVCollectionInventory.accessionNumber (int *statID*)

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. The value of ODV.getMissUINT32() is returned in these cases.

See also:

accessionNumberData()

public SWIGTYPE_p_unsigned_int ODVCollectionInventory.accessionNumberData ()

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.

Note:

The returned pointer object can not be used directly. It has to be converted to an ODVLongData object which then can be used to retrieve the values.

Example:

```
// Obtain collection inventory object (collection may be given here)
ODVCollectionInventory colInv = collection.collectionInventory();
// Obtain accession numbers and convert to ODVLongData
ODVLongData accNums = ODVLongData.frompointer(colInv.accessionNumberData());
// Check if we got valid data pointer
if (accNums != null)
  // Check if there are accession numbers at all
  if (accNums.getitem(0) != ODV.getMissUINT32())
     // We can use accession numbers!
```

long firstNum = accNums.getitem(0);

See also:

accessionNumber(), ODVLongData

public int ODVCollectionInventory.cruiseCount ()

Returns:

The number of cruises in this ODVCollectionInventory.

public int ODVCollectionInventory.cruiseID (int *statID*)

Returns:

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

public SWIGTYPE_p_int ODVCollectionInventory.cruiseIdData ()

Returns:

An array holding the cruise IDs of every station in the collection.

Index values range from `0` to `ODVCollection::stationCount()-1` .

Note:

The returned pointer object can not be used directly. It has to be converted to an ODVIntData object which then can be used to retrieve the values.

Example:

```
// Obtain collection inventory object (collection may be given here)
ODVCollectionInventory colInv = collection.collectionInventory();
// Obtain cruise IDs and convert to ODVIntData
ODVIntData cruiseIDs = ODVIntData.frompointer(colInv.cruiseIdData());
// Check if we got valid data pointer
if (cruiseIDs != null)
  // We can use the data!
```

int cruiseIDofFirstStation = cruiseIDs.getitem(0);

See also:

cruiseID(), ODVIntData

public ODVCruiseInfo ODVCollectionInventory.cruiseInfo (QString *cruiseName*)

Returns:

An ODVCruiseInfo object of cruise *cruiseName* , or `null` , if no such cruise exists.

public QStringList ODVCollectionInventory.cruiseNames ()

Returns:

A list of all cruise names in the collection.

public int ODVCollectionInventory.dataCount (int *varID*)

Returns:

The total number of non–ODV.getMissDOUBLE() samples for variable ID *varID* of all cruises in this ODVCollectionInventory.

public SWIGTYPE_p_short ODVCollectionInventory.dayTimeData ()

An array 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.getMissINT16()` indicates that day-time is not available or incomplete.

Note:

The returned pointer object cannot be used directly. It has to be converted to a ODVShortData object which then can be used to retrieve the values.

Example:

```
// Obtain collection inventory object (collection may be given here)
ODVCollectionInventory colInv = collection.collectionInventory();
// Obtain day-time data and convert to ODVShortData
ODVShortData dayTime = ODVShortData.frompointer(colInv.dayTimeData());
// Check if we got valid data pointer
if (dayTime != null)
  for (int i=0; i<collection.stationCount(); i++)
    // Check if there is a day time for this station
if (dayTime.getitem(i) != ODV.getMissINT16())
{  // We can use day time!
   short time = dayTime.getitem(i);
   // Make use of it.
   short hour = time / 600;
}
```

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

See also:

ODVShortData

public double ODVCollectionInventory.decimalYear (int *statID*)

Returns:

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

Stations without date and time information return the value of ODV.getMissDOUBLE(). If daytime is missing or incomplete `00:00h` is assumed.

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

See also:

ODVDate

public boolean ODVCollectionInventory.decimalYears (SWIGTYPE_p_double *decYears*, int *nStats*, SWIGTYPE_p_int *statIDs*)

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

Stations without date and time information return the value of ODV.getMissDOUBLE(). If daytime is missing or incomplete `00:00h` is assumed.

Returns:

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

Note:

The pointer objects for *decYears* and *statIDs* have to be created first as ODVDoubleData resp. ODVIntData objects. *decYears* must be sufficient in size to hold *nStats* `double` values for all the entries in *statIDs* . See latitudes() for an analogues usage example.

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

See also:

latitudes(), ODVDoubleData, ODVIntData, ODVDate

public SWIGTYPE_p_int ODVCollectionInventory.gregorianDayData ()

Returns an array of the date data (in Gregorian days) of all stations in the collection.

The value of ODV.getMissINT32() indicates that date information is not available or incomplete.

Note:

The returned pointer object can not be used directly. It has to be converted to an ODVIntData object which then can be used to retrieve the values. See cruiseIdData() for an analogues usage example.

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

See also:

cruiseIdData(), ODVIntData, ODVDate

public double ODVCollectionInventory.latitude (int *statID*)

Returns:

The decimal latitude of station *statID* .

public boolean ODVCollectionInventory.latitudes (SWIGTYPE_p_double *lats*, int *nStats*, SWIGTYPE_p_int *statIDs*)

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

Returns:

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

Note:

The pointer objects for *lats* and *statIDs* have to be created first as ODVDoubleData resp. ODVIntData objects. *lats* must be sufficient in size to hold *nStats* `double` values for all the entries in *statIDs* .

Example:

```
   // Obtain collection inventory object (collection may be given here)
   ODVCollectionInventory colInv = collection.collectionInventory();
/* Make an array with the ID of the stations we want the
   latitudes from (here: all stations)                        */
ODVIntData statIDs = new ODVIntData(statCount);
for (int stat=0; stat<collection.statCount; stat++)
    statIDs.setitem(stat, stat);
/* Obtain the latitudes for the wanted stations.              */
ODVDoubleData lats = new ODVDoubleData(statCount);
colInv.latitudes(lats.cast(), statCount, statIDs.cast());
/* Use the obtained latitudes.
double latitude;                       */
for (stat=0; stat<collection.statCount; stat++)
```

latitude = lats.getitem(stat);

See also:

longitudes(), ODVDoubleData, ODVIntData

public double ODVCollectionInventory.longitude (int *statID*)

Returns:

The decimal longitude of station *statID* .

public boolean ODVCollectionInventory.longitudes (SWIGTYPE_p_double *lons*, int *nStats*, SWIGTYPE_p_int *statIDs*)

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

Returns:

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

Note:

The pointer objects for *lats* and *statIDs* have to be created first as ODVDoubleData resp. ODVIntData objects. *lons* must be sufficient in size to hold *nStats* `double` values for all the entries in *statIDs* . See latitudes() for an analogues example.

See also:

latitudes(), ODVDoubleData, ODVIntData

public ODVMapDomain ODVCollectionInventory.nativeMapDomain ()

Returns:

The native map domain covered by this collection.

public int ODVCollectionInventory.sampleCount (int *statID*)

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).

public int ODVCollectionInventory.sampleCount ()

This is an overloaded function.

Returns:

The total number of samples of all cruises in this ODVCollectionInventory.

public int ODVCollectionInventory.sampleCount (int *nStats*, SWIGTYPE_p_int *statIDs*)

This is an overloaded function.

Returns:

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

Note:

The pointer object for *statIDs* has to be created first as ODVIntData object. It has to be created and set as the statIDs in the example in latitudes().

See also:

latitudes(), ODVIntData

public SWIGTYPE_p_int ODVCollectionInventory.sampleCountData ()

Returns an array of 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* .

Note:

The returned pointer object can not be used directly. It has to be converted to an ODVIntData object which then can be used to retrieve the values. See cruiseIdData() for an analogues usage example.

See also:

cruiseIdData(), ODVIntData

public int ODVCollectionInventory.stationIDFromAccessionNumber (long *accNum*)

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.

See also:

accessionNumber()

public ODVCruiseInfo ODVCollectionInventory.summaryCruiseInfo ()

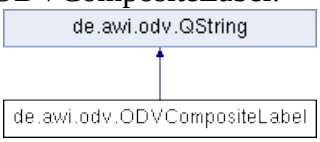
Returns:

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

de.awi.odv.ODVCompositeLabel Class Reference

ODVCompositeLabel is a QString which represents a variable label.

Inheritance diagram for de.awi.odv.ODVCompositeLabel:



Public Member Functions

- ODVCompositeLabel ()
- ODVCompositeLabel (QString other)
- ODVCompositeLabel (String str)
- ODVCompositeLabel (QByteArray ba)
- ODVCompositeLabel (QString nameLabel, QString unitsLabel)
- ODVCompositeLabel compositeLabel (ODVVariable var, int primaryVarID)
- ODVCompositeLabel compositeLabel (ODVVariable var)
- ODVCompositeLabel fullVariableLabel (QString nameLabel, QString unitsLabel)
- QString nameLabel (boolean allowSubstitution)
- QString nameLabel ()
- ODVCompositeLabel qfCompositeLabel (ODVQualityFlagSet qfSet)
- ODVQualityFlagSet.QFSetID qfSetID ()
- QString qualifierLabel ()
- QString unitLabel ()

Additional Inherited Members

Detailed Description

ODVCompositeLabel is a QString which represents a variable label.

This label consists of the variable's name, unit, quality flag and possibly further qualifiers regarding the type of the data values and if the variable is a meta or a primary variable.

A set of extended functions allows to deal comfortably with 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"* .

Constructor & Destructor Documentation

ODVCompositeLabel.ODVCompositeLabel ()

Creates an empty composite label.

ODVCompositeLabel.ODVCompositeLabel (QString *other*)

Constructs a copy of *other* .

The string is simply memorized and no checks are made if it is a valid composite label.

ODVCompositeLabel.ODVCompositeLabel (String *str*)

Constructs a composite label initialized with the Java string *str* . The characters in the given string are interpreted as 8-bit Latin-1 characters.

The string is simply memorized and no checks are made if it is a valid composite label.

ODVCompositeLabel.ODVCompositeLabel (QByteArray *ba*)

Constructs a composite label initialized with the byte array *ba* . The characters in the given string are interpreted as 8-bit Latin-1 characters. Stops copying at the first 0 character, otherwise copies the entire byte array.

The string is simply memorized and no checks are made if it is a valid composite label.

ODVCompositeLabel.ODVCompositeLabel (QString *nameLabel*, QString *unitsLabel*)

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

See also:

fullVariableLabel()

Member Function Documentation

ODVCompositeLabel ODVCompositeLabel.compositeLabel (ODVVariable *var*, int *primaryVarID*)

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

Appropriate qualifiers are appended to the label if it is a variable of known type and if the data type or the data byte length differs from the default values.

In case *var* is a primary variable the *primaryVarID* of the collection is needed to determine if it is necessary to append `":PRIMARYVAR"` to the label.

See also:

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

ODVCompositeLabel ODVCompositeLabel.compositeLabel (ODVVariable *var*)

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

Appropriate qualifiers are appended to the label if it is a variable of known type and if the data type or the data byte length differs from the default values.

No check for the necessity for a mark as primary variable is made.

See also:

compositeLabel(ODVVariable, int)

ODVCompositeLabel ODVCompositeLabel.fullVariableLabel (QString *nameLabel*, QString *unitsLabel*)

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

See also:

ODVVariable.fullLabel()

QString ODVCompositeLabel.nameLabel (boolean *allowSubstitution*)

Returns:

The name part from *compositeLabel* .

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

If *allowSubstitution* is `true` 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()

QString ODVCompositeLabel.nameLabel ()

Returns:

The name part from *compositeLabel* .

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

If 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:

nameLabel(boolean)

ODVCompositeLabel ODVCompositeLabel.qfCompositeLabel (ODVQualityFlagSet *qfSet*)

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

See also:

qfSetID(), compositeLabel()

ODVQualityFlagSet.QFSetID ODVCompositeLabel.qfSetID ()

Returns:

The Quality Flag schema ID from this composite label or ODVQualityFlagSet.QFSetID.ODV, if no valid schema name is found.

See also:

qfCompositeLabel()

QString ODVCompositeLabel.qualifierLabel ()

Returns:

The qualifier part from this composite label.

See also:

nameLabel(), unitLabel()

QString ODVCompositeLabel.unitLabel ()

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()

de.awi.odv.ODVCruiseInfo Class Reference

Holds summary information for one cruise of a collection.

Public Member Functions

- QString availabilityString ()
- int cruiseID ()
- int dataCount (int varID)
- QString dateString (boolean endDate, ODV.DateForm dateForm)
- QString dateString (boolean endDate)
- QString latitudeRangeString ()
- QString longitudeRangeString ()
- int maxGregorianDay ()
- int maxStationID ()
- int minGregorianDay ()
- int minStationID ()
- ODVMapDomain nativeMapDomain ()
- int sampleCount ()
- int stationCount ()
- int variableCount ()

Static Public Member Functions

- static QString getMissString ()

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.

Note:

There is no public constructor for this class because ODVCruiseInfo objects are only created by ODVCollectionInventory.

Member Function Documentation

public QString ODVCruiseInfo.availabilityString ()

Returns:

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

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. You may obtain the indicator for a certain variable with QString.at() for instance.

public int ODVCruiseInfo.cruiseID ()

Returns:

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

public int ODVCruiseInfo.dataCount (int *varID*)

Returns:

The number of non–miss samples, i.e. samples not having value ODV.getMissDOUBLE(), for basic variable ID *varID* in this cruise.

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

public QString ODVCruiseInfo.dateString (boolean *endDate*, ODV.DateForm *dateForm*)

Returns:

The start (*endDate* = `false`) or end (*endDate* = `true`) date of the cruise as QString.

The string format can be chosen via *dateForm* .

See also:

ODV.DateForm

public QString ODVCruiseInfo.dateString (boolean *endDate*)

This is an overloaded function.

Returns:

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

The string format is as in `"30 Sep 2007"` for example.

public static QString ODVCruiseInfo.getMissString ()`[static]`

Returns:

The string which is used for unavailable values. It is set to `"unavailable"` .

public QString ODVCruiseInfo.latitudeRangeString ()

Returns:

The latitudinal range of the cruise as QString.

public QString ODVCruiseInfo.longitudeRangeString ()

Returns:

The longitudinal range of the cruise as QString.

public int ODVCruiseInfo.maxGregorianDay ()

Returns:

The maximal gregorian day of this cruise.

See also:

minGregorianDay()

public int ODVCruiseInfo.maxStationID ()

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()

public int ODVCruiseInfo.minGregorianDay ()

Returns:

The minimal gregorian day of this cruise.

See also:

maxGregorianDay()

public int ODVCruiseInfo.minStationID ()

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()

public ODVMapDomain ODVCruiseInfo.nativeMapDomain ()

Returns:

The native map domain of this cruise.

public int ODVCruiseInfo.sampleCount ()

Returns:

The total number of samples in this cruise.

public int ODVCruiseInfo.stationCount ()

Returns:

The number of stations in this cruise.

public int ODVCruiseInfo.variableCount ()

Returns:

The number of basic variables for which counts are kept.

See also:

dataCount()

de.awi.odv.ODVDate Class Reference

This class provides date & time handling methods.

Static Public Member Functions

- static void dateFromDecimalYear (double decYear, short[] sYear, short[] sMonth, short[] sDay, short[] sHH, short[] sMM, double[] dSS)
- static void dateFromJulianDay (double julDay, int[] year, int[] month, int[] day, int[] hour, int[] min, double[] sec, boolean isChronological)
- static QString dateString (ODV.DateForm dateForm, short sYear, short sMonth, short sDay)
- static QString dateString (ODV.DateForm dateForm, double dYear, double dMonth, double dDay)
- static void daytimeFromFractionalDay (double fracDay, int[] hour, int[] min, double[] sec)
- static double decimalDay (short sHH, short sMM, double dSS)
- static double decimalDay (short sHH, short sMM)
- static double decimalYear (short sYear, short sMonth, short sDay, short sHH, short sMM, double dSS)
- static double decimalYear (short sYear, short sMonth, short sDay, short sHH, short sMM)
- static double decimalYearFromGregorianDay (int gregDay)
- static double decimalYearFromGregorianDay (double dGregDay)
- static int getDayOfYear (short sYear, short sMonth, short sDay)
- static int getDayOfYear (double decYear)
- static void gregorianDate (int days, short[] sYear, short[] sMonth, short[] sDay)
- static void gregorianDateInYear (int year, int dayOfYear, short[] sMonth, short[] sDay)
- static int gregorianDay (int year, int month, int day)
- static int gregorianDayOfWeek (int year, int month, int day)
- static int gregorianDayOfYear (int year, int month, int day)
- static int gregorianDaysInMonth (int year, int month)
- static int gregorianDaysInYear (int year)
- static boolean isGregorianLeapYear (int year)
- static QString isoDateFromGregorianDay (double gd)
- static int julianDay (int year, int month, int day)
- static QString timeString (double dHH, double dMM, double dSS)
- static boolean validateDate (short[] sYear, short[] sMonth, short[] sDay, short[] sHH, short[] sMM, double[] sec)
- static boolean validateDate (int[] year, int[] month, int[] day, int[] hour, int[] min, double[] sec)
- static boolean validateTime (short[] sHH, short[] sMM, double[] sec, short[] dayShift)

  Ensures that the specified time values are valid, and make modifications if necessary. All parameter arrays need to be of length 1 containing the respective values which are modified if necessary.
- static boolean validateTime (int[] hour, int[] min, double[] sec, int[] dayShift)

Detailed Description

This class provides date & time handling methods.

They allow to convert Gregorian days resp. years given as decimal values to dates & times and vice versa. Date & time strings can also be generated from given values.

Member Function Documentation

public static void ODVDate.dateFromDecimalYear (double *decYear*, short[] *sYear*, short[] *sMonth*, short[] *sDay*, short[] *sHH*, short[] *sMM*, double[] *dSS*)[static]

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 first and only element of the parameter arrays.

public static void ODVDate.dateFromJulianDay (double *julDay*, int[] *year*, int[] *month*, int[] *day*, int[] *hour*, int[] *min*, double[] *sec*, boolean *isChronological*)[static]

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` . 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 first and only element of the parameter arrays.

public static QString ODVDate.dateString (ODV.DateForm *dateForm*, short *sYear*, short *sMonth*, short *sDay*)[static]

Returns:

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

public static QString ODVDate.dateString (ODV.DateForm *dateForm*, double *dYear*, double *dMonth*, double *dDay*)[static]

Returns:

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

public static void ODVDate.daytimeFromFractionalDay (double *fracDay*, int[] *hour*, int[] *min*, double[] *sec*)[static]

Returns:

Daytime in the first and only element of arrays *hour* , *min* , and *sec* retrieved from fractional day *fracDay* . *fracDay* must be between 0 and 1.

public static double ODVDate.decimalDay (short *sHH*, short *sMM*, double *dSS*)[static]

Returns:

Daytime (in decimal days) for a given time.

public static double ODVDate.decimalDay (short *sHH*, short *sMM*)[static]

Returns:

Daytime (in decimal days) for a given time.

public static double ODVDate.decimalYear (short *sYear*, short *sMonth*, short *sDay*, short *sHH*, short *sMM*, double *dSS*)[static]

Returns:

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

public static double ODVDate.decimalYear (short *sYear*, short *sMonth*, short *sDay*, short *sHH*, short *sMM*)[static]

Returns:

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

public static double ODVDate.decimalYearFromGregorianDay (int *gregDay*)[static]

Returns:

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

public static double ODVDate.decimalYearFromGregorianDay (double *dGregDay*)[static]

Returns:

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

public static int ODVDate.getDayOfYear (short *sYear*, short *sMonth*, short *sDay*)[static]

Returns:

Gregorian day of the year for a Gregorian calendar date.

public static int ODVDate.getDayOfYear (double *decYear*)[static]

Returns:

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

public static void ODVDate.gregorianDate (int *days*, short[] *sYear*, short[] *sMonth*, short[] *sDay*)[static]

Given the Gregorian *days* calculates the associated date.

Returns:

The date in the first and only elements of *sYear* , *sMonth* , and *sDay* .

public static void ODVDate.gregorianDateInYear (int *year*, int *dayOfYear*, short[] *sMonth*, short[] *sDay*)[static]

Given the year *year* and the day in this year *dayOfYear* calculates the month *sMonth* and day *sDay* returned in their first and only elements.

public static int ODVDate.gregorianDay (int *year*, int *month*, int *day*)[static]

Returns:

The Gregorian days for a date on the Gregorian calender.

public static int ODVDate.gregorianDayOfWeek (int *year*, int *month*, int *day*)[static]

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

public static int ODVDate.gregorianDayOfYear (int *year*, int *month*, int *day*)[static]

Returns:

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

public static int ODVDate.gregorianDaysInMonth (int *year*, int *month*)[static]

Returns:

The last day of *month* for the Gregorian calendar.

public static int ODVDate.gregorianDaysInYear (int *year*)[static]

Returns:

The number of days in *year* .

public static boolean ODVDate.isGregorianLeapYear (int *year*)[static]

Returns:

`true` if *year* is a leap year according to the Gregorian calendar, or `false` otherwise.

public static QString ODVDate.isoDateFromGregorianDay (double *dGregDay*)[static]

Returns:

The ISO date/time string for fractional Gregorian day *dGregDay* .

public static int ODVDate.julianDay (int *year*, int *month*, int *day*)[static]

Returns:

Julian day for a given date on the Gregorian calender.

public static QString ODVDate.timeString (double *dHH*, double *dMM*, double *dSS*)[static]

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.

public static boolean ODVDate.validateDate (short[] *sYear*, short[] *sMonth*, short[] *sDay*, short[] *sHH*, short[] *sMM*, double[] *sec*)`[static]`

Ensures that the specified date values are valid, and make modifications if necessary. All parameter arrays are of length 1 containing the respective values which are modified if necessary.

Returns:

`true` if modifications were made and `false` otherwise.

public static boolean ODVDate.validateDate (int[] *year*, int[] *month*, int[] *day*, int[] *hour*, int[] *min*, double[] *sec*)`[static]`

Overloaded method.

See also:

validateDate(short[],short[],short[],short[],short[],double[])

public static boolean ODVDate.validateTime (short[] *sHH*, short[] *sMM*, double[] *sec*, short[] *dayShift*)`[static]`

Ensures that the specified time values are valid, and make modifications if necessary. All parameter arrays need to be of length 1 containing the respective values which are modified if necessary.

Returns:

The day shift arising from the modifications.

public static boolean ODVDate.validateTime (int[] *hour*, int[] *min*, double[] *sec*, int[] *dayShift*)`[static]`

Overloaded method.

See also:

validateTime(short[],short[],double[],short[])

de.awi.odv.ODVDateJNI Class Reference

de.awi.odv.ODVDoubleData Class Reference

Class for access to double value arrays.

Public Member Functions

- ODVDoubleData (int nelements)
- double getitem (int index)
- void setitem (int index, double value)
- SWIGTYPE_p_double cast ()

Static Public Member Functions

- static ODVDoubleData frompointer (SWIGTYPE_p_double t)

Detailed Description

Class for access to double value arrays.

Some functions make use of arrays of double data values. In the Java API they are of Java type `SWIGTYPE_p_double` . This can not be used directly, so it is necessary to convert it to an ODVDoubleData object with frompointer(). For access to the elements of the array use getitem().

Constructor & Destructor Documentation

ODVDoubleData.ODVDoubleData (int *nelements*)

Creates an ODVDoubleData array with *nelements* elements.

Member Function Documentation

SWIGTYPE_p_double ODVDoubleData.cast ()

Cast the ODVDoubleData object to a `SWIGTYPE_p_double` pointer and returns that.

static ODVDoubleData ODVDoubleData.frompointer (SWIGTYPE_p_double *t*)`[static]`

Takes a `SWIGTYPE_p_double` pointer *t* , casts it to an ODVDoubleData object and returns this object.

double ODVDoubleData.getitem (int *index*)

Returns:

The element at position *index* from the array.

double ODVDoubleData.setitem (int *index*, double *value*)

Set the element at position *index* to *value* .

de.awi.odv.ODVIntData Class Reference

Class for access to int value arrays.

Public Member Functions

- ODVIntData (int nelements)
- int getitem (int index)
- void setitem (int index, int value)
- SWIGTYPE_p_int cast ()

Static Public Member Functions

- static ODVIntData frompointer (SWIGTYPE_p_int t)

Detailed Description

Class for access to int value arrays.

Some functions in ODVCollectionInventory have array parameters or return values of Java type `SWIGTYPE_p_int` . This can not be used directly, so it is necessary to convert it to an ODVIntData object with frompointer(). For access to the elements of the array use getitem().

Constructor & Destructor Documentation

ODVIntData.ODVIntData (int *nelements*)

Creates an ODVIntData array with *nelements* elements.

Member Function Documentation

SWIGTYPE_p_int ODVIntData.cast ()

Cast the ODVIntData object to a `SWIGTYPE_p_int` pointer and returns that.

static ODVIntData ODVIntData.frompointer (SWIGTYPE_p_int *t*)`[static]`

Takes a `SWIGTYPE_p_int` pointer *t* , casts it to an ODVIntData object and returns this object.

int ODVIntData.getitem (int *index*)

Returns:

The element at position *index* from the array.

int ODVIntData.setitem (int *index*, int *value*)

Set the element at position *index* to *value* .

de.awi.odv.ODVLongData Class Reference

Class for access to long value arrays.

Public Member Functions

- ODVLongData (int nelements)
- long getitem (int index)
- void setitem (int index, long value)
- SWIGTYPE_p_unsigned_int cast ()

Static Public Member Functions

- static ODVLongData frompointer (SWIGTYPE_p_unsigned_int t)

Detailed Description

Class for access to long value arrays.

ODVCollectionInventory.accessionNumberData() returns the values in an array of Java type `SWIGTYPE_p_unsigned_int` . This can not be used directly, so it is necessary to convert it to an ODVLongData object with frompointer(). For access to the elements of the array use getitem().

Constructor & Destructor Documentation

ODVLongData.ODVLongData (int *nelements*)

Creates an ODVLongData array with *nelements* elements.

Member Function Documentation

SWIGTYPE_p_unsigned_int ODVLongData.cast ()

Cast the ODVLongData object to a `SWIGTYPE_p_unsigned_int` pointer and returns that.

static ODVLongData ODVLongData.frompointer (SWIGTYPE_p_unsigned_int *t*)[static]

Takes a `SWIGTYPE_p_unsigned_int` pointer *t* , casts it to an ODVLongData object and returns this object.

long ODVLongData.getitem (int *index*)

Returns:

The element at position *index* from the array.

long ODVLongData.setitem (int *index*, long *value*)

Set the element at position *index* to *value* .

de.awi.odv.ODVMapDomain Class Reference

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

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 (ODVMapDomain otherDomain)
  Creates a copy of ODVMapDomain otherDomain .
- void append (double lon, double lat)
  Appends point lon / lat to the domain.
- void append (int n, SWIGTYPE_p_double lon, SWIGTYPE_p_double lat)
  Appends n points in lon / lat to the domain.
- void append (int nLon, SWIGTYPE_p_double lon, int nLat, SWIGTYPE_p_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 (ODVMapDomain otherDomain)
  Appends ODVMapDomain otherDomain .
- double centerLatitude ()
- double centerLongitude ()
- void clear ()
- boolean domain (double[] lonmin, double[] lonmax, double[] latmin, double[] latmax, boolean autoEnlarge)
  Retrieves the ODVMapDomain's domain.
- double eastLongitude ()
- boolean intersects (ODVMapDomain otherDomain)
- boolean isEmpty ()
- boolean isInsideOf (ODVMapDomain otherDomain)
- double latitudeRange ()
- QString latitudeRangeString (int nDigits)
- QString latitudeRangeString ()
- double longitudeRange ()
- QString longitudeRangeString (int nDigits)
- QString longitudeRangeString ()
- double northLatitude ()
- void setDomain (double lonmin, double lonmax, double latmin, double latmax)
- double southLatitude ()
- double westLongitude ()
- ODVMapDomain odvmapdomain_assign (ODVMapDomain otherDomain)

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

public ODVMapDomain.ODVMapDomain ()

Creates an empty ODVMapDomain object.

public ODVMapDomain.ODVMapDomain (double *lonmin*, double *lonmax*, double *latmin*, double *latmax*)

Creates an ODVMapDomain object with specified longitude/latitude bounds.

public ODVMapDomain.ODVMapDomain (ODVMapDomain *otherDomain*)

Creates a copy of ODVMapDomain *otherDomain* .

Member Function Documentation

public 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.getMissDOUBLE().

public void ODVMapDomain.append (int *n*, SWIGTYPE_p_double *lon*, SWIGTYPE_p_double *lat*)

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

Points with either *lon* or *lat* equal to ODV.getMissDOUBLE() are not used.

Note:

The pointer objects for *lon* and *lat* have to be created first as ODVDoubleData objects and set with the appropriate values.

Example:

```
ODVMapDomain mapDomain = new ODVMapDomain();
/* Have 2 points to add. */
int n=2;
/* Create the longitudes and latitudes arrays. */
ODVDoubleData lons = new ODVDoubleData(n);
ODVDoubleData lats = new ODVDoubleData(n);
/* Set the points. */
lons.setitem(0) = 345.6;
lats.setitem(0) = 23.4;
lons.setitem(1) = -25;
lats.setitem(1) = -30.1;
/* Add the points to the domain. */
```

mapDomain.append(n, lons.cast(), lats.cast());

See also:

ODVDoubleData

public void ODVMapDomain.append (int *nLon*, SWIGTYPE_p_double *lon*, int *nLat*, SWIGTYPE_p_double *lat*)

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

Note:

The pointer objects for *lon* and *lat* have to be created first as ODVDoubleData objects and set with the appropriate values. See the previous append function for an analogues example.

See also:

append(int, SWIGTYPE_p_double, SWIGTYPE_p_double), ODVDoubleData

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

Appends the domain specified by the input parameters.

public void ODVMapDomain.append (ODVMapDomain *otherDomain*)

Appends ODVMapDomain *otherDomain* .

public double ODVMapDomain.centerLatitude ()

Returns:

The latitude of the map domain center.

public double ODVMapDomain.centerLongitude ()

Returns:

The longitude of the map domain center.

public void ODVMapDomain.clear ()

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

public boolean ODVMapDomain.domain (double[] *lonmin*, double[] *lonmax*, double[] *latmin*, double[] *latmax*, boolean *autoEnlarge*)

Retrieves the ODVMapDomain's domain.

Returns:

`true` if successful and `false` if the object is empty. The domain boundaries are returned in the first and only element of the array parameters. That means the parameters must be arrays with a size of 1.

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.

public double ODVMapDomain.eastLongitude ()

Returns:

The eastern border longitude of the map domain.

public boolean ODVMapDomain.intersects (ODVMapDomain *otherDomain*)

Returns:

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

public boolean ODVMapDomain.isEmpty ()

Returns:
`true` if this ODVMapDomain is empty.

public boolean ODVMapDomain.isInsideOf (ODVMapDomain *otherDomain*)

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

public double ODVMapDomain.latitudeRange ()

Returns:
The latitudinal range of the map domain.

public QString ODVMapDomain.latitudeRangeString (int *nDigits*)

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

public QString ODVMapDomain.latitudeRangeString ()

Returns:
The latitudinal range as a QString.
The latitude values are rounded to 1 significant digit before producing the text representation.

public double ODVMapDomain.longitudeRange ()

Returns:
The longitudinal range of the map domain.

public QString ODVMapDomain.longitudeRangeString (int *nDigits*)

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

public QString ODVMapDomain.longitudeRangeString ()

Returns:
The longitudinal range as a QString.
The longitude values are rounded to 1 significant digit before producing the text representation.

public double ODVMapDomain.northLatitude ()

Returns:
The northern border latitude of the map domain.

public ODVMapDomain ODVMapDomain.odvmapdomain_assign (ODVMapDomain *otherDomain*)

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

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

Sets the domain to the specified longitude/latitude bounds.

public double ODVMapDomain.southLatitude ()

Returns:

The southern border latitude of the map domain.

public double ODVMapDomain.westLongitude ()

Returns:

The western border longitude of the map domain.

de.awi.odv.ODVQualityFlagSet Class Reference

Represents an ODV Quality Flag Schema.

Classes

- enum QFSetID

List of available quality flag set identifiers for quality flag sets. Public Member Functions

- ODVQualityFlagSet (ODVQualityFlagSet.QFSetID setID)
- ODVQualityFlagSet ()
- ODVQualityFlagSet (ODVQualityFlagSet other)
- char badQfVal ()
- char defaultQfVal ()
- QStringList descriptions ()
- char genericQfVal (char qfVal)
- char goodQfVal ()
- int indexOf (char qfVal)
- char mapTo (char qfVal, ODVQualityFlagSet.QFSetID targetSetID)
- char mapTo (char qfVal, ODVQualityFlagSet targetSet)
  Maps quality flag qfVal to corresponding quality flag of ODVQualityFlagSet targetSet .
- char missQfVal ()
- QString qfString (int i)
- QString qfString (char qfVal)
- char qfVal (int i)
- int safeIndexOf (char qfVal)
- ODVQualityFlagSet.QFSetID setID ()
- int size ()
- QString text ()
- QStringList values ()
- ODVQualityFlagSet odvqualityflagset_assign (ODVQualityFlagSet.QFSetID setID)
- ODVQualityFlagSet odvqualityflagset_assign (ODVQualityFlagSet other)
- boolean odvqualityflagset_compare (ODVQualityFlagSet.QFSetID setID)
- boolean odvqualityflagset_compare (ODVQualityFlagSet other)

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.QFSetID.ODV quality flag set.

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

Constructor & Destructor Documentation

ODVQualityFlagSet.ODVQualityFlagSet (ODVQualityFlagSet.QFSetID *setID*)

Creates a ODVQualityFlagSet of type *setID* .

ODVQualityFlagSet.ODVQualityFlagSet ()

Creates a ODVQualityFlagSet of type ODVQualityFlagSet.QFSetID.ODV.

ODVQualityFlagSet.ODVQualityFlagSet (ODVQualityFlagSet *other*)

Creates a ODVQualityFlagSet from *other* one.

Member Function Documentation

public char ODVQualityFlagSet.badQfVal ()

Returns:

The bad quality flag of this set.

char ODVQualityFlagSet.defaultQfVal ()

Returns:

The default quality flag for collection variables of this set.

QStringList ODVQualityFlagSet.descriptions ()

Returns:

The list of flag descriptions.

char ODVQualityFlagSet.genericQfVal (char *qfVal*)

Returns:

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

public char ODVQualityFlagSet.goodQfVal ()

Returns:

The good quality flag of this set.

int ODVQualityFlagSet.indexOf (char *qfVal*)

Returns:

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

char ODVQualityFlagSet.mapTo (char *qfVal*, ODVQualityFlagSet.QFSetID *targetSetID*)

Maps quality flag *qfVal* to corresponding quality flag of ODVQualityFlagSet whose id is *targetSetID* .

If *qfVal* is an invalid member of the quality flag set, the default quality flag of the set is mapped.

char ODVQualityFlagSet.mapTo (char *qfVal*, ODVQualityFlagSet *targetSet*)

Maps quality flag *qfVal* to corresponding quality flag of ODVQualityFlagSet *targetSet* .

If *qfVal* is an invalid member of the quality flag set, the default quality flag of the set is mapped.

char ODVQualityFlagSet.missQfVal ()

Returns:
The quality flag for missing values.

ODVQualityFlagSet ODVQualityFlagSet.odvqualityflagset_assign (ODVQualityFlagSet.QFSetID *setID*)
Assignment operator: Sets ODVQualityFlagSet to type *setID* .

ODVQualityFlagSet ODVQualityFlagSet.odvqualityflagset_assign (ODVQualityFlagSet *other*)
Assignment operator: Sets ODVQualityFlagSet to type of *other* .

boolean ODVQualityFlagSet.odvqualityflagset_compare (ODVQualityFlagSet.QFSetID *setID*)

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

boolean ODVQualityFlagSet.odvqualityflagset_compare (ODVQualityFlagSet *other*)

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*)

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

QString ODVQualityFlagSet.qfString (char *qfVal*)

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

char ODVQualityFlagSet.qfVal (int *i*)

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*)

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 ()

Returns:
The quality flag set ID.

int ODVQualityFlagSet.size ()

Returns:
The number of flags in set.

QString ODVQualityFlagSet.text ()

Returns:
The title resp. description of the quality flag set.

QStringList ODVQualityFlagSet.values ()

Returns:
The list of flag values.

de.awi.odv.ODVShortData Class Reference

Class for access to short value arrays.

Public Member Functions

- ODVShortData (int nelements)
- short getitem (int index)
- void setitem (int index, short value)
- SWIGTYPE_p_short cast ()

Static Public Member Functions

- static ODVShortData frompointer (SWIGTYPE_p_short t)

Detailed Description

Class for access to short value arrays.

ODVCollectionInventory.dayTimeData() returns the data values in an array of Java type `SWIGTYPE_p_short` . This can not be used directly, so it is necessary to convert it to an ODVShortData object with frompointer(). For access to the elements of the array use getitem().

Constructor & Destructor Documentation

ODVShortData.ODVShortData (int *nelements*)

Creates an ODVShortData array with *nelements* elements.

Member Function Documentation

SWIGTYPE_p_short ODVShortData.cast ()

Cast the ODVShortData object to a `SWIGTYPE_p_short` pointer and returns that.

static ODVShortData ODVShortData.frompointer (SWIGTYPE_p_short *t*)`[static]`

Takes a `SWIGTYPE_p_short` pointer *t* , casts it to an ODVShortData object and returns this object.

short ODVShortData.getitem (int *index*)

Returns:

The element at position *index* from the array.

short ODVShortData.setitem (int *index*, short *value*)

Set the element at position *index* to *value* .

de.awi.odv.ODVStation Class Reference

Class for maintaining metadata and data of one station.

Classes

- enum MetaVarIndex

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

- ODVStation (ODVCollection col)
- long accessionNumber ()
  Retrieves the accession number of the station.
- void clear ()
- boolean containsDataErrors (int varID)
- boolean containsDataInfos (int varID)
- SWIGTYPE_p_double data (ODVVariable var)
- int dataCount (int varID)
- long dataTotalByteSize ()
- SWIGTYPE_p_double errorData (ODVVariable var)
- double errorValue (ODVVariable var, int sampleID)
- QString errorStringValue (ODVVariable var, int sampleID, int decimalCount)
- QString errorStringValue (ODVVariable var, int sampleID)
- QStringList historyStrings ()
  Retrieves all history strings for this station.
- QString identifierHeaderString ()
- QString identifierString ()
- QString infoStringValue (ODVVariable var, int sampleID)
- QStringList infoStringValues (ODVVariable var)
- double metaDecimalDay ()
- QString metaFullName ()
- double metaLatitude ()
- double metaLongitude ()
- QString metaName ()
- QString metaStringDate (ODV.DateForm dateForm)
- QString metaStringDate ()
- QString metaStringIsoDateTime ()
- QString metaStringPosition (int decimalCount)
- QString metaStringPosition ()
- QString metaStringPrimVarRange ()
- QString metaStringStatType ()
- QString metaStringTime ()
- QString metaStringValue (ODVStation.MetaVarIndex mvIndex)
- QString metaStringValue (ODVVariable.VarType metaVarType)
- QString metaStringValue (ODVVariable var)
- double metaValue (ODVVariable var)
- double metaValue (ODVVariable.VarType metaVarType)
- double metaValue (ODVStation.MetaVarIndex mvIndex)
- String qfData (ODVVariable var)
- ODV.Status readData (int statID)
- ODV.Status readMetaData (int statID)
- int sampleCount ()

- int stationID ()
- int stationLabelToInt (int dfltVal)
- int stationLabelToInt ()
- QString stringValue (ODVVariable var, int sampleID)
- String textData (ODVVariable var)
- String textValue (ODVVariable var, int sampleID)
- String textValue (ODVVariable var)
- double value (ODVVariable var, int sampleID)

Static Public Member Functions

- static String stationTypeFromSampleCount (int sampleCount)

Detailed Description

Class for maintaining metadata and data of one station.

The data of the station include values of all collection variables for all samples of the station as well as the metadata of the station.

Reading the metadata and/or the data of one station is achieved with the functions readMetaData() and readData(). These functions transfer the station's raw data from disk.

The metadata of the station can be accessed via a large number of meta<...>() functions. The data of a collection variable are accessed via value(ODVVariable, int) and stringValue(ODVVariable, int).

Constructor & Destructor Documentation

ODVStation.ODVStation (ODVCollection *col*)

Creates a new ODVStation object for collection *col* .

Member Function Documentation

public long ODVStation.accessionNumber ()

Retrieves the accession number of the station.

Returns:

The accession number of the station, or the value returned by ODV.getMissUINT32() if the station does not have one.

void ODVStation.clear ()

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

public boolean ODVStation.containsDataErrors (int *varID*)

Returns:

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

public boolean ODVStation.containsDataInfos (int *varID*)

Returns:

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

SWIGTYPE_p_double ODVStation.data (ODVVariable *var*)

Returns:

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

Note:

The returned pointer object can not be used directly. It has to be converted to a ODVDoubleData object which then can be used to retrieve the values.

Example:

```
double value = 0.0;
// Create station object (collection may be given here)
ODVStation station = new ODVStation(collection);
// Read data for first station
station.readData(0);
// Get the variable with ID 0
ODVVariable var = collection.var(0);
// Get data from station for variable and convert to ODVDoubleData
ODVDoubleData dataVals = ODVDoubleData.frompointer(station.data(var));
// Check if we got valid data pointer
if (dataVals == null)
  // no data :-(  -> set to miss value
  value = ODV.getMissDOUBLE();
else
  // Retrieve data value of first sample
  value = dataVals.getitem(0);
```

See also:

value(ODVVariable, int), ODVDoubleData

int ODVStation.dataCount (int *varID*)

Returns:

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

public long ODVStation.dataTotalByteSize ()

Returns:

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

public SWIGTYPE_p_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, non-numeric, or there are no error values.

Note:

The returned pointer object can not be used directly. It has to be converted to a ODVDoubleData object which then can be used to retrieve the values. For an example see data(ODVVariable).

public QString ODVStation.errorStringValue (ODVVariable *var*, int *sampleID*, int *decimalCount*)

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.

public QString ODVStation.errorStringValue (ODVVariable *var*, int *sampleID*)

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 the variable's significant digit count value (see ODVVariable::decimalCount()) before producing the text representation.

public double ODVStation.errorValue (ODVVariable *var*, int *sampleID*)

Returns:

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

The value of ODV.getMissDOUBLE() is returned if no error value exists for this sample or the sample ID is out of range.

Warning:

Always returns ODV.getMissDOUBLE() for meta variables.

public QStringList ODVStation.historyStrings ()

Retrieves all history strings for this station.

Returns:

The list of strings.

QString ODVStation.identifierHeaderString ()

Returns:

The description of the station identifier string contents.

See also:

metaFullName(), identifierString()

QString ODVStation.identifierString ()

Returns:

A string that best identifies the station.

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

See also:

metaFullName(), identifierHeaderString()

public 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.

public 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 ()

Returns:

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

QString ODVStation.metaFullName ()

Returns:

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

See also:

metaName()

double ODVStation.metaLatitude ()

Returns:

The latitude of the station.

double ODVStation.metaLongitude ()

Returns:

The longitude of the station.

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*)

Returns:

The metadata date in *dateForm* format.

QString ODVStation.metaStringDate ()

Returns:

The metadata date in format ODV.DateForm.mmmddyyyyDate.

QString ODVStation.metaStringIsoDateTime ()

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*)

Returns:

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

lon and *lat* are rounded to *decimalCount* digits if supplied and greater than or equal to zero, or to the decimal count property of the respective variable otherwise.

QString ODVStation.metaStringPosition ()

Returns:

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

This is an overloaded function.

lon and *lat* are rounded to the decimal count property of the respective variable.

See also:

metaStringPosition(int)

QString ODVStation.metaStringPrimVarRange ()

Returns:

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

QString ODVStation.metaStringStatType ()

Returns:

The station type as string.

QString ODVStation.metaStringTime ()

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.

QString ODVStation.metaStringValue (ODVStation.MetaVarIndex *mvIndex*)

Returns:

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

Using MetaVarIndex is the most efficent way of access but it is restricted to the mandatory meta variables.

If the meta variable is numeric it is rounded to ODVVariable.decimalCount() significant digits before producing the text representation.

QString ODVStation.metaStringValue (ODVVariable.VarType *metaVarType*)

Returns:

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

If the meta variable is numeric it is rounded to ODVVariable.decimalCount() significant digits before producing the text representation.

Warning:

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

QString ODVStation.metaStringValue (ODVVariable *var*)

Returns:

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

If the meta variable is numeric it is rounded to ODVVariable.decimalCount() significant digits before producing the text representation.

double ODVStation.metaValue (ODVVariable *var*)

Returns:

The numeric value of meta variable *var* for the current station, or the value returned by ODV.getMissDOUBLE() if the variable does not exist, the value is missing or the variable is a text variable.

double ODVStation.metaValue (ODVVariable.VarType *metaVarType*)

Returns:

The numeric value of meta variable of type *metaVarType* for the current station, or the value returned by ODV.getMissDOUBLE() if the variable does not exist, the value is missing or the variable is a text variable.

Warning:

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

double ODVStation.metaValue (ODVStation.MetaVarIndex *mvIndex*)

Returns:

The numeric value of meta variable with index *mvIndex* for this station, or the value returned by ODV.getMissDOUBLE() if the value is missing or the variable is a text variable.

Using MetaVarIndex is the most efficent way of access but it is restricted to the mandatory meta variables.

String ODVStation.qfData (ODVVariable *var*)

Returns:

A string with the quality flags of variable *var* , or `null` if no quality flags exist.

Note:

This function must not be called for derived meta variables, i.e. ones with type ODVVariable.VarType.METATIME and ODVVariable.VarType.METADAYOFYEAR.

The single characters of the string contain the quality flags for the samples of the station for the requested variable.

Example:

```
char qflag;
// Create station object (collection may be given here)
ODVStation station = new ODVStation(collection);
// Read data for first station
station.readData(0);
// Get the variable with ID 0
ODVVariable var = collection.var(0);
// Get the quality flags for that variable and this station
String qFlags = station.qfData(var);
// Check if there are quality flags
if (qFlags != null)
   // Yes, we have! Get quality flag for first sample
   qflag = qFlags.charAt(0));
else
  // No flags :-( -> use default
   qflag = var.defaultQfVal();
```

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

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

This function must be called before any meta or data variable values or quality flags can be accessed for the station with this *statID* .

Returns:

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

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

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

Returns:

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

int ODVStation.sampleCount ()

Returns:

Number of samples for this station.

int ODVStation.stationID ()

Returns:

The 0-based ID of 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*)

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

Returns:

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

int ODVStation.stationLabelToInt ()

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

Returns:

The station number, if successful, or the value returned by ODV.getMissINT32() otherwise.

static String ODVStation.stationTypeFromSampleCount (int *sampleCount*)`[static]`

Returns:

Station type depending on a station's *sampleCount* .

Up to a sample count of 250 stations `"B"` is returned and `"C"` otherwise.

QString ODVStation.stringValue (ODVVariable *var*, int *sampleID*)

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.

If the variable is numeric it is rounded to ODVVariable.decimalCount() significant digits before producing the text representation.

Warning:

Always returns an empty string for numeric meta variables.

String ODVStation.textData (ODVVariable *var*)

Returns:

The text data of variable *var* , or `null` if the variable has no text data.

String ODVStation.textValue (ODVVariable *var*, int *sampleID*)

Returns:

The text value of variable *var* for sample *sampleID* (0-based index) of this station or `null` , if this is a numeric variable or *sampleID* is out of range.

String ODVStation.textValue (ODVVariable *var*)

Returns:

The text value of variable *var* for first sample of this station or `null`, if this is a numeric variable.

This is an overloaded function.

See also:

textValue(ODVVariable, int)

double ODVStation.value (ODVVariable *var*, int *sampleID*)

Returns:

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

ODV.getMissDOUBLE() is returned if no value exists for this sample or the *sampleID* is out of range.

Warning:

Always returns ODV.getMissDOUBLE() for meta variables.

See also:

data(ODVVariable)

de.awi.odv.ODVVariable Class Reference

Represents a collection variable.

Classes

- enum ValueType
- *The enumeration values represent the type of the variable's values.* enum VarType

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

- ODVVariable (QString varName, QString varUnits, ODVVariable.VarType varType, int decimals, ODVVariable.ValueType valType, int valBytes, ODVQualityFlagSet.QFSetID qfSetID)
- ODVVariable (QString varName, QString varUnits, ODVVariable.VarType varType, int decimals, ODVVariable.ValueType valType, int valBytes)
- ODVVariable (QString varName, QString varUnits, ODVVariable.VarType varType, int decimals, ODVVariable.ValueType valType)
- ODVVariable (QString varName, QString varUnits, ODVVariable.VarType varType, int decimals)
- ODVVariable (QString varName, QString varUnits, ODVVariable.VarType varType)
- ODVVariable (QString varName)
- ODVVariable ()
- ODVVariable (ODVVariable var)
- char badQfVal ()
- QString commentLabel ()
- boolean compareFullName (QString otherName, QString otherUnit, Qt_casesensitivity cs)
  Compares variable's name and unit string with otherName and otherUnit .
- boolean compareFullName (QString otherName, QString otherUnit)
  Compares variable's name and unit string with otherName and otherUnit . The comparison is case-insensitive.
- boolean compareName (QString otherName, Qt_casesensitivity cs)
  Compares variable's name string with otherName .
- boolean compareName (QString otherName)
  Compares variable's name string with otherName . The comparison is case-insensitive.
- boolean compareUnit (QString otherUnit, Qt_casesensitivity cs)
  Compares variable's unit string with otherUnit .
- boolean compareUnit (QString otherUnit)
  Compares variable's unit string with otherUnit . The comparison is case-insensitive.
- ODVVariable createClone ()
- int decimalCount ()
- char defaultQfVal ()
- char defaultQfVal (double dVal)
- int errorVarID ()
- QString fullLabel (boolean doClean, boolean prependID)
  Returns the full label of the variable, consisting of variable name and units.
- QString fullLabel (boolean doClean)
  Returns the full label of the variable, consisting of variable name and units.
- QString fullLabel ()
  Returns the full label of the variable, consisting of variable name and units.
- char goodQfVal ()
- boolean isLatitude ()
- boolean isLongitude ()

- boolean isMandatoryMetaVar ()
- boolean isMeta ()
- boolean isNumeric ()
- double maxVal ()
- double minVal ()
- char missQfVal ()
- QString nameLabel (boolean doClean)
- QString nameLabel ()
- int nativeDecimalCount ()
- char qfGenericValue (char qfVal)
- ODVQualityFlagSet qfSet ()
- void range (double[] min, double[] max)
- void setComment (QString cmt)
- void setDecimalCount (int decimals)
- void setErrorVarID (int errID)
- 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 (QString nameLabel)
- void setProperties (ODVVariable var)
- void setQFSet (ODVQualityFlagSet.QFSetID qfSetID)
- void setRange (double min, double max)
  Sets the current value range for this variable to [min , max].
- void setType (ODVVariable.VarType type)
  Sets the variable type of this variable to type .
- void setUnits (QString unitLabel)
- void setValueType (ODVVariable.ValueType vt, long byteLengths)
- void setValueType (ODVVariable.ValueType vt)
- void setVarID (int varID)
- QString stringValue (double d, int decimals)
- ODVVariable.VarType type ()
- QString unitLabel (boolean doClean)
- QString unitLabel ()
- void updateRange (double min, double max, boolean doAutoScale)
- void updateRange (double min, double max)
- long valueByteSize ()
- ODVVariable.ValueType valueType ()
- int varID ()

Static Public Member Functions

- static long valueByteSize (ODVVariable.ValueType valueType, long nBytes)
- static long valueByteSize (ODVVariable.ValueType valueType)

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.

Constructor & Destructor Documentation

public ODVVariable.ODVVariable (QString *varName*, QString *varUnits*, ODVVariable.VarType *varType*, int *decimals*, ODVVariable.ValueType *valType*, int *valBytes*, ODVQualityFlagSet.QFSetID *qfSetID*)

Constructs an ODVVariable object with the given parameter properties. *decimals* is the digit count for fractions of a number which shall be shown to the user. The saved precision of data values is not influenced by this value! *valBytes* are only relevant for ODVVariable.ValueType.TEXT and specifies the amount of text which can be stored. For all other types the system determines the size. All other parameters should be self-explanatory.

public ODVVariable.ODVVariable (QString *varName*, QString *varUnits*, ODVVariable.VarType *varType*, int *decimals*, ODVVariable.ValueType *valType*, int *valBytes*)

Overloaded constructor. qfSetID will be set to ODVQualityFlagSet.QFSetID.ODV.

public ODVVariable.ODVVariable (QString *varName*, QString *varUnits*, ODVVariable.VarType *varType*, int *decimals*, ODVVariable.ValueType *valType*)

Overloaded constructor. *qfSetID* will be set to ODVQualityFlagSet.QFSetID.ODV. *valBytes* will be set to 4 but the above comment about its relevance is still valid.

public ODVVariable.ODVVariable (QString *varName*, QString *varUnits*, ODVVariable.VarType *varType*, int *decimals*)

Overloaded constructor. *qfSetID* will be set to ODVQualityFlagSet.QFSetID.ODV. *valBytes* will be set to 4 . *valType* will be set to ODVVariable.ValueType.FLOAT.

public ODVVariable.ODVVariable (QString *varName*, QString *varUnits*, ODVVariable.VarType *varType*)

Overloaded constructor. *qfSetID* will be set to ODVQualityFlagSet.QFSetID.ODV. *valBytes* will be set to 4 . *valType* will be set to ODVVariable.ValueType.FLOAT. *decimals* , the digit count for fractions of a number, will be set to 2 .

public ODVVariable.ODVVariable (QString *varName*)

Overloaded constructor. *qfSetID* will be set to ODVQualityFlagSet.QFSetID.ODV. *valBytes* will be set to 4 . *valType* will be set to ODVVariable.ValueType.FLOAT. *decimals* , the digit count for fractions of a number, will be set to 2 . *varType* will be set to ODVVariable.VarType.BASIC. *varUnits* will be an empty string.

public ODVVariable.ODVVariable ()

Overloaded constructor. *qfSetID* will be set to ODVQualityFlagSet.QFSetID.ODV. *valBytes* will be set to 4 . *valType* will be set to ODVVariable.ValueType.FLOAT. *decimals* , the digit count for fractions of a number, will be set to 2 . *varType* will be set to ODVVariable.VarType.BASIC. *varUnits* and *varName* will be an empty string.

public ODVVariable.ODVVariable (ODVVariable *var*)

Copy constructor. Builds the variable with the same properties as in *var* .

Member Function Documentation

public char ODVVariable.badQfVal ()

Returns:
The bad quality flag value of this variable.

public QString ODVVariable.commentLabel ()

Returns:
The comment associated with the variable.

public boolean ODVVariable.compareFullName (QString *otherName*, QString *otherUnit*, Qt_casesensitivity *cs*)

Compares variable's name and unit string with *otherName* and *otherUnit* .

Note:
The raw name and unit strings are used (ODVVariable.nameLabel(false), ODVVariable.unitLabel(false)).

Parameters:

in	*otherName*	name of other variable to compare
in	*otherUnit*	unit of other variable to compare
in	*cs*	case-sensitivity of the comparison

Returns:
`true` if strings are equal; otherwise returns `false` .

See also:
compareName(), compareUnit()

public boolean ODVVariable.compareFullName (QString *otherName*, QString *otherUnit*)

Compares variable's name and unit string with *otherName* and *otherUnit* . The comparison is case-insensitive.

Note:
The raw name and unit strings are used (ODVVariable.nameLabel(false), ODVVariable.unitLabel(false)).

Parameters:

in	*otherName*	name of other variable to compare
in	*otherUnit*	unit of other variable to compare

Returns:
`true` if strings are equal; otherwise returns `false` .

See also:
compareName(), compareUnit()

public boolean ODVVariable.compareName (QString *otherName*, Qt_casesensitivity *cs*)

Compares variable's name string with *otherName* .

Note:

The raw name string is used (ODVVariable.nameLabel(false)).

Parameters:

in	*otherName*	name of other variable to compare
in	*cs*	case-sensitivity of the comparison

Returns:

`true` if string *otherName* is equal to this name string; otherwise returns `false` .

See also:

compareFullName(), compareUnit()

public boolean ODVVariable.compareName (QString *otherName*)

Compares variable's name string with *otherName* . The comparison is case-insensitive.

Note:

The raw name string is used (ODVVariable.nameLabel(false)).

Parameters:

in	*otherName*	name of other variable to compare

Returns:

`true` if string *otherName* is equal to this name string; otherwise returns `false` .

See also:

compareFullName(), compareUnit()

public boolean ODVVariable.compareUnit (QString *otherUnit*, Qt::CaseSensitivity *cs*)

Compares variable's unit string with *otherUnit* .

Note:

The raw unit string is used (ODVVariable.unitLabel(false)).

Parameters:

in	*otherUnit*	unit of other variable to compare
in	*cs*	case-sensitivity of the comparison

Returns:

`true` if string *otherUnit* is equal to this unit string; otherwise returns `false` .

See also:

compareFullName(), compareName()

public boolean ODVVariable.compareUnit (QString *otherUnit*)

Compares variable's unit string with *otherUnit* . The comparison is case-insensitive.

Note:

The raw unit string is used (ODVVariable.unitLabel(false)).

Parameters:

in	*otherUnit*	unit of other variable to compare

Returns:

`true` if string *otherUnit* is equal to this unit string; otherwise returns `false` .

See also:

compareFullName(), compareName()

public ODVVariable ODVVariable.createClone ()

Returns:

A new ODVVariable object with the same properties, i.e. it is cloned from this object.

public int ODVVariable.decimalCount ()

Returns:

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

public char ODVVariable.defaultQfVal ()

Returns:

The default quality flag value of this variable.

See also:

badQfVal(), goodQfVal(), missQfVal()

public char ODVVariable.defaultQfVal (double *dVal*)

Returns:

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

See also:

badQfVal(), goodQfVal(), missQfVal()

public int ODVVariable.errorVarID ()

Returns:

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

public QString ODVVariable.fullLabel (boolean *doClean*, boolean *prependID*)

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()

public QString ODVVariable.fullLabel (boolean *doClean*)

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

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

See also:

nameLabel(), unitLabel()

public QString ODVVariable.fullLabel ()

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

~-style control sequences are not removed.

See also:

nameLabel(), unitLabel()

public char ODVVariable.goodQfVal ()

Returns:

The good quality flag value of this variable.

See also:

badQfVal(), defaultQfVal(), missQfVal()

public boolean ODVVariable.isLatitude ()

Returns:

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

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

public boolean ODVVariable.isLongitude ()

Returns:

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

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

public boolean ODVVariable.isMandatoryMetaVar ()

Returns:

`true` if it is a mandatory variable and `false` otherwise.

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

public boolean ODVVariable.isMeta ()

Returns:

`true` if variable is a meta variable.

public boolean ODVVariable.isNumeric ()

Returns:

`true` if variable's data type is ValueType.FLOAT, ValueType.DOUBLE, ValueType.BYTE, ValueType.SHORT or ValueType.INTEGER.

public double ODVVariable.maxVal ()

Returns:
The current maximal value for this variable.

See also:
setMaxVal(), minVal()

public double ODVVariable.minVal ()

Returns:
The current minimal value for this variable.

See also:
setMinVal(), maxVal()

public char ODVVariable.missQfVal ()

Returns:
The missing value quality flag value of this variable.

Note:
Some quality flag schemes such as ODVQualityFlagSet.QFSetID.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()

public QString ODVVariable.nameLabel (boolean *doClean*)

Returns:
The label of the variable without units.
~-style control sequences are removed, if *doClean* is `true` .

See also:
fullLabel(), unitLabel()

public QString ODVVariable.nameLabel ()

Returns:
The label of the variable without units.

See also:
fullLabel(), unitLabel()

public int ODVVariable.nativeDecimalCount ()

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

See also:
decimalCount()

public char ODVVariable.qfGenericValue (char *qfVal*)

Returns:

The corresponding ODVQualityFlagSet.QFSetID.ODV quality flag value for quality flag *qfVal* of variable's quality flag set.

public ODVQualityFlagSet ODVVariable.qfSet ()

Returns:

The variable's quality flag set.

public void ODVVariable.range (double[] *min*, double[] *max*)

Returns:

The current range of the variable values in the first and only element of arrays *min* & *max* .

public void ODVVariable.setComment (QString *cmt*)

Set the comment *cmt* for the variable.

public void ODVVariable.setDecimalCount (int *decimals*)

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

public void ODVVariable.setErrorVarID (int *i*)

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

public void ODVVariable.setMaxVal (double *max*)

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

See also:

setRange()

public void ODVVariable.setMinVal (double *min*)

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

See also:

setRange()

void ODVVariable.setName (QString *nameLabel*)

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

public void ODVVariable.setProperties (ODVVariable *var*)

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

public void ODVVariable.setQFSet (ODVQualityFlagSet.QFSetID *qfSetID*)

Sets the quality flag schema of this variable to the one with ID *qfSetID* .

public void ODVVariable.setRange (double *min*, double *max*)

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

See also:
setMinVal(), setMaxVal()

public void ODVVariable.setType (ODVVariable.VarType *type*)

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

Warning:
It is potentially risky to change the type of the variable!

public void ODVVariable.setUnits (QString *unitLabel*)

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

public void ODVVariable.setValueType (ODVVariable.ValueType *vt*, long *byteLengths*)

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

byteLengths is only used for ValueType.TEXT. In this case add 1 Byte for the terminating zero character.

public void ODVVariable.setValueType (ODVVariable.ValueType *vt*)

Change the value type of the variable to *vt* . For ValueType.TEXT the size is `21` .

public void ODVVariable.setVarID (int *varID*)

Sets the 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()

public QString ODVVariable.stringValue (double *d*, int *nDecimals*)

Returns:
The text representation of the value *d* or an empty string, if *d* equals to `ODV::getMissDOUBLE()` .

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

public ODVVariable.VarType ODVVariable.type ()

Returns:
The type of the variable.

public QString ODVVariable.unitLabel (boolean *doClean*)

Returns:

The units of the variable.

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

See also:

fullLabel(), nameLabel()

public QString ODVVariable.unitLabel ()

Returns:

The units of the variable.

~-style control sequences are not removed.

See also:

fullLabel(), nameLabel()

public void ODVVariable.updateRange (double *min*, double *max*, boolean *doAutoScale*)

Updates the range of the variable to encompass the [*min* , *max*] range. Autoscales the range if *doAutoScale* is true.

public void ODVVariable.updateRange (double *min*, double *max*)

Updates the range of the variable to encompass the [*min* , *max*] range. Autoscales the range.

public static long ODVVariable.valueByteSize (ODVVariable.ValueType *valueType*, long *nBytes*)`[static]`

Returns:

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

public static long ODVVariable.valueByteSize (ODVVariable.ValueType *valueType*)`[static]`

Overloaded function with *nBytes* set to `21` .

public long ODVVariable.valueByteSize ()

Returns:

The number of bytes per raw value of this variable.

public ODVVariable.ValueType ODVVariable.valueType ()

Returns:

The value type of the variable.

public int ODVVariable.varID ()

Returns:

The ID of the variable.

See also:

setVarID()

de.awi.odv.ODVVariablePtrList Class Reference

The ODVVariablePtrList class provides a list of ODVVariable pointers.

Public Member Functions

- ODVVariablePtrList ()
- ODVVariablePtrList (ODVVariablePtrList other)
- void append (ODVVariable value)
- void append (ODVVariablePtrList value)
- ODVVariable at (int i)
- int count (ODVVariable value)
- int count ()
- boolean empty ()
- void clear ()
- ODVVariable first ()
- int indexOf (ODVVariable value, int from)
- int indexOf (ODVVariable value)
- void insert (int i, ODVVariable value)
- boolean isEmpty ()
- ODVVariable last ()
- ODVVariablePtrList mid (int pos, int length)
- ODVVariablePtrList mid (int pos)
- void move (int from, int to)
- void prepend (ODVVariable value)
- int removeAll (ODVVariable value)
- void removeAt (int i)
- void removeFirst ()
- void removeLast ()
- boolean removeOne (ODVVariable value)
- void replace (int i, ODVVariable value)
- void swap (int i, int j)
- ODVVariable takeAt (int i)
- ODVVariable takeFirst ()
- ODVVariable takeLast ()
- ODVVariable value (int i)
- ODVVariable value (int i, ODVVariable defaultValue)
- ODVVariablePtrList qlist_assign (ODVVariablePtrList other)

Detailed Description

The ODVVariablePtrList class provides a list of ODVVariable pointers.

This class is a subset of the Qt `QList<T>` template class with `T = ODVVariable pointers`. See Qt documentation for further details.

It stores a list of pointers to ODVVariable objects and provides fast index-based access as well as fast insertions and removals.

Note:

Because Java does not know pointers the access functions will return **ODVVariable objects** so you can think of it as a list of ODVVariable **objects** . Please do not get confused by the name ODVVariablePtrList.

ODVVariablePtrList provides these basic functions to add, move, and remove items: insert(), replace(), removeAt(), move(), and swap(). In addition, it provides the following convenience functions: append(), prepend(), removeFirst(), and removeLast().

To avoid failures when your list can be empty, call isEmpty() before calling other member functions. If you must pass an index value that might not be in the valid range, check that it is less than the value returned by count() but not less than 0.

Constructor & Destructor Documentation

ODVVariablePtrList.ODVVariablePtrList ()

Constructs an empty list.

ODVVariablePtrList.ODVVariablePtrList (ODVVariablePtrList *other*)

Constructs a copy of *other* .

This operation takes constant time, because ODVVariablePtrList is implicitly shared. This makes returning a ODVVariablePtrList from a function very fast. If a shared instance is modified, it will be copied (copy-on-write), and that takes linear time.

Member Function Documentation

ODVVariablePtrList.append (ODVVariable *value*)

Inserts *value* at the end of the list.

This is the same as `insert(count(), value)` .

Appends the items of the value list to this list.

ODVVariablePtrList.append (ODVVariablePtrList *value*)

Appends the items of the *value* list to this list.

ODVVariablePtrList.at (int *i*)

Returns:

The item at index position *i* in the list.

i must be a valid index position in the list (i.e., 0 <= *i* < count()).

This function is very fast (constant time).

int ODVVariablePtrList.clear ()

Removes all items from the list.

int ODVVariablePtrList.count (ODVVariable *value*)

Returns:

The number of occurrences of *value* in the list.

Note:

The search takes place only on the object addresses and is not based on the ODVVariable object content. So it only finds this very object itself in the list.

int ODVVariablePtrList.count ()

Returns:

The number of items in the list.

boolean ODVVariablePtrList.empty ()

This function is equivalent to isEmpty() and returns `true` if the list is empty.

ODVVariable ODVVariablePtrList.first ()

Returns:

The first item in the list. The list must not be empty. If the list can be empty, call isEmpty() before calling this function.

See also:

last() and isEmpty().

int ODVVariablePtrList.indexOf (ODVVariable *value*, int *from*)

Returns:

The index position of the first occurrence of *value* in the list, searching forward from index position *from* . Returns `-1` if no item matched.

Note that the list uses 0-based indexes, just like C++ arrays. Negative indexes are not supported with the exception of the value mentioned above.

Note:

The search takes place only on the object addresses and is not based on the ODVVariable object content. So it only finds this very object itself in the list.

int ODVVariablePtrList.indexOf (ODVVariable *value*)

Returns:

The index position of the first occurrence of *value* in the list, searching forward from the beginning of the list.

See also:

indexOf(ODVVariable, int)

void ODVVariablePtrList.insert (int *i*, ODVVariable *value*)

Inserts *value* at index position *i* in the list. If *i* is 0, the value is prepended to the list. If *i* is count(), the value is appended to the list.

See also:

prepend()

boolean ODVVariablePtrList.isEmpty ()

Returns:

`true` if the list contains no items; otherwise returns `false` ..

ODVVariable ODVVariablePtrList.last ()

Returns:

The last item in the list. The list must not be empty. If the list can be empty, call isEmpty() before calling this function.

ODVVariablePtrList ODVVariablePtrList.mid (int *pos*, int *length*)

Returns:

A list whose elements are copied from this list, starting at position *pos* . If *length* is -1, all elements from *pos* are copied; otherwise *length* elements (or all remaining elements if there are less than *length* elements) are copied.

ODVVariablePtrList ODVVariablePtrList.mid (int *pos*)

Returns:

A list whose elements are copied from this list, starting at position *pos* .

See also:

mid(int, int)

void ODVVariablePtrList.move (int *from*, int *to*)

Moves the item at index position *from* to index position *to* .

This is the same as `insert(to, takeAt(from))` . This function assumes that both *from* and *to* are at least `0` but less than count(). To avoid failure, test that both *from* and *to* are at least `0` and less than count().

void ODVVariablePtrList.prepend (ODVVariable *value*)

Inserts *value* at the beginning of the list.

This is the same as `insert(0, value)` . This operation is usually very fast (constant time), because ODVVariablePtrList preallocates extra space on both sides of its internal buffer to allow for fast growth at both ends of the list.

See also:

append(), insert()

ODVVariablePtrList ODVVariablePtrList.qlist_assign (ODVVariablePtrList *other*)

Assigns other to this list and returns this list.

int ODVVariablePtrList.removeAll (ODVVariable *value*)

Removes all occurrences of *value* in the list and returns the number of entries removed.

void ODVVariablePtrList.removeAt (int *i*)

Removes the item at index position *i* . *i* must be a valid index position in the list (i.e., 0 <= *i* < count()).

void ODVVariablePtrList.removeFirst ()

Removes the first item in the list. Calling this function is equivalent to calling removeAt(0). The list must not be empty. If the list can be empty, call isEmpty() before calling this function.

void ODVVariablePtrList.removeLast ()

Removes the last item in the list. Calling this function is equivalent to calling `removeAt(count() - 1)` . The list must not be empty. If the list can be empty, call isEmpty() before calling this function.

boolean ODVVariablePtrList.removeOne (ODVVariable *value*)

Removes the first occurrence of *value* in the list and returns `true` on success; otherwise returns `false`.

See also:

removeAll(), removeAt()

void ODVVariablePtrList.replace (int *i*, ODVVariable *value*)

Replaces the item at index position *i* with *value*. *i* must be a valid index position in the list (i.e., 0 <= *i* < count()).

void ODVVariablePtrList.swap (int *i*, int *j*)

Exchange the item at index position *i* with the item at index position *j*. This function assumes that both *i* and *j* are at least 0 but less than count(). To avoid failure, test that both *i* and *j* are at least 0 and less than count().

ODVVariable ODVVariablePtrList.takeAt (int *i*)

Removes the item at index position *i* and returns it. *i* must be a valid index position in the list (i.e., 0 <= *i* < count()).

If you don't use the return value, removeAt() is more efficient.

ODVVariable ODVVariablePtrList.takeFirst ()

Removes the first item in the list and returns it. This is the same as `takeAt(0)`. This function assumes the list is not empty. To avoid failure, call isEmpty() before calling this function.

This operation takes constant time.

If you don't use the return value, removeFirst() is more efficient.

ODVVariable ODVVariablePtrList.takeLast ()

Removes the last item in the list and returns it. This is the same as `takeAt(count() - 1)`. This function assumes the list is not empty. To avoid failure, call isEmpty() before calling this function.

This operation takes constant time.

If you don't use the return value, removeLast() is more efficient.

ODVVariable ODVVariablePtrList.value (int *i*)

Returns:

The value at index position *i* in the list.

If the index *i* is out of bounds, the function returns a default-constructed value, i.e. an empty string. If you are certain that the index is going to be within bounds, you can use at() instead, which is slightly faster.

ODVVariable ODVVariablePtrList.value (int *i*, ODVVariable *defaultValue*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

If the index *i* is out of bounds, the function returns *defaultValue*.

See also:

value(int)

de.awi.odv.QByteArray Class Reference

The QByteArray class provides an array of bytes.

Public Member Functions

- QByteArray ()
- QByteArray (String str)
- QByteArray (QByteArray ba)
- QByteArray append (char c)
- QByteArray append (String s)
- QByteArray append (String s, int len)
- QByteArray append (QByteArray a)
- char at (int i)
- void chop (int n)
- void clear ()
- String constData ()
- boolean contains (char ch)
- boolean contains (String str)
- boolean contains (QByteArray ba)
- int count (char ch)
- int count (String str)
- int count (QByteArray ba)
- String data ()
- QByteArray fill (char c, int size)
- QByteArray fill (char c)
- int indexOf (char ch, int from)
- int indexOf (char ch)
- int indexOf (String str, int from)
- int indexOf (String str)
- int indexOf (QByteArray ba, int from)
- int indexOf (QByteArray ba)
- QByteArray insert (int i, char c)
- QByteArray insert (int i, String s)
- QByteArray insert (int i, String s, int len)
- QByteArray insert (int i, QByteArray a)
- boolean isEmpty ()
- int lastIndexOf (char ch, int from)
- int lastIndexOf (char ch)
- int lastIndexOf (String str, int from)
- int lastIndexOf (String str)
- int lastIndexOf (QByteArray ba, int from)
- int lastIndexOf (QByteArray ba)
- QByteArray left (int len)
- QByteArray mid (int index, int len)
- QByteArray mid (int index)
- QByteArray prepend (char c)
- QByteArray prepend (String s)
- QByteArray prepend (String s, int len)
- QByteArray prepend (QByteArray a)
- QByteArray remove (int index, int len)
- void resize (int size)
- QByteArray right (int len)

- QByteArray setNum (int arg0, int base)
- QByteArray setNum (int arg0)
- QByteArray setNum (double arg0, char f, int prec)
- QByteArray setNum (double arg0, char f)
- QByteArray setNum (double arg0)
- QByteArray simplified ()
- int size ()
- double toDouble (boolean[] ok)
- double toDouble ()
- int toInt (boolean[] ok, int base)
- int toInt (boolean[] ok)
- int toInt ()
- QByteArray toLower ()
- QByteArray toUpper ()
- QByteArray trimmed ()
- void truncate (int pos)

Detailed Description

The QByteArray class provides an array of bytes.

This class is a subset of the Qt QByteArray class. See Qt documentation for further details.

QByteArray can be used to store both raw bytes (including '\0's) and traditional 8-bit '\0'-terminated strings. Behind the scenes, it always ensures that the data is followed by a '\0' terminator, and uses implicit sharing (copy-on-write) to reduce memory usage and avoid needless copying of data.

One way to initialize a QByteArray is simply to pass a Java string to its constructor. For example, the following code creates a byte array of size 5 containing the data "Hello":

QByteArray ba = new QByteArray("Hello");

To obtain the actual Java string, call data() or constData().

QByteArray provides the following basic functions for modifying the byte data: append(), prepend(), insert(), and remove(). For example:

```
QByteArray x = new QByteArray("and");
x.prepend("rock ");      // x == "rock and"
x.append(" roll");       // x == "rock and roll"
x.insert(5, "&");       // x == "rock &and roll"
```

x.remove(6, 3); // x == "rock & roll" The remove() functions' first argument are the position from which to start erasing and the number of bytes that should be erased.

Constructor & Destructor Documentation

public QByteArray.QByteArray ()

Constructs an empty byte array.

public QByteArray.QByteArray (String *str*)

Constructs a byte array initialized with the string *str* .

public QByteArray.QByteArray (QByteArray *ba*)

Constructs a copy of *ba* .

Member Function Documentation

public QByteArray QByteArray.append (char *c*)

Appends the character *c* to this byte array. The resulting QByteArray is returned.

public QByteArray QByteArray.append (String *s*)

Appends the string *s* to this byte array. The resulting QByteArray is returned.

public QByteArray QByteArray.append (String *s*, int *len*)

Appends the first *len* characters of the string *s* to this byte array. The resulting QByteArray is returned.

public QByteArray QByteArray.append (QByteArray *a*)

Appends the byte array *a* to this byte array. The resulting QByteArray is returned.

public char QByteArray.at (int *i*)

Returns:

The character at index position *i* in the byte array. *i* must be a valid index position in the byte array (i.e., `0` <= *i* < size()).

public void QByteArray.chop (int *n*)

Removes *n* bytes from the end of the byte array. If *n* is greater than size(), the result is an empty byte array.

public void QByteArray.clear ()

Clears the contents of the byte array and makes it empty.

public String QByteArray.constData ()

Returns:

The data stored in the byte array as Java string.

This is the same as data().

public boolean QByteArray.contains (char *ch*)

Returns:

`true` if the byte array contains the character *ch* ; otherwise returns `false` .

public boolean QByteArray.contains (String *str*)

Returns:

`true` if the byte array contains the string *str* ; otherwise returns `false` .

public boolean QByteArray.contains (QByteArray *ba*)

Returns:

`true` if the byte array contains an occurrence of the byte array *ba* ; otherwise returns `false` .

public int QByteArray.count (char *ch*)

Returns:

The number of occurrences of character *ch* in the byte array.

public int QByteArray.count (String *str*)

Returns:

The number of (potentially overlapping) occurrences of string *str* in the byte array.

public int QByteArray.count (QByteArray *ba*)

Returns:

The number of (potentially overlapping) occurrences of byte array *ba* in this byte array.

public String QByteArray.data ()

Returns:

The data stored in the byte array as Java string.

public QByteArray QByteArray.fill (char *c*, int *size*)

Sets every byte in the byte array to character *c* . If *size* is different from -1, the byte array is resized to size *size* beforehand. The resulting QByteArray is returned.

Example:

```
QByteArray ba = new("Istanbul");
ba.fill('o', -1);            // ba == "oooooooo"
```

ba.fill('X', 2); // ba == "XX"

public QByteArray QByteArray.fill (char *c*)

This is an overloaded function. The size remains unchanged.

See also:

fill(char, int)

public int QByteArray.indexOf (char *ch*, int *from*)

Returns:

The index position of the first occurrence of the character *ch* in the byte array, searching forward from index position *from* . Returns `-1` if *ch* could not be found.

Example:

```
QByteArray ba = new QByteArray("ABCBA");
ba.indexOf("B", 0);         // returns 1
ba.indexOf("B", 1);         // returns 1
ba.indexOf("B", 2);         // returns 3
```

ba.indexOf("X", 0); // returns -1

public int QByteArray.indexOf (char *ch*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. The search starts from the beginning of the QByteArray.

See also:

indexOf(char, int)

public int QByteArray.indexOf (String *str*, int *from*)

Returns:

The index position of the first occurrence of the string *str* in the byte array, searching forward from index position *from* . Returns `-1` if *str* could not be found.

public int QByteArray.indexOf (String *str*)

Returns:

The index position of the first occurrence of the string *str* in the byte array. The search starts from the beginning of the QByteArray.

public int QByteArray.indexOf (QByteArray *ba*, int *from*)

Returns:

The index position of the first occurrence of the byte array *ba* in this byte array, searching forward from index position *from* . Returns `-1` if *ba* could not be found.

Example:

```
QByteArray x = new QByteArray("sticky question");
QByteArray y = new QByteArray("sti");
x.indexOf(y, 0);            // returns 0
x.indexOf(y, 1);            // returns 10
x.indexOf(y, 10);           // returns 10
x.indexOf(y, 11);           // returns -1
```

public int QByteArray.indexOf (QByteArray *ba*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. The search starts from the beginning of the QByteArray.

See also:

indexOf(QByteArray, int)

public QByteArray QByteArray.insert (int *i*, char *c*)

Inserts character *c* at index position *i* in the byte array. If *i* is greater than size(), the array is first extended using resize(). The resulting QByteArray is returned.

public QByteArray QByteArray.insert (int *i*, String *s*)

Inserts the string *s* at index position *i* in the byte array. If *i* is greater than size(), the array is first extended using resize(). The resulting QByteArray is returned.

public QByteArray QByteArray.insert (int *i*, String *s*, int *len*)

Inserts *len* bytes of the string *s* at index position *i* in the byte array. If *i* is greater than size(), the array is first extended using resize(). The resulting QByteArray is returned.

public QByteArray QByteArray.insert (int *i*, QByteArray *a*)

Inserts the byte array *a* at index position *i* in the byte array. The resulting QByteArray is returned.

public boolean QByteArray.isEmpty ()

Returns:

`true` if the byte array has size 0; otherwise returns `false` .

public int QByteArray.lastIndexOf (char *ch*, int *from*)

Returns:

The index position of the last occurrence of the character *ch* in the byte array, searching backward from index position *from* . If *from* is `-1` , the search starts at the last (size() - 1) byte. Returns `-1` if *ch* could not be found.

Example:

```
QByteArray ba = new QByteArray("ABCBA");
ba.lastIndexOf("B", -1);    // returns 3
ba.lastIndexOf("B", 3);     // returns 3
ba.lastIndexOf("B", 2);     // returns 1
```

ba.lastIndexOf("X", -1); // returns -1

public int QByteArray.lastIndexOf (char *ch*)

This is an overloaded function. The search starts from the last byte of the QByteArray.

See also:

lastIndexOf(char, int)

public int QByteArray.lastIndexOf (String *str*, int *from*)

Returns:

The index position of the last occurrence of the string *str* in the byte array, searching backward from index position *from* . If *from* is `-1` , the search starts at the last (size() - 1) byte. Returns `-1` if *str* could not be found.

public int QByteArray.lastIndexOf (String *str*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. The search starts from the last byte of the QByteArray.

See also:

lastIndexOf(String, int)

public int QByteArray.lastIndexOf (QByteArray *ba*, int *from*)

Returns:

The index position of the last occurrence of the byte array *ba* in this byte array, searching backward from index position *from* . If *from* is `-1` , the search starts at the last (size() - 1) byte. Returns `-1` if *ch* could not be found.

Example:

```
QByteArray x = new QByteArray("crazy azimuths");
QByteArray y = new QByteArray("az");
x.lastIndexOf(y, -1);       // returns 6
x.lastIndexOf(y, 6);        // returns 6
x.lastIndexOf(y, 5);        // returns 2
```

x.lastIndexOf(y, 1); // returns -1

public int QByteArray.lastIndexOf (QByteArray *ba*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. The search starts from the last byte of the QByteArray.

See also:

lastIndexOf(QByteArray, int)

public QByteArray QByteArray.left (int *len*)

Returns:

A byte array that contains the leftmost *len* bytes of this byte array. The entire byte array is returned if len is greater than size().

Example:

```
QByteArray x = new QByteArray("Pineapple");
QByteArray y = x.left(4);                    // y == "Pine"
```

public QByteArray QByteArray.mid (int *index*, int *len*)

Returns:

A byte array containing *len* bytes from this byte array, starting at position *index* . If *len* is -1, or *index* + *len* >= size(), returns a byte array containing all bytes starting at position *index* until the end of the byte array.

Example:

```
QByteArray x = new QByteArray("Five pineapples");
QByteArray y = x.mid(5, 4);     // y == "pine"
QByteArray z = x.mid(5, -1);     // z == "pineapples"
```

public QByteArray QByteArray.mid (int *index*)

This is an overloaded function. All bytes from the position *index* are returned.

See also:

mid(int, int)

public QByteArray QByteArray.prepend (char *c*)

Prepends the character *c* to this byte array. The resulting QByteArray is returned.

public QByteArray QByteArray.prepend (String *s*)

Prepends the string *s* to this byte array. The resulting QByteArray is returned.

public QByteArray QByteArray.prepend (String *s*, int *len*)

Prepends *len* bytes of the string *s* to this byte array. The resulting QByteArray is returned.

public QByteArray QByteArray.prepend (QByteArray *a*)

Prepends the byte array *a* to this byte array. The resulting QByteArray is returned.

public QByteArray QByteArray.remove (int *pos*, int *len*)

Removes *len* bytes from the array, starting at index position *pos* , and returns the resulting array.

If *pos* is out of range, nothing happens. If *pos* is valid, but *pos* + *len* is larger than the size of the array, the array is truncated at position *pos* .

void QByteArray.resize (int *size*)

Sets the size of the byte array to *size* bytes. If *size* is greater than the current size, the byte array is extended to make it *size* bytes with the extra bytes added to the end. The new bytes are uninitialized.

If *size* is less than the current size, bytes are removed from the end.

See also:

size() and truncate().

public QByteArray QByteArray.right (int *len*)

Returns:

A byte array that contains the rightmost *len* bytes of this byte array. The entire byte array is returned if len is greater than size().

Example:

```
QByteArray x = new QByteArray("Pineapple");
QByteArray y = x.right(5);                  // y == "apple"
```

public QByteArray QByteArray.setNum (int *arg0*, int *base*)

Sets the byte array to the printed value of *arg0* in base *base* and returns the resulting byte array. The base can be any value between 2 and 36.

Example:

```
QByteArray ba = new QByteArray();
int arg0 = 63;
ba.setNum(arg0, 10);       // ba == "63"
ba.setNum(arg0, 16);         // ba == "3f"
```

public QByteArray QByteArray.setNum (int *arg0*)

This is an overloaded function. The base is 10.

See also:

setNum(int, int)

public QByteArray QByteArray.setNum (double *arg0*, char *f*, int *prec*)

Sets the byte array to the printed value of *arg0* , formatted in format *f* with precision *prec* , and returns the resulting byte array.

The format *f* can be any of the following:

Format	Meaning
e	format as [-]9.9e[+|-]999
E	format as [-]9.9E[+|-]999
f	format as [-]9.9
g	use e or f format, whichever is the most concise
G	use E or f format, whichever is the most concise

With 'e', 'E', and 'f', *prec* is the number of digits after the decimal point. With 'g' and 'G', *prec* is the maximum number of significant digits (trailing zeroes are omitted).

public QByteArray QByteArray.setNum (double *arg0*, char *f*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. The precision is 6.

See also:

setNum(double, char, int)

public QByteArray QByteArray.setNum (double *arg0*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. The format is 'g' and the precision is 6.

See also:

setNum(double, char, int)

public QByteArray QByteArray.simplified ()

Returns:

The byte array that has whitespace removed from the start and the end, and which has each sequence of internal whitespace replaced with a single space. Whitespace means any character for which the standard C++ isspace() function returns `true` . This includes the ASCII characters '\t', '\n', '\v', '\f', '\r', and ' '.

See also:

trimmed()

public int QByteArray.size ()

Returns:

The number of bytes in this byte array. The last byte in the byte array is at position size() - 1. In addition, QByteArray ensures that the byte at position size() is always '\0', so that you can use the return value of data() and constData() as arguments to functions that expect '\0'-terminated strings.

public double QByteArray.toDouble (boolean[] *ok*)

Returns:

The byte array converted to a double value. Returns `0.0` if the conversion fails.

If a conversion error occurs, *ok* [0] is set to `false` ; otherwise *ok* [0] is set to `true` .

Note:

The conversion of the number is performed in the default C locale, irrespective of the user's locale.

public double QByteArray.toDouble ()

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. The conversion success can not be checked.

See also:

toDouble(boolean[])

public int QByteArray.toInt (boolean[] *ok*, int *base*)

Returns:

The byte array converted to an `int` using base *base* , which must be between 2 and 36, or 0. If *base* is 0, the base is determined automatically using the following rules: If the byte array begins with "0x", it is assumed to be hexadecimal; if it begins with "0", it is assumed to be octal; otherwise it is assumed to be decimal. Returns `0` if the conversion fails.

If a conversion error occurs, *ok* [0] is set to `false` ; otherwise *ok* [0] is set to `true` .

Example:

```
QByteArray str = new QByteArray("FF");
boolean ok[] = new boolean[1];
int hex = str.toInt(ok, 16);     // hex == 255, ok[0] == true
```

int dec = str.toInt(ok, 10); // dec == 0, ok[0] == false

Note:

The conversion of the number is performed in the default C locale, irrespective of the user's locale.

public int QByteArray.toInt (boolean[] *ok*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. The base is 10.

See also:

toInt(boolean[], int)

public int QByteArray.toInt ()

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. The base is 10. The conversion success can not be checked.

See also:

toInt(boolean[], int)

public QByteArray QByteArray.toLower ()

Returns:

A lowercase copy of the byte array. The byte array is interpreted as a Latin-1 encoded string.

public QByteArray QByteArray.toUpper ()

Returns:

A uppercase copy of the byte array. The byte array is interpreted as a Latin-1 encoded string.

public QByteArray QByteArray.trimmed ()

Returns:

The byte array that has whitespace removed from the start and the end. Whitespace means any character for which the standard C++ isspace() function returns `true` . This includes the ASCII characters '\t', '\n', '\v', '\f', '\r', and ' '.

See also:

simplified()

public void QByteArray.truncate (int *pos*)

Truncates the byte array at index position *pos* . If *pos* is beyond the end of the array, nothing happens.

de.awi.odv.QChar Class Reference

The QChar class provides a 16-bit Unicode character.

Public Member Functions

- QChar (short code)
- QChar (int code)
- QChar (char latin1char)
- int digitValue ()
- boolean isNull ()
- boolean isPrint ()
- boolean isPunct ()
- boolean isSpace ()
- boolean isMark ()
- boolean isLetter ()
- boolean isNumber ()
- boolean isLetterOrNumber ()
- boolean isDigit ()
- boolean isSymbol ()
- boolean isLower ()
- boolean isUpper ()
- char toLatin1 ()
- QChar toLower ()
- QChar toUpper ()
- int unicode ()

Static Public Member Functions

- static QChar fromLatin1 (char c)

Detailed Description

The QChar class provides a 16-bit Unicode character.

This class is a subset of the Qt QChar class. See Qt documentation for further details.

In Qt, Unicode characters are 16-bit entities without any markup or structure. This class represents such an entity. It is lightweight, so it can be used everywhere. Most compilers treat it like a unsigned short.

The classification functions include functions like those in the standard C++ header <cctype> (formerly <ctype.h>), but operating on the full range of Unicode characters. They all return true if the character is a certain type of character; otherwise they return false. These classification functions are isNull() (returns true if the character is '\0'), isPrint() (true if the character is any sort of printable character, including whitespace), isPunct() (any sort of punctation), isMark() (Unicode Mark), isLetter() (a letter), isNumber() (any sort of numeric character, not just 0-9), isLetterOrNumber(), and isDigit() (decimal digits).

The conversion functions include unicode() (to a scalar), toLatin1() (to scalar, but converts all non-Latin-1 characters to 0), digitValue() (gives the integer value of any of the numerous digit characters), and a host of constructors.

Constructor & Destructor Documentation

public QChar.QChar (short *code*)

Constructs a QChar for the character with Unicode code point *code* .

public QChar.QChar (int *code*)

Constructs a QChar for the character with Unicode code point *code* .

public QChar.QChar (char *latin1char*)

Constructs a QChar corresponding to ASCII/Latin-1 character *latin1char* .

Member Function Documentation

public int QChar.digitValue ()

Returns:

The numeric value of the digit, or -1 if the character is not a digit.

public static QChar QChar.fromLatin1 (char *ch*)[static]

Converts the Latin-1 character *ch* to its equivalent QChar.

public boolean QChar.isDigit ()

Returns:

`true` if the character is a decimal digit; otherwise returns `false` .

public boolean QChar.isLetter ()

Returns:

`true` if the character is a letter; otherwise returns `false` .

public boolean QChar.isLetterOrNumber ()

Returns:

`true` if the character is a letter or number; otherwise returns `false` .

public boolean QChar.isLower ()

Returns:

`true` if the character is a lowercase letter; otherwise returns `false` .

public boolean QChar.isMark ()

Returns:

`true` if the character is a mark; otherwise returns `false` .

public boolean QChar.isNull ()

Returns:

`true` if the character is the Unicode character 0x0000 ('\0'); otherwise returns `false` .

public boolean QChar.isNumber ()

Returns:

`true` if the character is a number; otherwise returns `false` .

public boolean QChar.isPrint ()

Returns:

`true` if the character is a printable character; otherwise returns `false` .

Note:

This gives no indication of whether the character is available in a particular font.

public boolean QChar.isPunct ()

Returns:

`true` if the character is a punctuation mark; otherwise returns `false` .

public boolean QChar.isSpace ()

Returns:

`true` if the character is a separator character; otherwise returns `false` .

public boolean QChar.isSymbol ()

Returns:

`true` if the character is a symbol; otherwise returns `false` .

public boolean QChar.isUpper ()

Returns:

`true` if the character is an uppercase letter; otherwise returns `false` .

public char QChar.toLatin1 ()

Returns:

The Latin-1 character equivalent to the QChar, or 0.

public QChar QChar.toLower ()

Returns:

The lowercase equivalent if the character is uppercase or titlecase; otherwise returns the character itself.

public QChar QChar.toUpper ()

Returns:

The uppercase equivalent if the character is lowercase or titlecase; otherwise returns the character itself.

public int QChar.unicode ()

Returns:

The numeric Unicode value of the QChar.

de.awi.odv.ODVQualityFlagSet.QFSetID Enum Reference

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

Public Member Functions

- final int swigValue ()

Static Public Member Functions

- static QFSetID swigToEnum (int swigValue)

Public Attributes

- 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)

Detailed Description

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

You may apply `int swigValue()` to the QFSetID enum object to obtain the corresponding integer value.

Member Function Documentation

QFSetID ODVQualityFlagSet.QFSetID.swigToEnum (int *swigValue*)[static]

Returns:

The QFSetID enum value corresponding to the supplied integer *swigValue* .

final int ODVQualityFlagSet.QFSetID.swigValue ()

Returns:

The integer value corresponding to the QFSetID enum value.

Member Data Documentation

ODVQualityFlagSet.QFSetID.ARGO =(2)
ARGO quality flags

ODVQualityFlagSet.QFSetID.BODC =(11)
British Oceanographic Data Centre quality flags

ODVQualityFlagSet.QFSetID.ESEAS =(4)
ESEAS quality flags

ODVQualityFlagSet.QFSetID.GTSPP =(1)
GTSPP quality flags

ODVQualityFlagSet.QFSetID.IODE =(15)
IODE quality flags

ODVQualityFlagSet.QFSetID.OCEANSITES =(14)
OceanSITES quality flags

ODVQualityFlagSet.QFSetID.ODV =(0)
Ocean Data View quality flags (default)

ODVQualityFlagSet.QFSetID.PANGAEA =(12)
PANGAEA quality flags

ODVQualityFlagSet.QFSetID.QARTOD =(10)
QARTOD quality flags

ODVQualityFlagSet.QFSetID.SEADATANET =(3)
SEADATANET quality flags

ODVQualityFlagSet.QFSetID.SMHI =(13)
Swedish Meteorological and Hydrographic Institute quality codes

ODVQualityFlagSet.QFSetID.WOCEBOTTLE =(7)
WOCE bottle data quality flags

ODVQualityFlagSet.QFSetID.WOCECTD =(8)
WOCE CTD data quality flags

ODVQualityFlagSet.QFSetID.WOCESAMPLE =(9)
WOCE quality flags for the water bottle itself

ODVQualityFlagSet.QFSetID.WOD =(5)
World Ocean Database observed level quality codes

ODVQualityFlagSet.QFSetID.WODSTATION =(6)
World Ocean Database entire station quality flags

de.awi.odv.QIntList Class Reference

The QIntList class provides a list of integers.

Public Member Functions

- QIntList ()
- QIntList (QIntList other)
- void append (int value)
- void append (QIntList value)
- int at (int i)
- int count (int value)
- int count ()
- boolean empty ()
- void clear ()
- int first ()
- int indexOf (int value, int from)
- int indexOf (int value)
- void insert (int i, int value)
- boolean isEmpty ()
- int last ()
- QIntList mid (int pos, int length)
- QIntList mid (int pos)
- void move (int from, int to)
- void prepend (int value)
- int removeAll (int value)
- void removeAt (int i)
- void removeFirst ()
- void removeLast ()
- boolean removeOne (int value)
- void replace (int i, int value)
- void swap (int i, int j)
- int takeAt (int i)
- int takeFirst ()
- int takeLast ()
- int value (int i)
- int value (int i, int defaultValue)
- QIntList qlist_assign (QIntList other)

Detailed Description

The QIntList class provides a list of integers.

This class is a subset of the Qt `QList<int>` class. See Qt documentation for further details.

It stores a list of ints and provides fast index-based access as well as fast insertions and removals.

QIntList provides these basic functions to add, move, and remove items: insert(), replace(), removeAt(), move(), and swap(). In addition, it provides the following convenience functions: append(), prepend(), removeFirst(), and removeLast().

To avoid failures when your list can be empty, call isEmpty() before calling other member functions. If you must pass an index value that might not be in the valid range, check that it is less than the value returned by count() but not less than 0.

Constructor & Destructor Documentation

QIntList.QIntList ()

Constructs an empty list.

QIntList.QIntList (QIntList *other*)

Constructs a copy of *other* .

This operation takes constant time, because QIntList is implicitly shared. This makes returning a QIntList from a function very fast. If a shared instance is modified, it will be copied (copy-on-write), and that takes linear time.

Member Function Documentation

QIntList.append (int *value*)

Inserts *value* at the end of the list.

This is the same as `insert(count(), value)` .

Appends the items of the value list to this list.

QIntList.append (QIntList *value*)

Appends the items of the *value* list to this list.

QIntList.at (int *i*)

Returns:

The item at index position *i* in the list.

i must be a valid index position in the list (i.e., 0 <= *i* < count()).

This function is very fast (constant time).

int QIntList.clear ()

Removes all items from the list.

int QIntList.count (int *value*)

Returns:

The number of occurrences of *value* in the list.

int QIntList.count ()

Returns:

The number of items in the list.

boolean QIntList.empty ()

This function is equivalent to isEmpty() and returns `true` if the list is empty.

int QIntList.first ()

Returns:

The first item in the list. The list must not be empty. If the list can be empty, call isEmpty() before calling this function.

See also:

last() and isEmpty().

int QIntList.indexOf (int *value*, int *from*)

Returns:

The index position of the first occurrence of *value* in the list, searching forward from index position *from* . Returns `-1` if no item matched.

Note that the list uses 0-based indexes, just like C++ arrays. Negative indexes are not supported with the exception of the value mentioned above.

int QIntList.indexOf (int *value*)

Returns:

The index position of the first occurrence of *value* in the list, searching forward from the beginning of the list.

See also:

indexOf(int, int)

void QIntList.insert (int *i*, int *value*)

Inserts *value* at index position *i* in the list. If *i* is 0, the value is prepended to the list. If *i* is count(), the value is appended to the list.

See also:

prepend()

boolean QIntList.isEmpty ()

Returns:

`true` if the list contains no items; otherwise returns `false` ..

int QIntList.last ()

Returns:

The last item in the list. The list must not be empty. If the list can be empty, call isEmpty() before calling this function.

QIntList QIntList.mid (int *pos*, int *length*)

Returns:

A list whose elements are copied from this list, starting at position *pos* . If *length* is -1, all elements from *pos* are copied; otherwise *length* elements (or all remaining elements if there are less than *length* elements) are copied.

QIntList QIntList.mid (int *pos*)

Returns:

A list whose elements are copied from this list, starting at position *pos* .

See also:

mid(int, int)

void QIntList.move (int *from*, int *to*)

Moves the item at index position *from* to index position *to* .

This is the same as `insert(to, takeAt(from))` . This function assumes that both *from* and *to* are at least `0` but less than count(). To avoid failure, test that both *from* and *to* are at least `0` and less than count().

void QIntList.prepend (int *value*)

Inserts *value* at the beginning of the list.

This is the same as `insert(0, value)` . This operation is usually very fast (constant time), because QIntList preallocates extra space on both sides of its internal buffer to allow for fast growth at both ends of the list.

See also:

append(), insert()

QIntList QIntList.qlist_assign (QIntList *other*)

Assigns other to this list and returns this list.

int QIntList.removeAll (int *value*)

Removes all occurrences of *value* in the list and returns the number of entries removed.

void QIntList.removeAt (int *i*)

Removes the item at index position *i* . *i* must be a valid index position in the list (i.e., 0 <= *i* < count()).

void QIntList.removeFirst ()

Removes the first item in the list. Calling this function is equivalent to calling removeAt(0). The list must not be empty. If the list can be empty, call isEmpty() before calling this function.

void QIntList.removeLast ()

Removes the last item in the list. Calling this function is equivalent to calling `removeAt(count() - 1)` . The list must not be empty. If the list can be empty, call isEmpty() before calling this function.

boolean QIntList.removeOne (int *value*)

Removes the first occurrence of *value* in the list and returns `true` on success; otherwise returns `false` .

See also:

removeAll(), removeAt()

void QIntList.replace (int *i*, int *value*)

Replaces the item at index position *i* with *value* . *i* must be a valid index position in the list (i.e., 0 <= *i* < count()).

void QIntList.swap (int *i*, int *j*)

Exchange the item at index position *i* with the item at index position *j*. This function assumes that both *i* and *j* are at least 0 but less than count(). To avoid failure, test that both *i* and *j* are at least 0 and less than count().

int QIntList.takeAt (int *i*)

Removes the item at index position *i* and returns it. *i* must be a valid index position in the list (i.e., 0 <= *i* < count()).

If you don't use the return value, removeAt() is more efficient.

int QIntList.takeFirst ()

Removes the first item in the list and returns it. This is the same as `takeAt(0)`. This function assumes the list is not empty. To avoid failure, call isEmpty() before calling this function.

This operation takes constant time.

If you don't use the return value, removeFirst() is more efficient.

int QIntList.takeLast ()

Removes the last item in the list and returns it. This is the same as `takeAt(count() - 1)`. This function assumes the list is not empty. To avoid failure, call isEmpty() before calling this function.

This operation takes constant time.

If you don't use the return value, removeLast() is more efficient.

int QIntList.value (int *i*)

Returns:

The value at index position *i* in the list.

If the index *i* is out of bounds, the function returns a default-constructed value, i.e. `0`. If you are certain that the index is going to be within bounds, you can use at() instead, which is slightly faster.

int QIntList.value (int *i*, int *defaultValue*)

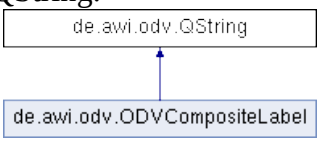
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

If the index *i* is out of bounds, the function returns *defaultValue*.

de.awi.odv.QString Class Reference

The QString class provides a Unicode character string.

Inheritance diagram for de.awi.odv.QString:



Public Member Functions

- QString ()
- QString (String str)
- QString (QChar unicode)
- QString (int size, QChar c)
- QString (QByteArray ba)
- QString (QString other)
- QString append (QChar c)
- QString append (QString s)
- QString append (String str)
- QString append (QByteArray ba)
- QChar at (int position)
- void chop (int n)
- void clear ()
- int count ()
- int count (QChar ch)
- QString fill (QChar c, int size)
- QString fill (QChar c)
- boolean isEmpty ()
- int indexOf (QChar c, int from)
- int indexOf (QChar c)
- int indexOf (QString s, int from)
- int indexOf (QString s)
- QString insert (int i, QString s)
- int lastIndexOf (QChar c, int from)
- int lastIndexOf (QChar c)
- int lastIndexOf (QString s, int from)
- int lastIndexOf (QString s)
- QString left (int n)
- int length ()
- QString mid (int position, int n)
- QString mid (int position)
- QString prepend (QChar c)
- QString prepend (QString s)
- QString prepend (String str)
- QString prepend (QByteArray ba)
- QString remove (int i, int len)
- QString remove (QChar c)
- QString remove (QString s)
- void resize (int size)
- QString right (int n)
- QString setNum (int arg0, int base)
- QString setNum (int arg0)

- QString setNum (double arg0, char f, int prec)
- QString setNum (double arg0, char f)
- QString setNum (double arg0)
- QString simplified ()
- int size ()
- double toDouble (boolean[] ok)
- double toDouble ()
- int toInt (boolean[] ok, int base)
- int toInt (boolean[] ok)
- int toInt ()
- QByteArray toLatin1 ()
- QByteArray toLocal8Bit ()
- QByteArray toUtf8 ()
- QString trimmed ()
- void truncate (int pos)
- QString qstring_assign (QChar c)
- QString qstring_assign (QString arg0)
- QString qstring_append (QString s)

Static Public Member Functions

- static int compare (QString s1, QString s2)

Detailed Description

The QString class provides a Unicode character string.

This class is a subset of the Qt `QString` class. See Qt documentation for further details.

QString stores a string of 16-bit QChars, where each QChar corresponds one Unicode 4.0 character. (Unicode characters with code values above 65535 are stored using surrogate pairs, i.e., two consecutive QChars.) Unicode is an international standard that supports most of the writing systems in use today. It is a superset of US-ASCII (ANSI X3.4-1986) and Latin-1 (ISO 8859-1), and all the US-ASCII/Latin-1 characters are available at the same code positions.

Behind the scenes, QString uses implicit sharing (copy-on-write) to reduce memory usage and to avoid the needless copying of data. This also helps reduce the inherent overhead of storing 16-bit characters instead of 8-bit characters.

In addition to QString, Qt also provides the QByteArray class to store raw bytes and traditional 8-bit '\0'-terminated strings.

One way to initialize a QString is simply to pass a Java string to its constructor. For example, the following code creates a QString of size 5 containing the data "Hello":

QString str = new QString("Hello"); QString converts the string data into Unicode by interpreting them as Latin-1 characters.

To convert a QString into a Java string you may do as follows:

```
QString qstr = new QString("Hello");
String jstr = qstr.toLatin1().data();
```

Constructor & Destructor Documentation

QString.QString ()

Constructs a null string. Null strings are also empty.

See also:

isEmpty()

QString.QString (String *str*)

Constructs a string initialized with the Java string *str* . The characters in the given string are interpreted as 8-bit Latin-1 characters.

QString.QString (QChar *unicode*)

Constructs a string of size 1 containing the unicode character *unicode* .

QString.QString (int *size*, QChar *c*)

Constructs a string of the given *size* with every character set to *c* .

See also:

fill()

QString.QString (QByteArray *ba*)

Constructs a string initialized with the byte array *ba* . The characters in the given string are interpreted as 8-bit Latin-1 characters. Stops copying at the first 0 character, otherwise copies the entire byte array.

QString.QString (QString *other*)

Constructs a copy of *other* .

This operation takes constant time, because QString is implicitly shared. This makes returning a QString from a function very fast. If a shared instance is modified, it will be copied (copy-on-write), and that takes linear time.

See also:

qstring_assign()

Member Function Documentation

QString QString.append (QChar *c*)

Appends the character *c* to this string.

QString QString.append (QString *s*)

Appends the string *s* onto the end of this string.

QString QString.append (String *str*)

Appends the Java string *str* onto the end of this string. The characters in the given string are interpreted as 8-bit Latin-1 characters.

QString & QString.append (QByteArray *ba*)

Appends the byte array *ba* to this string. The characters in the given array are interpreted as 8-bit Latin-1 characters.

QChar QString.at (int *position*)

Returns:

The character at the given index *position* in the string.

The *position* must be a valid index position in the string (i.e., `0` <= *position* < size()).

void QString.chop (int *n*)

Removes *n* characters from the end of the string.

If *n* is greater than size(), the result is an empty string.

If you want to remove characters from the *beginning* of the string, use remove() instead.

See also:

truncate(), resize(), remove()

void QString.clear ()

Clears the contents of the string and makes it empty.

See also:

resize(), isEmpty()

public static int QString.compare (QString *s1*, QString *s2*)[static]

Compares *s1* with *s2* and returns an integer less than, equal to, or greater than zero if *s1* is less than, equal to, or greater than *s2* .

The comparison is case sensitive. Case sensitive comparison is based exclusively on the numeric Unicode values of the characters and is very fast, but is not always what a human would expect.

int QString.count ()

Same as size().

int QString.count (QChar *ch*)

Returns:

The number of occurrences of character *ch* in the string.

QString QString.fill (QChar *c*, int *size*)

Sets every character in the string to character *c* . If *size* is different from `-1` , the string is resized to *size* beforehand.

QString QString.fill (QChar *c*)

Sets every character in the string to character *c* .

int QString.indexOf (QChar *c*, int *from*)

Returns:

The index position of the first occurrence of the character *c* in the string, searching forward from index position *from* . Returns -1 if *c* could not be found.

int QString.indexOf (QChar *c*)

Returns:

The index position of the first occurrence of the character *c* in the string, searching forward from the beginning of the string.

See also:

indexOf(QString, int)

int QString.indexOf (QString *s*, int *from*)

Returns:

The index position of the first occurrence of the string *s* in this string, searching forward from index position *from* . Returns `-1` if *s* is not found.

If *from* is `-1` , the search starts at the last character; if it is `-2` , at the next to last character and so on.

int QString.indexOf (QString *s*)

Returns:

The index position of the first occurrence of the string *s* in this string, searching forward from the beginning of the string.

See also:

indexOf(QString, int)

QString QString.insert (int *i*, QString *s*)

Inserts the string *s* at the given index *i* and returns a this string.

If the given position *i* is greater than size(), the array is first extended using resize().

See also:

append(), prepend(), remove()

boolean QString.isEmpty ()

Returns:

`true` if the string has no characters; otherwise returns `false` .

See also:

size()

int QString.lastIndexOf (QChar *c*, int *from*)

Returns:

The index position of the last occurrence of the character *c* , searching backward from position *from* .

int QString.lastIndexOf (QChar *c*)

Returns:

The index position of the last occurrence of the character *c* , searching backward from the end of the string.

See also:

lastIndexOf(QChar, int)

int QString.lastIndexOf (QString *s*, int *from*)

Returns:

The index position of the last occurrence of the string *s* in this string, searching backward from index position *from* . If *from* is `-1` , the search starts at the last character; if *from* is `-2` , at the next to last character and so on. Returns `-1` if *s* is not found.

See also:

indexOf(), count()

int QString.lastIndexOf (QString *s*)

Returns:

The index position of the last occurrence of the string *s* in this string, searching backward from the end of the string.

See also:

lastIndexOf(QString, int)

QString QString.left (int *n*)

Returns:

A substring that contains the *n* leftmost characters of the string.

The entire string is returned if *n* is greater than size() or less than zero.

See also:

right(), mid()

int QString.length ()

Returns:

The number of characters in this string. Equivalent to size().

See also:

resize()

QString QString.mid (int *position*, int *n*)

Returns:

A string that contains *n* characters of this string, starting at the specified *position* index.

Returns a null string if the *position* index exceeds the length of the string. If there are less than *n* characters available in the string starting at the given *position* , or if *n* is `-1` , the function returns all characters that are available from the specified *position* .

See also:

left(), right()

QString QString.mid (int *position*)

Returns:

A string that contains all characters that are available from the specified *position* index.

Returns a null string if the *position* index exceeds the length of the string.

See also:

mid(int, int), left(), right()

QString QString.prepend (QChar *c*)

Prepends the character *c* to this string and returns this string.

QString QString.prepend (QString *s*)

Prepends the string *s* to the beginning of this string and returns this string.

See also:

append()

QString QString.prepend (String *str*)

Prepends the Java string *str* to this string. The characters in the given string are interpreted as 8-bit Latin-1 characters.

QString QString.prepend (QByteArray *ba*)

Prepends the byte array *ba* to this string. The characters in the given array are interpreted as 8-bit Latin-1 characters.

QString QString.qstring_append (QString *other*)

Appends the string *other* onto the end of this string and returns this string.

This operation is typically very fast (constant time), because QString preallocates extra space at the end of the string data so it can grow without reallocating the entire string each time.

See also:

append(), prepend()

QString QString.qstring_assign (QChar *c*)

Sets the string to contain the single character *c* .

QString QString.qstring_assign (QString *arg0*)

Assigns *arg0* to this string and returns this string.

QString QString.remove (int *i*, int *len*)

Removes *len* characters from the string, starting at the given *position* *i* , and returns the string.

If the specified position *i* is within the string, but *i* + *len* is beyond the end of the string, the string is truncated at the specified position *i* .

See also:

insert()

QString QString.remove (QChar *c*)

Removes every occurrence of the character *c* in this string, and returns this string.

QString QString.remove (QString *s*)

Removes every occurrence of the given *s* string in this string, and returns this string.

void QString.resize (int *size*)

Sets the size of the string to *size* characters.

If *size* is greater than the current size, the string is extended to make it *size* characters long with the extra characters added to the end. The new characters are uninitialized.

If *size* is less than the current size, characters are removed from the end.

If *size* is negative, it is equivalent to passing zero.

QString QString.right (int *n*)

Returns:

A substring that contains the *n* rightmost characters of the string.

The entire string is returned if *n* is greater than size() or less than zero.

See also:

left(), mid()

QString QString.setNum (int *arg0*, int *base*)

Sets the string to the printed value of *arg0* in the specified *base* , and returns the string.

The base must be between 2 and 36. For bases other than 10, *n* is treated as an unsigned integer.

The formatting always uses QLocale::C, i.e., English/UnitedStates.

QString QString.setNum (int *arg0*)

Sets the string to the printed value of *arg0* assuming base 10, and returns the string.

The formatting always uses QLocale::C, i.e., English/UnitedStates.

See also:

setNum(int, int)

QString QString.setNum (double *arg0*, char *f*, int *prec*)

Sets the string to the printed value of *arg0* , formatted according to the given format *f* and precision *prec* , and returns the string.

The *format* can be 'f', 'F', 'e', 'E', 'g' or 'G'.

Argument Formats

The format argument *f* can be one of the following:

Format	Meaning
`e`	format as [-]9.9e[+|-]999
`E`	format as [-]9.9E[+|-]999
`f`	format as [-]9.9
`g`	use `e` or `f` format, whichever is the most concise
`G`	use `E` or `f` format, whichever is the most concise

A precision *prec* is also specified with the argument *f* . For the 'e', 'E', and 'f' formats, the *prec* represents the number of digits after the decimal point. For the 'g' and 'G' formats, the *prec* represents the maximum number of significant digits (trailing zeroes are omitted).
The formatting always uses QLocale::C, i.e., English/UnitedStates.

QString QString.setNum (double *arg0*, char *f*)

Sets the string to the printed value of *arg0* , formatted according to the given format *f* and with a precision of `6` , and returns the string.

The formatting always uses QLocale::C, i.e., English/UnitedStates.

The format *f* can be 'f', 'F', 'e', 'E', 'g' or 'G'.

See also:

setNum(double, char, int)

QString QString.setNum (double *arg0*)

Sets the string to the printed value of *arg0* , formatted according to `'g'` and with a precision of `6` , and returns the string.

The formatting always uses QLocale::C, i.e., English/UnitedStates.

See also:

setNum(double, char, int)

QString QString.simplified ()

Returns:

A string that has whitespace removed from the start and the end, and that has each sequence of internal whitespace replaced with a single space.

Whitespace means any character for which QChar::isSpace() returns true. This includes the ASCII characters '\t', '\n', '\v', '\f', '\r', and ' '.

See also:

trimmed()

int QString.size ()

Returns:

The number of characters in this string.

The last character in the string is at position size() - 1. In addition, QString ensures that the character at position size() is always '\0', so that you can use the return value of data() and constData() as arguments to functions that expect '\0'-terminated strings.

See also:

isEmpty(), resize()

double QString.toDouble (boolean[] *ok*)

Returns:

The string converted to a `double` value.

Returns 0.0 if the conversion fails.

If a conversion error occurs, *ok* [0] is set to false; otherwise *ok* [0] is set to true.

Due to the ambiguity between the decimal point and thousands group separator in various locales, this function does not handle thousands group separators.

double QString.toDouble ()

Returns:

The string converted to a `double` value.

This is an overloaded function. The conversion success can not be checked.

See also:

toDouble(boolean[])

int QString.toInt (boolean[] *ok*, int *base*)

Returns:

The string converted to an `int` using base *base* , which must be between 2 and 36, or 0. Returns 0 if the conversion fails.

If a conversion error occurs, *ok* [0] is set to false; otherwise *ok* [0] is set to true.

If *base* is 0, the C language convention is used: If the string begins with "0x", base 16 is used; if the string begins with "0", base 8 is used; otherwise, base 10 is used.

See also:

toDouble(boolean[])

int QString.toInt (boolean[] *ok*)

Returns:

The string converted to an `int` value. The base is assumed to be 10.

This is an overloaded function.

If a conversion error occurs, *ok* [0] is set to false; otherwise *ok* [0] is set to true.

See also:

toInt(boolean[], int)

int QString.toInt ()

Returns:

The string converted to an `int` value. The base is assumed to be 10.

This is an overloaded function. The conversion success can not be checked.

See also:

toInt(boolean[], int)

QByteArray QString.toLatin1 ()

Returns a Latin-1 representation of the string as a QByteArray.

The returned byte array is undefined if the string contains non-Latin1 characters. Those characters may be suppressed or replaced with a question mark.

QByteArray QString.toLocal8Bit ()

Returns:

The local 8-bit representation of the string as a QByteArray. The returned byte array is undefined if the string contains characters not supported by the local 8-bit encoding. If the locale encoding could not be determined, this function does the same as toLatin1().

If this string contains any characters that cannot be encoded in the locale, the returned byte array is undefined. Those characters may be suppressed or replaced by another.

See also:

toLatin1()

QByteArray QString.toUtf8 ()

Returns:

An UTF-8 representation of the string as a QByteArray.

UTF-8 is a Unicode codec and can represent all characters in a Unicode string like QString.

However, in the Unicode range, there are certain codepoints that are not considered characters. The Unicode standard reserves the last two codepoints in each Unicode Plane (U+FFFE, U+FFFF, U+1FFFE, U+1FFFF, U+2FFFE, etc.), as well as 16 codepoints in the range U+FDD0..U+FDDF, inclusive, as non-characters. If any of those appear in the string, they may be discarded and will not appear in the UTF-8 representation, or they may be replaced by one or more replacement characters.

See also:

toAscii(), toLatin1(), toLocal8Bit()

QString QString.trimmed ()

Returns:

A string that has whitespace removed from the start and the end.

Whitespace means any character for which QChar::isSpace() returns true. This includes the ASCII characters '\t', '\n', '\v', '\f', '\r', and ' '.

Unlike simplified(), trimmed() leaves internal whitespace alone.

See also:

simplified()

void QString.truncate (int *position*)

Truncates the string at the given *position* index.

If the specified *position* index is beyond the end of the string, nothing happens.

If *position* is negative, it is equivalent to passing zero.

See also:

chop(), resize(), left()

de.awi.odv.QStringList Class Reference

The QStringList class provides a list of Unicode character strings.

Public Member Functions

- QStringList ()
- QStringList (QStringList other)
- void append (QString value)
- void append (QStringList value)
- QString at (int i)
  Returns the item at index position i in the list.
- int count (QString value)
- int count ()
- boolean empty ()
- void clear ()
- QString first ()
- int indexOf (QString value, int from)
- int indexOf (QString value)
- void insert (int i, QString value)
  Inserts value at index position i in the list. If i is 0, the value is prepended to the list. If i is count(), the value is appended to the list.
- boolean isEmpty ()
- QString last ()
- QStringList mid (int pos, int length)
- QStringList mid (int pos)
- void move (int from, int to)
- void prepend (QString value)
  Inserts value at the beginning of the list.
- int removeAll (QString value)
- void removeAt (int i)
- void removeFirst ()
- void removeLast ()
- boolean removeOne (QString value)
  Removes the first occurrence of value in the list and returns `true` on success; otherwise returns `false`.
- void replace (int i, QString value)
- void swap (int i, int j)
- QString takeAt (int i)
- QString takeFirst ()
- QString takeLast ()
- QString value (int i)
- QString value (int i, QString defaultValue)
- QStringList qlist_assign (QStringList other)

Detailed Description

The QStringList class provides a list of Unicode character strings.

This class is a subset of the Qt `QStringList` class. See Qt documentation for further details.

It stores a list of strings and provides fast index-based access as well as fast insertions and removals.

QStringList provides these basic functions to add, move, and remove items: insert(), replace(), removeAt(), move(), and swap(). In addition, it provides the following convenience functions: append(), prepend(), removeFirst(), and removeLast().

To avoid failures when your list can be empty, call isEmpty() before calling other member functions. If you must pass an index value that might not be in the valid range, check that it is less than the value returned by count() but not less than 0.

Constructor & Destructor Documentation

QStringList.QStringList ()

Constructs an empty list.

QStringList.QStringList (QStringList *other*)

Constructs a copy of *other* .

This operation takes constant time, because QStringList is implicitly shared. This makes returning a QStringList from a function very fast. If a shared instance is modified, it will be copied (copy-on-write), and that takes linear time.

Member Function Documentation

QStringList.append (QString *value*)

Inserts *value* at the end of the list.

This is the same as `insert(count(), value)` .

Appends the items of the value list to this list.

QStringList.append (QStringList *value*)

Appends the items of the *value* list to this list.

QStringList.at (int *i*)

Returns the item at index position *i* in the list.

i must be a valid index position in the list (i.e., 0 <= *i* < count()).

This function is very fast (constant time).

int QStringList.clear ()

Removes all items from the list.

int QStringList.count (QString *value*)

Returns:

The number of occurrences of *value* in the list.

int QStringList.count ()

Returns:

The number of items in the list.

boolean QStringList.empty ()

This function is equivalent to isEmpty() and returns `true` if the list is empty.

QString QStringList.first ()

Returns:

The first item in the list. The list must not be empty. If the list can be empty, call isEmpty() before calling this function.

See also:

last() and isEmpty().

int QStringList.indexOf (QString *value*, int *from*)

Returns:

The index position of the first occurrence of *value* in the list, searching forward from index position *from* . Returns `-1` if no item matched.

Note that the list uses 0-based indexes, just like C++ arrays. Negative indexes are not supported with the exception of the value mentioned above.

int QStringList.indexOf (QString *value*)

Returns:

The index position of the first occurrence of *value* in the list, searching forward from the beginning of the list.

See also:

indexOf(QString, int)

void QStringList.insert (int *i*, QString *value*)

Inserts *value* at index position *i* in the list. If *i* is 0, the value is prepended to the list. If *i* is count(), the value is appended to the list.

See also:

prepend()

boolean QStringList.isEmpty ()

Returns:

`true` if the list contains no items; otherwise returns `false` .

QString QStringList.last ()

Returns:

The last item in the list. The list must not be empty. If the list can be empty, call isEmpty() before calling this function.

QStringList QStringList.mid (int *pos*, int *length*)

Returns:

A list whose elements are copied from this list, starting at position *pos* . If *length* is -1, all elements from *pos* are copied; otherwise *length* elements (or all remaining elements if there are less than *length* elements) are copied.

QStringList QStringList.mid (int *pos*)

Returns:

A list whose elements are copied from this list, starting at position *pos* .

See also:

mid(int, int)

void QStringList.move (int *from*, int *to*)

Moves the item at index position *from* to index position *to* .

This is the same as `insert(to, takeAt(from))` . This function assumes that both *from* and *to* are at least `0` but less than count(). To avoid failure, test that both *from* and *to* are at least `0` and less than count().

void QStringList.prepend (QString *value*)

Inserts *value* at the beginning of the list.

This is the same as `insert(0, value)` . This operation is usually very fast (constant time), because QStringList preallocates extra space on both sides of its internal buffer to allow for fast growth at both ends of the list.

See also:

append(), insert()

QStringList QStringList.qlist_assign (QStringList *other*)

Assigns other to this list and returns this list.

int QStringList.removeAll (QString *value*)

Removes all occurrences of *value* in the list and returns the number of entries removed.

void QStringList.removeAt (int *i*)

Removes the item at index position *i* . *i* must be a valid index position in the list (i.e., 0 <= *i* < count()).

void QStringList.removeFirst ()

Removes the first item in the list. Calling this function is equivalent to calling removeAt(0). The list must not be empty. If the list can be empty, call isEmpty() before calling this function.

void QStringList.removeLast ()

Removes the last item in the list. Calling this function is equivalent to calling `removeAt(count() - 1)` . The list must not be empty. If the list can be empty, call isEmpty() before calling this function.

boolean QStringList.removeOne (QString *value*)

Removes the first occurrence of *value* in the list and returns `true` on success; otherwise returns `false` .

See also:

removeAll(), removeAt()

void QStringList.replace (int *i*, QString *value*)

Replaces the item at index position *i* with *value* . *i* must be a valid index position in the list (i.e., 0 <= *i* < count()).

void QStringList.swap (int *i*, int *j*)

Exchange the item at index position *i* with the item at index position *j* . This function assumes that both *i* and *j* are at least 0 but less than count(). To avoid failure, test that both *i* and *j* are at least 0 and less than count().

QString QStringList.takeAt (int *i*)

Removes the item at index position *i* and returns it. *i* must be a valid index position in the list (i.e., 0 <= *i* < count()).

If you don't use the return value, removeAt() is more efficient.

QString QStringList.takeFirst ()

Removes the first item in the list and returns it. This is the same as `takeAt(0)` . This function assumes the list is not empty. To avoid failure, call isEmpty() before calling this function.

This operation takes constant time.

If you don't use the return value, removeFirst() is more efficient.

QString QStringList.takeLast ()

Removes the last item in the list and returns it. This is the same as `takeAt(count() - 1)` . This function assumes the list is not empty. To avoid failure, call isEmpty() before calling this function.

This operation takes constant time.

If you don't use the return value, removeLast() is more efficient.

QString QStringList.value (int *i*)

Returns:

The value at index position *i* in the list.

If the index *i* is out of bounds, the function returns a default-constructed value, i.e. an empty string. If you are certain that the index is going to be within bounds, you can use at() instead, which is slightly faster.

QString QStringList.value (int *i*, QString *defaultValue*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

If the index *i* is out of bounds, the function returns *defaultValue* .

de.awi.odv.Qt_casesensitivity Enum Reference

Enumeration values to specifiy if operations should be case-sensitiv or not.

Public Member Functions

- final int swigValue ()

Static Public Member Functions

- static Qt_casesensitivity swigToEnum (int swigValue)

Public Attributes

- CaseInsensitive =(0)
- CaseSensitive =(1)

Detailed Description

Enumeration values to specifiy if operations should be case-sensitiv or not.

You may apply `int swigValue()` to the Qt_casesensitivity enum object to obtain the corresponding integer value.

Member Function Documentation

Qt_casesensitivity Qt_casesensitivity.swigToEnum (int *swigValue*)[static]

Returns:

The Qt_casesensitivity enum value corresponding to the supplied integer *swigValue* .

final int Qt_casesensitivity.swigValue ()

Returns:

The integer value corresponding to the Qt_casesensitivity enum value.

Member Data Documentation

Qt_casesensitivity.CaseInsensitive =(0)

Use if operation shall be case-insensitive.

Qt_casesensitivity.CaseSensitive =(1)

Use if operation shall be case-sensitive.

de.awi.odv.ODVCollection.StateFlag Enum Reference

This flag describes the current state of the collection.

Public Member Functions

- final int swigValue ()

Static Public Member Functions

- static StateFlag swigToEnum (int swigValue)

Public Attributes

- ColUndefined =(0)
- ColDefined =(1)
- ColDeleteWhenClosed =(2)
- ColEmptyNetCDF =(16)
- ColVarsRead =(32)
- ColOpen =(64)

Detailed Description

This flag describes the current state of the collection.

You may apply `int swigValue()` to the StateFlag enum object to obtain the corresponding integer value.

Member Function Documentation

StateFlag ODVCollection.StateFlag.swigToEnum (int *swigValue*)`[static]`

Returns:

The StateFlag enum value corresponding to the supplied integer *swigValue* .

final int ODVCollection.StateFlag.swigValue ()

Returns:

The integer value corresponding to the StateFlag enum value.

Member Data Documentation

ODVCollection.StateFlag.ColDefined =(1)

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.

ODVCollection.StateFlag.ColDeleteWhenClosed =(2)

This flag is currently not used.

ODVCollection.StateFlag.ColEmptyNetCDF =(16)

This flag applies to NetCDF collections only.

ODVCollection.StateFlag.ColOpen =(64)

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

ODVCollection.StateFlag.ColUndefined =(0)

Collection object is not initialized, format unknown.

ODVCollection.StateFlag.ColVarsRead =(32)

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

de.awi.odv.ODV.Status Enum Reference

Returned error status of functions. Not all of them apply to the API.

Public Member Functions

- final int swigValue ()

Static Public Member Functions

- static Status swigToEnum (int swigValue)

Public Attributes

- 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

Detailed Description

Returned error status of functions. Not all of them apply to the API.

You may apply `int swigValue()` to the Status enum object to obtain the corresponding integer value.

Note:

There is an additional value in this enum class available which is unfortunately not documented correctly due to technical reasons:
ODV.Status.NotImplemented : The requested functionality is not implemented yet.

Member Function Documentation

Status ODV.Status.swigToEnum (int *swigValue*)[static]

Returns:
The Status enum value corresponding to the supplied integer *swigValue* .

final int ODV.Status.swigValue ()

Returns:
The integer value corresponding to the Status enum value.

Member Data Documentation

ODV.Status.AnalyzeFileErr
Error on analyzing file

ODV.Status.BadParameter
A bad parameter value was supplied

ODV.Status.CollCopyErr
Could not copy collection

ODV.Status.CollCreateErr
Could not create collection

ODV.Status.CollDelErr
Could not delete collection

ODV.Status.CollFormatUnsupported
The collection format is not supported

ODV.Status.CollNotSorted
Could not sort & condense collection

ODV.Status.CollOpenErr
Could not open collection

ODV.Status.CollReadErr
Error on read in collection

ODV.Status.CollReadOnly
Collection not writable

ODV.Status.CollWriteErr
Error on write to collection

ODV.Status.DirCreateErr
Could not create directory

ODV.Status.EOD
End of data reached

ODV.Status.FileErr
A general error on file operation occured

ODV.Status.FileOpenErr
Could not open file

ODV.Status.FileReadErr
Error on read in file

ODV.Status.FileWriteErr
Error on write to file

ODV.Status.ImportErr
Error on import

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

ODV.Status.InvalidStationData
Data of station is invalid or unreasonable

ODV.Status.MacroErr
Syntax error in macro

ODV.Status.NoErr =(0)
OK

ODV.Status.NoSuchDirErr
Directory does not exist

ODV.Status.OutOfMemory
Machine is out of memory.

ODV.Status.PSPreambleNotFound
Postscript preamble file not found

ODV.Status.StatIDOutOfRange
ID of requested station is out of range

ODV.Status.UnknownFileTypeErr
Unknown file type

ODV.Status.UserAbort
Aborted by user

ODV.Status.VarNotFound

A variable could not be identified

de.awi.odv.ODVVariable.ValueType Enum Reference

The enumeration values represent the type of the variable's values.

Public Member Functions

- final int swigValue ()

Static Public Member Functions

- static ValueType swigToEnum (int swigValue)

Public Attributes

- 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)

Detailed Description

The enumeration values represent the type of the variable's values.

Note:

UNICODETEXT is not supported yet.

You may apply `int swigValue()` to the ValueType enum object to obtain the corresponding integer value.

Member Function Documentation

ValueType ODVVariable.ValueType.swigToEnum (int *swigValue*)`[static]`

Returns:

The ValueType enum value corresponding to the supplied integer *swigValue* .

final int ODVVariable.ValueType.swigValue ()

Returns:

The integer value corresponding to the ValueType enum value.

Member Data Documentation

ODVVariable.ValueType.BYTE =(3)

1-byte unsigned number or character

ODVVariable.ValueType.DOUBLE =(2)

8-byte double precision number

ODVVariable.ValueType.FLOAT =(1)

4-byte floating point number

ODVVariable.ValueType.INDEXED_TEXT =(12)

character string of arbitrary length (only available in CF6 collections)

ODVVariable.ValueType.INTEGER =(5)

4-byte signed integer number

ODVVariable.ValueType.SHORT =(4)

2-byte signed integer number

ODVVariable.ValueType.SIGNED_BYTE =(8)

1-byte signed number or character

ODVVariable.ValueType.TEXT =(10)

character string of variable length (needs to be specified)

ODVVariable.ValueType.UNICODETEXT =(15)

not supported yet

ODVVariable.ValueType.UNKNOWN =(32768)

placeholder for unknown value type

ODVVariable.ValueType.UNSIGNED_INTEGER =(7)

4-byte unsigned integer number

ODVVariable.ValueType.UNSIGNED_SHORT =(6)

2-byte unsigned integer number

de.awi.odv.ODVVariable.VarType Enum Reference

The enumeration values represent the type of the variable.

Public Member Functions

- final int swigValue ()

Static Public Member Functions

- static VarType swigToEnum (int swigValue)

Public Attributes

- BASIC =(-1)
- 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
- METAORIGCRUISE
- METAORIGSTATION
- METAGTSPPDATATYPE
- METACOMMENTSLINK
- METACRUISEREPORTLINK

Detailed Description

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"* .

You may apply `int swigValue()` to the VarType enum object to obtain the corresponding integer value.

Note:

There is an additional value in this enum class available which is unfortunately not documented correctly due to technical reasons:
ODVVariable.VarType.METAREFERENCE : Provides a link to web resources related to the station's data

---

Member Function Documentation

VarType ODVVariable.VarType.swigToEnum (int *swigValue*)[static]

Returns:

The VarType enum value corresponding to the supplied integer *swigValue* .

final int ODVVariable.VarType.swigValue ()

Returns:

The integer value corresponding to the VarType enum value.

Member Data Documentation

ODVVariable.VarType.BASIC =(-1)

Type for all collection variables

ODVVariable.VarType.METAACCESSIONNUMBER

Unique and fixed ID of a station

ODVVariable.VarType.METABASIC =(2000)

Type of all additional, i.e. self-made meta variables

ODVVariable.VarType.METABOTDEPTH

Bottom depth at position

ODVVariable.VarType.METACOMMENTSLINK

URL to more meta information

ODVVariable.VarType.METACRUISE

Cruise name

ODVVariable.VarType.METACRUISEREPORTLINK

URL to cruise report

ODVVariable.VarType.METADAY

Day of observation

ODVVariable.VarType.METADAYOFYEAR =(1002)

Day in year derived from date

ODVVariable.VarType.METADURATION

Duration of observation

ODVVariable.VarType.METAEDMOCODE
SeaDataNet institution identifier

ODVVariable.VarType.METAGTSPPDATATYPE

ODVVariable.VarType.METAHOUR
Hour of observation

ODVVariable.VarType.METALATITUDE
Latitude of position

ODVVariable.VarType.METALOCALCDIID
SeaDataNet station textual identifier

ODVVariable.VarType.METALONGITUDE
Longitude of position

ODVVariable.VarType.METAMINUTE
Minute of observation

ODVVariable.VarType.METAMONTH
Month of observation

ODVVariable.VarType.METAORIGCRUISE

ODVVariable.VarType.METAORIGSTATION

ODVVariable.VarType.METAPRIMVARMAX
Atomatically generated variable. Keeps the maximum value of the primary value for the station.

ODVVariable.VarType.METAPRIMVARMIN
Atomatically generated variable. Keeps the minimum value of the primary value for the station.

ODVVariable.VarType.METASECOND
Second of observation

ODVVariable.VarType.METASENSORDEPTH
Depth of sensor

ODVVariable.VarType.METASTATION
Station name

ODVVariable.VarType.METATIME =(1001)
Meta time

ODVVariable.VarType.METATYPE
Station type

ODVVariable.VarType.METAYEAR

Year of observation

Index