kvQc2

Generated by Doxygen 1.5.5

Sun Nov 1 23:03:54 2009

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

Class Documentation

2.1 CheckedDataCommandBase Class Reference

`#include <CheckedDataCommandBase.h>`

Public Member Functions

- **CheckedDataCommandBase** (const CKvalObs::StationInfoList &stInfo)
- CheckedDataHelper & **helper** ()
- void **helper** (CheckedDataHelper *helper__)

Protected Attributes

- CheckedDataHelper * **helper_**

2.1.1 Detailed Description

Qc2 Context: A class definition necessary for CheckedDataHelper taken from the Kvalobs Qc1 manager. Necessary for communicating to the kvServiced that an update has been made to the database.

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

- CheckedDataCommandBase.h

2.2 Distribute Class Reference

The class manages the redistribution of 24 hour precipitation data.

`#include <Distribute.h>`

Public Member Functions

- **Distribute** (const std::list< kvalobs::kvStation > &slist, ReadProgramOptions params)
- void add_element (int &sid, float &data, float &intp, float &corr, float &newd, miutil::miTime &tbtime, miutil::miTime &time, int &sensor, int &level, int &d_tid, kvalobs::kvControlInfo &d_control, kvalobs::kvUseInfo &d_use, miutil::miString &cfailed)

  Add a row to the data object holding items to be redistributed.

- void clean_station_entry (int &sid)

  Clear single station entry from the redistribution data object.

- void RedistributeStationData (int &sid, std::list< kvalobs::kvData > &ReturnData)

  Algorithm to redistribute data based on interpolated model data.

- void clear_all ()

  Clear all data from the redistribution data object.

Public Attributes

- ReadProgramOptions **params**
- std::map< int, std::vector< float > > **dst_data**
- std::map< int, std::vector< float > > **dst_intp**
- std::map< int, std::vector< float > > **dst_corr**
- std::map< int, std::vector< float > > **dst_newd**
- std::map< int, std::vector< miutil::miTime > > **dst_time**
- std::map< int, std::vector< miutil::miTime > > **dst_tbtime**
- std::map< int, std::vector< int > > **d_sensor**
- std::map< int, std::vector< int > > **d_level**
- std::map< int, std::vector< int > > **d_typeid**
- std::map< int, std::vector< kvalobs::kvControlInfo > > **d_controlinfo**
- std::map< int, std::vector< kvalobs::kvUseInfo > > **d_useinfo**
- std::map< int, std::vector< miutil::miString > > **d_cfailed**

2.2.1 Detailed Description

The class manages the redistribution of 24 hour precipitation data.

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

- algorithms/Distribute.h
- algorithms/Distribute.cc

2.3 kvQABase::par_values Struct Reference

parametervalue, -missingstatus and -controlflag

`#include <kvQABaseTypes.h>`

Public Member Functions

- **par_values** (const std::string &val_, int status_, kvalobs::kvControlInfo &ci)

Public Attributes

- std::string value
  data-value
- int status
  kvQCFlagTypes::missing_status
- kvalobs::kvControlInfo cinfo
  control data-flag

2.3.1 Detailed Description

parametervalue, -missingstatus and -controlflag

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

- algorithms/kvQABaseTypes.h

2.4 ProcessImpl Class Reference

Handles the interface to different processing algorithms.

`#include <ProcessImpl.h>`

Public Member Functions

- **ProcessImpl** (Qc2App &app_, dnmi::db::Connection &con_)
- void **GetStationList** (std::list< kvalobs::kvStation > &StationList)
- void **GetStationList** (std::list< kvalobs::kvStation > &StationList, miutil::miTime ProcessTime)
- int **select** (ReadProgramOptions params)
- int Redistribute (ReadProgramOptions params)
- int Variability (ReadProgramOptions params)
- int **locust_alg** (ReadProgramOptions params)
- int Process4D (ReadProgramOptions params)
- int ProcessUnitT (ReadProgramOptions params)
- int Interpolate (ReadProgramOptions params)
- int ProcessSpaceCheck (ReadProgramOptions params)

2.4.1 Detailed Description

Handles the interface to different processing algorithms.

2.4.2 Member Function Documentation

2.4.2.1 int ProcessImpl::Redistribute (ReadProgramOptions *params*)

StationList is all the possible stations

TODO: interpolate across all type ids and check for effective duplicates.

solution for memory cleanup ... maybe needs to be improved.

2.4.2.2 int ProcessImpl::Variability (ReadProgramOptions *params*)

Possibly this should be checked for every time interval ?????

2.4.2.3 int ProcessImpl::Process4D (ReadProgramOptions *params*)

Need to integrate multiple handling of different type ids OR resolve this issue by separate program that scan kvalobs database and identifies the value of each duplicate measurement to use ...

carry window with the paramter StepH

This algorithm steps back in time.

StationList is all the possible stations

Does this not have to go in the loop ...

The window is determined by the params.StepH (need to add a uniques parameter in config options for this), fed into HW The time series selected is checked for missing values and the optimum number of niegbours x: value m: missing value

m x x x x x m m m m x x x x x x m time —> 1 2 3 4

1: maxlower 2: minlower 3: minupper 4: maxupper

The indices for the missing values 'm' are held in gap_index.

OBS! The length of the akima spline is longer than the interval requested ... is this a behaviour of the library or a bug ??? CHECK

CHECK out this point !!! what exactly happens here ??????

2.4.2.4 int ProcessImpl::ProcessUnitT (ReadProgramOptions *params*)

Need to integrate multiple handling of different type ids OR resolve this issue by separate program that scan kvalobs database and identifies the value of each duplicate measurement to use ...

fixtime here for tests

StationList is all the possible stations ... Check

or is it better here ??? YES YES YES

If one or more of the analysis flags are set then will not process further!

Update if correction is out of TAN TAX range!

REMOVE ALL THE CONTROLS

and a control in the configuration file to turn totally on or off!!!!

2.4.2.5 int ProcessImpl::Interpolate (ReadProgramOptions *params*)

StationList is all the possible stations

End of experiments ... this

2.4.2.6 int ProcessImpl::ProcessSpaceCheck (ReadProgramOptions *params*)

StationList is all the possible stations

Make time step completely arbitrary etc ..... TODO

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

- ProcessImpl.h
- algorithms/Interpolate.cc
- algorithms/Process4D.cc
- algorithms/ProcessRedistribution.cc
- algorithms/ProcessSpaceCheck.cc
- algorithms/ProcessUnitT.cc
- algorithms/ProcessVariability.cc
- ProcessImpl.cc

2.5 Qc2App Class Reference

`#include <Qc2App.h>`

Public Member Functions

- **Qc2App** (int argn, char **argv, const std::string &driver_, const std::string &connect_, const char *options[ ][2]=0)
- virtual bool **isOk** () const
- bool **sendDataToKvService** (const kvalobs::kvStationInfoList &info_, bool &busy)
- CKvalObs::CService::DataReadyInput_ptr **lookUpKvService** (bool forceNS, bool &usedNS)
- void **doShutdown** ()
- bool shutdown ()
- dnmi::db::Connection * getNewDbConnection ()
- void **releaseDbConnection** (dnmi::db::Connection *con)

2.5.1 Detailed Description

Application type. The Qc2 application class: follows the standard Qc1 kvalobs model.

2.5.2 Member Function Documentation

2.5.2.1 bool Qc2App::shutdown ()

shutdown returns true when the application is in the terminating state.

2.5.2.2 dnmi::db::Connection * Qc2App::getNewDbConnection ()

Creates a new connection to the database. The caller must call releaseDbConnection after use.

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

- Qc2App.h
- Qc2App.cc

2.6 Qc2D Class Reference

`#include <Qc2D.h>`

Public Member Functions

- void **istindex** (int stid)
- std::vector< float > **intp** ()
- void **istid** (int stid)
- void **iobstime** (miutil::miTime obstime)
- void **ioriginal** (float original_value)
- void **iparamid** (int parameter)
- void **itbtime** (miutil::miTime tbtime)
- void **itypeid** (int type_id)
- void **isensor** (int sensor)
- void **ilevel** (int level)
- void **icorrected** (float corrected_value)
- void **icontrolinfo** (kvalobs::kvControlInfo controlinfo)
- void **iuseinfo** (kvalobs::kvUseInfo useinfo)
- void **icfailed** (miutil::miString cfailed)
- void **iintp** (float interp_value)
- void **iredis** (float redistributed_value)
- void **ilat** (float latitude)
- void **ilon** (float longitude)
- void **iht** (float height)
- void **icp** (float CP)
- int **stationID** () const
- **Qc2D** (std::list< kvalobs::kvData > &QD, std::list< kvalobs::kvStation > &SL, ReadProgramOptions params)
- Qc2D (std::list< kvalobs::kvData > &QD, std::list< kvalobs::kvStation > &SL, ReadProgramOptions params, std::string GenerateMissing)
- void clean ()

  Method to clear() all of the vectors held in the Qc2D data structure.

- void Qc2_interp ()

  Pointless interface to the interpolation method. Replace with interpolation algorithm strategy!!!

- void distributor (const std::list< kvalobs::kvStation > &slist, std::list< kvalobs::kvData > &ReturnData, int ClearFlag)

  Method to pass Qc2D data for redistribution of accumulated values. ((Needs to be reworked!! Encapsulate!)).

- void calculate_intp_all (unsigned int index)

  perform an inverse distanced weighted interpolation based on all the neighbours.

- void calculate_intp_temp (unsigned int index)
- void idw_intp_limit (unsigned int index)
- void intp_delaunay (unsigned int index)
- void intp_dummy (unsigned int index)

  dummy modeule |

- void intp_temp (unsigned int index)
- void calculate_intp_wet_dry (unsigned int index)
- void calculate_intp_h (unsigned int index)
- void calculate_intp_sl (unsigned int index, std::list< int > BestStations)

  perform an interpolation based on a list of allowed stations (sl).

- void calculate_trintp_sl (unsigned int index, std::list< int > BestStations)
- int SampleSemiVariogram ()

  Calculates the semivarigram for the loaded data.

- int SpaceCheck ()
- int write_cdf (const std::list< kvalobs::kvStation > &slist)

  Method to append the interpolated model data to a netCDF file.

- void **filter** (std::vector< float > &fdata, float Min, float Max, float IfMod, float Mod)

Public Attributes

- std::vector< int > **stid_**
- std::vector< miutil::miTime > **obstime_**
- std::vector< float > **original_**
- std::vector< int > **paramid_**
- std::vector< miutil::miTime > **tbtime_**
- std::vector< int > **typeid_**
- std::vector< int > **sensor_**
- std::vector< int > **level_**
- std::vector< float > **corrected_**
- std::vector< kvalobs::kvControlInfo > **controlinfo_**
- std::vector< kvalobs::kvUseInfo > **useinfo_**
- std::vector< miutil::miString > **cfailed_**
- std::vector< float > **intp_**
- std::vector< float > **redis_**
- std::vector< float > **lat_**
- std::vector< float > **lon_**
- std::vector< float > **ht_**
- std::vector< float > **CP_**
- ReadProgramOptions **params**
- std::map< int, int > **stindex**

Friends

- std::ostream & **operator**<< (std::ostream &stm, const Qc2D &Q)

2.6.1 Detailed Description

The Qc2Data object is a custom container for handling data subject to Qc2 algorithms. It is designed to hold data from the whole network and include the geographic co-ordinates and altitude of each point. The geo-statistical algorithms which make up Qc2 space controls require this information to be managed together. Each structure includes a full kvalobs::kvData record with the associated station location (height, latitude, longitude).

2.6.2 Constructor & Destructor Documentation

2.6.2.1 Qc2D::Qc2D (std::list< kvalobs::kvData > & *QD*, std::list< kvalobs::kvStation > & *SL*, ReadProgramOptions *params*, std::string *GenerateMissing*)

Includes handling of missing rows. This block will add entries for stations with no values. If an accumulated value is found for these stations a reaccumulation is performed.

Need to check this logic!!!

2.6.3 Member Function Documentation

2.6.3.1 void Qc2D::calculate_intp_temp (unsigned int *index*)

perform an inverse distanced weighted interpolation based on all the neighbours. This one modified for temperature ... working ... not for use

2.6.3.2 void Qc2D::idw_intp_limit (unsigned int *index*)

Inverse distance weighting interpolation prototype. Algorithm to construct a model value by inverse distance weighting from neighbouring stations. This optionincludes experimental investigation of various uniformity tests.

How many neighbours to use is an open question. Best option is to do the triangulation !!!!!!!!!!!!!!!!!!! ??????????????+ if (inv_dist > 0.0 && NeighboursUsed.size() > 0 && NeighboursUsed.size() < 7) {

2.6.3.3 void Qc2D::intp_delaunay (unsigned int *index*)

Interpolation based on the construction of local Delaunay triangles and subsequent linear interpolation.

Only makes sense to perform this interpolation if there are realvalues in the mesh

We have the neighbours must check to see if there are any duplicate points

Problem to solve here we triangulate the nodes but lose track of the data corresponding to the node !!!!!!

Use DEG_TO_RAD instead !!!!! ????????

Funny non-intuitive counting here array is (1+N) where n=1 is the centre??? Check all this array dimensioning ... I think it is fishy!

We are assuming that table is not reordered !!!!!

This does nothing more yet !!!!!!!!!!!!!!!!!!!

Perform "Point in triangle test" according to Barycentric Technique (see http://www.blackpawn.com/texts/pointinpoly/default.html)

PLotting test triangles gives memory fault on virtual boxes !!!!ZZ

This is wrong! It is not the right data point.

This is only for RAINFALL

if (rr[0] > params.MinimumValue && rr[1] > params.MinimumValue && rr[2] > params.MinimumValue && OKTRI){ Need something like the above for any generalisation (currently triangulate only rainfall).

2.6.3.4 void Qc2D::intp_temp (unsigned int *index*)

Inverse distance weighting interpolation prototype. Algorithm to construct a model value by inverse distance weighting from neighbouring stations. This optionincludes experimental investigation of various uniformity tests.

This is a placeholder for spatial interpretation for temperature. Currently the algorithm just picks the same value as the nearest neighbour with a measurement.

includes rudimentary height correction ...

2.6.3.5 void Qc2D::calculate_intp_wet_dry (unsigned int *index*)

Inverse distance weighting interpolation prototype. Algorithm to construct a model value by inverse distance weighting from neighbouring stations. This optionincludes experimental investigation of various uniformity tests.

MP | PP

_ _|_ _

|

MM | PM

2.6.3.6 void Qc2D::calculate_intp_h (unsigned int *index*)

Inverse distance weighting interpolation prototype. Algorithm to construct a model value by inverse distance weighting from neighbouring stations. Includes a 10 % modification to the rainfall with 100 m or altitude up to 1000m and 5% for every 100m above 1000m.

2.6.3.7 void Qc2D::calculate_trintp_sl (unsigned int *index*, std::list< int > *BestStations*)

perform interpolation based on a list triangulation points. assuming a linear trend in precipitation in the latitude and longitude directions. This is an approximation, strict interpolation should be along geodescis???

This needs checking and testing

2.6.3.8 int Qc2D::SpaceCheck ()

TODO at the moment this is just a copy of the interpolation code block, a SpaceCheck to be generated here!

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

- algorithms/Qc2D.h
- algorithms/Qc2D.cc

2.7 Qc2Work Class Reference

The main Qc2 thread.

`#include <Qc2Thread.h>`

Public Member Functions

- **Qc2Work** (Qc2App &app_, const std::string &logpath="./log")
- void **operator()** ()

2.7.1 Detailed Description

The main Qc2 thread.

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

- Qc2Thread.h
- Qc2Thread.cc

2.8 ReadProgramOptions Class Reference

Selects and reads the configuration files driving each of the Qc2 algorithms.

`#include <ReadProgramOptions.h>`

Public Member Functions

- int Parse (std::string filename)

  Parses the configuration files.

- int **SelectConfigFiles** (std::vector< std::string > &config_files)
- int clear ()

Public Attributes

- miutil::miTime **UT0**
- miutil::miTime **UT1**
- int **StepD**
- int **StepH**
- int **AlgoCode**
- int **InterpCode**
- int **LastN**
- std::string **ControlInfoString**
- std::vector< int > **ControlInfoVector**
- int **RunAtMinute**
- int **RunAtHour**
- int **pid**
- int **tid**
- int **missing**
- int **MinimumValue**
- float **InterpolationLimit**
- bool **newfile**
- std::map< int, unsigned char > **zflag**
- std::map< int, unsigned char > **Rflag**
- std::map< int, unsigned char > **Iflag**
- std::map< int, unsigned char > **Aflag**
- std::map< int, unsigned char > **Wflag**
- std::map< int, unsigned char > **Sflag**
- std::vector< unsigned char > **Vfqclevel**
- std::vector< unsigned char > **Vfr**
- std::vector< unsigned char > **Vfcc**
- std::vector< unsigned char > **Vfs**
- std::vector< unsigned char > **Vfnum**
- std::vector< unsigned char > **Vfpos**
- std::vector< unsigned char > **Vfmis**
- std::vector< unsigned char > **Vftime**
- std::vector< unsigned char > **Vfw**
- std::vector< unsigned char > **Vfstat**
- std::vector< unsigned char > **Vfcp**

- std::vector< unsigned char > **Vfclim**
- std::vector< unsigned char > **Vfd**
- std::vector< unsigned char > **Vfpre**
- std::vector< unsigned char > **Vfcombi**
- std::vector< unsigned char > **Vfhqc**

2.8.1 Detailed Description

Selects and reads the configuration files driving each of the Qc2 algorithms.

2.8.2 Member Function Documentation

2.8.2.1 int ReadProgramOptions::Parse (std::string *filename*)

Parses the configuration files.

could also rely on fmis here !!??

2.8.2.2 int ReadProgramOptions::clear ()

Check these are cleared correctly

TBD

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

- ReadProgramOptions.h
- ReadProgramOptions.cc

2.9 kvQABase::script_par Struct Reference

script parameter

`#include <kvQABaseTypes.h>`

Public Attributes

- std::string signature
  varname in script
- std::string name
  official parametername
- int paramid
  official parameterid
- int sensor
  sensor
- int level
  level
- int typeID
  typeID
- bool normal
  normal parameter (data or model_data)
- std::map< int, std::map< int, par_values > > values

2.9.1 Detailed Description

script parameter

2.9.2 Member Data Documentation

2.9.2.1 std::map<int, std::map<int, par_values> > kvQABase::script_par::values

timeseries (timeoffset/value hash) hashed by stationid

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

- algorithms/kvQABaseTypes.h

2.10 kvQABase::script_var Struct Reference

script variables from one source

`#include <kvQABaseTypes.h>`

Public Member Functions

- void **clear** ()

Public Attributes

- int dsource
  data_source type
- std::string source
  sourcename of data (obs,refobs,model,meta)
- int timestart
  starttime in minutes from obstime
- int timestop
  stoptime in minutes from obstime
- bool missing_data
  any missing data for this source
- bool allnormal
  only normal variables in pars
- std::vector< int > allpos
  all station-ids
- std::vector< int > alltimes
  all timeoffsets for source
- std::vector< script_par > pars
  each parameter

2.10.1 Detailed Description

script variables from one source

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

- algorithms/kvQABaseTypes.h

2.11 stopwatch Class Reference

`#include <StopWatch.h>`

2.11.1 Detailed Description

Simple class to record cpu time usage for performance monitoring during development and testing.

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

- StopWatch.cc
- StopWatch.h

Index