

kvQc2

Generated by Doxygen 1.5.5

Tue Oct 27 21:45:23 2009



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>Class Documentation</b>	<b>3</b>
2.1	CheckedDataCommandBase Class Reference . . . . .	3
2.2	Distribute Class Reference . . . . .	4
2.3	kvQABase::par_values Struct Reference . . . . .	5
2.4	ProcessImpl Class Reference . . . . .	6
2.5	Qc2App Class Reference . . . . .	8
2.6	Qc2D Class Reference . . . . .	9
2.7	Qc2Work Class Reference . . . . .	14
2.8	ReadProgramOptions Class Reference . . . . .	15
2.9	kvQABase::script_par Struct Reference . . . . .	17
2.10	kvQABase::script_var Struct Reference . . . . .	18
2.11	stopwatch Class Reference . . . . .	19



# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">CheckedDataCommandBase</a> . . . . .	3
<a href="#">Distribute</a> (The class manages the redistribution of 24 hour precipitation data ) . . . . .	4
<a href="#">kvQABase::par_values</a> (Parametervalue, -missingstatus and -controlflag ) . . . . .	5
<a href="#">ProcessImpl</a> (Handles the interface to different processing algorithms ) . . . . .	6
<a href="#">Qc2App</a> . . . . .	8
<a href="#">Qc2D</a> . . . . .	9
<a href="#">Qc2Work</a> (The main Qc2 thread ) . . . . .	14
<a href="#">ReadProgramOptions</a> (Selects and reads the configuration files driving each of the Qc2 algorithms ) . . . . .	15
<a href="#">kvQABase::script_par</a> (Script parameter ) . . . . .	17
<a href="#">kvQABase::script_var</a> (Script variables from one source ) . . . . .	18
<a href="#">stopwatch</a> . . . . .	19



## 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 m m m m 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

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

Update if correction is out of TAN TAX range!

#### 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 std)
- std::vector< float > **intp** ()
- void **istid** (int std)
- 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 > &slst, 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 semivariogram 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 option includes 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 real values 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 option includes 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 option includes 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 geodesics???

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!

#### 2.6.3.9 int Qc2D::write\_cdf (const std::list< kvalobs::kvStation > & *slist*)

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

Fix this here

```
int month=obstime_[0].month(); ***** Fix this here int month=obstime_[0].month();
***** Fix this here int month=obstime_[0].month(); ***** Fix this here int
month=obstime_[0].month(); ***** Fix this 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 > **Vfqcleve**
- 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 > Vfhq`

### 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

- calculate\_intp\_h
  - Qc2D, [12](#)
- calculate\_intp\_temp
  - Qc2D, [11](#)
- calculate\_intp\_wet\_dry
  - Qc2D, [12](#)
- calculate\_trintp\_sl
  - Qc2D, [12](#)
- CheckedDataCommandBase, [3](#)
- clear
  - ReadProgramOptions, [16](#)
- Distribute, [4](#)
- getNewDbConnection
  - Qc2App, [8](#)
- idw\_intp\_limit
  - Qc2D, [11](#)
- Interpolate
  - ProcessImpl, [7](#)
- intp\_de launay
  - Qc2D, [11](#)
- intp\_temp
  - Qc2D, [11](#)
- kvQABase::par\_values, [5](#)
- kvQABase::script\_par, [17](#)
  - values, [17](#)
- kvQABase::script\_var, [18](#)
- Parse
  - ReadProgramOptions, [16](#)
- Process4D
  - ProcessImpl, [6](#)
- ProcessImpl, [6](#)
  - Interpolate, [7](#)
  - Process4D, [6](#)
  - ProcessSpaceCheck, [7](#)
  - ProcessUnitT, [7](#)
  - Redistribute, [6](#)
  - Variability, [6](#)
- ProcessSpaceCheck
  - ProcessImpl, [7](#)
- ProcessUnitT
  - ProcessImpl, [7](#)
- Qc2App, [8](#)
  - getNewDbConnection, [8](#)
  - shutdown, [8](#)
- Qc2D, [9](#)
  - calculate\_intp\_h, [12](#)
  - calculate\_intp\_temp, [11](#)
  - calculate\_intp\_wet\_dry, [12](#)
  - calculate\_trintp\_sl, [12](#)
  - idw\_intp\_limit, [11](#)
  - intp\_de launay, [11](#)
  - intp\_temp, [11](#)
  - Qc2D, [11](#)
  - SpaceCheck, [12](#)
  - write\_cdf, [12](#)
- Qc2Work, [14](#)
- ReadProgramOptions, [15](#)
  - clear, [16](#)
  - Parse, [16](#)
- Redistribute
  - ProcessImpl, [6](#)
- shutdown
  - Qc2App, [8](#)
- SpaceCheck
  - Qc2D, [12](#)
- stopwatch, [19](#)
- values
  - kvQABase::script\_par, [17](#)
- Variability
  - ProcessImpl, [6](#)
- write\_cdf
  - Qc2D, [12](#)