Dikes Overtopping Kernel - Technical documentation

Generated by Doxygen 1.8.9.1

Tue Sep 29 2015 10:07:35

Contents

Chapter 1

Todo List

Subprogram conversionfunctions::betafromq (q, beta)

Is it necessay to extend the routine betaFromQ with a check on the probability of non-exceedance (q) and a error message? Q has to be a between 0 and 1.

This routine is extreemly often called! Therefore with this check the speed of the computation is decreased. There is another argument not to implement this check: Sometimes by combining the computed probabilities the combined probability is slightly more than 1 or slightly less than 0 (errors of rounding), then the computation will end eith this error message. Now this error of rounding is repaired in this routine betaFromQ. If a probability is outside the allowed area the probability is changed to the lower or upper bound. The disadvantage of this method is that programming errors are no longer visible: a probability of for intance two doesn't generate an error!

Module Feedback

Add multilingual support

Chapter 2

Module Index

2.1 Libraries

Here is a list of all modules:

Chapter 3

Modules Index

3.1 Modules List

Here is a list of all modules with brief descriptions:

Chapter 4

Data Type Index

4.1 Class List

Here are the data types with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Module Documentation

6.1 Feedback library

The feedback library provides a mechanism for reporting errors and progress.

Functions/Subroutines

- integer function feedback::p ()

  Returns the pointer to the feedback dll, avoiding many loadlibrary calls.
- logical function, public feedback::todll ()

  Is feedback writing to the LogMessage function in HydraRing.IO.Native.dll ?
- logical function, public feedback::fatalerroroccured (value)

  Has a fatal error occured? Mainly used in unit tests.
- subroutine, public feedback::resetfatalerroroccured ()

  Reset fatal error occurance Mainly used in unit tests.
- subroutine feedback::setfatalerroroccured (value)

  Set fatal error occurance Mainly used in unit tests.
- logical function, public feedback::getfatalerrorexpected (value)

  Get value of FatalErrorExpected Mainly used in unit tests.
- subroutine, public feedback::setfatalerrorexpected (value)

  Set the value of FatalErrorExpected Mainly used in unit tests.
- logical function, public feedback::allow_log ()

  Is logging active?
- subroutine, public feedback::setallowlogging (value)

  Set logging on/off.
- subroutine, public feedback::getthreadid (threadId)

  Get the tread ID Note that multi threading is not implemented, so always the value 0 is returned.
- subroutine, public feedback::setfatalerroraction (action)

  Set the value of FatalErrorAction possible values: onfatalerrorStop = 1 : Action on fatal error: exit the program onfatalerrorThrowException = 2 : Action on fatal error: throw exception (for use as dll) onfatalerrorUnittest = 3 : Action on fatal error: handle as unit test.
- subroutine, public feedback::getfatalerrormessage (message)

  Get the last of FatalErrorMessage.
- subroutine, public feedback::feedbackerror (error)

  Return the error status of the current thread If an error has been reported (via the errorMessage routines), this routine will return true, otherwise false.
- subroutine, public feedback::feedbackclose

Close the log file.

- subroutine, public feedback::feedbackinitialise (output, logfile, verbosity)

  Initialise the feedback subsystem.
- subroutine feedback::progressreporttxt (level, msgtext)

  Write a single message to the output for the benefit of the user.
- subroutine feedback::progressreportint (level, msgtext, value)

  Write a single message with an integer value.
- subroutine feedback::progressreportreal (level, msgtext, value)

  Write a single message with a real value.
- subroutine feedback::progressreportdouble (level, msgtext, value)

  Write a single message with a double precision real value.
- subroutine feedback::progressreportarray (level, msgText, x, size)

  Print array values to report file.
- subroutine feedback::progressreportvalueandarray (level, msgText, z, x, size)

  Print z-value and array values to report file.
- subroutine, public feedback::feedbackenter (routine)

  Register a new routine level.
- subroutine, public feedback::feedbackleave

  Drop a routine level from the stack trace.
- subroutine, public feedback::feedbackprintstack

  Print the stack trace as known via feedbackEnter and feedbackLeave.
- subroutine feedback::errormessagetxt (msgtext)

  Write an error message.
- subroutine feedback::errormessageint (msgtext, value)

  Write an error message with an integer value.
- subroutine feedback::errormessagereal (msgtext, value)

  Write an error message with a real value.
- subroutine feedback::errormessagedouble (msgtext, value)

  Write an error message with a double precision real value.
- subroutine feedback::warningmessagetxt (msgtext)

  Write a warning message.
- subroutine feedback::warningmessageint (msgtext, value)

  Write a warning message with an integer value.
- subroutine feedback::warningmessagereal (msgtext, value)

  Write a warning message with a real value.
- subroutine feedback::warningmessagedouble (msgtext, value)

  Write a warning message with a double precision real value.
- subroutine feedback::fatalerrortxt (msgtext)

  Write a fatal error message.
- subroutine feedback::fatalerrorint (msgtext, value)

  Write a fatal error message with an integer value.
- subroutine feedback::fatalerrorreal (msgtext, value)

  Write a fatal error message with a real value.
- subroutine feedback::fatalerrordouble (msgtext, value)

  Write a fatal error message with a double precision real value.
- subroutine, public feedback::feedbackdropcontext

  Drop the entire context.
- subroutine, public feedback::feedbackprintcontext

  Register a context for error messages.
- subroutine feedback::feedbackcontexttxt (text)

  Register a context for error messages.

- subroutine feedback::feedbackcontextint (text, value)

  Register a context for error messages - with integer value.
- subroutine feedback::feedbackcontextreal (text, value)

  Register a context for error messages - with real value.
- subroutine feedback::feedbackcontextdouble (text, value)

  Register a context for error messages - with double precision real value.
- subroutine, public feedback::throw (msgText)

  Stop execution by throwing an exception Useful if running in a dll and main program is a GUI in e.g. C++ or C#.

Variables

- integer, parameter feedback_parameters::onlyscreen = 0

  Mode: write to screen only.
- integer, parameter feedback_parameters::onlyfile = 1

  Mode: write to file only.
- integer, parameter feedback_parameters::screenandfile = 2

  Mode: write to both screen and file.
- integer, parameter feedback_parameters::dll = 3

  Mode: write to dll.
- integer, parameter feedback_parameters::verbosenone = -1

  Reporting: NO messages.
- integer, parameter feedback_parameters::verbosebasic = 0

  Reporting: basic messages only.
- integer, parameter feedback_parameters::verbosedetailed = 1

  Reporting: detailed messages (including traceback)
- integer, parameter feedback_parameters::verbosedebugging = 2

  Reporting: all messages.

6.1.1 Detailed Description

The feedback library provides a mechanism for reporting errors and progress.

Short documentation of intended usage:

The library consists of the following types of routines:

- Managing the library itself
- Write progress messages (detailed or basic)
- Write error messages
- Define the context of the computation

Managing the library: The routine `feedbackInitialise()` initialises the library with:

- Whether to write to screen only or to a file or both
- Name of the log file
- Detailed output or not (parameters: verboseNone to verboseDebugging)

For performance reasons, you can set and examine the variable `traceback`. This variable indicates whether keeping track of the stack is required. Typical usage in tight loops:

```
use feedback

...
if ( traceback ) call feedbackenter( "section or routine" )
...
if ( traceback ) call feedbackleave
```

See below for more details

(Note: initialisation has to be done for each thread and each thread should have its own logfile to avoid interlacing the messages)

Write progress messages: Progress messages tell the user what the program is doing. For users who are interested in the details of the computation the level of detail can be set high, for others a minimum amount of information is printed.

Use the routine `progressReport` to print a short message indicating the stage in the computation.

To communicate with the user-interface, use the `progressBarPercent` or `progressBarIncrement` routines.

Write error messages: Error messages are written with the context in which the error occurred and if details are required with the known stack.

An error message is written with the subroutine `errorMessage`, a warning with `warningMessage` and a fatal error (which stops the program) with `fatalError`

To indicate where in the program the error occurs, you can use the subroutine `feedbackEnter` - this routine registers where in the program (source) you are, this may be a routine you enter but also a section within the computation that is of interest. If you leave the subroutine or section, use `feedbackLeave` to indicate this. You probably should not use it inside a tight loop, but here is a sketch of typical use:

```
subroutine examplecomputation( ... )
use feedback
...
if ( traceback ) call feedbackenter( "exampleComputation" )
... some computation ...

if ( traceback ) call feedbackenter( "Main do-loop" )
do i = 1,1000000

    ... some computation ...
enddo
if ( traceback ) call feedbackleave
if ( traceback ) call feedbackleave
```

Note: `feedbackEnter` and `feedbackLeave` must be used in matching pairs!

Define the context of the computation: A bare error message may not be enough to understand what went wrong. For the purpose of standardising the extra information that should accompany an error message, the library offers the routine `feedbackContext`. You can use it to define relevant text to be shown in conjunction with the error message. For instance: the name of the computation method that is being run or the case that you are dealing with.

To define a completely new context, use `feedbackDropContext`.

Todo Add multilingual support

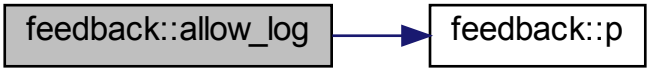
6.1.2 Function/Subroutine Documentation

6.1.2.1 logical function, public feedback::allow_log ( )

Is logging active?

Definition at line 273 of file feedback.f90.

Here is the call graph for this function:



6.1.2.2 subroutine feedback::errormessagedouble (character(len=∗), intent(in) *msgtext,* real(kind=dp), intent(in) *value*) [private]

Write an error message with a double precision real value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 661 of file feedback.f90.

6.1.2.3 subroutine feedback::errormessageint (character(len=∗), intent(in) *msgtext,* integer, intent(in) *value*) [private]

Write an error message with an integer value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 631 of file feedback.f90.

6.1.2.4 subroutine feedback::errormessagereal (character(len=∗), intent(in) *msgtext,* real, intent(in) *value*) [private]

Write an error message with a real value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 646 of file feedback.f90.

6.1.2.5 subroutine feedback::errormessagetxt (character(len=∗), intent(in) *msgtext*) [private]

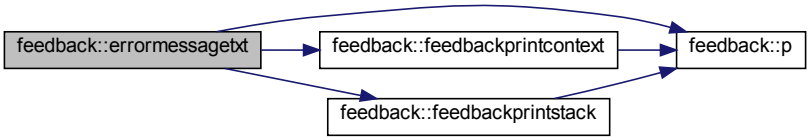
Write an error message.

Parameters

in	*msgtext*	Error message to write

Definition at line 608 of file feedback.f90.

Here is the call graph for this function:



6.1.2.6 subroutine feedback::fatalerrordouble (character(len=∗), intent(in) *msgtext,* real(kind=dp), intent(in) *value*) [private]

Write a fatal error message with a double precision real value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 815 of file feedback.f90.

6.1.2.7 subroutine feedback::fatalerrorint (character(len=∗), intent(in) *msgtext,* integer, intent(in) *value*) [private]

Write a fatal error message with an integer value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

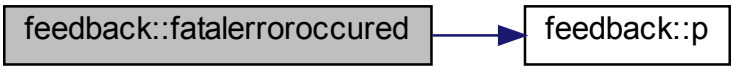
Definition at line 785 of file feedback.f90.

6.1.2.8 logical function, public feedback::fatalerroroccured (*result,* *value*)

Has a fatal error occured? Mainly used in unit tests.

Definition at line 212 of file feedback.f90.

Here is the call graph for this function:



6.1.2.9 subroutine feedback::fatalerrorreal (character(len=∗), intent(in) *msgtext,* real, intent(in) *value*) [private]

Write a fatal error message with a real value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 800 of file feedback.f90.

6.1.2.10 subroutine feedback::fatalerrortxt (character(len=∗), intent(in) *msgtext*) [private]

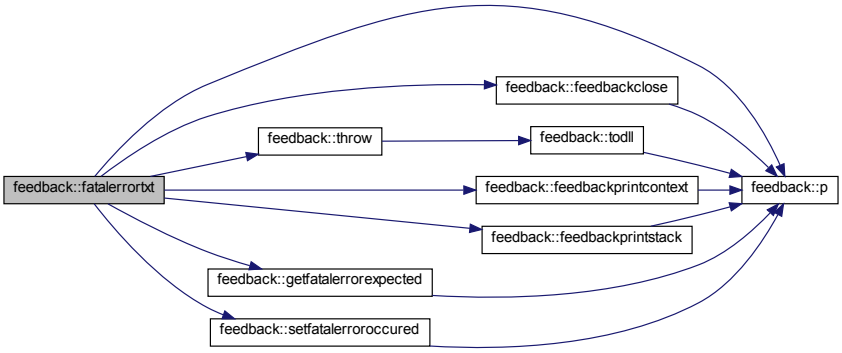
Write a fatal error message.

Parameters

in	*msgtext*	Error message to write

Definition at line 744 of file feedback.f90.

Here is the call graph for this function:

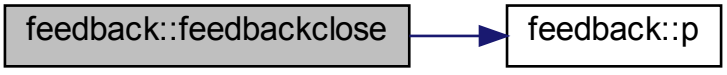


6.1.2.11 subroutine, public feedback::feedbackclose ( )

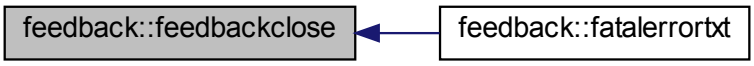
Close the log file.

Definition at line 355 of file feedback.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.12 subroutine feedback::feedbackcontextdouble (character(len=∗), intent(in) *text,* real(kind=dp), intent(in) *value*) [private]

Register a context for error messages - with double precision real value.

Parameters

in	*text*	text to be added to the context stack
in	*value*	number to be added to the text

Definition at line 900 of file feedback.f90.

6.1.2.13 subroutine feedback::feedbackcontextint (character(len=∗), intent(in) *text,* integer, intent(in) *value*) [private]

Register a context for error messages - with integer value.

Parameters

in	*text*	text to be added to the context stack
in	*value*	number to be added to the text

Definition at line 870 of file feedback.f90.

6.1.2.14 subroutine feedback::feedbackcontextreal (character(len=∗), intent(in) *text,* real(kind=sp), intent(in) *value*) [private]

Register a context for error messages - with real value.

Parameters

in	*text*	text to be added to the context stack
in	*value*	number to be added to the text

Definition at line 885 of file feedback.f90.

6.1.2.15 subroutine feedback::feedbackcontexttxt (character(len=∗), intent(in) *text*) [private]

Register a context for error messages.

Parameters

in	*text*	text to be added to the context stack

Definition at line 852 of file feedback.f90.

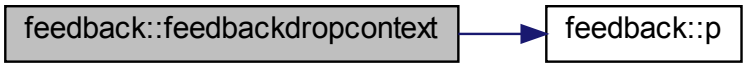
Here is the call graph for this function:

6.1.2.16 subroutine, public feedback::feedbackdropcontext ( )

Drop the entire context.

Definition at line 830 of file feedback.f90.

Here is the call graph for this function:



6.1.2.17 subroutine, public feedback::feedbackenter (character(len=*) *routine*)

Register a new routine level.

Parameters

routine	Name of the routine to add

Definition at line 563 of file feedback.f90.

Here is the call graph for this function:

6.1.2.18 subroutine, public feedback::feedbackerror (logical *error*)

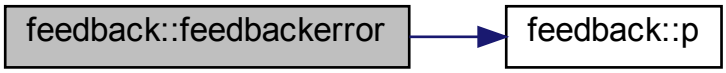
Return the error status of the current thread If an error has been reported (via the errorMessage routines), this routine will return true, otherwise false.

Parameters

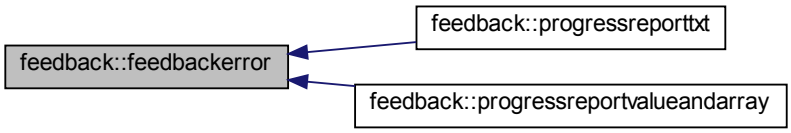
error	True if there has been some error reported

Definition at line 341 of file feedback.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.19 subroutine, public feedback::feedbackinitialise (integer, intent(in) *output,* character(len=∗), intent(in) *logfile,* integer, intent(in) *verbosity*)

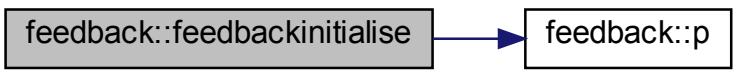
Initialise the feedback subsystem.

Parameters

in	*output*	option: output to screen or file
in	*logfile*	name of the log file to be used
in	*verbosity*	level for displaying the messages

Definition at line 382 of file feedback.f90.

Here is the call graph for this function:

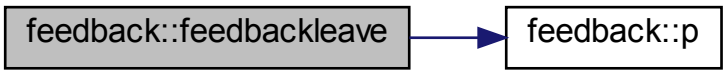


6.1.2.20 subroutine, public feedback::feedbackleave ( )

Drop a routine level from the stack trace.

Definition at line 580 of file feedback.f90.

Here is the call graph for this function:



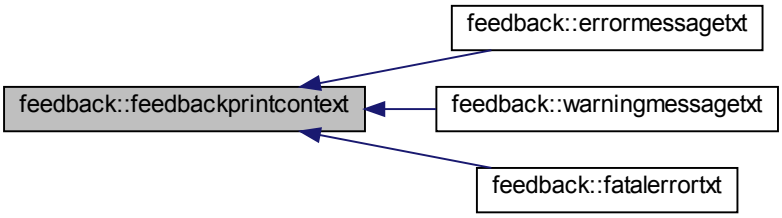
6.1.2.21 subroutine, public feedback::feedbackprintcontext ( )

Register a context for error messages.

Definition at line 841 of file feedback.f90.

Here is the call graph for this function:

Here is the caller graph for this function:



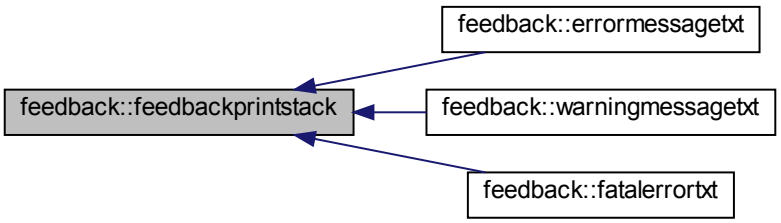
6.1.2.22 subroutine, public feedback::feedbackprintstack ( )

Print the stack trace as known via feedbackEnter and feedbackLeave.

Definition at line 597 of file feedback.f90.

Here is the call graph for this function:

Here is the caller graph for this function:



6.1.2.23 logical function, public feedback::getfatalerrorexpected (*result*, *value*)

Get value of FatalErrorExpected Mainly used in unit tests.

Definition at line 249 of file feedback.f90.

Here is the call graph for this function:

Here is the caller graph for this function:

6.1.2.24 subroutine, public feedback::getfatalerrormessage (character(len=∗), intent(out) *message*)

Get the last of FatalErrorMessage.

Parameters

out	*message*	the last error message

Definition at line 326 of file feedback.f90.

Here is the call graph for this function:

6.1.2.25 subroutine, public feedback::getthreadid (integer, intent(out) *threadId*)

Get the tread ID Note that multi threading is not implemented, so always the value 0 is returned.

Parameters

out	*threadid*	the current thread ID (always 0)

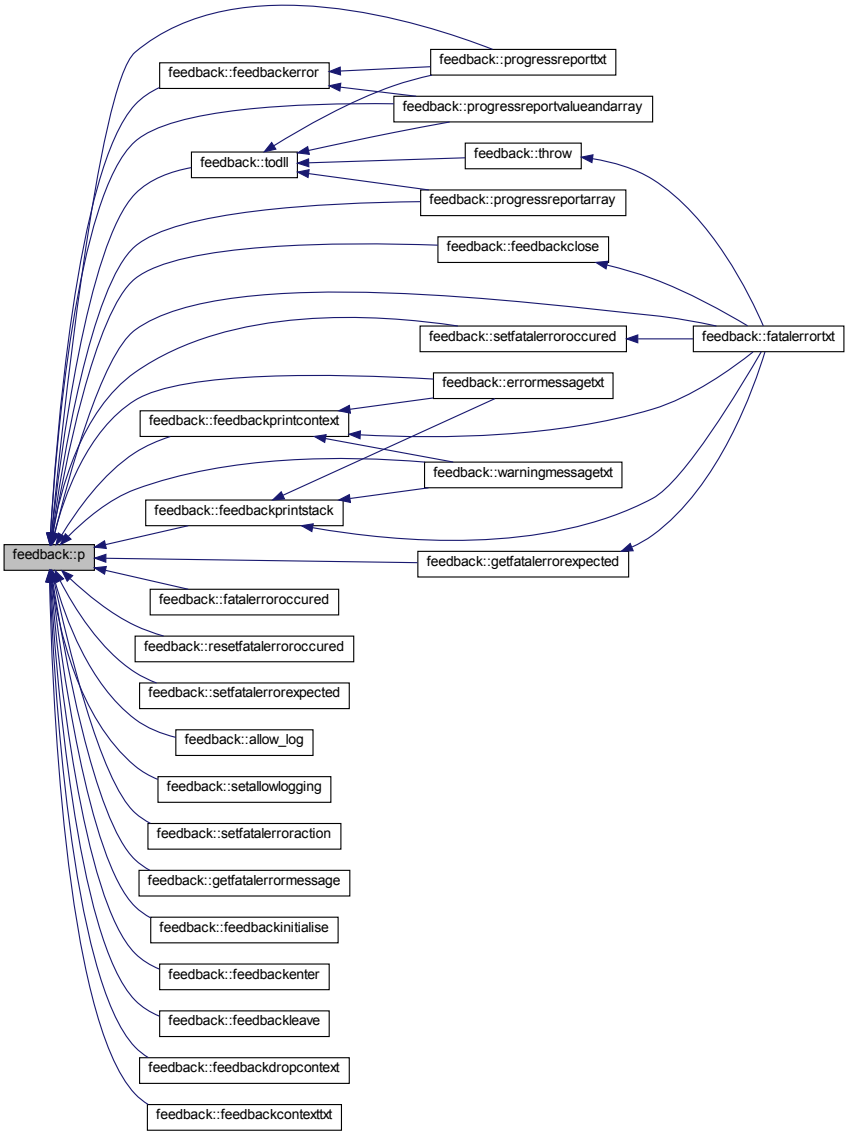
Definition at line 300 of file feedback.f90.

6.1.2.26 integer function feedback::p ( ) [private]

Returns the pointer to the feedback dll, avoiding many loadlibrary calls.

Definition at line 186 of file feedback.f90.

Here is the caller graph for this function:



6.1.2.27 subroutine feedback::progressreportarray (integer, intent(in) *level,* character(len=∗), intent(in) *msgText,* real(kind=wp), dimension(:), intent(in) *x,* integer, intent(in) *size*) [private]

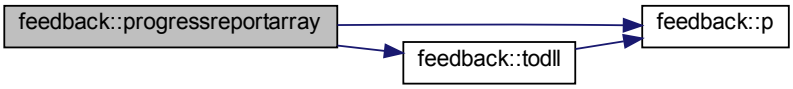
Print array values to report file.

Parameters

in	*level*	Level of verbosity
in	*msgtext*	Error message to write
in	*x*	x vector to write
in	*size*	the length of x

Definition at line 489 of file feedback.f90.

Here is the call graph for this function:



6.1.2.28 subroutine feedback::progressreportdouble (integer, intent(in) *level,* character(len=∗), intent(in) *msgtext,* real(kind=dp), intent(in) *value*) [private]

Write a single message with a double precision real value.

Parameters

in	*level*	Level of verbosity
in	*msgtext*	Error message to write
in	*value*	Double value to add

Definition at line 472 of file feedback.f90.

6.1.2.29 subroutine feedback::progressreportint (integer, intent(in) *level,* character(len=∗), intent(in) *msgtext,* integer, intent(in) *value*) [private]

Write a single message with an integer value.

Parameters

in	*level*	Level of verbosity
in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 440 of file feedback.f90.

6.1.2.30 subroutine feedback::progressreportreal (integer, intent(in) *level,* character(len=∗), intent(in) *msgtext,* real, intent(in) *value*) [private]

Write a single message with a real value.

Parameters

in	*level*	Level of verbosity
in	*msgtext*	Error message to write
in	*value*	Real value to add

Definition at line 456 of file feedback.f90.

6.1.2.31 subroutine feedback::progressreporttxt (integer, intent(in) *level,* character(len=∗), intent(in) *msgtext*) [private]

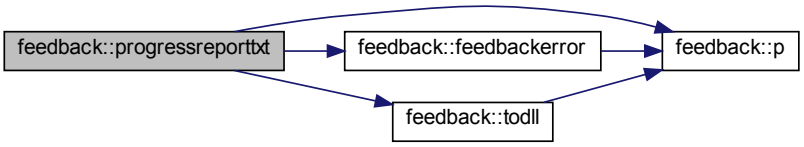
Write a single message to the output for the benefit of the user.

Parameters

in	*level*	Level of verbosity
in	*msgtext*	Text to write

Definition at line 400 of file feedback.f90.

Here is the call graph for this function:



6.1.2.32 subroutine feedback::progressreportvalueandarray (integer, intent(in) *level,* character(len=∗), intent(in) *msgText,* real(kind=wp), intent(in) *z,* real(kind=wp), dimension(:), intent(in) *x,* integer, intent(in) *size*) [private]

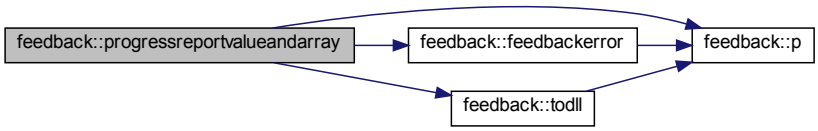
Print z-value and array values to report file.

Parameters

in	*level*	Level of verbosity
in	*msgtext*	Error message to write
in	*z*	z value to write
in	*x*	x vector to write
in	*size*	the length of x

Definition at line 525 of file feedback.f90.

Here is the call graph for this function:

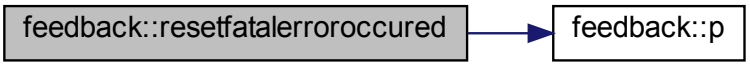


6.1.2.33 subroutine, public feedback::resetfatalerroroccured ( )

Reset fatal error occurance Mainly used in unit tests.

Definition at line 224 of file feedback.f90.

Here is the call graph for this function:



6.1.2.34 subroutine, public feedback::setallowlogging (logical, intent(in) *value*)

Set logging on/off.

Parameters

in	*value*	the value of allow_logging to be set

Definition at line 286 of file feedback.f90.

Here is the call graph for this function:

6.1.2.35 subroutine, public feedback::setfatalerroraction (integer, intent(in) *action*)

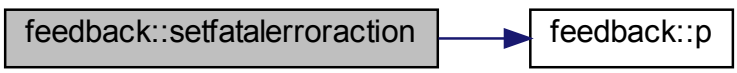
Set the value of FatalErrorAction possible values: onfatalerrorStop = 1 : Action on fatal error: exit the program onfatalerrorThrowException = 2 : Action on fatal error: throw exception (for use as dll) onfatalerrorUnittest = 3 : Action on fatal error: handle as unit test.

Parameters

in	*action*	the value of FatalErrorAction to be set (1, 2 or 3)

Definition at line 313 of file feedback.f90.

Here is the call graph for this function:



6.1.2.36 subroutine, public feedback::setfatalerrorexpected (logical, intent(in) *value*)

Set the value of FatalErrorExpected Mainly used in unit tests.

Parameters

in	*value*	the value of FatalErrorExpected to be set

Definition at line 261 of file feedback.f90.

Here is the call graph for this function:

6.1.2.37 subroutine feedback::setfatalerroroccured (logical, intent(in) *value*) [private]

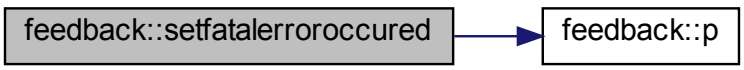
Set fatal error occurance Mainly used in unit tests.

Parameters

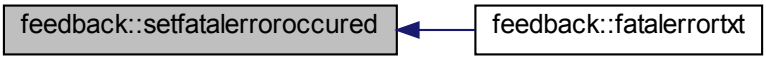
in	*value*	the value of FatalErrorOccured to be set

Definition at line 236 of file feedback.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.38 subroutine, public feedback::throw (character(len=∗), intent(in) *msgText*)

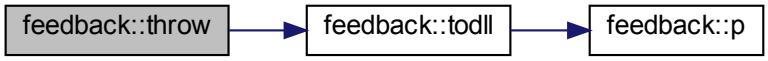
Stop execution by throwing an exception Useful if running in a dll and main program is a GUI in e.g. C++ or C#.

Parameters

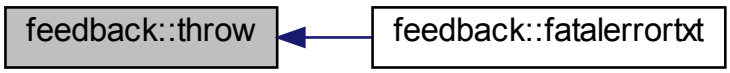
in	*msgtext*	Error message to write

Definition at line 917 of file feedback.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

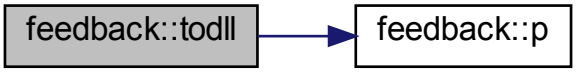


6.1.2.39 logical function, public feedback::todll ( )

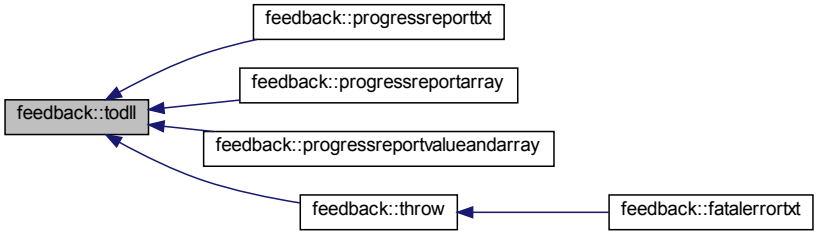
Is feedback writing to the LogMessage function in HydraRing.IO.Native.dll ?

Definition at line 200 of file feedback.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.40 subroutine feedback::warningmessagedouble (character(len=∗), intent(in) *msgtext,* real(kind=dp), intent(in) *value*) [private]

Write a warning message with a double precision real value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Real (double) value to add

Definition at line 729 of file feedback.f90.

6.1.2.41 subroutine feedback::warningmessageint (character(len=∗), intent(in) *msgtext,* integer, intent(in) *value*) [private]

Write a warning message with an integer value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 699 of file feedback.f90.

6.1.2.42 subroutine feedback::warningmessagereal (character(len=∗), intent(in) *msgtext,* real, intent(in) *value*) [private]

Write a warning message with a real value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Real value to add

Definition at line 714 of file feedback.f90.

6.1.2.43 subroutine feedback::warningmessagetxt (character(len=∗), intent(in) *msgtext*) [private]

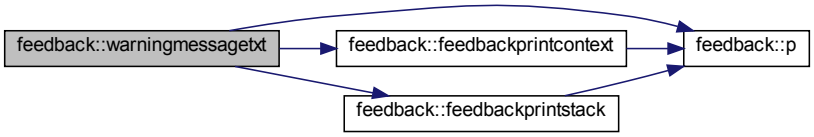
Write a warning message.

Parameters

in	*msgtext*	Error message to write

Definition at line 676 of file feedback.f90.

Here is the call graph for this function:



6.1.3 Variable Documentation

6.1.3.1 integer, parameter feedback_parameters::dll = 3

Mode: write to dll.

Definition at line 17 of file feedback_parameters.f90.

6.1.3.2 integer, parameter feedback_parameters::onlyfile = 1

Mode: write to file only.

Definition at line 15 of file feedback_parameters.f90.

6.1.3.3 integer, parameter feedback_parameters::onlyscreen = 0

Mode: write to screen only.

Definition at line 14 of file feedback_parameters.f90.

6.1.3.4 integer, parameter feedback_parameters::screenandfile = 2

Mode: write to both screen and file.

Definition at line 16 of file feedback_parameters.f90.

6.1.3.5 integer, parameter feedback_parameters::verbosebasic = 0

Reporting: basic messages only.

Definition at line 19 of file feedback_parameters.f90.

6.1.3.6 integer, parameter feedback_parameters::verbosedebugging = 2

Reporting: all messages.

Definition at line 21 of file feedback_parameters.f90.

6.1.3.7 integer, parameter feedback_parameters::verbosedetailed = 1

Reporting: detailed messages (including traceback)

Definition at line 20 of file feedback_parameters.f90.

6.1.3.8 integer, parameter feedback_parameters::verbosenone = -1

Reporting: NO messages.

Definition at line 18 of file feedback_parameters.f90.

Chapter 7

Module Documentation

7.1 angleutilities Module Reference

Module with direction utility functions.

Functions/Subroutines

- real(kind=wp) function anglebetween2directions (direction1, direction2)

  Function angleBetween2Directions() for the computation of the angle between two directions.
  The two directions must be given in degrees and within [0, 360]
  The computed angle between these two directions is also in degrees, and within [0, 180]
  .

7.1.1 Detailed Description

Module with direction utility functions.

7.1.2 Function/Subroutine Documentation

7.1.2.1 real(kind= wp) function angleutilities::anglebetween2directions (real(kind= wp), intent(in) *direction1,* real(kind= wp), intent(in) *direction2*)

Function angleBetween2Directions() for the computation of the angle between two directions.
The two directions must be given in degrees and within [0, 360]
The computed angle between these two directions is also in degrees, and within [0, 180]
.

Parameters

in	*direction1*	first direction (an angle in degrees)
in	*direction2*	second direction (an angle in degrees)

Returns

angle between the two directions

Definition at line 24 of file angleUtilities.f90.

7.2 conversionfunctions Module Reference

Functions/Subroutines

- subroutine, public pqfrombeta (beta, p, q)

  Subroutine for computing the exceedance and non-exceedance probabilities for a given reliability index (beta) assuming a standard normal distribution.
- real(kind=wp) function, public pfrombeta (beta)

  Function for computing the non-exceedance probability (P) for a given reliability index (beta), assuming a standard normal distribution.

  .
- real(kind=wp) function, public qfrombeta (beta)

  Function for computing the exceedance probability (Q) for a given reliability index (beta), assuming a standard normal distribution.

  .
- subroutine, public returntimefrombeta (beta, returnTime)

  Subroutine for computing the return time for a given reliability index (beta) assuming a standard normal distribution.
- subroutine, public freqfrombeta (beta, freq)

  Subroutine for computing the frequency for a given reliability index (beta) assuming a standard normal distribution.
- subroutine, public logqfrombeta (beta, logQ)

  Subroutine for computing the (negative) logarithm of the non-exceedance probability for a given reliability index (beta) assuming a standard normal distribution.
- subroutine, public betafromq (q, beta)

  Subroutine for computing the reliability index (beta) for a given exceedance probability. The reliability index is corrected with the subroutine PQfromBeta.

Variables

- real(kind=wp), parameter, private qmin = 1.0d-300
- real(kind=wp), parameter, private upperlog = 700.0d0
- real(kind=wp), parameter, private ulimit = 5.6d0

7.2.1 Function/Subroutine Documentation

7.2.1.1 subroutine, public conversionfunctions::betafromq (real(kind=wp), intent(in) *q*, real(kind=wp), intent(out) *beta*)

Subroutine for computing the reliability index (beta) for a given exceedance probability. The reliability index is corrected with the subroutine PQfromBeta.

This routine is essentially the code for a standard normal distribution, in which a probability is input and a u-value (beta) is returned.

Note: Theoretically the probability q has to be between 0.0 and 1.0
Values outside this range can be used in this routine but are set equal to the bounds of the theoretical range:

- values larger than 1.0 are set equal to 1.0 (or, to be precisely: 1 - 1e-300), and
- values smaller than 0.0 are set equal to 0.0 (or, to be precisely: 1e-300)
  This is done with the construction "qsr = max(qsr, 1.0d-300)"

Todo Is it necessay to extend the routine betaFromQ with a check on the probability of non-exceedance (q) and a error message? Q has to be a between 0 and 1.

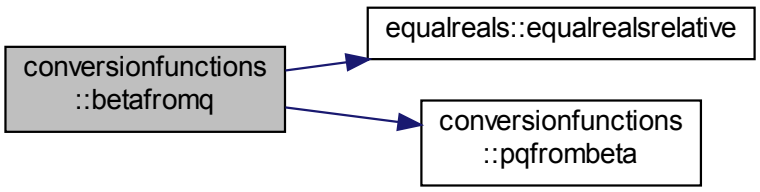
This routine is extreemly often called! Therefore with this check the speed of the computation is decreased. There is another argument not to implement this check: Sometimes by combining the computed probabilities the combined probability is slightly more than 1 or slightly less than 0 (errors of rounding), then the computation will end eith this error message. Now this error of rounding is repaired in this routine betaFromQ. If a probability is outside the allowed area the probability is changed to the lower or upper bound. The disadvantage of this method is that programming errors are no longer visible: a probability of for intance two doesn't generate an error!

Parameters

in	*q*	Probability of exceedance
out	*beta*	Reliability index

Definition at line 234 of file conversionFunctions.f90.

Here is the call graph for this function:



7.2.1.2 subroutine, public conversionfunctions::freqfrombeta (real(kind=wp), intent(in) *beta,* real(kind=wp), intent(out) *freq*)

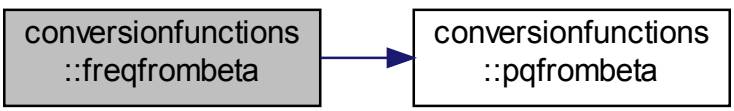
Subroutine for computing the frequency for a given reliability index (beta) assuming a standard normal distribution.

Parameters

in	*beta*	Reliability index
out	*freq*	Return time

Definition at line 165 of file conversionFunctions.f90.

Here is the call graph for this function:



7.2.1.3 subroutine, public conversionfunctions::logqfrombeta (real(kind=wp), intent(in) *beta,* real(kind=wp), intent(out) *logQ*)

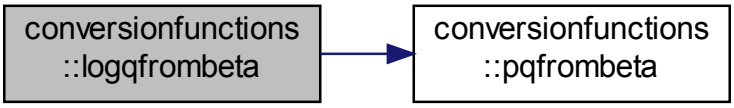
Subroutine for computing the (negative) logarithm of the non-exceedance probability for a given reliability index (beta) assuming a standard normal distribution.

Parameters

in	*beta*	Reliability index
out	*logq*	Log q

Definition at line 188 of file conversionFunctions.f90.

Here is the call graph for this function:



7.2.1.4 real(kind = wp) function, public conversionfunctions::pfrombeta (real(kind = wp), intent(in) *beta*)

Function for computing the non-exceedance probability (P) for a given
reliability index (beta), assuming a standard normal distribution.

.

Parameters

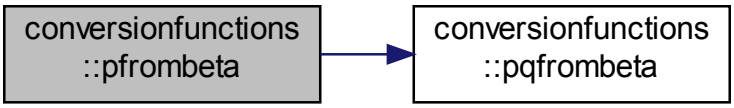
in	*beta*	Reliability index

Returns

Probability of non-exceedance

Definition at line 90 of file conversionFunctions.f90.

Here is the call graph for this function:



7.2.1.5 subroutine, public conversionfunctions::pqfrombeta (real(kind=wp), intent(in) *beta,* real(kind=wp), intent(out) *p,* real(kind=wp), intent(out) *q*)

Subroutine for computing the exceedance and non-exceedance probabilities for a given reliability index (beta) assuming a standard normal distribution.

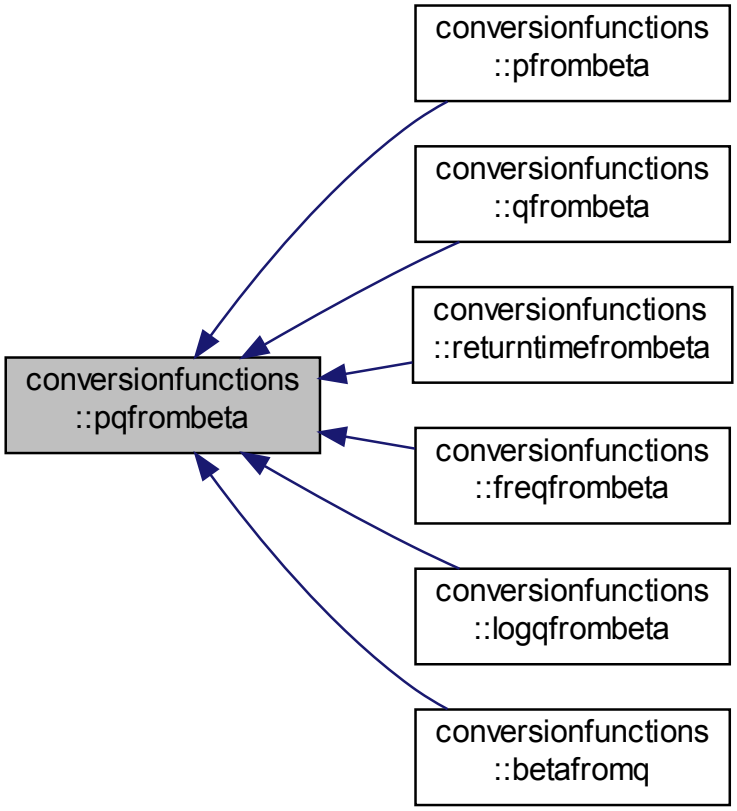
Approximation method is taken from: Approximation Formula 26.2.17 for Normal Probability Function, Handbook of Mathematical Functions, Abramowitz & Stegun, page 932

Parameters

in	*beta*	Reliability index
out	*p*	Probability of non-exceedance
out	*q*	Probability of exceedance

Definition at line 44 of file conversionFunctions.f90.

Here is the caller graph for this function:



7.2.1.6 real(kind = wp) function, public conversionfunctions::qfrombeta (real(kind = wp), intent(in) *beta*)

Function for computing the exceedance probability (Q) for a given
reliability index (beta), assuming a standard normal distribution.

.

Parameters

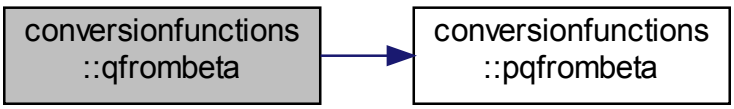
in	*beta*	Reliability index

Returns

Probability of exceedance

Definition at line 116 of file conversionFunctions.f90.

Here is the call graph for this function:



7.2.1.7 subroutine, public conversionfunctions::returntimefrombeta (real(kind=wp), intent(in) *beta,* real(kind=wp), intent(out) *returnTime*)

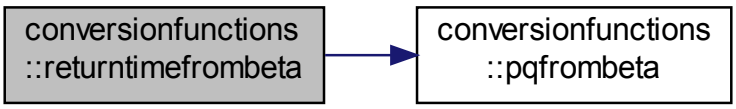
Subroutine for computing the return time for a given reliability index (beta) assuming a standard normal distribution.

Parameters

in	*beta*	Reliability index
out	*returntime*	Return time

Definition at line 142 of file conversionFunctions.f90.

Here is the call graph for this function:



7.2.2 Variable Documentation

7.2.2.1 real(kind = wp), parameter, private conversionfunctions::qmin = 1.0d-300

Definition at line 23 of file conversionFunctions.f90.

7.2.2.2 real(kind = wp), parameter, private conversionfunctions::ulimit = 5.6d0

Definition at line 31 of file conversionFunctions.f90.

7.2.2.3 real(kind = wp), parameter, private conversionfunctions::upperlog = 700.0d0

Definition at line 27 of file conversionFunctions.f90.

7.3 dllovertopping Module Reference

Calculate one type of overtopping.

Functions/Subroutines

- subroutine, public calculateqo (load, geometryInput, dikeHeight, modelFactors, overtopping, success, errorText, verbosity, logFile)

  Subroutine that calculates the discharge needed for the Z-function DikesOvertopping Wrapper for calculateQoF: convert C-like input structures to Fortran input structures.
- subroutine, public calculateqof (load, geometryF, dikeHeight, modelFactors, overtopping, success, errorText, logging)

  Subroutine that calculates the discharge needed for the Z-function DikesOvertopping.
- subroutine, public calczvalue (criticalOvertoppingRate, modelFactors, Qo, z, success, errorMessage)

  Subroutine that calculates the Z-function DikesOvertopping based on the discharge calculated with calculateQoF.
- subroutine, public validateinputc (geometryInput, dikeHeight, modelFactors, success, errorText)

  Subroutine that validates the geometry Wrapper for ValidateInputF: convert C-like input structures to Fortran input structures.
- subroutine, public validateinputf (geometryF, dikeHeight, modelFactors, success, errorText)

  Subroutine that validates the geometry.
- subroutine, public versionnumber (version)

  Subroutine that delivers the version number.
- type(overtoppinggeometrytypef) function geometry_c_f (geometryInput)

  Private subroutine that converts geometry from c-pointer to fortran struct.

7.3.1 Detailed Description

Calculate one type of overtopping.

7.3.2 Function/Subroutine Documentation

7.3.2.1 subroutine, public dllovertopping::calculateqo (type(tpload), intent(in) *load,* type(overtoppinggeometrytype), intent(in) *geometryInput,* real(kind=wp), intent(in) *dikeHeight,* type(tpovertoppinginput), intent(inout) *modelFactors,* type (tpovertopping), intent(out) *overtopping,* logical, intent(out) *success,* character(len=*), intent(out) *errorText,* integer, intent(in) *verbosity,* character(len=*), intent(in) *logFile*)

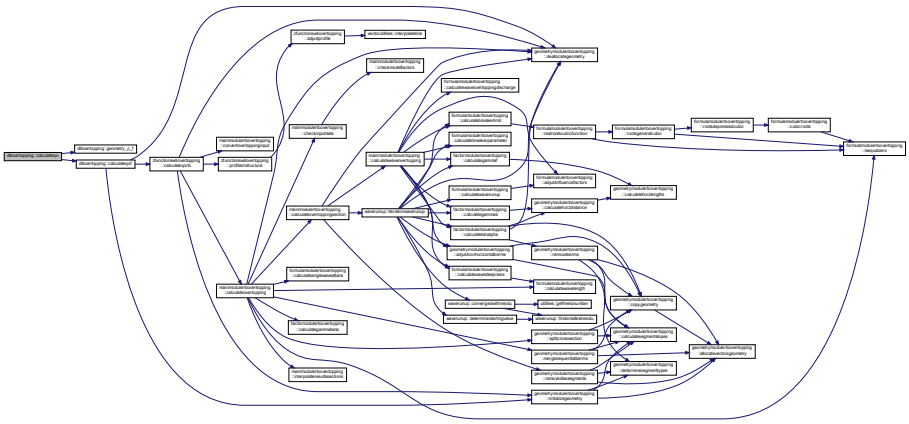
Subroutine that calculates the discharge needed for the Z-function DikesOvertopping Wrapper for calculateQoF: convert C-like input structures to Fortran input structures.

Parameters

in	*geometryinput*	struct with geometry and roughness as c-pointers
in	*load*	struct with waterlevel and wave parameters
in,out	*modelfactors*	struct with modelfactors
out	*overtopping*	structure with overtopping results
out	*success*	flag for success
out	*errortext*	error message (only set if not successful)
in	*verbosity*	level of verbosity
in	*logfile*	filename of logfile

Definition at line 40 of file dllOvertopping.f90.

Here is the call graph for this function:



7.3.2.2 subroutine, public dllovertopping::calculateqof (type(tpload), intent(in) *load,* type(overtoppinggeometrytypef), intent(in) *geometryF,* real(kind=wp), intent(in) *dikeHeight,* type(tpovertoppinginput), intent(inout) *modelFactors,* type (tpovertopping), intent(out) *overtopping,* logical, intent(out) *success,* character(len=∗), intent(out) *errorText,* type(tlogging), intent(in) *logging*)

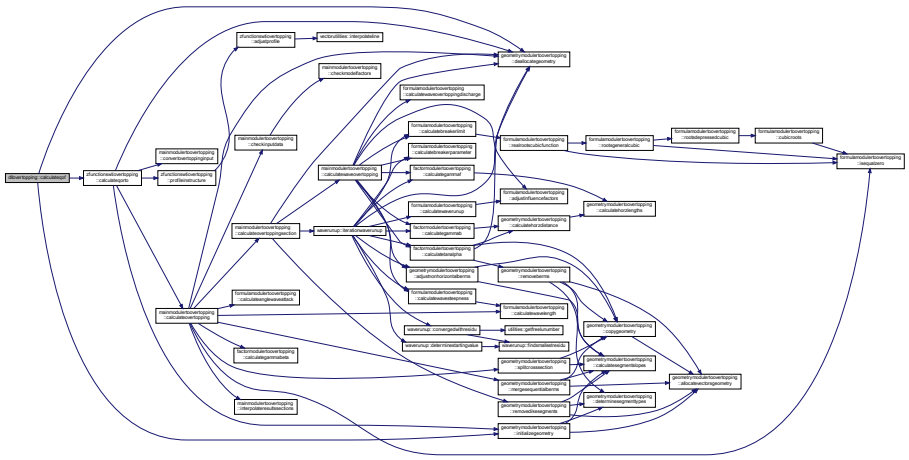
Subroutine that calculates the discharge needed for the Z-function DikesOvertopping.

Parameters

in	*geometryf*	struct with geometry and roughness
in	*load*	struct with waterlevel and wave parameters
in,out	*modelfactors*	struct with modelFactors
out	*overtopping*	structure with overtopping results
out	*success*	flag for success
out	*errortext*	error message (only set if not successful)
in	*logging*	logging struct

Definition at line 71 of file dllOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.3.2.3 subroutine, public dllovertopping::calczvalue (real(kind=wp), intent(in) *criticalOvertoppingRate,* type(tpovertoppinginput), intent(inout) *modelFactors,* real(kind=wp), intent(in) *Qo,* real(kind=wp), intent(out) *z,* logical, intent(out) *success,* character(len=∗), intent(out) *errorMessage*)

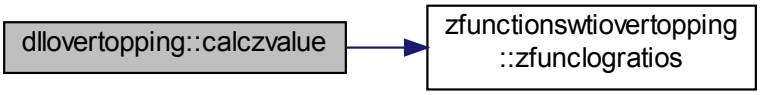
Subroutine that calculates the Z-function DikesOvertopping based on the discharge calculated with calculateQoF.

Parameters

in	*criticalovertop-pingrate*	critical overtoppingrate
in,out	*modelfactors*	struct with modelfactors
in	*qo*	calculated discharge
out	*z*	z value
out	*errormessage*	error message (only if not successful)
out	*success*	flag for success

Definition at line 104 of file dllOvertopping.f90.

Here is the call graph for this function:



7.3.2.4 type(overtoppinggeometrytypef) function dllovertopping::geometry_c_f (type(overtoppinggeometrytype), intent(in) *geometryInput*) [private]

Private subroutine that converts geometry from c-pointer to fortran struct.

Parameters

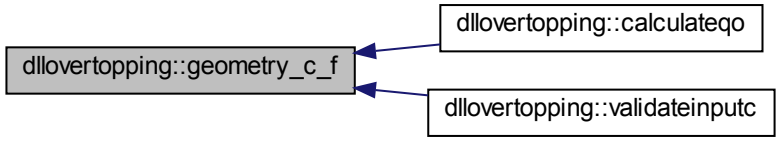
in	*geometryinput*	struct with geometry and roughness as c-pointers

Returns

fortran struct with geometry and roughness

Definition at line 225 of file dllOvertopping.f90.

Here is the caller graph for this function:



7.3.2.5 subroutine, public dllovertopping::validateinputc (type(overtoppinggeometrytype), intent(in) *geometryInput,* real(kind=wp), intent(in) *dikeHeight,* type(tpovertoppinginput), intent(inout) *modelFactors,* logical, intent(out) *success,* character(len=∗), intent(out) *errorText*)

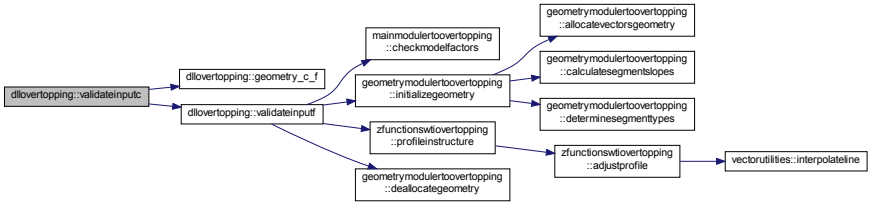
Subroutine that validates the geometry Wrapper for ValidateInputF: convert C-like input structures to Fortran input structures.

Parameters

in	*geometryinput*	struct with geometry and roughness as c-pointers
in,out	*modelfactors*	struct with modelfactors
out	*success*	flag for success
out	*errortext*	error message (only set if not successful)

Definition at line 124 of file dllOvertopping.f90.

Here is the call graph for this function:



7.3.2.6 subroutine, public dllovertopping::validateinputf (type(overtoppinggeometrytypef), intent(in) *geometryF,* real(kind=wp), intent(in) *dikeHeight,* type(tpovertoppinginput), intent(inout) *modelFactors,* logical, intent(out) *success,* character(len=∗), intent(out) *errorText*)

Subroutine that validates the geometry.

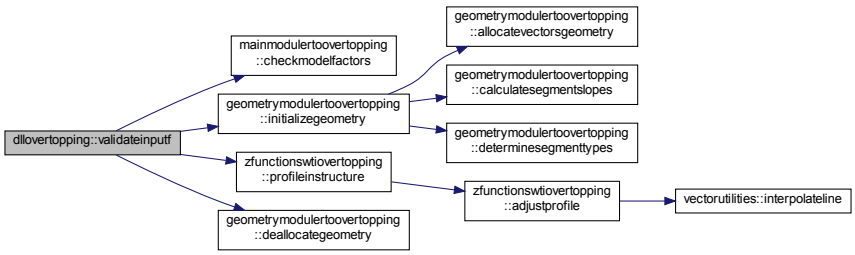
Parameters

in	*geometryf*	struct with geometry and roughness
in,out	*modelfactors*	struct with modelFactors
out	*success*	flag for success

out	*errortext*	error message (only set if not successful)

Definition at line 146 of file dllOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.3.2.7 subroutine, public dllovertopping::versionnumber (character(len=∗), intent(out) *version*)

Subroutine that delivers the version number.

Parameters

out	*version*	version number

Definition at line 203 of file dllOvertopping.f90.

7.4 equalreals Module Reference

Module with functions to test quality of reals.

Functions/Subroutines

- logical function equalrealsrelative (x, y, margin)

  equalRealsRelative Function to test for equality of two reals with a relative value for the margin equalRealsRelative = |x-y| <= margin ∗ (|x|+|y|)/2
- logical function equalrealsabsolute (x, y, margin)

  equalRealsAbsolute Function to test for equality of two reals with an absolute value for the margin equalRealsAbsolute = |x-y| <= margin
- logical function equalrealsinvector (n, vector, margin)

  Function to test for equality of reals in a vector with a relative value of the margin.
- logical function equalvectorswithreals (n, vector1, vector2, margin, equality)

  Function to test for equality of two vectors with reals Two vectors are equal if all elements in the vector are equal.

- logical function equaltableswithreals (matrix1, matrix2, margin)

  Function to test for equality of two vectors with reals Two vectors are equal if all elements in the vector are equal.

- logical function equalvectorswithrealsweightedmargin (n, vector1, vector2, margin, equality)

  Function to test for equality of two vectors with reals. Two vectors are equal if all elements in the vector are almost equal Here, margin is weighted according to tested numbers.

- logical function equalvectorswithintegers (n, vector1, vector2, equality)

7.4.1 Detailed Description

Module with functions to test quality of reals.

7.4.2 Function/Subroutine Documentation

7.4.2.1 logical function equalreals::equalrealsabsolute (real(kind=wp), intent(in) *x,* real(kind=wp), intent(in) *y,* real(kind=wp), intent(in) *margin*)

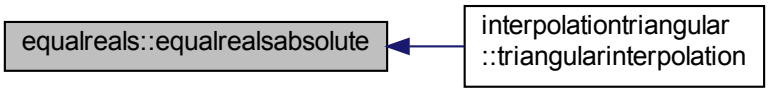
equalRealsAbsolute Function to test for equality of two reals with an absolute value for the margin equalReals↩ Absolute = |x-y| <= margin

Parameters

in	*x*	First real for the comparison
in	*y*	Second real for the comparison
in	*margin*	Absolute value for the margin

Definition at line 40 of file equalReals.f90.

Here is the caller graph for this function:



7.4.2.2 logical function equalreals::equalrealsinvector (integer, intent(in) *n,* real(kind=wp), dimension(n), intent(in) *vector,* real(kind=wp), intent(in) *margin*)

Function to test for equality of reals in a vector with a relative value of the margin.

Parameters

in	*n*	Vector length
in	*vector*	Vector with reals
in	*margin*	Relative value for the margin

Definition at line 55 of file equalReals.f90.

Here is the call graph for this function:

7.4.2.3 logical function equalreals::equalrealsrelative (real(kind=wp), intent(in) *x,* real(kind=wp), intent(in) *y,* real(kind=wp), intent(in) *margin*)

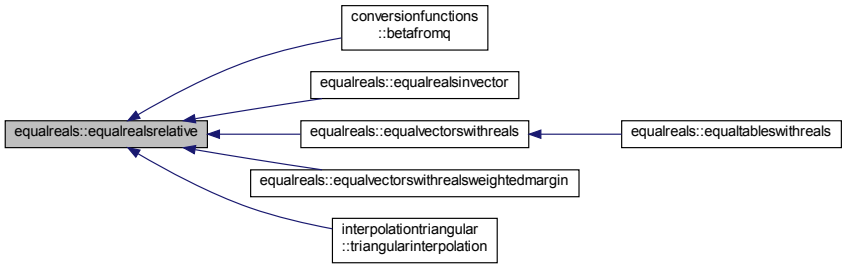
equalRealsRelative Function to test for equality of two reals with a relative value for the margin equalRealsRelative = |x-y| <= margin * (|x|+|y|)/2

Parameters

in	*x*	First real for the comparison
in	*y*	Second real for the comparison
in	*margin*	Relative value for the margin

Definition at line 24 of file equalReals.f90.

Here is the caller graph for this function:



7.4.2.4 logical function equalreals::equaltableswithreals (real(kind=wp), dimension(:,:), intent(in) *matrix1,* real(kind=wp), dimension(:,:), intent(in) *matrix2,* real(kind=wp), intent(in) *margin*)

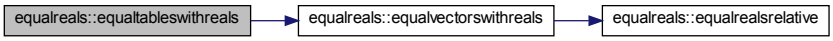
Function to test for equality of two vectors with reals Two vectors are equal if all elements in the vector are equal.

Parameters

in	*matrix1*	Matrix 1 with reals
in	*matrix2*	Matrix 2 with reals
in	*margin*	Relative value for the margin

Definition at line 110 of file equalReals.f90.

Here is the call graph for this function:



7.4.2.5 logical function equalreals::equalvectorswithintegers (integer, intent(in) *n,* integer, dimension(n), intent(in) *vector1,* integer, dimension(n), intent(in) *vector2,* logical, dimension(n), intent(out) *equality*)

Parameters

in	*n*	Vector length
in	*vector1*	Vector1 with reals (size n)
in	*vector2*	Vector2 with reals (size n)
out	*equality*	Vector (size n) with elements which indicates if the corresponding elements of the two input vectors are the same

Definition at line 200 of file equalReals.f90.

7.4.2.6 logical function equalreals::equalvectorswithreals (integer, intent(in) *n,* real(kind=wp), dimension(n), intent(in) *vector1,* real(kind=wp), dimension(n), intent(in) *vector2,* real(kind=wp), intent(in) *margin,* logical, dimension(n), intent(out) *equality*)

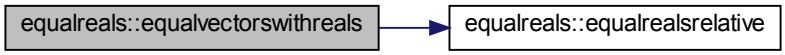
Function to test for equality of two vectors with reals Two vectors are equal if all elements in the vector are equal.

Parameters

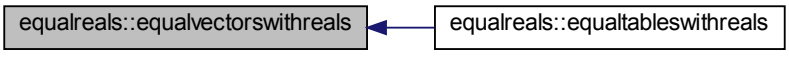
in	*n*	Vector length
in	*vector1*	Vector1 with reals (size n)
in	*vector2*	Vector2 with reals (size n)
in	*margin*	Relative value for the margin
out	*equality*	Vector (size n) with elements which indicates if the corresponding elements of the two input vectors are the same

Definition at line 77 of file equalReals.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.2.7 logical function equalreals::equalvectorswithrealsweightedmargin (integer, intent(in) *n,* real(kind=wp), dimension(n), intent(in) *vector1,* real(kind=wp), dimension(n), intent(in) *vector2,* real(kind=wp), intent(in) *margin,* logical, dimension(n), intent(out) *equality*)

Function to test for equality of two vectors with reals. Two vectors are equal if all elements in the vector are almost equal Here, margin is weighted according to tested numbers.

Parameters

in	*n*	Vector length
in	*vector1*	Vector1 with reals (size n)
in	*vector2*	Vector2 with reals (size n)
in	*margin*	Relative value for the margin
out	*equality*	Vector (size n) with elements which indicates if the corresponding elements of the two input vectors are the same

Definition at line 159 of file equalReals.f90.

Here is the call graph for this function:

7.5 factormodulertoovertopping Module Reference

Functions/Subroutines

- subroutine, public calculatetanalpha (h, Hm0, z2, geometry, tanAlpha, succes, errorMessage)

  calculateTanAlpha representative slope angle
- subroutine, public calculategammabeta (Hm0, Tm_10, beta, gammaBeta_z, gammaBeta_o)

  calculateGammaBeta influence factor angle of wave attack
- subroutine, public calculategammaf (h, ksi0, ksi0Limit, gammaB, z2, geometry, gammaF, succes, error↩ Message)

  calculateGammaF influence factor roughness
- subroutine, public calculategammab (h, Hm0, z2, geometry, NbermSegments, gammaB, succes, error↩ Message)

  calculateGammaB influence factor berms

7.5.1 Function/Subroutine Documentation

7.5.1.1 subroutine, public factormodulertoovertopping::calculategammab (real(wp), intent(in) *h,* real(wp), intent(in) *Hm0,* real(wp), intent(in) *z2,* type(tpgeometry), intent(in) *geometry,* integer, intent(in) *NbermSegments,* real(wp), intent(out) *gammaB,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

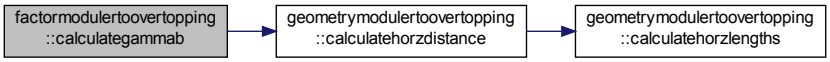
calculateGammaB influence factor berms

Parameters

in	*h*	local water level (m+NAP)
in	*hm0*	significant wave height (m)
in	*z2*	2% wave run-up (m)
in	*geometry*	structure with geometry data
in	*nbermsegments*	number of berm segments
out	*gammab*	influence factor berms
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 271 of file factorModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.5.1.2 subroutine, public factormodulertoovertopping::calculategammabeta (real(wp), intent(inout) *Hm0,* real(wp), intent(inout) *Tm_10,* real(wp), intent(in) *beta,* real(wp), intent(out) *gammaBeta_z,* real(wp), intent(out) *gammaBeta_o*)

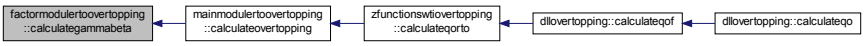
calculateGammaBeta influence factor angle of wave attack

Parameters

in,out	*hm0*	significant wave height (m)
in,out	*tm_10*	spectral wave period (s)
in	*beta*	angle of wave attack (degree)
out	*gammabeta_z*	influence factor angle of wave attack 2% wave run-up
out	*gammabeta_o*	influence factor angle of wave attack overtopping

Definition at line 114 of file factorModuleRTOovertopping.f90.

Here is the caller graph for this function:



7.5.1.3 subroutine, public factormodulertoovertopping::calculategammaf (real(wp), intent(in) *h,* real(wp), intent(in) *ksi0,* real(wp), intent(in) *ksi0Limit,* real(wp), intent(in) *gammaB,* real(wp), intent(in) *z2,* type(tpgeometry), intent(in) *geometry,* real(wp), intent(out) *gammaF,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

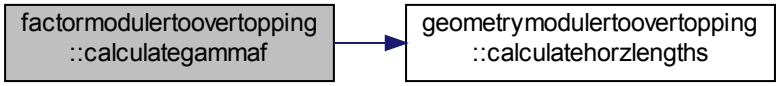
calculateGammaF influence factor roughness

Parameters

in	*h*	local water level (m+NAP)
in	*ksi0*	breaker parameter
in	*ksi0limit*	limit value breaker parameter
in	*gammab*	influence factor berms
in	*z2*	2% wave run-up (m)
in	*geometry*	structure with geometry data
out	*gammaf*	influence factor roughness
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 154 of file factorModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.5.1.4 subroutine, public factormodulertoovertopping::calculatetanalpha (real(wp), intent(in) *h,* real(wp), intent(in) *Hm0,* real(wp), intent(in) *z2,* type(tpgeometry), intent(in) *geometry,* real(wp), intent(out) *tanAlpha,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

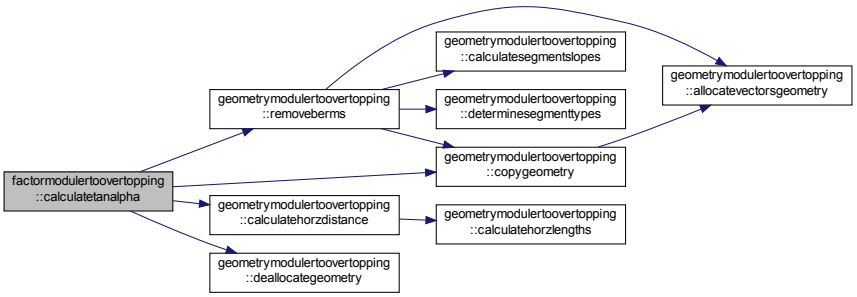
calculateTanAlpha representative slope angle

Parameters

in	*h*	local water level (m+NAP)
in	*hm0*	significant wave height (m)
in	*z2*	2% wave run-up (m)
in	*geometry*	structure with geometry data
out	*tanalpha*	representative slope angle
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 35 of file factorModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.6 feedback Module Reference

Data Types

- interface errormessage
- interface fatalerror
- interface feedbackcontext
- interface progressreport
- interface warningmessage

Functions/Subroutines

- integer function p ()

  Returns the pointer to the feedback dll, avoiding many loadlibrary calls.
- logical function, public todll ()

  Is feedback writing to the LogMessage function in HydraRing.IO.Native.dll ?
- logical function, public fatalerroroccured (value)

  Has a fatal error occured? Mainly used in unit tests.
- subroutine, public resetfatalerroroccured ()

  Reset fatal error occurance Mainly used in unit tests.
- subroutine setfatalerroroccured (value)

  Set fatal error occurance Mainly used in unit tests.
- logical function, public getfatalerrorexpected (value)

  Get value of FatalErrorExpected Mainly used in unit tests.
- subroutine, public setfatalerrorexpected (value)

  Set the value of FatalErrorExpected Mainly used in unit tests.
- logical function, public allow_log ()

  Is logging active?
- subroutine, public setallowlogging (value)

Set logging on/off.

- subroutine, public getthreadid (threadId)

  Get the tread ID Note that multi threading is not implemented, so always the value 0 is returned.
- subroutine, public setfatalerroraction (action)

  Set the value of FatalErrorAction possible values: onfatalerrorStop = 1 : Action on fatal error: exit the program onfatalerrorThrowException = 2 : Action on fatal error: throw exception (for use as dll) onfatalerrorUnittest = 3 : Action on fatal error: handle as unit test.
- subroutine, public getfatalerrormessage (message)

  Get the last of FatalErrorMessage.
- subroutine, public feedbackerror (error)

  Return the error status of the current thread If an error has been reported (via the errorMessage routines), this routine will return true, otherwise false.
- subroutine, public feedbackclose

  Close the log file.
- subroutine, public feedbackinitialise (output, logfile, verbosity)

  Initialise the feedback subsystem.
- subroutine progressreporttxt (level, msgtext)

  Write a single message to the output for the benefit of the user.
- subroutine progressreportint (level, msgtext, value)

  Write a single message with an integer value.
- subroutine progressreportreal (level, msgtext, value)

  Write a single message with a real value.
- subroutine progressreportdouble (level, msgtext, value)

  Write a single message with a double precision real value.
- subroutine progressreportarray (level, msgText, x, size)

  Print array values to report file.
- subroutine progressreportvalueandarray (level, msgText, z, x, size)

  Print z-value and array values to report file.
- subroutine, public feedbackenter (routine)

  Register a new routine level.
- subroutine, public feedbackleave

  Drop a routine level from the stack trace.
- subroutine, public feedbackprintstack

  Print the stack trace as known via feedbackEnter and feedbackLeave.
- subroutine errormessagetxt (msgtext)

  Write an error message.
- subroutine errormessageint (msgtext, value)

  Write an error message with an integer value.
- subroutine errormessagereal (msgtext, value)

  Write an error message with a real value.
- subroutine errormessagedouble (msgtext, value)

  Write an error message with a double precision real value.
- subroutine warningmessagetxt (msgtext)

  Write a warning message.
- subroutine warningmessageint (msgtext, value)

  Write a warning message with an integer value.
- subroutine warningmessagereal (msgtext, value)

  Write a warning message with a real value.
- subroutine warningmessagedouble (msgtext, value)

  Write a warning message with a double precision real value.
- subroutine fatalerrortxt (msgtext)

Write a fatal error message.

- subroutine fatalerrorint (msgtext, value)

  Write a fatal error message with an integer value.
- subroutine fatalerrorreal (msgtext, value)

  Write a fatal error message with a real value.
- subroutine fatalerrordouble (msgtext, value)

  Write a fatal error message with a double precision real value.
- subroutine, public feedbackdropcontext

  Drop the entire context.
- subroutine, public feedbackprintcontext

  Register a context for error messages.
- subroutine feedbackcontexttxt (text)

  Register a context for error messages.
- subroutine feedbackcontextint (text, value)

  Register a context for error messages - with integer value.
- subroutine feedbackcontextreal (text, value)

  Register a context for error messages - with real value.
- subroutine feedbackcontextdouble (text, value)

  Register a context for error messages - with double precision real value.
- subroutine, public throw (msgText)

  Stop execution by throwing an exception Useful if running in a dll and main program is a GUI in e.g. C++ or C#.

Variables

- integer, parameter id = 0

  Thread ID; multi threading is not supported yet!
- integer, parameter sp = kind(1.0)
- integer, parameter dp = kind(1.0d0)
- integer, parameter wp = kind(1.0d0)

7.6.1 Variable Documentation

7.6.1.1 integer, parameter feedback::dp = kind(1.0d0)

Definition at line 114 of file feedback.f90.

7.6.1.2 integer, parameter feedback::id = 0

Thread ID; multi threading is not supported yet!

Definition at line 111 of file feedback.f90.

7.6.1.3 integer, parameter feedback::sp = kind(1.0)

Definition at line 113 of file feedback.f90.

7.6.1.4 integer, parameter feedback::wp = kind(1.0d0)

Definition at line 115 of file feedback.f90.

7.7 feedback_parameters Module Reference

Variables

- integer, parameter onfatalerrorstop = 1
  Action on fatal error: exit the program.
- integer, parameter onfatalerrorthrowexception = 2
  Action on fatal error: throw exception (for use as dll)
- integer, parameter onfatalerrorunittest = 3
  Action on fatal error: handle as unit test.
- integer, parameter onlyscreen = 0
  Mode: write to screen only.
- integer, parameter onlyfile = 1
  Mode: write to file only.
- integer, parameter screenandfile = 2
  Mode: write to both screen and file.
- integer, parameter dll = 3
  Mode: write to dll.
- integer, parameter verbosenone = -1
  Reporting: NO messages.
- integer, parameter verbosebasic = 0
  Reporting: basic messages only.
- integer, parameter verbosedetailed = 1
  Reporting: detailed messages (including traceback)
- integer, parameter verbosedebugging = 2
  Reporting: all messages.

7.7.1 Variable Documentation

7.7.1.1 integer, parameter feedback_parameters::onfatalerrorstop = 1

Action on fatal error: exit the program.

Definition at line 10 of file feedback_parameters.f90.

7.7.1.2 integer, parameter feedback_parameters::onfatalerrorthrowexception = 2

Action on fatal error: throw exception (for use as dll)

Definition at line 11 of file feedback_parameters.f90.

7.7.1.3 integer, parameter feedback_parameters::onfatalerrorunittest = 3

Action on fatal error: handle as unit test.

Definition at line 12 of file feedback_parameters.f90.

7.8 feedbackdll Module Reference

Data Types

- type feedbackdata

Functions/Subroutines

- subroutine initialise (output, logfile, verbosity)

  Initialise the feedback dll.
- subroutine printmessage (msgtext)

  Write a message to output device (as selected in Initialise)
- subroutine outputtodll (active)

  Is output is to another dll?
- subroutine getallowlogging (value)

  Is logging active.
- subroutine setallowlogging (value)

  Set logging on/off.
- subroutine getverbosity (level)

  Get the current verbosity level.
- subroutine setonfatalerroraction (action)

  Set the OnFatalErrorAction.
- subroutine getonfatalerroraction (onfatalerrorAction)

  Get the current OnFatalErrorAction.
- subroutine getfeedbackerror (error)

  Get the error status (true/false)
- subroutine setfatalerrormessage (message)

  Set the errormessage.
- subroutine getfatalerrormsg (message)

  Get the last error message.
- subroutine getverbositylevel (verbosityLevel)

  Get the verbosity level.
- subroutine pushroutine (routine, success)

  Add a routine name to the call stack.
- subroutine poproutine (success)

  Removes and get a routine name from the call stack.
- subroutine printstack

  Print the call stack.
- subroutine getstacklevel (level)

  Get the number of routines in the call stack.
- subroutine closefile

  Closes the output file.
- subroutine getunitnumber (lun)

  Get the unit number of the logfile.
- subroutine dropcontext

  Reset the context stack.
- subroutine printcontext

  Print the context stack.
- subroutine contexttxt (text, success)

  Add a line of text to the context stack.
- subroutine getsetfatalerrorexpected (func, value)

  Get or Set the FatalErrorExpected if func=get: get the value if func=set: set the value.
- subroutine getsetfatalerroroccured (func, value)

  Get or Set the FatalErrorOccured if func=get: get the value if func=set: set the value.
- subroutine writetodll (msgText)

  Write message to another dll.

Variables

- type(feedbackdata) fbdata

  Array holding the actual information per thread (multiple threads not yet implemented)
- integer fatalerroraction = 1
- character(len=255) fatalerrormessage
- integer, parameter id = 0
- logical allow_logging = .true.
- logical isfatalerrorexpected = .false.

  if isFatalErrorExpected = .true. then log error message in fatalErrorMsgtext and set boolean isFatalErrorOccured to true if isFatalErrorExpected = .false. then same action as onfatalerrorStop
- logical isfatalerroroccured = .false.

7.8.1 Function/Subroutine Documentation

7.8.1.1 subroutine feedbackdll::closefile ( ) [private]

Closes the output file.

Definition at line 288 of file feedbackDLL.f90.

7.8.1.2 subroutine feedbackdll::contexttxt (character(len=*), intent(in) *text*, logical, intent(out) *success*) [private]

Add a line of text to the context stack.

Parameters

in	*text*	line of text to be pushed on the context stack
out	*success*	.false., if the stack already reached it maximum size

Definition at line 335 of file feedbackDLL.f90.

7.8.1.3 subroutine feedbackdll::dropcontext ( ) [private]

Reset the context stack.

Definition at line 307 of file feedbackDLL.f90.

7.8.1.4 subroutine feedbackdll::getallowlogging (logical, intent(out) *value*) [private]

Is logging active.

Parameters

out	*value*	the current value of allow_logging

Definition at line 133 of file feedbackDLL.f90.

7.8.1.5 subroutine feedbackdll::getfatalerrormsg (character(len=*), intent(out) *message*) [private]

Get the last error message.

Parameters

out	*message*	current error message

Definition at line 203 of file feedbackDLL.f90.

7.8.1.6 subroutine feedbackdll::getfeedbackerror (logical, intent(out) *error*) [private]

Get the error status (true/false)

Parameters

out	*error*	last error status

Definition at line 183 of file feedbackDLL.f90.

7.8.1.7 subroutine feedbackdll::getonfatalerroraction (integer, intent(out) *onfatalerrorAction*) [private]

Get the current OnFatalErrorAction.

Parameters

out	*onfatalerroraction*	current FatalErrorAction: 1, 2 or 3

Definition at line 172 of file feedbackDLL.f90.

7.8.1.8 subroutine feedbackdll::getsetfatalerrorexpected (character(len=3), intent(in) *func,* logical, intent(inout) *value*) [private]

Get or Set the FatalErrorExpected if func=get: get the value if func=set: set the value.

Parameters

in,out	*value*	the value of FatalErrorExpected
in	*func*	the action: 'get' or 'set' (case sensitive!)

Definition at line 357 of file feedbackDLL.f90.

7.8.1.9 subroutine feedbackdll::getsetfatalerroroccured (character(len=3), intent(in) *func,* logical, intent(inout) *value*) [private]

Get or Set the FatalErrorOccured if func=get: get the value if func=set: set the value.

Parameters

in,out	*value*	the value of FatalErrorOccured
in	*func*	the action: 'get' or 'set' (case sensitive!)

Definition at line 374 of file feedbackDLL.f90.

7.8.1.10 subroutine feedbackdll::getstacklevel (integer, intent(out) *level*) [private]

Get the number of routines in the call stack.

Parameters

out	*level*	number of routines in the call stack

Definition at line 278 of file feedbackDLL.f90.

7.8.1.11 subroutine feedbackdll::getunitnumber (integer, intent(out) *lun*) [private]

Get the unit number of the logfile.

Parameters

out	*lun*	integer unit number of logfile

Definition at line 297 of file feedbackDLL.f90.

7.8.1.12 subroutine feedbackdll::getverbosity (integer, intent(out) *level*) [private]

Get the current verbosity level.

Parameters

out	*level*	the current verbosity level: -1, .. , 2

Definition at line 151 of file feedbackDLL.f90.

7.8.1.13 subroutine feedbackdll::getverbositylevel (integer, intent(out) *verbosityLevel*) [private]

Get the verbosity level.

Parameters

out	*verbositylevel*	current verbosityLevel: -1, .., 2

Definition at line 213 of file feedbackDLL.f90.

7.8.1.14 subroutine feedbackdll::initialise (integer, intent(in) *output,* character(len=∗), intent(in) *logfile,* integer, intent(in) *verbosity*) [private]

Initialise the feedback dll.

Parameters

in	*output*	option: output to screen, file, both, or another dll
in	*logfile*	name of the log file to be used (only used if writing to file)
in	*verbosity*	level for displaying the messages

Definition at line 51 of file feedbackDLL.f90.

7.8.1.15 subroutine feedbackdll::outputtodll (logical, intent(out) *active*) [private]

Is output is to another dll?

Parameters

out	*active*	.true., if output is to another dll

Definition at line 123 of file feedbackDLL.f90.

7.8.1.16 subroutine feedbackdll::poproutine (logical, intent(out) *success*) [private]

Removes and get a routine name from the call stack.

Parameters

out	*success*	if .false., the stack is already empty

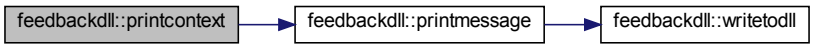
Definition at line 240 of file feedbackDLL.f90.

7.8.1.17 subroutine feedbackdll::printcontext ( ) [private]

Print the context stack.

Definition at line 317 of file feedbackDLL.f90.

Here is the call graph for this function:



7.8.1.18 subroutine feedbackdll::printmessage (character(len=*), intent(in) *msgtext*) [private]

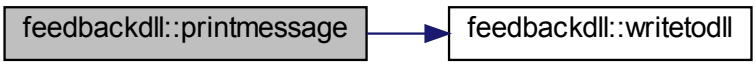
Write a message to output device (as selected in Initialise)

Parameters

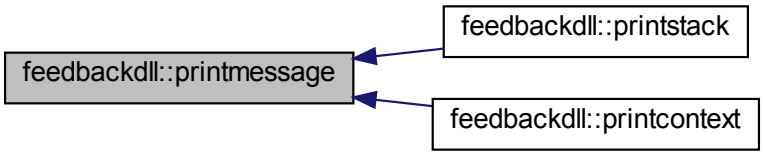
in	*msgtext*	text to be printed to screen, file or dll

Definition at line 96 of file feedbackDLL.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

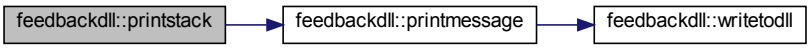


7.8.1.19 subroutine feedbackdll::printstack ( ) [private]

Print the call stack.

Definition at line 255 of file feedbackDLL.f90.

Here is the call graph for this function:



7.8.1.20 subroutine feedbackdll::pushroutine (character(len=∗), intent(in) *routine,* logical, intent(out) *success*) [private]

Add a routine name to the call stack.

Parameters

in	*routine*	routine name to be added to the stack
out	*success*	.false., if the stack already reached it maximum size

Definition at line 223 of file feedbackDLL.f90.

7.8.1.21 subroutine feedbackdll::setallowlogging (logical, intent(in) *value*) [private]

Set logging on/off.

Parameters

in	*value*	the value for allow_logging to be set

Definition at line 142 of file feedbackDLL.f90.

7.8.1.22 subroutine feedbackdll::setfatalerrormessage (character(len=∗), intent(in) *message*) [private]

Set the errormessage.

Parameters

in	*message*	error message to be set

Definition at line 193 of file feedbackDLL.f90.

7.8.1.23 subroutine feedbackdll::setonfatalerroraction (integer, intent(in) *action*) [private]

Set the OnFatalErrorAction.

Parameters

in	*action*	FatalErrorAction to be set. Must be in range: 1, 2 or 3

Definition at line 162 of file feedbackDLL.f90.

7.8.1.24 subroutine feedbackdll::writetodll (character(len=∗), intent(in) *msgText*) [private]

Write message to another dll.

Parameters

in	*msgtext*	Error message to write

Definition at line 391 of file feedbackDLL.f90.

Here is the caller graph for this function:

7.8.2 Variable Documentation

7.8.2.1 logical feedbackdll::allow_logging = .true.

Definition at line 40 of file feedbackDLL.f90.

7.8.2.2 integer feedbackdll::fatalerroraction = 1

Definition at line 37 of file feedbackDLL.f90.

7.8.2.3 character(len=255) feedbackdll::fatalerrormessage

Definition at line 38 of file feedbackDLL.f90.

7.8.2.4 type(feedbackdata) feedbackdll::fbdata

Array holding the actual information per thread (multiple threads not yet implemented)

Definition at line 35 of file feedbackDLL.f90.

7.8.2.5 integer, parameter feedbackdll::id = 0

Definition at line 39 of file feedbackDLL.f90.

7.8.2.6 logical feedbackdll::isfatalerrorexpected = .false.

if isFatalErrorExpected = .true. then log error message in fatalErrorMsgtext and set boolean isFatalErrorOccured to true if isFatalErrorExpected = .false. then same action as onfatalerrorStop

Definition at line 41 of file feedbackDLL.f90.

7.8.2.7 logical feedbackdll::isfatalerroroccured = .false.

Definition at line 43 of file feedbackDLL.f90.

7.9 fileutilities Module Reference

Functions/Subroutines

- subroutine readtable (filename, nColumns, table)
- subroutine writetablevalues (filename, table)
- subroutine removefile (filename)

  Routine to throw away a file.

7.9.1 Function/Subroutine Documentation

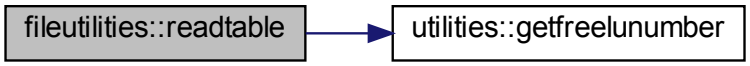
7.9.1.1 subroutine fileutilities::readtable (character(len=∗), intent(in) *filename,* integer, intent(in) *nColumns,* real(wp), dimension(:,:), allocatable *table*)

Parameters

in	*filename*	Name of the file
in	*ncolumns*	Number of colums
	table	Table with contents of file

Definition at line 20 of file fileUtilities.f90.

Here is the call graph for this function:



7.9.1.2 subroutine fileutilities::removefile (character(len=∗), intent(in) *filename*)

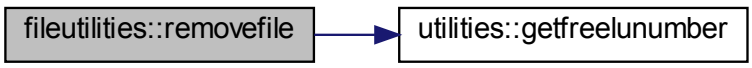
Routine to throw away a file.

Parameters

in	*filename*	file to delete

Definition at line 78 of file fileUtilities.f90.

Here is the call graph for this function:



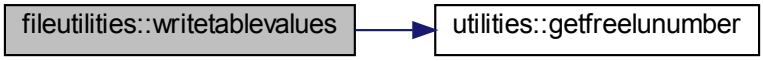
7.9.1.3 subroutine fileutilities::writetablevalues (character(len=∗), intent(in) *filename,* real(wp), dimension(:,:), intent(in) *table*)

Parameters

in	*filename*	Name of the file to write
in	*table*	Table with contents to write in file

Definition at line 51 of file fileUtilities.f90.

Here is the call graph for this function:



7.10 formulamodulertoovertopping Module Reference

Functions/Subroutines

- subroutine, public calculatewaverunup (Hm0, ksi0, ksi0Limit, gammaB, gammaF, gammaBeta, modelFactors, z2, succes, errorMessage)

  calculateWaveRunup: calculate wave runup
- subroutine, public calculatewaveovertoppingdischarge (h, Hm0, tanAlpha, gammaB, gammaF, gammaBeta, ksi0, hCrest, modelFactors, Qo, succes, errorMessage)

  calculateWaveOvertoppingDischarge: calculate the wave overtopping discharge
- subroutine, public calculatewavelength (Tm_10, L0)

  calculateWaveLength: calculate the wave length
- subroutine, public calculatewavesteepness (Hm0, Tm_10, s0, succes, errorMessage)

  calculateWaveSteepness: calculate the wave steepness
- subroutine, public calculatebreakerparameter (tanAlpha, s0, ksi0, succes, errorMessage)

  calculateBreakerParameter: calculate the breaker parameter
- subroutine, public calculateanglewaveattack (phi, psi, beta)

  calculateAngleWaveAttack: calculate the angle of wave attack
- subroutine, public calculatebreakerlimit (modelFactors, gammaB, ksi0Limit, succes, errorMessage)

  calculateBreakerLimit: calculate the breaker limit
- subroutine, public adjustinfluencefactors (gammaB, gammaF, gammaBeta, gammaBetaType, ksi0, ksi0Limit, succes, errorMessage)

  adjustInfluenceFactors: adjust the influence factors
- subroutine, public realrootscubicfunction (a, b, c, d, N, x, succes, errorMessage)

  realRootsCubicFunction: calculate the roots of a cubic function
- subroutine, public rootsgeneralcubic (a, b, c, d, z, succes, errorMessage)

  rootsGeneralCubic: calculate the roots of a generic cubic function
- subroutine, public rootsdepressedcubic (p, q, z)

  rootsDepressedCubic: calculate the roots of a depressed cubic function
- subroutine, public cubicroots (z, roots)

  cubicRoots: calculate the roots of a cubic function
- logical function, public isequalreal (x1, x2)

  isEqualReal: are two reals (almost) equal
- logical function, public isequalzero (x)

  isEqualZero: is a real (almost) zero

7.10.1 Function/Subroutine Documentation

7.10.1.1 subroutine, public formulamodulertoovertopping::adjustinfluencefactors (real(wp), intent(inout) *gammaB,* real(wp), intent(inout) *gammaF,* real(wp), intent(inout) *gammaBeta,* integer, intent(in) *gammaBetaType,* real(wp), intent(in) *ksi0,* real(wp), intent(in) *ksi0Limit,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

adjustInfluenceFactors: adjust the influence factors

Parameters

in,out	*gammab*	influence factor berms
in,out	*gammaf*	influence factor roughness
in,out	*gammabeta*	influence factor angle of wave attack
in	*gammabetatype*	type influence factor angle of wave attack: 1 = wave run-up, 2 = overtopping
in	*ksi0*	breaker parameter
in	*ksi0limit*	limit value breaker parameter
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 382 of file formulaModuleRTOovertopping.f90.

Here is the caller graph for this function:

7.10.1.2 subroutine, public formulamodulertoovertopping::calculateanglewaveattack (real(wp), intent(in) *phi,* real(wp), intent(in) *psi,* real(wp), intent(out) *beta*)

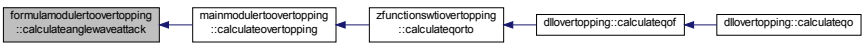
calculateAngleWaveAttack: calculate the angle of wave attack

Parameters

in	*phi*	wave direction (degree)
in	*psi*	dike normal (degree)
out	*beta*	angle of wave attack (degree)

Definition at line 285 of file formulaModuleRTOovertopping.f90.

Here is the caller graph for this function:

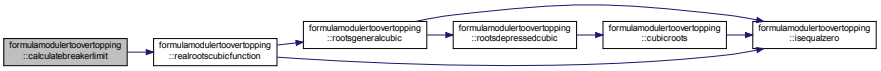


7.10.1.3 subroutine, public formulamodulertoovertopping::calculatebreakerlimit (type (tpovertoppinginput), intent(in) *modelFactors,* real(wp), intent(in) *gammaB,* real(wp), intent(out) *ksi0Limit,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

calculateBreakerLimit: calculate the breaker limit

Definition at line 308 of file formulaModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.10.1.4 subroutine, public formulamodulertoovertopping::calculatebreakerparameter (real(wp), intent(in) *tanAlpha,* real(wp), intent(in) *s0,* real(wp), intent(out) *ksi0,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

calculateBreakerParameter: calculate the breaker parameter

Parameters

in	*tanalpha*	representative slope angle
in	*s0*	wave steepness
out	*ksi0*	breaker parameter
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 244 of file formulaModuleRTOovertopping.f90.

Here is the caller graph for this function:

7.10.1.5 subroutine, public formulamodulertoovertopping::calculatewavelength (real(wp), intent(in) *Tm_10,* real(wp), intent(out) *L0*)

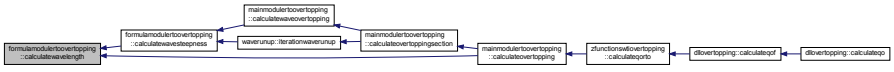
calculateWaveLength: calculate the wave length

Parameters

in	*tm_10*	spectral wave period (s)
out	*l0*	wave length (m)

Definition at line 181 of file formulaModuleRTOovertopping.f90.

Here is the caller graph for this function:



7.10.1.6 subroutine, public formulamodulertoovertopping::calculatewaveovertoppingdischarge (real(wp), intent(in) *h,* real(wp), intent(in) *Hm0,* real(wp), intent(in) *tanAlpha,* real(wp), intent(in) *gammaB,* real(wp), intent(in) *gammaF,* real(wp), intent(in) *gammaBeta,* real(wp), intent(in) *ksi0,* real(wp), intent(in) *hCrest,* type(tpovertoppinginput), intent(in) *modelFactors,* real(wp), intent(out) *Qo,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

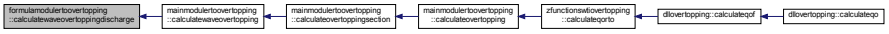
calculateWaveOvertoppingDischarge: calculate the wave overtopping discharge

Parameters

in	*h*	local water level (m+NAP)
in	*hm0*	significant wave height (m)
in	*tanalpha*	representative slope angle
in	*gammab*	influence factor berms
in	*gammaf*	influence factor roughness
in	*gammabeta*	influence factor angle of wave attack
in	*ksi0*	breaker parameter
in	*hcrest*	crest level (m+NAP)
in	*modelfactors*	structure with model factors
out	*qo*	wave overtopping discharge (l/m per s)
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 83 of file formulaModuleRTOovertopping.f90.

Here is the caller graph for this function:



7.10.1.7 subroutine, public formulamodulertoovertopping::calculatewaverunup (real(wp), intent(in) *Hm0,* real(wp), intent(in) *ksi0,* real(wp), intent(in) *ksi0Limit,* real(wp), intent(inout) *gammaB,* real(wp), intent(inout) *gammaF,* real(wp), intent(inout) *gammaBeta,* type (tpovertoppinginput), intent(in) *modelFactors,* real(wp), intent(out) *z2,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

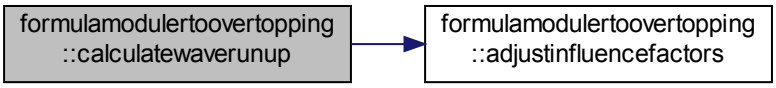
calculateWaveRunup: calculate wave runup

Parameters

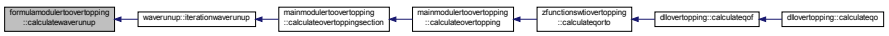
in	*hm0*	significant wave height (m)
in	*ksi0*	breaker parameter
in	*ksi0limit*	limit value breaker parameter
in,out	*gammab*	influence factor berms
in,out	*gammaf*	influence factor roughness
in,out	*gammabeta*	influence factor angle of wave attack
in	*modelfactors*	structure with model factors
out	*z2*	2% wave run-up (m)
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 34 of file formulaModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.1.8 subroutine, public formulamodulertoovertopping::calculatewavesteepness (real(wp), intent(in) *Hm0,* real(wp), intent(in) *Tm_10,* real(wp), intent(out) *s0,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

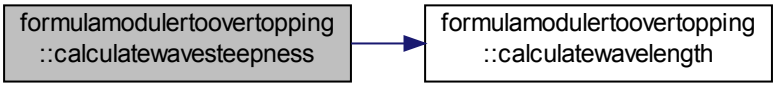
calculateWaveSteepness: calculate the wave steepness

Parameters

in	*hm0*	significant wave height (m)
in	*tm_10*	spectral wave period (s)
out	*s0*	wave steepness
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 202 of file formulaModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.10.1.9 subroutine, public formulamodulertoovertopping::cubicroots (double complex, intent(in) *z,* double complex, dimension(3), intent(out) *roots*)

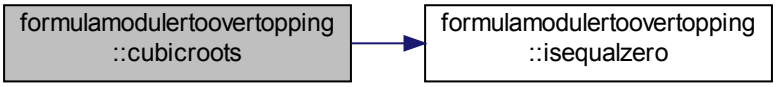
cubicRoots: calculate the roots of a cubic function

Parameters

in	*z*	complex number
out	*roots*	cubic roots

Definition at line 628 of file formulaModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.10.1.10 logical function, public formulamodulertoovertopping::isequalreal (real(wp), intent(in) *x1*, real(wp), intent(in) *x2*)

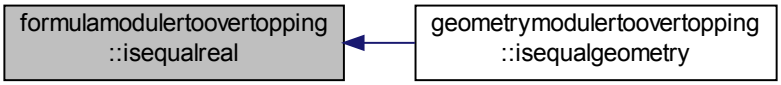
isEqualReal: are two reals (almost) equal

Parameters

in	*x1*	first real
in	*x2*	second real

Definition at line 669 of file formulaModuleRTOovertopping.f90.

Here is the caller graph for this function:



7.10.1.11 logical function, public formulamodulertoovertopping::isequalzero (real(wp), intent(in) *x*)

isEqualZero: is a real (almost) zero

Parameters

in	*x*	real number

Definition at line 693 of file formulaModuleRTOovertopping.f90.

Here is the caller graph for this function:

7.10.1.12 subroutine, public formulamodulertoovertopping::realrootscubicfunction (real(wp), intent(in) *a,* real(wp), intent(in) *b,* real(wp), intent(in) *c,* real(wp), intent(in) *d,* integer, intent(out) *N,* real(wp), dimension(3), intent(out) *x,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

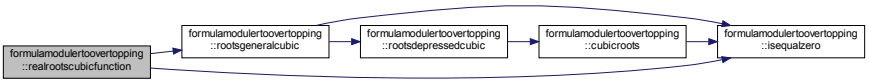
realRootsCubicFunction: calculate the roots of a cubic function

Parameters

in	*d*	coefficients cubic function
out	*n*	number of real roots cubic function
out	*x*	real roots cubic function
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 480 of file formulaModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.10.1.13 subroutine, public formulamodulertoovertopping::rootsdepressedcubic (real(wp), intent(in) *p,* real(wp), intent(in) *q,* double complex, dimension(3), intent(out) *z*)

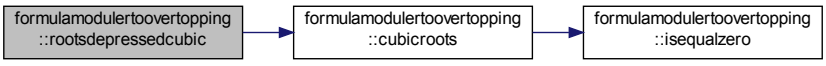
rootsDepressedCubic: calculate the roots of a depressed cubic function

Parameters

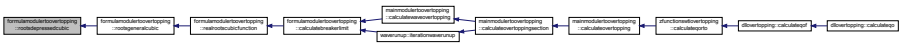
in	*q*	coefficients depressed cubic
out	*z*	roots depressed cubic

Definition at line 588 of file formulaModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.1.14 subroutine, public formulamodulertoovertopping::rootsgeneralcubic (real(wp), intent(in) *a,* real(wp), intent(in) *b,* real(wp), intent(in) *c,* real(wp), intent(in) *d,* double complex, dimension(3), intent(out) *z,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

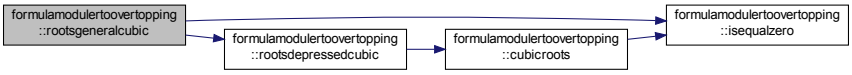
rootsGeneralCubic: calculate the roots of a generic cubic function

Parameters

in	*d*	coefficients cubic function
out	*z*	roots cubic function
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 538 of file formulaModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.11 ftnunit Module Reference

Data Types

- interface assert_comparable
- interface assert_equal
- interface assert_inbetween
- interface proc

Functions/Subroutines

- subroutine, public setruntestlevel (runtTestLevel)

  Set the run test level 0 = fast tests 1.. 2.. 3 = slow tests.
- subroutine, public settesttitle (message)

  Set the title that is displayed in the report of the test run.
- subroutine, public test (proc, text, ignore)
- subroutine, public testwithlevel (proc, text, testlevel, ignore)
- subroutine, public runtests_init
- subroutine runtests_init_priv
- subroutine, public runtests_final (stop)
- logical function issilent ()
- subroutine setworkingdir
- subroutine, public runtests (testproc)
- subroutine expect_program_stop
- subroutine, public assert_true (cond, text)
- subroutine, public assert_false (cond, text)
- subroutine assert_equal_logical (value1, value2, text)
- subroutine assert_equal_logical1d (array1, array2, text)
- subroutine assert_equal_string (value1, value2, text)
- subroutine assert_equal_string1d (array1, array2, text)

- subroutine assert_equal_int (value1, value2, text)
- subroutine assert_equal_int1d (array1, array2, text)
- subroutine assert_comparable_real (value1, value2, margin, text)
- subroutine assert_comparable_real1d (array1, array2, margin, text)
- subroutine assert_comparable_real2d (array1, array2, margin, text)
- subroutine assert_comparable_double (value1, value2, margin, text)
- subroutine assert_comparable_double1d (array1, array2, margin, text)
- subroutine assert_comparable_double2d (array1, array2, margin, text)
- subroutine assert_inbetween_real (value, vmin, vmax, text)
- subroutine assert_inbetween_double (value, vmin, vmax, text)
- subroutine, public assert_files_comparable (filename1, filename2, text, tolerance)
- logical function ftnunit_file_exists (filename)
- subroutine ftnunit_remove_file (filename)
- subroutine ftnunit_make_empty_file (filename)
- subroutine ftnunit_write_html_header
- subroutine ftnunit_write_html_footer
- subroutine colour_number (number, coloured, colour1, colour2, allowed)
- subroutine ftnunit_write_html_test_begin (text)
- subroutine ftnunit_write_html_ignored (lun)
- subroutine ftnunit_write_html_cpu (cpu)
- subroutine ftnunit_write_html_previous_failed
- subroutine ftnunit_write_html_previous_stopped
- subroutine ftnunit_write_html_close_row (lun)
- subroutine ftnunit_write_html_failed_logic (text, expected)
- subroutine ftnunit_write_html_failed_equivalent (text)
- subroutine ftnunit_write_html_failed_equivalent1d (text, idx, value1, value2, addtext)
- subroutine ftnunit_write_html_failed_string (text, value1, value2)
- subroutine ftnunit_write_html_failed_string1d (text, idx, value1, value2, addtext)
- subroutine ftnunit_write_html_failed_int (text, value1, value2)
- subroutine ftnunit_write_html_failed_int1d (text, idx, value1, value2, addtext)
- subroutine ftnunit_write_html_failed_real (text, value1, value2)
- subroutine ftnunit_write_html_failed_real1d (text, idx, value1, value2, addtext)
- subroutine ftnunit_write_html_failed_real2d (text, idx1, idx2, value1, value2, addtext)
- subroutine ftnunit_write_html_failed_double (text, value1, value2)
- subroutine ftnunit_write_html_failed_double1d (text, idx, value1, value2, addtext)
- subroutine ftnunit_write_html_failed_double2d (text, idx1, idx2, value1, value2, addtext)
- subroutine ftnunit_write_html_failed_inbetween_real (text, value, vmin, vmax)
- subroutine ftnunit_write_html_failed_inbetween_double (text, value, vmin, vmax)
- subroutine ftnunit_write_html_failed_files (text, string1, string2, string3)

Variables

- procedure(proc), pointer, public subptrtestprogaminit => null()
- procedure(proc), pointer, public subptrtestinit => null()
- integer, parameter mode_all = 0
- integer, parameter mode_single = 1
- integer, parameter mode_list = 2
- integer, parameter dp = kind(1.0d0)
- integer, save run_test_level = 99
- integer, save single_test = -1
- integer, save test_mode = mode_all
- integer, save last_test
- integer, save testno
- integer, save nofails

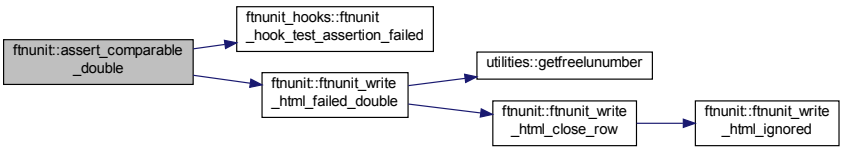
- integer, save nofails_prev
- integer, save notests_run
- integer, save notests_failed
- integer, save notests_ignored
- integer, save notests_work_in_progress
- integer, save noruns
- logical, save call_final = .true.
- logical, save previous = .false.
- integer, save failed_asserts = 0
- logical, save has_run = .false.
- logical, save ignore_test = .false.
- logical, save ignore_previous_test = .false.
- character(len=20), save html_file = 'ftnunit.html'
- character(len=80), save testname
- character(len=80), save testtitle = 'Result of unit tests'
- character(len=32), save ignore_reason
- character(len=32), save ignore_reason_previous_test = ' '
- character(len=∗), parameter work_in_progress = 'work-in-progress'

7.11.1 Function/Subroutine Documentation

7.11.1.1 subroutine ftnunit::assert_comparable_double (real(kind=dp), intent(in) *value1,* real(kind=dp), intent(in) *value2,* real(kind=dp), intent(in) *margin,* character(len=∗), intent(in) *text*) [private]

Definition at line 966 of file ftnunit.f90.

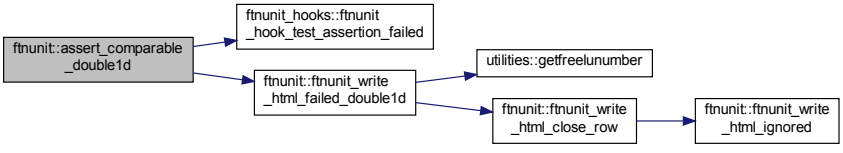
Here is the call graph for this function:



7.11.1.2 subroutine ftnunit::assert_comparable_double1d (real(kind=dp), dimension(:), intent(in) *array1,* real(kind=dp), dimension(:), intent(in) *array2,* real(kind=dp), intent(in) *margin,* character(len=∗), intent(in) *text*) [private]

Definition at line 997 of file ftnunit.f90.

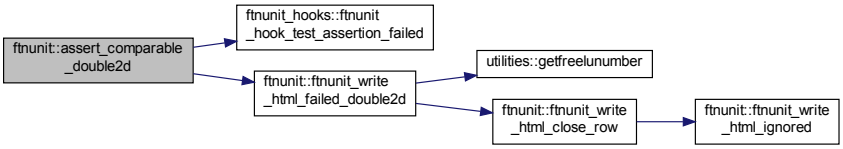
Here is the call graph for this function:



7.11.1.3 subroutine ftnunit::assert_comparable_double2d (real(kind=dp), dimension(:,:), intent(in) *array1,* real(kind=dp), dimension(:,:), intent(in) *array2,* real(kind=dp), intent(in) *margin,* character(len=∗), intent(in) *text*) [private]

Definition at line 1061 of file ftnunit.f90.

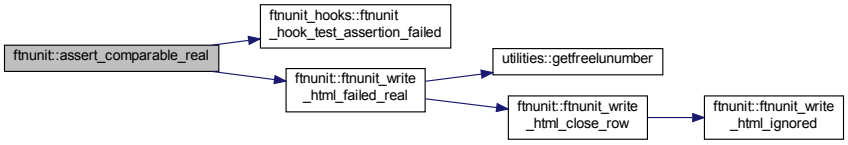
Here is the call graph for this function:



7.11.1.4 subroutine ftnunit::assert_comparable_real (real, intent(in) *value1,* real, intent(in) *value2,* real, intent(in) *margin,* character(len=∗), intent(in) *text*) [private]

Definition at line 802 of file ftnunit.f90.

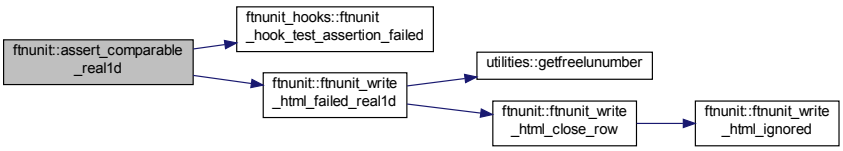
Here is the call graph for this function:



7.11.1.5 subroutine ftnunit::assert_comparable_real1d (real, dimension(:), intent(in) *array1,* real, dimension(:), intent(in) *array2,* real, intent(in) *margin,* character(len=∗), intent(in) *text*) [private]

Definition at line 834 of file ftnunit.f90.

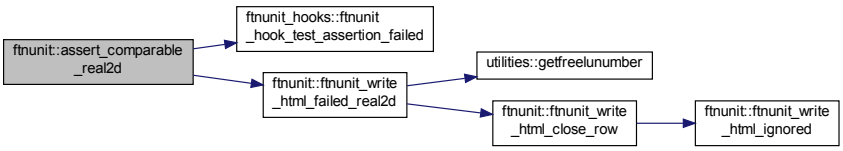
Here is the call graph for this function:



7.11.1.6 subroutine ftnunit::assert_comparable_real2d (real, dimension(:,:), intent(in) *array1,* real, dimension(:,:), intent(in) *array2,* real, intent(in) *margin,* character(len=∗), intent(in) *text*) [private]

Definition at line 898 of file ftnunit.f90.

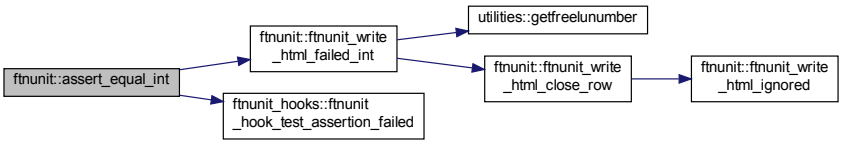
Here is the call graph for this function:



7.11.1.7 subroutine ftnunit::assert_equal_int (integer, intent(in) *value1,* integer, intent(in) *value2,* character(len=∗), intent(in) *text*) [private]

Definition at line 724 of file ftnunit.f90.

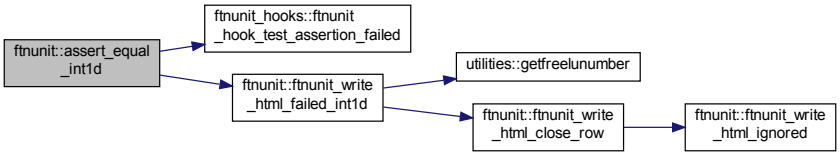
Here is the call graph for this function:



7.11.1.8 subroutine ftnunit::assert_equal_int1d (integer, dimension(:), intent(in) *array1,* integer, dimension(:), intent(in) *array2,* character(len=∗), intent(in) *text*) [private]

Definition at line 748 of file ftnunit.f90.

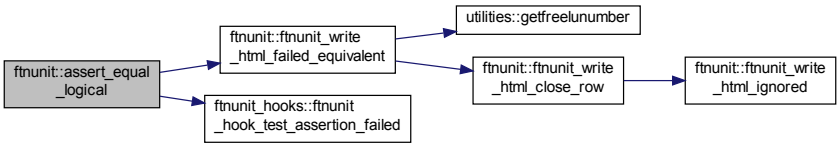
Here is the call graph for this function:



7.11.1.9 subroutine ftnunit::assert_equal_logical (logical, intent(in) *value1,* logical, intent(in) *value2,* character(len=∗), intent(in) *text*) [private]

Definition at line 575 of file ftnunit.f90.

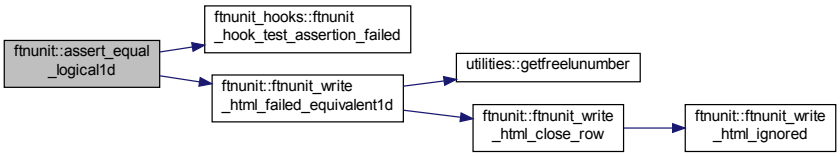
Here is the call graph for this function:



7.11.1.10 subroutine ftnunit::assert_equal_logical1d (logical, dimension(:), intent(in) *array1,* logical, dimension(:), intent(in) *array2,* character(len=∗), intent(in) *text*) [private]

Definition at line 599 of file ftnunit.f90.

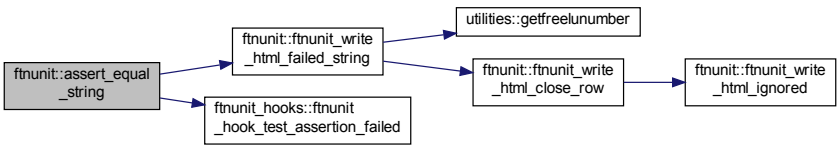
Here is the call graph for this function:



7.11.1.11 subroutine ftnunit::assert_equal_string (character(len=∗), intent(in) *value1,* character(len=∗), intent(in) *value2,* character(len=∗), intent(in) *text*) [private]

Definition at line 649 of file ftnunit.f90.

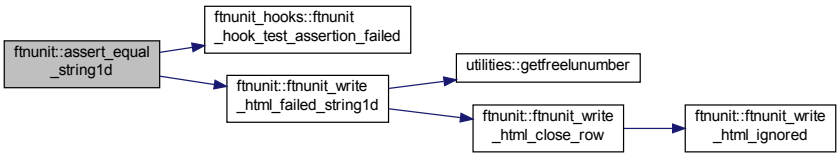
Here is the call graph for this function:



7.11.1.12 subroutine ftnunit::assert_equal_string1d (character(len=∗), dimension(:), intent(in) *array1,* character(len=∗), dimension(:), intent(in) *array2,* character(len=∗), intent(in) *text*) [private]

Definition at line 675 of file ftnunit.f90.

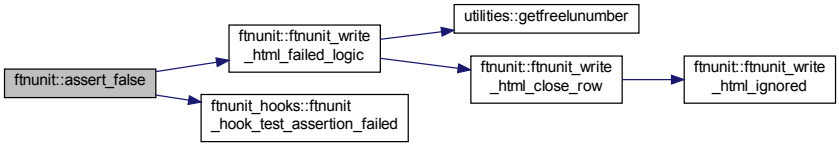
Here is the call graph for this function:



7.11.1.13 subroutine, public ftnunit::assert_false (logical, intent(in) *cond,* character(len=∗), intent(in) *text*)

Definition at line 552 of file ftnunit.f90.

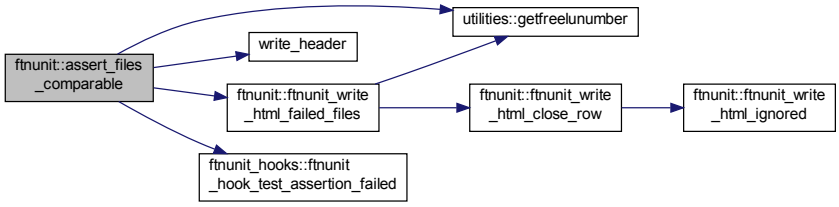
Here is the call graph for this function:



7.11.1.14 subroutine, public ftnunit::assert_files_comparable (character(len=∗), intent(in) *filename1,* character(len=∗), intent(in) *filename2,* character(len=∗), intent(in) *text,* real, intent(in), optional *tolerance*)

Definition at line 1213 of file ftnunit.f90.

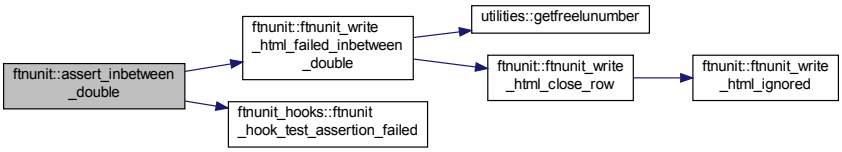
Here is the call graph for this function:



7.11.1.15 subroutine ftnunit::assert_inbetween_double (real(kind=dp), intent(in) *value,* real(kind=dp), intent(in) *vmin,* real(kind=dp), intent(in) *vmax,* character(len=∗), intent(in) *text*) [private]

Definition at line 1168 of file ftnunit.f90.

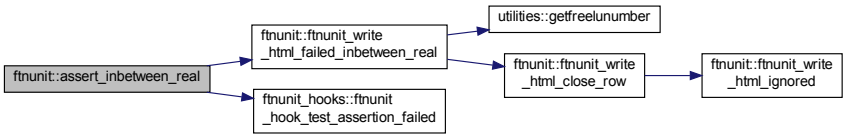
Here is the call graph for this function:



7.11.1.16 subroutine ftnunit::assert_inbetween_real (real, intent(in) *value,* real, intent(in) *vmin,* real, intent(in) *vmax,* character(len=∗), intent(in) *text*) [private]

Definition at line 1129 of file ftnunit.f90.

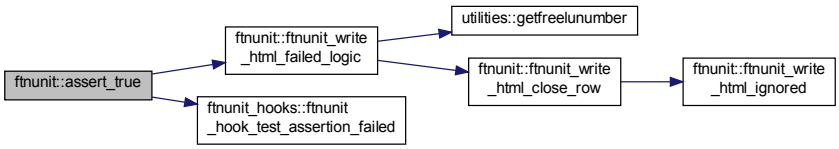
Here is the call graph for this function:



7.11.1.17 subroutine, public ftnunit::assert_true (logical, intent(in) *cond,* character(len=∗), intent(in) *text*)

Definition at line 530 of file ftnunit.f90.

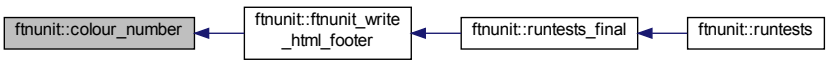
Here is the call graph for this function:



7.11.1.18 subroutine ftnunit::colour_number (integer, intent(in) *number,* character(len=∗), intent(out) *coloured,* character(len=∗), intent(in) *colour1,* character(len=∗), intent(in), optional *colour2,* integer, intent(in), optional *allowed*) [private]

Definition at line 1567 of file ftnunit.f90.

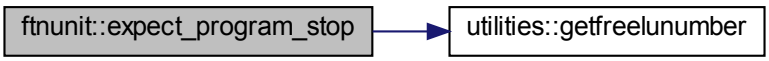
Here is the caller graph for this function:



7.11.1.19 subroutine ftnunit::expect_program_stop ( ) [private]

Definition at line 509 of file ftnunit.f90.

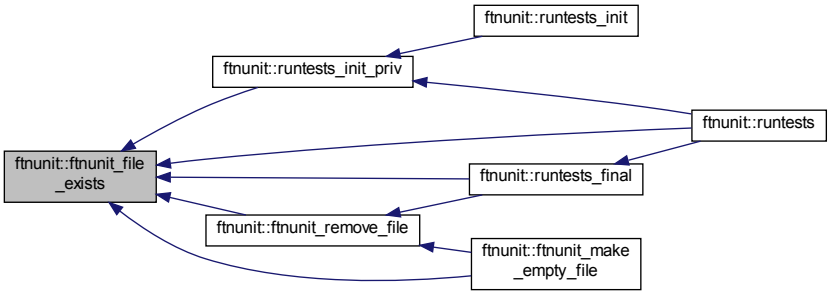
Here is the call graph for this function:



7.11.1.20 logical function ftnunit::ftnunit_file_exists (character(len=*), intent(in) *filename*) [private]

Definition at line 1424 of file ftnunit.f90.

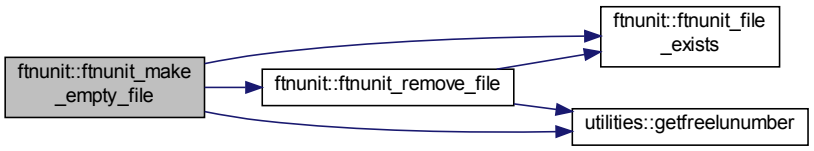
Here is the caller graph for this function:



7.11.1.21 subroutine ftnunit::ftnunit_make_empty_file (character(len=*), intent(in) *filename*) [private]

Definition at line 1461 of file ftnunit.f90.

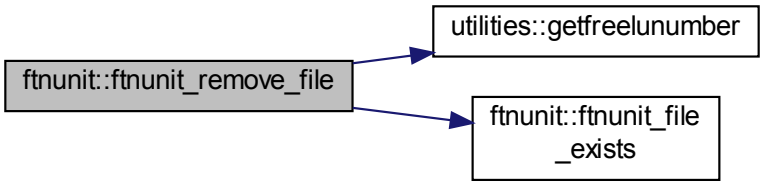
Here is the call graph for this function:



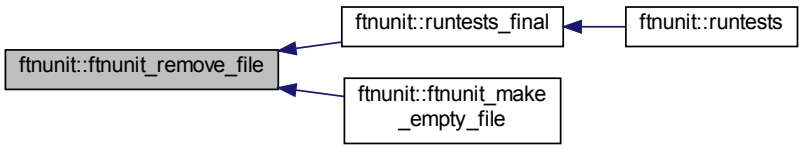
7.11.1.22 subroutine ftnunit::ftnunit_remove_file (character(len=∗), intent(in) *filename*) [private]

Definition at line 1435 of file ftnunit.f90.

Here is the call graph for this function:



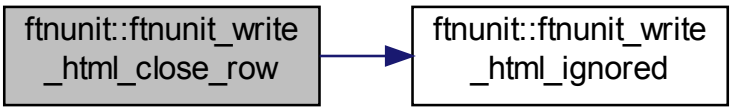
Here is the caller graph for this function:



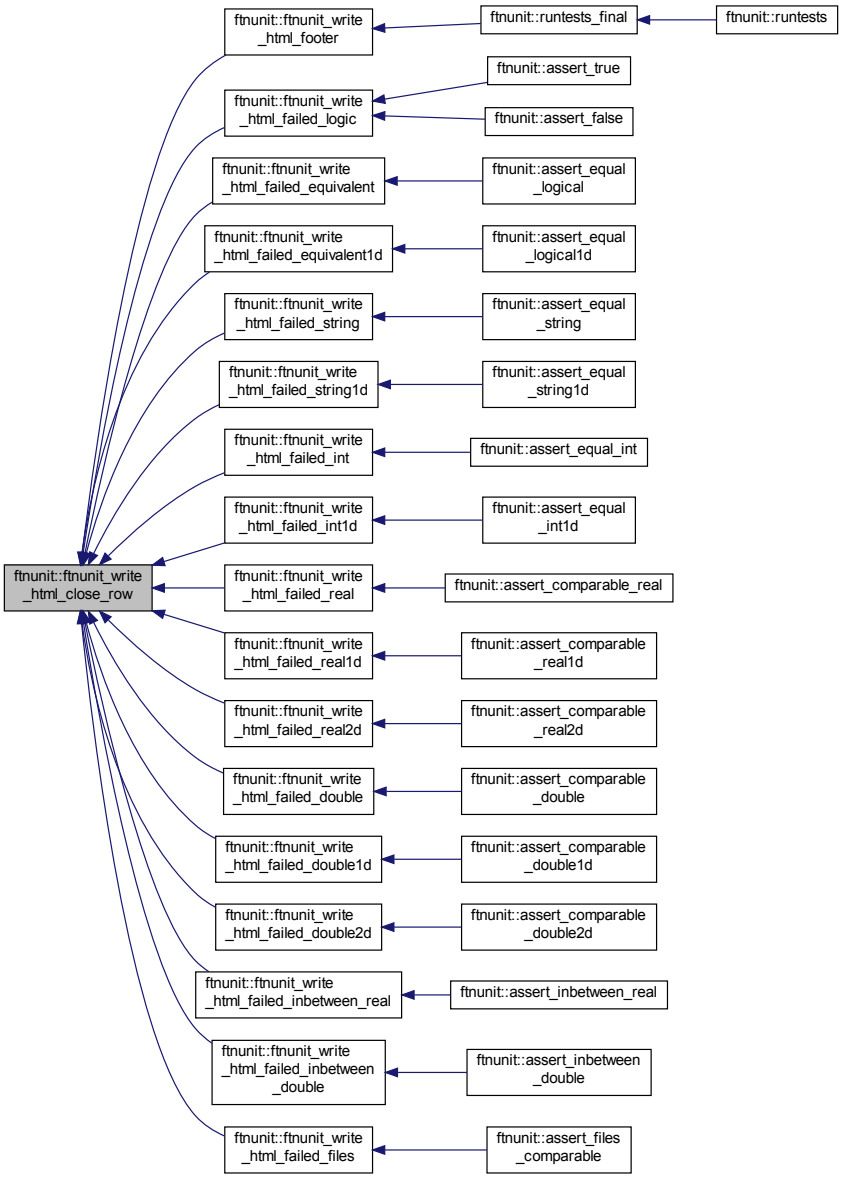
7.11.1.23 subroutine ftnunit::ftnunit_write_html_close_row (integer *lun*) [private]

Definition at line 1708 of file ftnunit.f90.

Here is the call graph for this function:



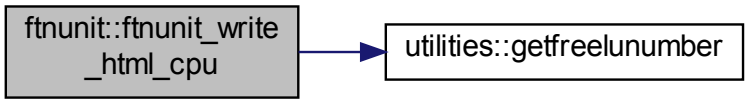
Here is the caller graph for this function:



7.11.1.24 subroutine ftnunit::ftnunit_write_html_cpu (real(kind=wp) *cpu*) [private]

Definition at line 1636 of file ftnunit.f90.

Here is the call graph for this function:

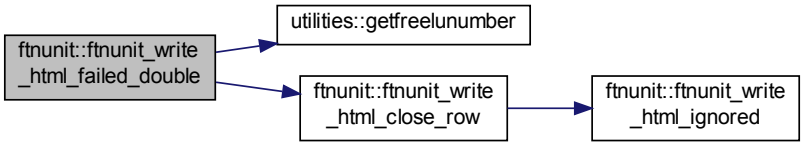


Here is the caller graph for this function:

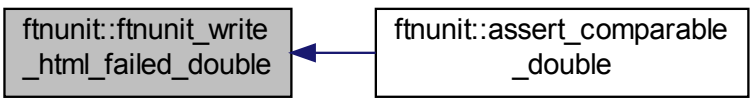
7.11.1.25 subroutine ftnunit::ftnunit_write_html_failed_double (character(len=∗) *text,* real(kind=dp) *value1,* real(kind=dp) *value2*) [private]

Definition at line 2086 of file ftnunit.f90.

Here is the call graph for this function:



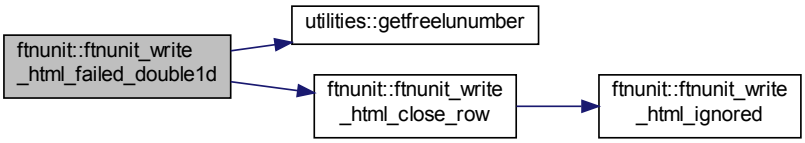
Here is the caller graph for this function:



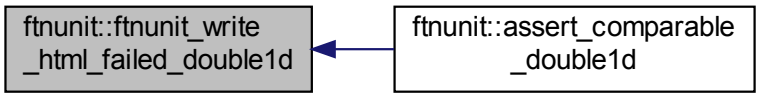
7.11.1.26 subroutine ftnunit::ftnunit_write_html_failed_double1d (character(len=∗) *text,* integer *idx,* real(kind=dp) *value1,* real(kind=dp) *value2,* logical *addtext*) [private]

Definition at line 2120 of file ftnunit.f90.

Here is the call graph for this function:



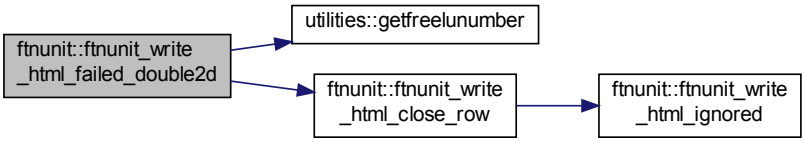
Here is the caller graph for this function:



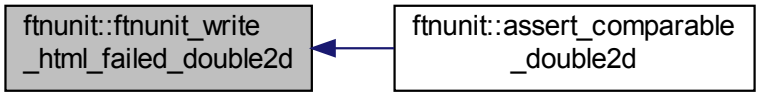
7.11.1.27 subroutine ftnunit::ftnunit_write_html_failed_double2d (character(len=∗) *text,* integer *idx1,* integer *idx2,* real(kind=dp) *value1,* real(kind=dp) *value2,* logical *addtext*) [private]

Definition at line 2162 of file ftnunit.f90.

Here is the call graph for this function:



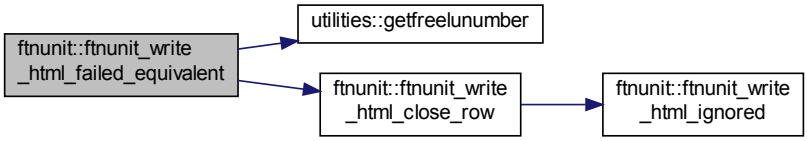
Here is the caller graph for this function:



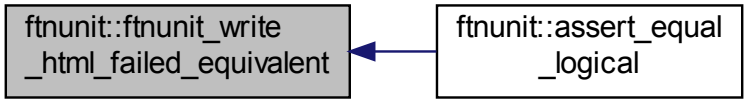
7.11.1.28 subroutine ftnunit::ftnunit_write_html_failed_equivalent (character(len=∗) *text*) [private]

Definition at line 1759 of file ftnunit.f90.

Here is the call graph for this function:



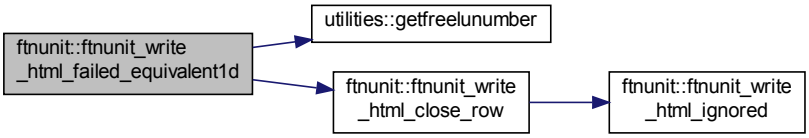
Here is the caller graph for this function:



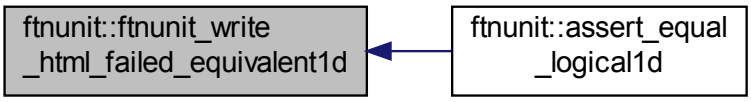
7.11.1.29 subroutine ftnunit::ftnunit_write_html_failed_equivalent1d (character(len=∗) *text,* integer *idx,* logical *value1,* logical *value2,* logical *addtext*) [private]

Definition at line 1788 of file ftnunit.f90.

Here is the call graph for this function:



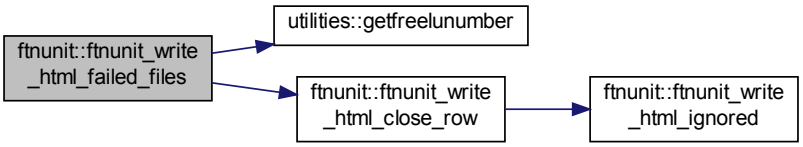
Here is the caller graph for this function:



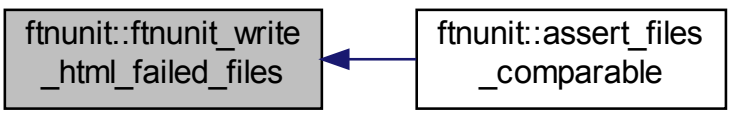
7.11.1.30 **subroutine ftnunit::ftnunit_write_html_failed_files (character(len=∗) *text,* character(len=∗) *string1,* character(len=∗) *string2,* character(len=∗) *string3*)** `[private]`

Definition at line 2267 of file ftnunit.f90.

Here is the call graph for this function:



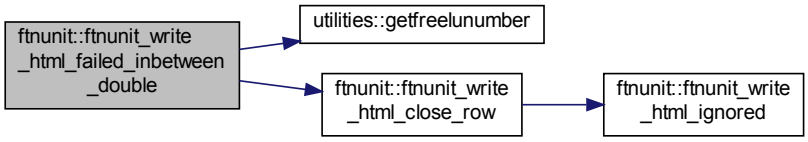
Here is the caller graph for this function:



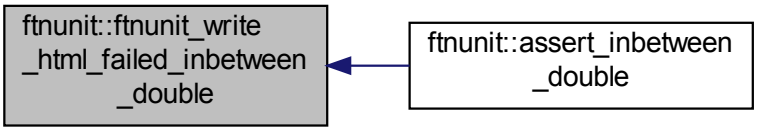
7.11.1.31 subroutine ftnunit::ftnunit_write_html_failed_inbetween_double (character(len=∗) *text*, real(kind=dp) *value*, real(kind=dp) *vmin*, real(kind=dp) *vmax*) [private]

Definition at line 2236 of file ftnunit.f90.

Here is the call graph for this function:



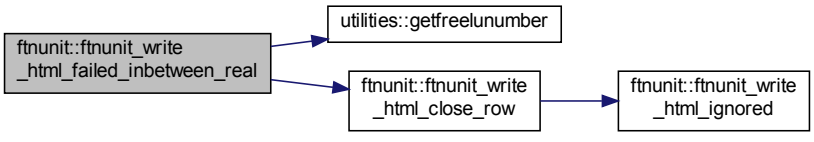
Here is the caller graph for this function:



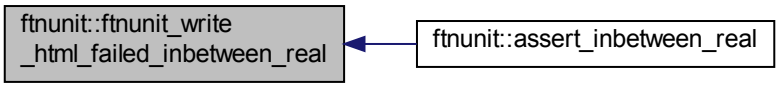
7.11.1.32 subroutine ftnunit::ftnunit_write_html_failed_inbetween_real (character(len=∗) *text*, real *value*, real *vmin*, real *vmax*) [private]

Definition at line 2204 of file ftnunit.f90.

Here is the call graph for this function:



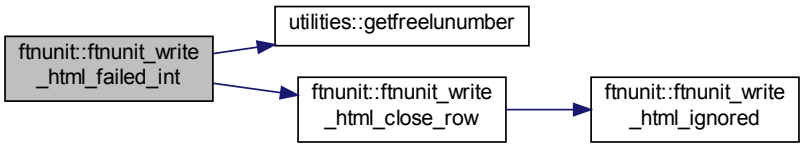
Here is the caller graph for this function:



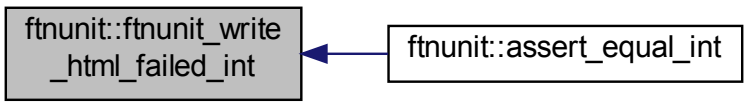
7.11.1.33 subroutine ftnunit::ftnunit_write_html_failed_int (character(len=*) *text,* integer *value1,* integer *value2*) [private]

Definition at line 1899 of file ftnunit.f90.

Here is the call graph for this function:



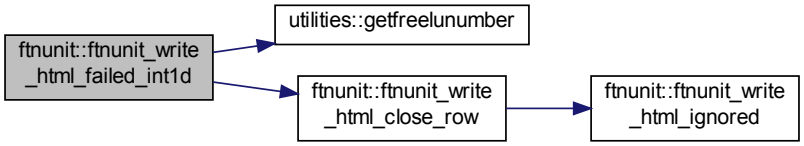
Here is the caller graph for this function:



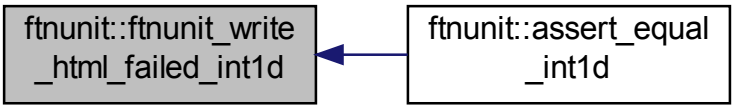
7.11.1.34 subroutine ftnunit::ftnunit_write_html_failed_int1d (character(len=*) *text,* integer *idx,* integer *value1,* integer *value2,* logical *addtext*) [private]

Definition at line 1932 of file ftnunit.f90.

Here is the call graph for this function:



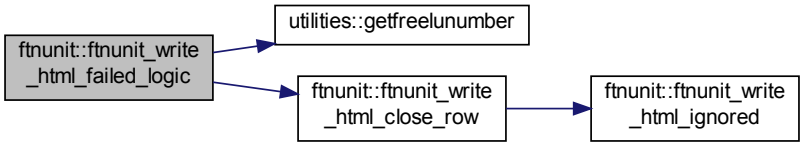
Here is the caller graph for this function:



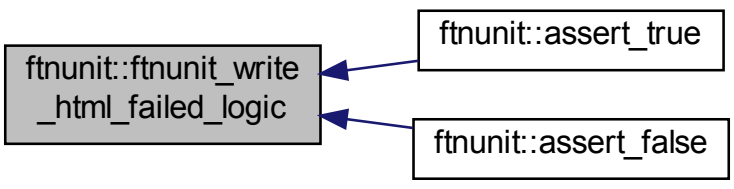
7.11.1.35 subroutine ftnunit::ftnunit_write_html_failed_logic (character(len=*) *text,* logical *expected*) [private]

Definition at line 1733 of file ftnunit.f90.

Here is the call graph for this function:



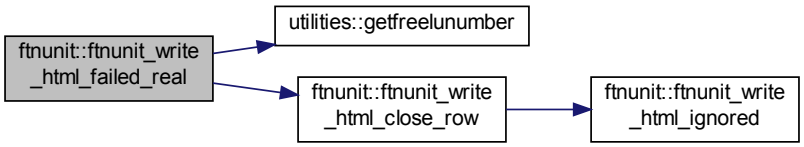
Here is the caller graph for this function:



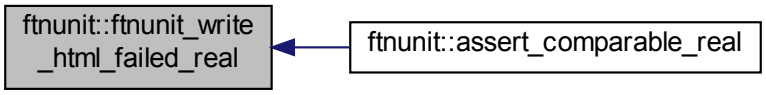
7.11.1.36 subroutine ftnunit::ftnunit_write_html_failed_real (character(len=*) *text,* real *value1,* real *value2*) [private]

Definition at line 1972 of file ftnunit.f90.

Here is the call graph for this function:



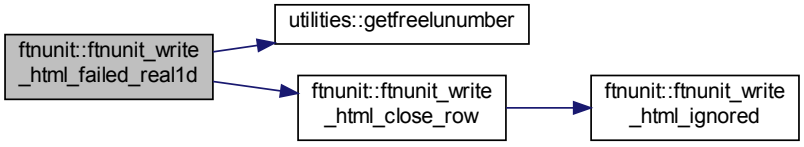
Here is the caller graph for this function:



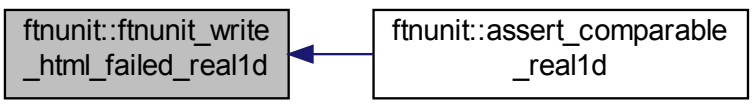
7.11.1.37 subroutine ftnunit::ftnunit_write_html_failed_real1d (character(len=*) *text,* integer *idx,* real *value1,* real *value2,* logical *addtext*) [private]

Definition at line 2004 of file ftnunit.f90.

Here is the call graph for this function:



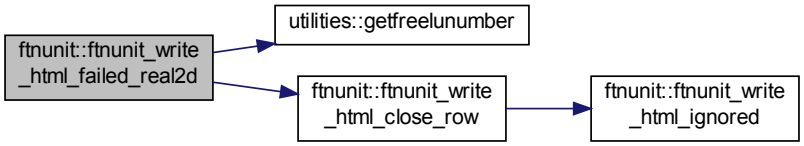
Here is the caller graph for this function:



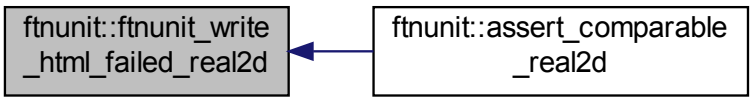
7.11.1.38 subroutine ftnunit::ftnunit_write_html_failed_real2d (character(len=*) *text,* integer *idx1,* integer *idx2,* real *value1,* real *value2,* logical *addtext*) [private]

Definition at line 2046 of file ftnunit.f90.

Here is the call graph for this function:



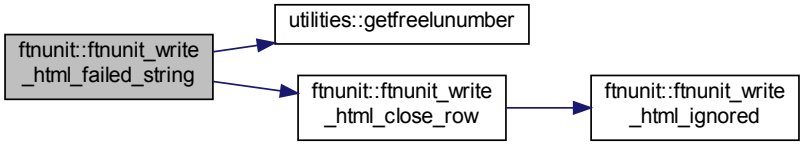
Here is the caller graph for this function:



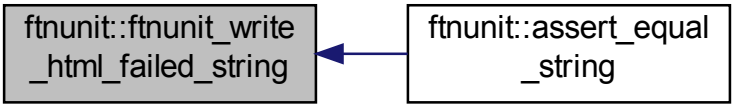
7.11.1.39 subroutine ftnunit::ftnunit_write_html_failed_string (character(len=*) *text,* character(len=*) *value1,* character(len=*) *value2*) [private]

Definition at line 1827 of file ftnunit.f90.

Here is the call graph for this function:



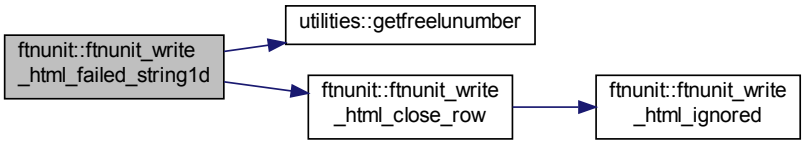
Here is the caller graph for this function:



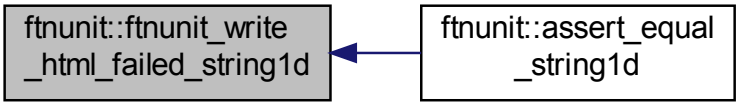
7.11.1.40 **subroutine ftnunit::ftnunit_write_html_failed_string1d (character(len=∗) *text,* integer *idx,* character(len=∗) *value1,* character(len=∗) *value2,* logical *addtext*)** [private]

Definition at line 1859 of file ftnunit.f90.

Here is the call graph for this function:



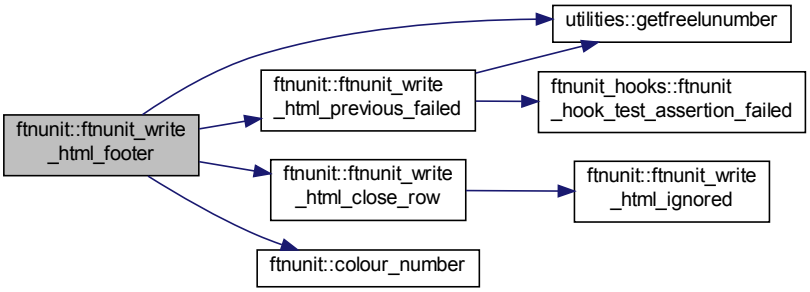
Here is the caller graph for this function:



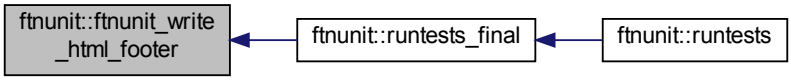
7.11.1.41 **subroutine ftnunit::ftnunit_write_html_footer ( )** [private]

Definition at line 1525 of file ftnunit.f90.

Here is the call graph for this function:



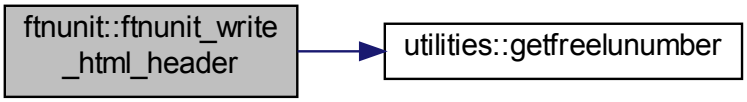
Here is the caller graph for this function:



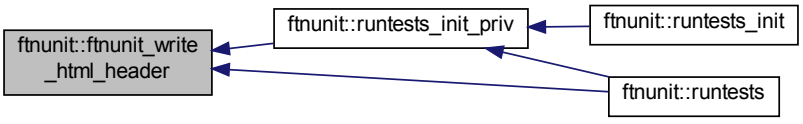
7.11.1.42 subroutine ftnunit::ftnunit_write_html_header ( ) [private]

Definition at line 1486 of file ftnunit.f90.

Here is the call graph for this function:



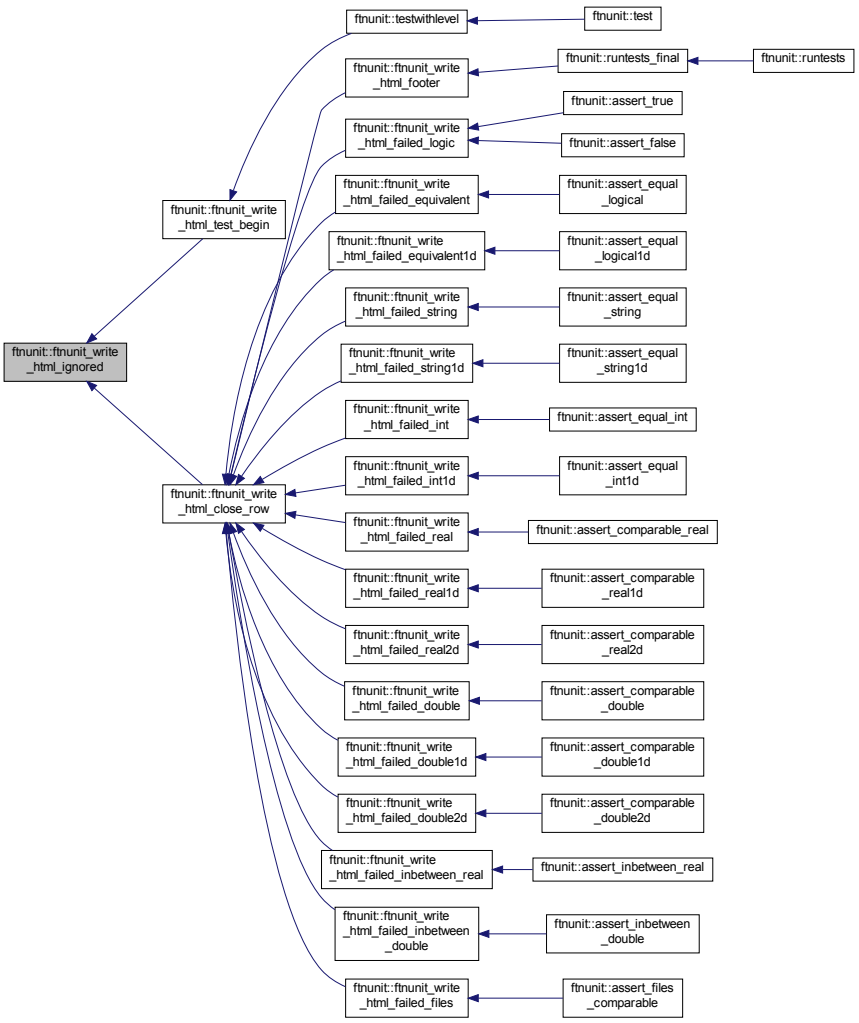
Here is the caller graph for this function:



7.11.1.43 subroutine ftnunit::ftnunit_write_html_ignored (integer, intent(in) *lun*) [private]

Definition at line 1622 of file ftnunit.f90.

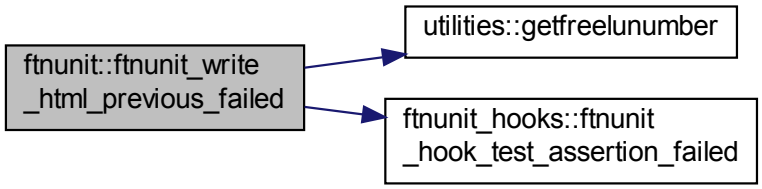
Here is the caller graph for this function:



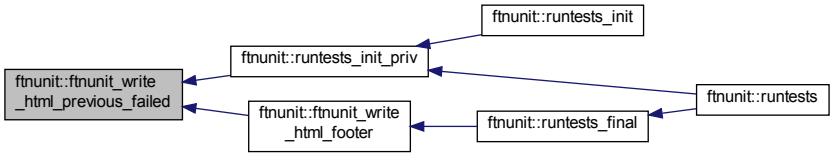
7.11.1.44 subroutine ftnunit::ftnunit_write_html_previous_failed ( )

Definition at line 1661 of file ftnunit.f90.

Here is the call graph for this function:



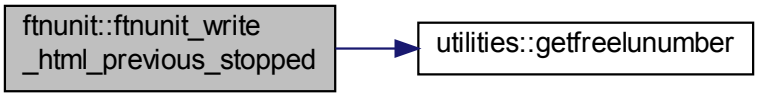
Here is the caller graph for this function:



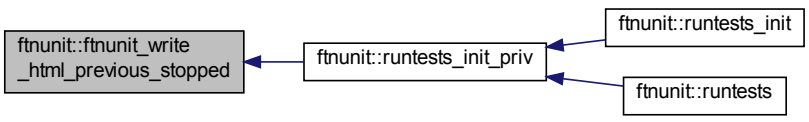
7.11.1.45 subroutine ftnunit::ftnunit_write_html_previous_stopped ( ) [private]

Definition at line 1688 of file ftnunit.f90.

Here is the call graph for this function:



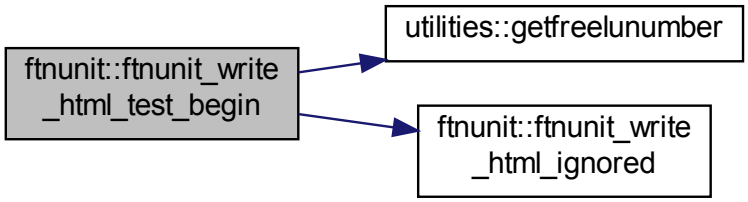
Here is the caller graph for this function:



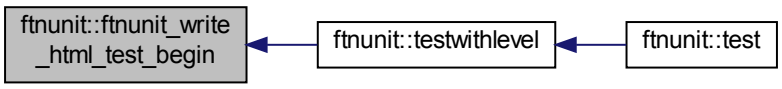
7.11.1.46 subroutine ftnunit::ftnunit_write_html_test_begin (character(len=*) *text*) [private]

Definition at line 1593 of file ftnunit.f90.

Here is the call graph for this function:



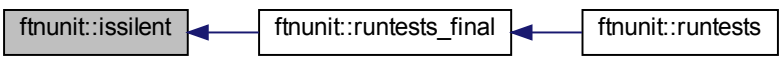
Here is the caller graph for this function:



7.11.1.47 logical function ftnunit::issilent ( ) [private]

Definition at line 411 of file ftnunit.f90.

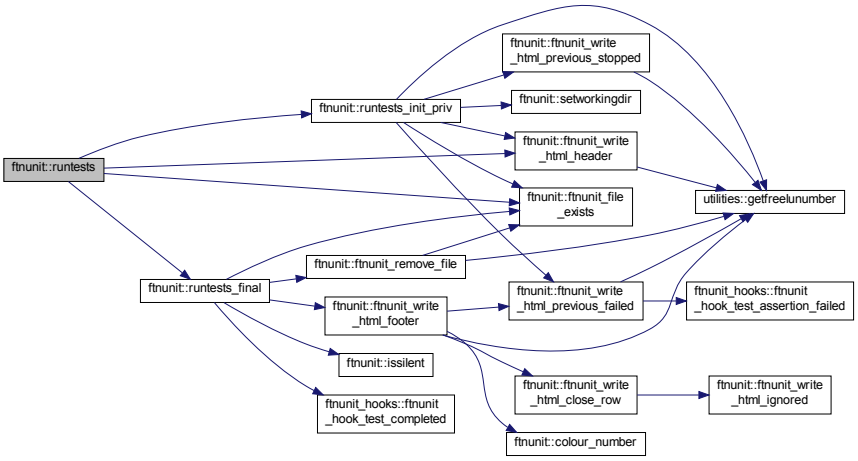
Here is the caller graph for this function:



7.11.1.48 subroutine, public ftnunit::runtests (*testproc*)

Definition at line 463 of file ftnunit.f90.

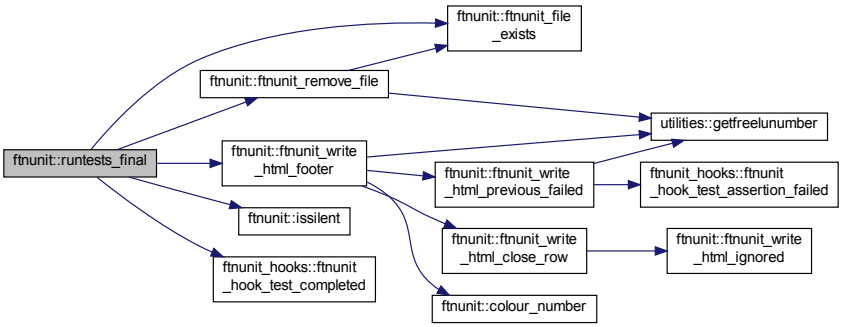
Here is the call graph for this function:



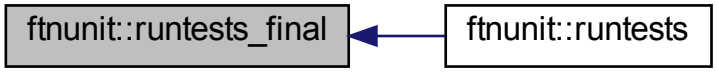
7.11.1.49 subroutine, public ftnunit::runtests_final (logical, intent(in), optional *stop*)

Definition at line 383 of file ftnunit.f90.

Here is the call graph for this function:



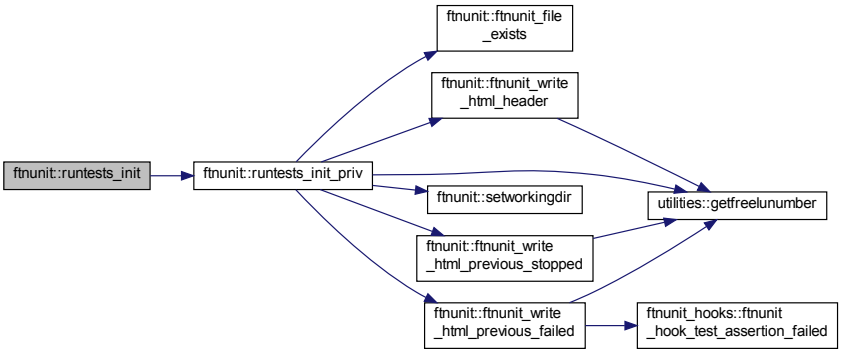
Here is the caller graph for this function:



7.11.1.50 subroutine, public ftnunit::runtests_init ( )

Definition at line 266 of file ftnunit.f90.

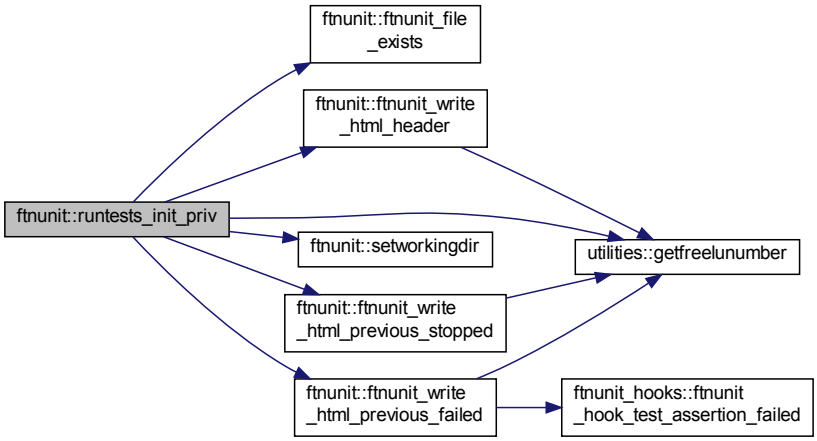
Here is the call graph for this function:



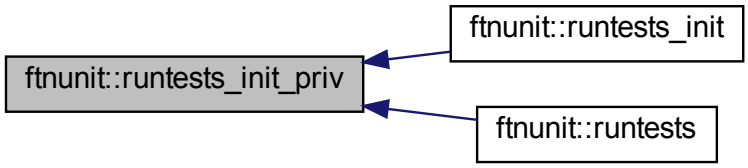
7.11.1.51 subroutine ftnunit::runtests_init_priv ( ) [private]

Definition at line 278 of file ftnunit.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.1.52 subroutine, public ftnunit::setruntestlevel (integer, intent(in) *runtTestLevel*)

Set the run test level 0 = fast tests 1.. 2.. 3 = slow tests.

Parameters

in	*runttestlevel*	set the new run testlevel to this value

Definition at line 112 of file ftnunit.f90.

7.11.1.53 subroutine, public ftnunit::settesttitle (character(len=∗), intent(in) *message*)

Set the title that is displayed in the report of the test run.

Parameters

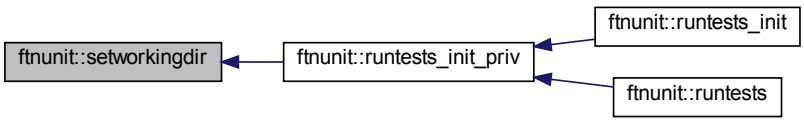
in	*message*	the title to be set

Definition at line 120 of file ftnunit.f90.

7.11.1.54 subroutine ftnunit::setworkingdir ( ) [private]

Definition at line 430 of file ftnunit.f90.

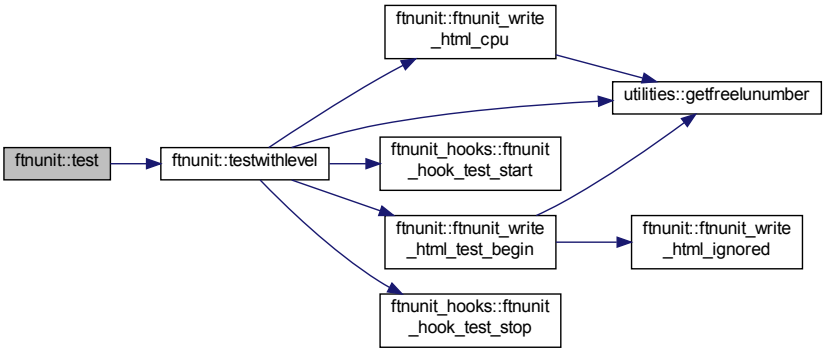
Here is the caller graph for this function:



7.11.1.55 subroutine, public ftnunit::test (external *proc,* character(len=∗) *text,* logical, intent(in), optional *ignore*)

Definition at line 131 of file ftnunit.f90.

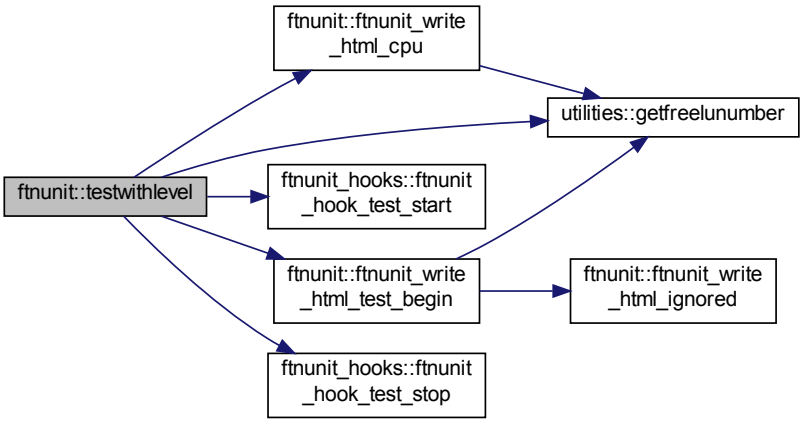
Here is the call graph for this function:



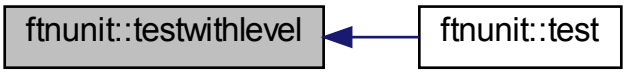
7.11.1.56 subroutine, public ftnunit::testwithlevel (external *proc,* character(len=∗) *text,* integer, intent(in) *testlevel,* logical, intent(in), optional *ignore*)

Definition at line 145 of file ftnunit.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.2 Variable Documentation

7.11.2.1 logical, save ftnunit::call_final = .true.

Definition at line 68 of file ftnunit.f90.

7.11.2.2 integer, parameter ftnunit::dp = kind(1.0d0)

Definition at line 54 of file ftnunit.f90.

7.11.2.3 integer, save ftnunit::failed_asserts = 0

Definition at line 70 of file ftnunit.f90.

7.11.2.4 logical, save ftnunit::has_run = .false.

Definition at line 71 of file ftnunit.f90.

7.11.2.5 character(len=20), save ftnunit::html_file = 'ftnunit.html'

Definition at line 74 of file ftnunit.f90.

7.11.2.6 logical, save ftnunit::ignore_previous_test = .false.

Definition at line 73 of file ftnunit.f90.

7.11.2.7 character(len=32), save ftnunit::ignore_reason

Definition at line 77 of file ftnunit.f90.

7.11.2.8 character(len=32), save ftnunit::ignore_reason_previous_test = ' '

Definition at line 78 of file ftnunit.f90.

7.11.2.9 logical, save ftnunit::ignore_test = .false.

Definition at line 72 of file ftnunit.f90.

7.11.2.10 integer, save ftnunit::last_test

Definition at line 59 of file ftnunit.f90.

7.11.2.11 integer, parameter ftnunit::mode_all = 0

Definition at line 50 of file ftnunit.f90.

7.11.2.12 integer, parameter ftnunit::mode_list = 2

Definition at line 52 of file ftnunit.f90.

7.11.2.13 integer, parameter ftnunit::mode_single = 1

Definition at line 51 of file ftnunit.f90.

7.11.2.14 integer, save ftnunit::nofails

Definition at line 61 of file ftnunit.f90.

7.11.2.15 integer, save ftnunit::nofails_prev

Definition at line 62 of file ftnunit.f90.

7.11.2.16 integer, save ftnunit::noruns

Definition at line 67 of file ftnunit.f90.

7.11.2.17 integer, save ftnunit::notests_failed

Definition at line 64 of file ftnunit.f90.

7.11.2.18 integer, save ftnunit::notests_ignored

Definition at line 65 of file ftnunit.f90.

7.11.2.19 integer, save ftnunit::notests_run

Definition at line 63 of file ftnunit.f90.

7.11.2.20 integer, save ftnunit::notests_work_in_progress

Definition at line 66 of file ftnunit.f90.

7.11.2.21 logical, save ftnunit::previous = .false.

Definition at line 69 of file ftnunit.f90.

7.11.2.22 integer, save ftnunit::run_test_level = 99

Definition at line 56 of file ftnunit.f90.

7.11.2.23 integer, save ftnunit::single_test = -1

Definition at line 57 of file ftnunit.f90.

7.11.2.24 procedure (proc), pointer, public ftnunit::subptrtestinit => null()

Definition at line 44 of file ftnunit.f90.

7.11.2.25 procedure (proc), pointer, public ftnunit::subptrtestprograminit => null()

Definition at line 43 of file ftnunit.f90.

7.11.2.26 integer, save ftnunit::test_mode = mode_all

Definition at line 58 of file ftnunit.f90.

7.11.2.27 character(len=80), save ftnunit::testname

Definition at line 75 of file ftnunit.f90.

7.11.2.28 integer, save ftnunit::testno

Definition at line 60 of file ftnunit.f90.

7.11.2.29 character(len=80), save ftnunit::testtitle = 'Result of unit tests'

Definition at line 76 of file ftnunit.f90.

7.11.2.30 character(len=*), parameter ftnunit::work_in_progress = 'work-in-progress'

Definition at line 79 of file ftnunit.f90.

7.12 ftnunit_hooks Module Reference

Functions/Subroutines

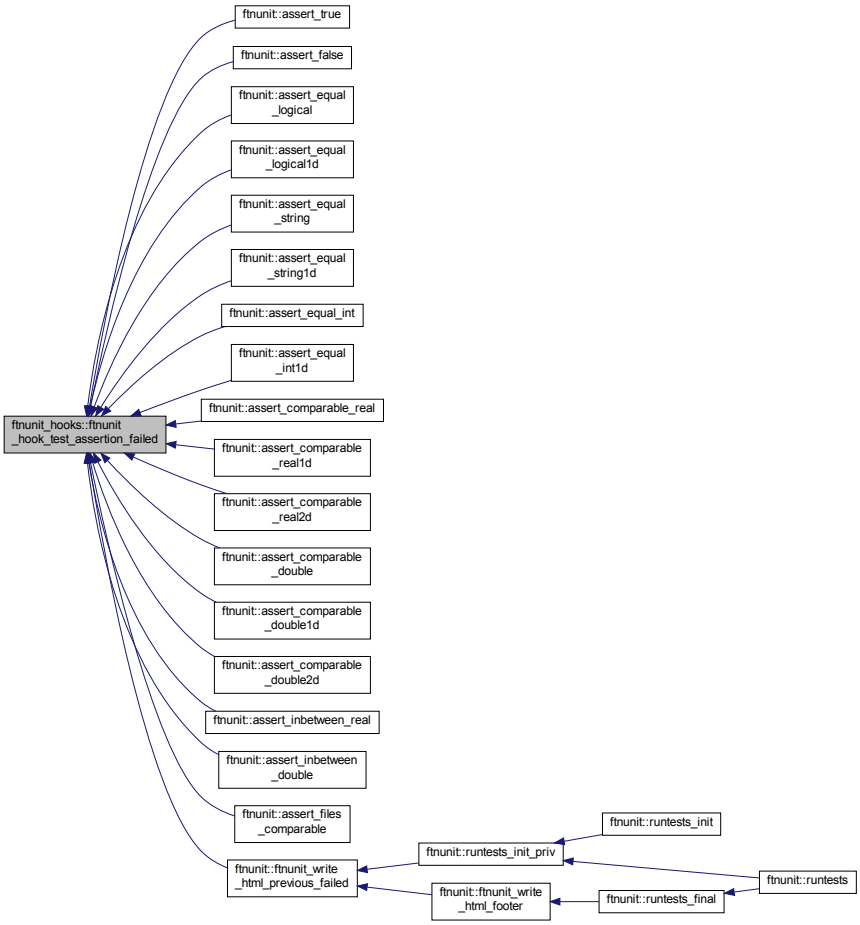
- subroutine ftnunit_hook_test_start (text)
- subroutine ftnunit_hook_test_stop (text)
- subroutine ftnunit_hook_test_assertion_failed (text, assert_text, failure_text)
- subroutine ftnunit_hook_test_completed

7.12.1 Function/Subroutine Documentation

7.12.1.1 subroutine ftnunit_hooks::ftnunit_hook_test_assertion_failed (character(len=*) *text*, character(len=*) *assert_text*, character(len=*) *failure_text*)

Definition at line 57 of file ftnunit_hooks.f90.

Here is the caller graph for this function:



7.12.1.2 subroutine ftnunit_hooks::ftnunit_hook_test_completed ( )

Definition at line 71 of file ftnunit_hooks.f90.

Here is the caller graph for this function:

7.12.1.3 subroutine ftnunit_hooks::ftnunit_hook_test_start (character(len=∗) *text*)

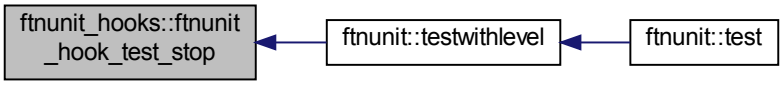
Definition at line 31 of file ftnunit_hooks.f90.

Here is the caller graph for this function:

7.12.1.4 subroutine ftnunit_hooks::ftnunit_hook_test_stop (character(len=∗) *text*)

Definition at line 43 of file ftnunit_hooks.f90.

Here is the caller graph for this function:



7.13 general Module Reference

7.14 geometrymodulertoovertopping Module Reference

Functions/Subroutines

- subroutine, public checkcrosssection (psi, nCoordinates, xCoordinates, yCoordinates, roughnessFactors, succes, errorMessage)

  checkCrossSection: check cross section
- subroutine, public initializegeometry (psi, nCoordinates, xCoordinates, yCoordinates, roughnessFactors, geometry, succes, errorMessage)

  initializeGeometry: initialize the geometry
- subroutine, public allocatevectorsgeometry (nCoordinates, geometry)

  allocateVectorsGeometry: allocate the geometry vectors
- subroutine, public deallocategeometry (geometry)

  deallocateGeometry: deallocate the geometry vectors
- subroutine, public calculatesegmentslopes (geometry, succes, errorMessage)

  calculateSegmentSlopes: calculate the segment slopes
- subroutine, public determinesegmenttypes (geometry)

  determineSegmentTypes: determine the segment types
- subroutine, public copygeometry (geometry, geometryCopy)

  copyGeometry: copy a geometry structure
- subroutine, public isequalgeometry (geometry1, geometry2, succes, errorMessage)

  isEqualGeometry: are two geometries equal
- subroutine, public mergesequentialberms (geometry, geometryMergedBerms, succes, errorMessage)

mergeSequentialBerms: merge sequential berms

- subroutine, public adjustnonhorizontalberms (geometry, geometryFlatBerms, succes, errorMessage)

  adjustNonHorizontalBerms: adjust non-horizontal berms
- subroutine, public removeberms (geometry, geometryNoBerms, succes, errorMessage)

  removeBerms: remove berms
- subroutine, public removedikesegments (geometry, index, geometryAdjusted, succes, errorMessage)

  removeDikeSegments: remove dike segments
- subroutine, public splitcrosssection (geometry, L0, NwideBerms, geometrysectionB, geometrysectionF, succes, errorMessage)

  splitCrossSection: split a cross section
- subroutine, public calculatehorzlengths (geometry, yLower, yUpper, horzLengths, succes, errorMessage)

  calculateHorzLengths: calculate horizontal lengths
- subroutine, public calculatehorzdistance (geometry, yLower, yUpper, dx, succes, errorMessage)

  calculateHorzDistance: calculate horizontal distance
- subroutine, public writecrosssection (geometry, geometryName)

  writeCrossSection: write a cross section

7.14.1 Function/Subroutine Documentation

7.14.1.1 subroutine, public geometrymodulertoovertopping::adjustnonhorizontalberms (type (tpgeometry), intent(in) *geometry,* type (tpgeometry), intent(out) *geometryFlatBerms,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

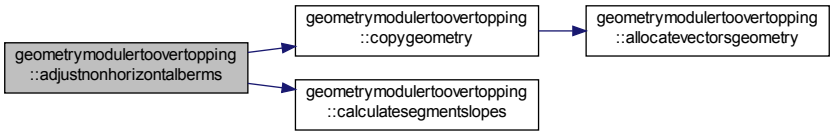
adjustNonHorizontalBerms: adjust non-horizontal berms

Parameters

in	*geometry*	structure with geometry data
out	*geometryflat-berms*	geometry data with horizontal berms
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 578 of file geometryModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.14.1.2 subroutine, public geometrymodulertoovertopping::allocatevectorsgeometry (integer, intent(in) *nCoordinates,* type (tpgeometry), intent(inout) *geometry*)

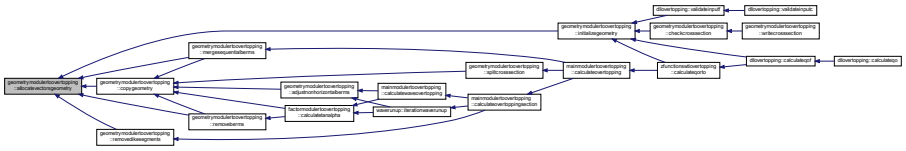
allocateVectorsGeometry: allocate the geometry vectors

Parameters

in	*ncoordinates*	number of coordinates
in,out	*geometry*	structure with geometry data

Definition at line 199 of file geometryModuleRTOovertopping.f90.

Here is the caller graph for this function:



7.14.1.3 subroutine, public geometrymodulertoovertopping::calculatehorzdistance (type (tpgeometry), intent(in) *geometry,* real(wp), intent(in) *yLower,* real(wp), intent(in) *yUpper,* real(wp), intent(out) *dx,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

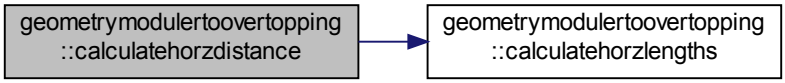
calculateHorzDistance: calculate horizontal distance

Parameters

in	*geometry*	structure with geometry data
in	*ylower*	y-coordinate lower bound (m+NAP)
in	*yupper*	y-coordinate upper bound (m+NAP)
out	*dx*	horizontal distance between bounds (m)
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 1023 of file geometryModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.14.1.4 subroutine, public geometrymodulertoovertopping::calculatehorzlengths (type (tpgeometry), intent(in) *geometry,* real(wp), intent(in) *yLower,* real(wp), intent(in) *yUpper,* real(wp), dimension(geometry%ncoordinates-1), intent(out) *horzLengths,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

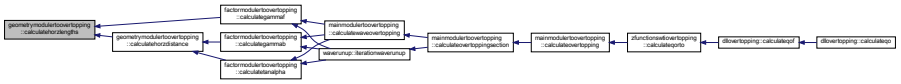
calculateHorzLengths: calculate horizontal lengths

Parameters

in	*geometry*	structure with geometry data
in	*ylower*	y-coord. lower bound (m+NAP)
in	*yupper*	y-coord. upper bound (m+NAP)
out	*horzlengths*	horizontal lengths segments (m)
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 927 of file geometryModuleRTOovertopping.f90.

Here is the caller graph for this function:



7.14.1.5 subroutine, public geometrymodulertoovertopping::calculatesegmentslopes (type (tpgeometry), intent(inout) *geometry,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

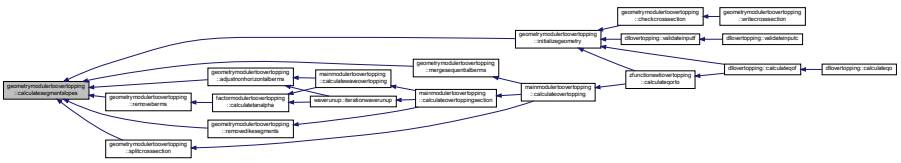
calculateSegmentSlopes: calculate the segment slopes

Parameters

in,out	*geometry*	structure with geometry data
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 248 of file geometryModuleRTOovertopping.f90.

Here is the caller graph for this function:



7.14.1.6 subroutine, public geometrymodulertoovertopping::checkcrosssection (real(wp), intent(in) *psi,* integer, intent(in) *nCoordinates,* real(wp), dimension (ncoordinates), intent(in) *xCoordinates,* real(wp), dimension (ncoordinates), intent(in) *yCoordinates,* real(wp), dimension(ncoordinates-1), intent(in) *roughnessFactors,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

checkCrossSection: check cross section

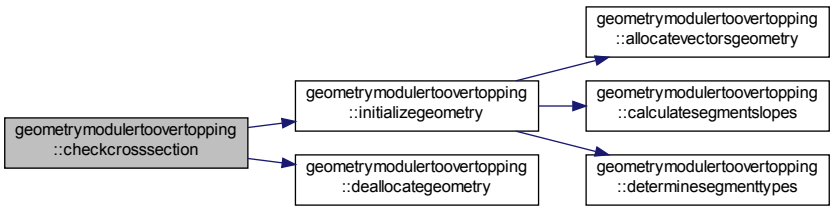
Parameters

in	*psi*	dike normal (degree)
in	*ncoordinates*	number of coordinates
in	*xcoordinates*	x-coordinates (m)

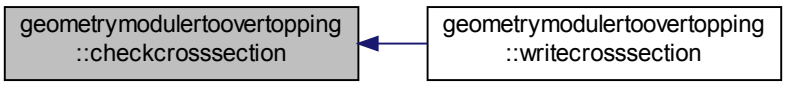
in	*ycoordinates*	y-coordinates (m+NAP)
in	*roughnessfactors*	roughness factors
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 34 of file geometryModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.7 subroutine, public geometrymodulertoovertopping::copygeometry (type (tpgeometry), intent(in) *geometry,* type (tpgeometry), intent(inout) *geometryCopy*)

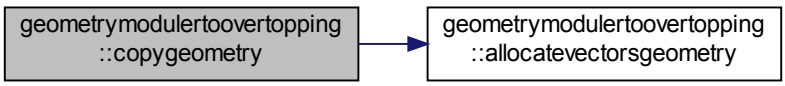
copyGeometry: copy a geometry structure

Parameters

in	*geometry*	structure with geometry data
in,out	*geometrycopy*	structure with geometry data copy

Definition at line 330 of file geometryModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.14.1.8 subroutine, public geometrymodulertoovertopping::deallocategeometry (type (tpgeometry), intent(inout) *geometry*)

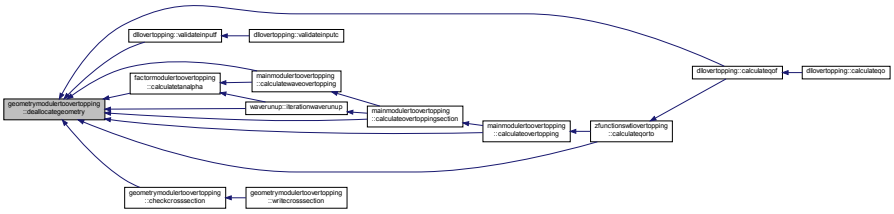
deallocateGeometry: deallocate the geometry vectors

Parameters

in,out	*geometry*	structure with geometry data

Definition at line 225 of file geometryModuleRTOovertopping.f90.

Here is the caller graph for this function:



7.14.1.9 subroutine, public geometrymodulertoovertopping::determinesegmenttypes (type (tpgeometry), intent(inout) *geometry*)

determineSegmentTypes: determine the segment types

Parameters

in,out	*geometry*	structure with geometry data

Definition at line 287 of file geometryModuleRTOovertopping.f90.

Here is the caller graph for this function:

7.14.1.10 subroutine, public geometrymodulertoovertopping::initializegeometry (real(wp), intent(in) *psi,* integer, intent(in) *nCoordinates,* real(wp), dimension (ncoordinates), intent(in) *xCoordinates,* real(wp), dimension (ncoordinates), intent(in) *yCoordinates,* real(wp), dimension(ncoordinates-1), intent(in) *roughnessFactors,* type (tpgeometry), intent(out) *geometry,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

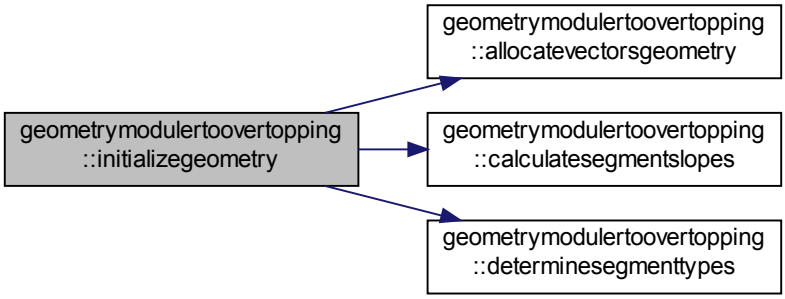
initializeGeometry: initialize the geometry

Parameters

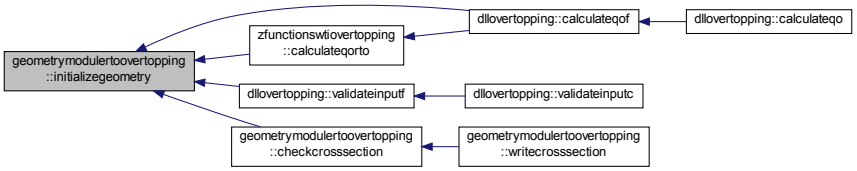
in	*psi*	dike normal (degree)
in	*ncoordinates*	number of coordinates
in	*xcoordinates*	x-coordinates (m)
in	*ycoordinates*	y-coordinates (m+NAP)
in	*roughnessfactors*	roughness factors
out	*geometry*	structure with geometry data
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 142 of file geometryModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.11 subroutine, public geometrymodulertoovertopping::isequalgeometry (type (tpgeometry), intent(in) *geometry1,* type (tpgeometry), intent(in) *geometry2,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

isEqualGeometry: are two geometries equal

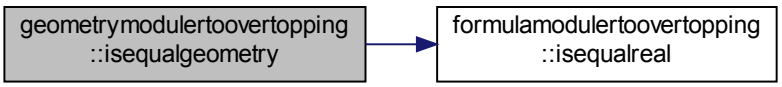
Parameters

in	*geometry1*	structure with geometry data 1
in	*geometry2*	structure with geometry data 2
out	*succes*	flag for succes

out	*errormessage*	error message

Definition at line 377 of file geometryModuleRTOovertopping.f90.

Here is the call graph for this function:



7.14.1.12 subroutine, public geometrymodulertoovertopping::mergesequentialberms (type (tpgeometry), intent(in) *geometry,* type (tpgeometry), intent(inout) *geometryMergedBerms,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

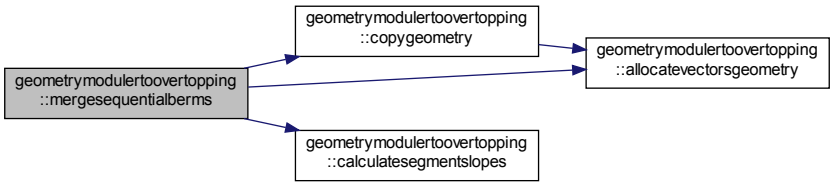
mergeSequentialBerms: merge sequential berms

Parameters

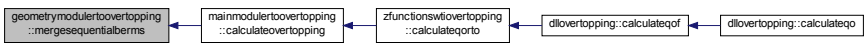
in	*geometry*	structure with geometry data
in,out	*geometrymergedberms*	geometry data with merged sequential berms
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 471 of file geometryModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.13 subroutine, public geometrymodulertoovertopping::removeberms (type (tpgeometry), intent(in) *geometry,* type (tpgeometry), intent(out) *geometryNoBerms,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

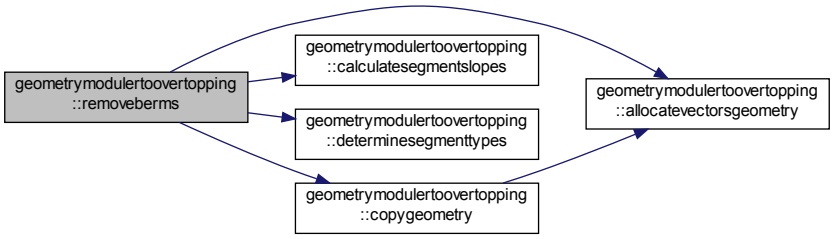
removeBerms: remove berms

Parameters

in	*geometry*	structure with geometry data
out	*geometrytynoberms*	geometry data withouth berms
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 664 of file geometryModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.14.1.14 subroutine, public geometrymodulertoovertopping::removedikesegments (type (tpgeometry), intent(in) *geometry,* integer, intent(in) *index,* type (tpgeometry), intent(out) *geometryAdjusted,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

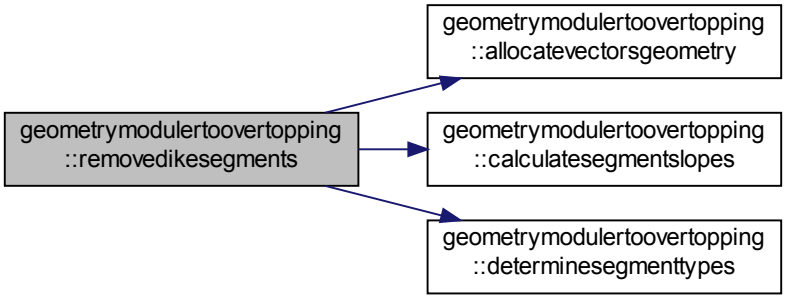
removeDikeSegments: remove dike segments

Parameters

in	*geometry*	structure with geometry data
in	*index*	index starting point new cross section
out	*geometryadjusted*	geometry data with removed dike segments
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 761 of file geometryModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.14.1.15 subroutine, public geometrymodulertoovertopping::splitcrosssection (type (tpgeometry), intent(in) *geometry,* real(wp), intent(in) *L0,* integer, intent(out) *NwideBerms,* type (tpgeometry), intent(out) *geometrysectionB,* type (tpgeometry), intent(out) *geometrysectionF,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

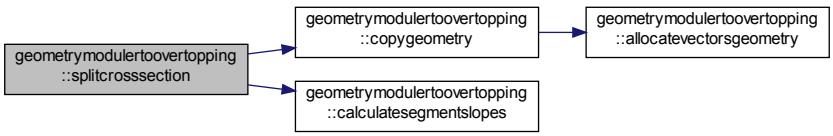
splitCrossSection: split a cross section

Parameters

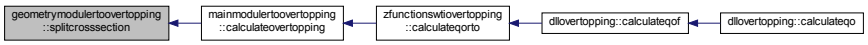
in	*geometry*	structure with geometry data
in	*l0*	wave length (m)
out	*nwideberms*	number of wide berms
out	*geometrysectionb*	geometry data with wide berms to ordinary berms
out	*geometrysectionf*	geometry data with wide berms to foreshores
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 825 of file geometryModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.16 subroutine, public geometrymodulertoovertopping::writecrosssection (type (tpgeometry), intent(in) *geometry,* character(len=∗), intent(in) *geometryName*)

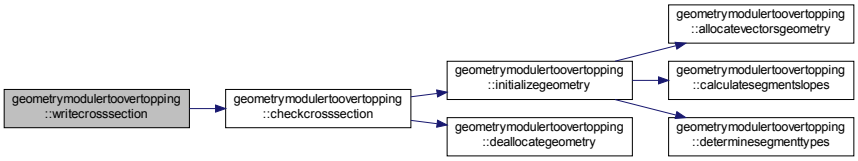
writeCrossSection: write a cross section

Parameters

in	*geometry*	structure with geometry data
in	*geometryname*	description of geometry data

Definition at line 1066 of file geometryModuleRTOovertopping.f90.

Here is the call graph for this function:



7.15 interpolationtriangular Module Reference

Module with function for triangular interpolation.

Functions/Subroutines

- real(kind=wp) function triangularinterpolation (x1, y1, h1, x2, y2, h2, x3, y3, h3, x, y)

  Function for for triangular interpolation Using three result values of three different stations on the Dutch RD grid a plane is spaned. The result values are in general water levels. For a fourth point on the Dutch RD grid the result value is calculated lying on this plane. In general a water level is calculated.
- real(kind=wp) function datainterpolation (var1, var2, weight, iscyclic)

7.15.1 Detailed Description

Module with function for triangular interpolation.

7.15.2 Function/Subroutine Documentation

7.15.2.1 real(kind=wp) function interpolationtriangular::datainterpolation (real(kind=wp), intent(in) *var1,* real(kind=wp), intent(in) *var2,* real(kind=wp), intent(in) *weight,* integer *iscyclic*)

Parameters

in	*var1*	First input argument for in interpolation functions using a scalar weight value
in	*var2*	Second input argument for in interpolation functions using a scalar weight value
in	*weight*	Value for weighing two variables: 'weight' holds for var1 and '(1-weight)' holds for var2
	iscyclic	If the input is of cyclic character, then iscyclic = 1 (for instance: wave direction)

Returns

Result value after linear interpolation between var1 and var2 using weightvalue weight

Definition at line 111 of file interpolationTriangular.f90.

7.15.2.2 real(kind=wp) function interpolationtriangular::triangularinterpolation (real(kind=wp), intent(in) *x1,* real(kind=wp), intent(in) *y1,* real(kind=wp), intent(in) *h1,* real(kind=wp), intent(in) *x2,* real(kind=wp), intent(in) *y2,* real(kind=wp), intent(in) *h2,* real(kind=wp), intent(in) *x3,* real(kind=wp), intent(in) *y3,* real(kind=wp), intent(in) *h3,* real(kind=wp), intent(in) *x,* real(kind=wp), intent(in) *y*)

Function for for triangular interpolation Using three result values of three different stations on the Dutch RD grid a plane is spaned. The result values are in general water levels. For a fourth point on the Dutch RD grid the result value is calculated lying on this plane. In general a water level is calculated.

Parameters

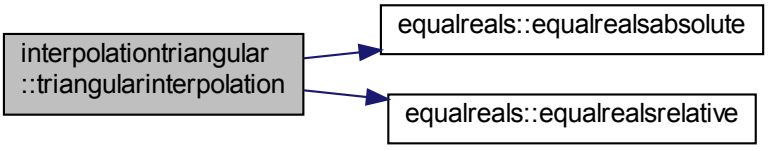
in	*x1*	x-coordinate station 1
in	*y1*	y-coordinate station 1
in	*h1*	Result value station 1 (water level)
in	*x2*	x-coordinate station 2
in	*y2*	y-coordinate station 2
in	*h2*	Result value station 2 (water level)
in	*x3*	x-coordinate station 3
in	*y3*	y-coordinate station 3
in	*h3*	Result value station 3 (water level)
in	*x*	x-coordinate point of examination
in	*y*	y-coordinate point of examination

Returns

Result value (water level) on point of examination after triangular interpolation

Definition at line 26 of file interpolationTriangular.f90.

Here is the call graph for this function:



7.16 mainmodulertoovertopping Module Reference

Functions/Subroutines

- subroutine, public calculateovertopping (geometry, load, modelFactors, overtopping, succes, errorMessage)

  calculateOvertopping: calculate the overtopping

- subroutine, public calculateovertoppingsection (geometry, h, Hm0, Tm_10, L0, gammaBeta_z, gammaBeta_o, modelFactors, overtopping, succes, errorMessage)

  calculateOvertoppingSection: calculate the overtopping for a section

- subroutine, public calculatewaveovertopping (geometry, h, Hm0, Tm_10, z2, gammaBeta_o, modelFactors, Qo, succes, errorMessage)

  calculateWaveOvertopping: calculate wave overtopping

- subroutine calculateovertoppingnegativefreeboard (load, geometry, overtopping, succes, errorMessage)

  calculateOvertoppingNegativeFreeboard: calculate overtopping in case of negative freeboard

- subroutine, public interpolateresultssections (geometry, L0, NwideBerms, overtoppingB, overtoppingF, overtopping, succes, errorMessage)

  interpolateResultsSections: interpolate results for split cross sections

- subroutine, public checkinputdata (geometry, load, modelFactors, succes, errorMessage)

  checkInputdata: check the input data

- subroutine, public checkmodelfactors (modelFactors, succes, errorMessage)

  checkModelFactors: check the input data

- subroutine, public convertovertoppinginput (modelFactors, success, errorMessage)

  convertOvertoppingInput: convert the model factors from C-like to Fortran

7.16.1 Function/Subroutine Documentation

7.16.1.1 subroutine, public mainmodulertoovertopping::calculateovertopping (type (tpgeometry), intent(in) *geometry,* type (tpload), intent(in) *load,* type (tpovertoppinginput), intent(in) *modelFactors,* type (tpovertopping), intent(out) *overtopping,* logical, intent(out) *succes,* character(len=*), intent(out) *errorMessage*)

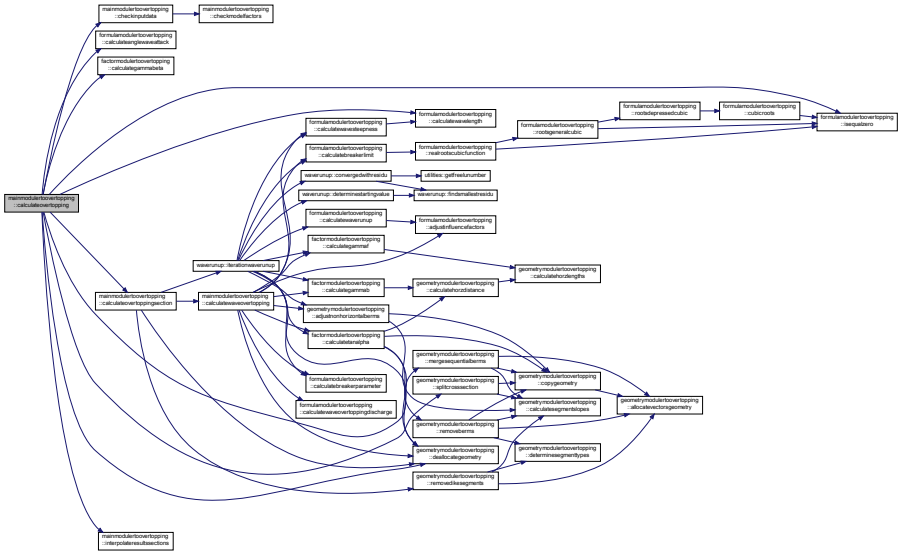
calculateOvertopping: calculate the overtopping

Parameters

in	*geometry*	structure with geometry data
in	*load*	structure with load parameters
in	*modelfactors*	structure with model factors
out	*overtopping*	structure with overtopping results
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 36 of file mainModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.16.1.2 subroutine mainmodulertoovertopping::calculateovertoppingnegativefreeboard (type (tpload), intent(in) *load,* type (tpgeometry), intent(in) *geometry,* type (tpovertopping), intent(inout) *overtopping,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*) [private]

calculateOvertoppingNegativeFreeboard: calculate overtopping in case of negative freeboard

Parameters

in	*geometry*	structure with geometry data
in	*load*	structure with load parameters
in,out	*overtopping*	structure with overtopping results
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 478 of file mainModuleRTOovertopping.f90.

7.16.1.3 subroutine, public mainmodulertoovertopping::calculateovertoppingsection (type (tpgeometry), intent(in) *geometry,* real(wp), intent(in) *h,* real(wp), intent(in) *Hm0,* real(wp), intent(in) *Tm_10,* real(wp), intent(in) *L0,* real(wp), intent(inout) *gammaBeta_z,* real(wp), intent(inout) *gammaBeta_o,* type (tpovertoppinginput), intent(in) *modelFactors,* type (tpovertopping), intent(out) *overtopping,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

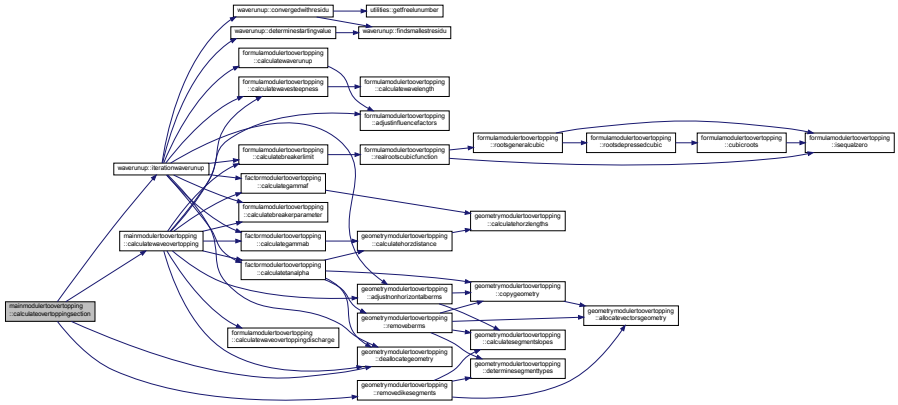
calculateOvertoppingSection: calculate the overtopping for a section

Parameters

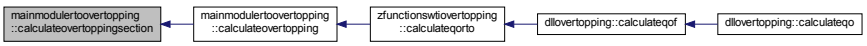
in	*geometry*	structure with geometry data
in	*h*	local water level (m+NAP)
in	*hm0*	significant wave height (m)
in	*tm_10*	spectral wave period (s)
in	*l0*	wave length (m)
in,out	*gammabeta_z*	influence angle wave attack wave run-up
in,out	*gammabeta_o*	influence angle wave attack overtopping
in	*modelfactors*	structure with model factors
out	*overtopping*	structure with overtopping results
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 164 of file mainModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.16.1.4 subroutine, public mainmodulertoovertopping::calculatewaveovertopping (type (tpgeometry), intent(in) *geometry,* real(wp), intent(in) *h,* real(wp), intent(in) *Hm0,* real(wp), intent(in) *Tm_10,* real(wp), intent(in) *z2,* real(wp), intent(inout) *gammaBeta_o,* type (tpovertoppinginput), intent(in) *modelFactors,* real(wp), intent(out) *Qo,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

calculateWaveOvertopping: calculate wave overtopping

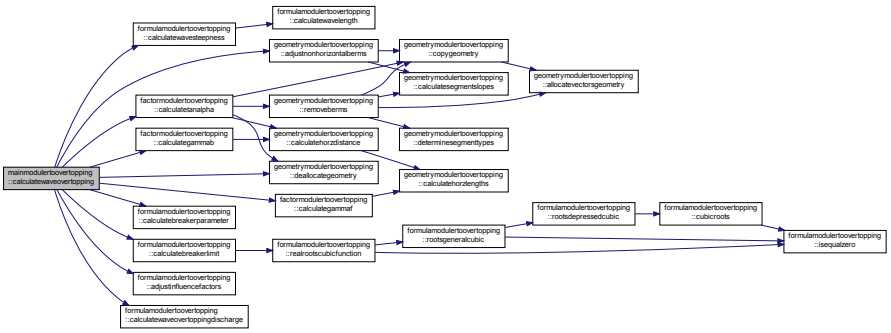
Parameters

in	*geometry*	structure with geometry data
in	*h*	local water level (m+NAP)
in	*hm0*	significant wave height (m)

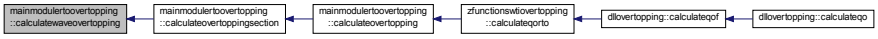
in	*tm_10*	spectral wave period (s)
in	*z2*	2% wave run-up (m)
in,out	*gammabeta_o*	influence angle wave attack overtopping
in	*modelfactors*	structure with model factors
out	*qo*	wave overtopping discharge (m3/m per s)
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 390 of file mainModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.16.1.5 subroutine, public mainmodulertoovertopping::checkinputdata (type (tpgeometry), intent(in) *geometry,* type (tpload), intent(in) *load,* type (tpovertoppinginput), intent(in) *modelFactors,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

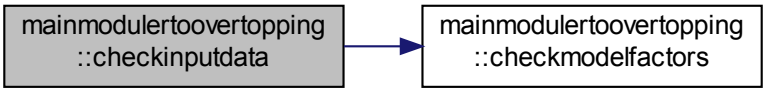
checkInputdata: check the input data

Parameters

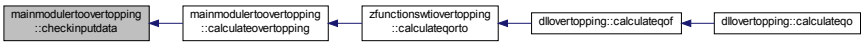
in	*geometry*	structure with geometry data
in	*load*	structure with load parameters
in	*modelfactors*	structure with model factors
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 595 of file mainModuleRTOovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.16.1.6 subroutine, public mainmodulertoovertopping::checkmodelfactors (type (tpovertoppinginput), intent(in) *modelFactors,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

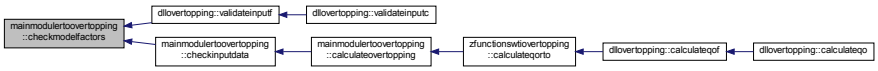
checkModelFactors: check the input data

Parameters

in	*modelfactors*	structure with model factors
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 649 of file mainModuleRTOovertopping.f90.

Here is the caller graph for this function:



7.16.1.7 subroutine, public mainmodulertoovertopping::convertovertoppinginput (type (tpovertoppinginput), intent(inout) *modelFactors,* logical, intent(out) *success,* character(len=∗), intent(inout) *errorMessage*)

convertOvertoppingInput: convert the model factors from C-like to Fortran

Parameters

in,out	*modelfactors*	model factors and other input for overtopping
out	*success*	flag for success
in,out	*errormessage*	error message; only set when not successful

Definition at line 744 of file mainModuleRTOovertopping.f90.

Here is the caller graph for this function:

7.16.1.8 subroutine, public mainmodulertoovertopping::interpolateresultssections (type (tpgeometry), intent(in) *geometry,* real(wp), intent(in) *L0,* integer, intent(in) *NwideBerms,* type (tpovertopping), intent(in) *overtoppingB,* type (tpovertopping), intent(in) *overtoppingF,* type (tpovertopping), intent(out) *overtopping,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

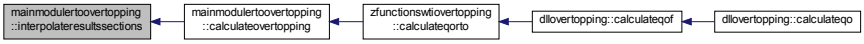
interpolateResultsSections: interpolate results for split cross sections

Parameters

in	*geometry*	structure with geometry data
in	*l0*	wave length (m)
in	*nwideberms*	number of wide berms
in	*overtoppingb*	structure with overtopping results ordinary berms
in	*overtoppingf*	structure with overtopping results foreshores
out	*overtopping*	structure with combined overtopping results
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 514 of file mainModuleRTOovertopping.f90.

Here is the caller graph for this function:



7.17 modulelogging Module Reference

Data Types

- type tlogging

  TLogging: structure for steering the logging.

Variables

- integer, parameter maxfilenamelength = 256

  maximum length of filename
- type(tlogging) currentlogging

  copy of argument logging

7.17.1 Variable Documentation

7.17.1.1 type(tlogging) modulelogging::currentlogging

copy of argument logging

Definition at line 21 of file ModuleLogging.f90.

7.17.1.2 integer, parameter modulelogging::maxfilenamelength = 256

maximum length of filename

Definition at line 13 of file ModuleLogging.f90.

7.18 overtoppinginterface Module Reference

Data Types

- type overtoppinggeometrytype
- type overtoppinggeometrytypef
- type tpprofilecoordinate

Variables

- integer, parameter, public varmodelfactorcriticalovertopping = 8
  Model factor critical overtopping.

7.18.1 Variable Documentation

7.18.1.1 integer, parameter, public overtoppinginterface::varmodelfactorcriticalovertopping = 8

Model factor critical overtopping.

Definition at line 17 of file overtoppingInterface.f90.

7.19 physics Module Reference

7.20 physics_breakwater Module Reference

Module with routines for breakwater.

Functions/Subroutines

- subroutine, public calculatebreakwater (breakwaterType, heightBreakwater, waterLevel, Hs)
  Function to transform the wave height for the influence of an occupant breakwater.

7.20.1 Detailed Description

Module with routines for breakwater.

7.20.2 Function/Subroutine Documentation

7.20.2.1 subroutine, public physics_breakwater::calculatebreakwater (integer, intent(in) *breakwaterType,* real (kind=wp), intent(in) *heightBreakwater,* real (kind=wp), intent(in) *waterLevel,* real (kind=wp), intent(inout) *Hs*)

Function to transform the wave height for the influence of an occupant breakwater.

This subroutine performs a transformation on the wave height, due to breakwater construction which breaks waves. The breakwater formulation is based on the formula of Goda (1969) and Seelig (1979). Reference is made to the report "Rekenregels voor waterbouwkundig ontwerpen" of RWS Directie Sluizen en Stuwen, Utrecht, 1990.

Determine the shape factors (alpha and beta) depending on the breakwater type (caisson, vertical wall, rubble mound) Compute first the transmittance coefficient and afterwards the reduced wave height.

Parameters

in	*breakwatertype*	breakwater type
in	*heightbreakwater*	height breakwater [m + NAP]
in	*waterlevel*	water level [m + NAP]
in,out	*hs*	wave height [m]

Definition at line 35 of file breakwater.f90.

7.21 physics_breakwatertypesenumerations Module Reference

Enumeration for breakwater types WHEN CHANGING THESE VALUES. ALSO CHANGE THEM IN HydraRing↩ Enumerations.CS.

Variables

- integer, parameter breakwaternotpresent = 0

  no breakwater
- integer, parameter breakwatercaisson = 1

  breakwater type is caisson
- integer, parameter breakwaterverticalwall = 2

  breakwater type is vertical wall
- integer, parameter breakwaterrubblemound = 3

  breakwater type is rubble mound 1:1.5

7.21.1 Detailed Description

Enumeration for breakwater types WHEN CHANGING THESE VALUES. ALSO CHANGE THEM IN HydraRing↩ Enumerations.CS.

7.21.2 Variable Documentation

7.21.2.1 integer, parameter physics_breakwatertypesenumerations::breakwatercaisson = 1

breakwater type is caisson

Definition at line 20 of file breakwaterEnumerations.f90.

7.21.2.2 integer, parameter physics_breakwatertypesenumerations::breakwaternotpresent = 0

no breakwater

Definition at line 17 of file breakwaterEnumerations.f90.

7.21.2.3 integer, parameter physics_breakwatertypesenumerations::breakwaterrubblemound = 3

breakwater type is rubble mound 1:1.5

Definition at line 26 of file breakwaterEnumerations.f90.

7.21.2.4 integer, parameter physics_breakwatertypesenumerations::breakwaterverticalwall = 2

breakwater type is vertical wall

Definition at line 23 of file breakwaterEnumerations.f90.

7.22 physics_bretschneider Module Reference

Module with routines for bretschneider.

Data Types

- type tpbretschneider

Functions/Subroutines

- subroutine, public calculatewavebretschneider (bretschneider, Hs, Ts)

  Function to compute wave parameters according to Bretschneider formula: wave height (Hs) wave period (Ts)

7.22.1 Detailed Description

Module with routines for bretschneider.

7.22.2 Function/Subroutine Documentation

7.22.2.1 subroutine, public physics_bretschneider::calculatewavebretschneider (type(tpbretschneider) *bretschneider,* real(kind = wp), intent(out) *Hs,* real(kind = wp), intent(out) *Ts*)

Function to compute wave parameters according to Bretschneider formula: wave height (Hs) wave period (Ts)

Parameters

	bretschneider	Bretschneider structure with necessary variables stored
out	*hs*	Significant wave height
out	*ts*	Significant wave period

Definition at line 37 of file bretschneider.f90.

7.23 physics_foreland Module Reference

wrapper for foreland dll !!

Functions/Subroutines

- subroutine, public physics_calculateforeland (retval, hDamType, hDamHeight, hAlpha, hFc, hInvalid, hDim↩ Hm0, hHm0, hTp, hWlev, hIncomingWaveAngle, hDikeNormal, hDimX, hX, hMinStepSize, hBottomLevel, hRatioDepth, hLogging, hLoggingFileName, hHm0Dike, hTpDike, hRefractedWaveAngleDike, hMessage)

  wrapper for foreland calculation in DynamicLib-DaF.dll

Variables

- integer, parameter message_length =1000

7.23.1 Detailed Description

wrapper for foreland dll !!

7.23.2 Function/Subroutine Documentation

7.23.2.1 subroutine, public physics_foreland::physics_calculateforeland (integer, intent(out) *retval,* integer, intent(in) *hDamType,* real(kind=wp), intent(in) *hDamHeight,* real(kind=wp), intent(in) *hAlpha,* real(kind=wp), intent(in) *hFc,* real(kind=wp), intent(in) *hInvalid,* integer, intent(in) *hDimHm0,* real(kind=wp), dimension(1:hdimhm0), intent(in) *hHm0,* real(kind=wp), dimension(1:hdimhm0), intent(in) *hTp,* real(kind=wp), dimension(1:hdimhm0), intent(in) *hWlev,* real(kind=wp), dimension(1:hdimhm0), intent(in) *hIncomingWaveAngle,* real(kind=wp), intent(in) *hDikeNormal,* integer, intent(in) *hDimX,* real(kind=wp), dimension(1:hdimx), intent(in) *hX,* real(kind=wp), intent(in) *hMinStepSize,* real(kind=wp), dimension(1:hdimx), intent(in) *hBottomLevel,* real(kind=wp), intent(in) *hRatioDepth,* integer, intent(in) *hLogging,* character(len=∗), intent(in) *hLoggingFileName,* real(kind=wp), dimension(1:hdimhm0), intent(out) *hHm0Dike,* real(kind=wp), dimension(1:hdimhm0), intent(out) *hTpDike,* real(kind=wp), dimension(1:hdimhm0), intent(out) *hRefractedWaveAngleDike,* character(len=message_length) *hMessage*)

wrapper for foreland calculation in DynamicLib-DaF.dll

Parameters

in	*hdamtype*	Dam types 1,2,3 [-]
in	*hdamheight*	Height of the dam [m]
in	*hdimhm0*	Total number of hydrodynamic parameters [-]
in	*hdimx*	Total number of locations [-]
in	*halpha*	Baldock coefficient [-]
in	*hfc*	Collins coefficient [-]
in	*hinvalid*	Value to be used for marking output as invalid [-]
in	*hhm0*	Array(1D) containing Hm0 values at different time steps [m]
in	*htp*	Array(1D) containing Tp values at different time steps [s]
in	*hwlev*	Array(1D) containing Water level values at different time steps [m]
in	*hincoming-waveangle*	Array(1D) containing incoming wave angles values at different time steps [degrees N]
in	*hdikenormal*	DikeNormal [degrees N]
in	*hx*	Array(1D) containing the x values of the profile (x(1) equals the location of the swan output point) [m]
in	*hminstepsize*	Minimum grid step size [m]
in	*hbottomlevel*	Array(1D) containing the depth values at the prescribed profile coordinates x(i) [-]

in	*hratiodepth*	Ratio between the wave height Hm0 (check with Alfons) and waterdepth above which breaking occurs.
in	*hlogging*	Switch which activates logging mechanism [-]
in	*hloggingfilename*	Name of the log file.
out	*hhm0dike*	Array(1D) containing the calculate Hm0 values at the toe of the dike for the provided time steps [m]
out	*htpdike*	Array(1D) containing the Tp values at the toe of the dike for the provided time steps [s]
out	*hrefracted-waveangledike*	Array(1D) containing the refracted angles [degrees N]
out	*retval*	Return 0 or -1. The first if no errors occur the later otherwise.
	hmessage	String containing an error message in case een error occurs [-]

Definition at line 27 of file foreland.f90.

7.23.3 Variable Documentation

7.23.3.1 integer, parameter physics_foreland::message_length =1000

Definition at line 19 of file foreland.f90.

7.24 physics_wavereduction Module Reference

Module with routines for wave reduction.

Data Types

- type tpwavereduction

Functions/Subroutines

- subroutine, public calculatewavereduction (wavereduction, errorCode, Hs)

  Computation of wave reduction due to shallow foreland (simple model) Wave reduction function to compute the wave reduction (simple model)

7.24.1 Detailed Description

Module with routines for wave reduction.

7.24.2 Function/Subroutine Documentation

7.24.2.1 subroutine, public physics_wavereduction::calculatewavereduction (type(tpwavereduction) *wavereduction,* integer, intent(out) *errorCode,* real(kind = wp), intent(out) *Hs*)

Computation of wave reduction due to shallow foreland (simple model) Wave reduction function to compute the wave reduction (simple model)

Parameters

	wavereduction	Definition of wavereduction input parameters
out	*errorcode*	errorCode; 0 = succes; -1 = failure
out	*hs*	wave height

Definition at line 37 of file waveReduction.f90.

7.25 precision Module Reference

Module with parameters for KINDs and mathematical constants.

Variables

- integer, parameter sp = kind(1.0)

  Single precision.
- integer, parameter dp = kind(1.0d0)

  Double precision.
- integer, parameter wp = dp

  Working precision.
- real(wp), parameter almostzero = 1.0d-30

  Definition of zero for the machine.
- real(wp), parameter almostinf = huge(1.0d0)

  Very large value as limit.
- real(wp), parameter rholimit = 0.99999d0

  Limit value for the correlation coefficient DO NOT CHANGE THIS VALUE WITHOUT SUFFICIENT RESEARCH.
- real(wp), parameter betalimit = 0.99999999d0

  Limit value for the beta ratio in combining DO NOT CHANGE ONLY USED FOR EXTRA WIND STOCHAST.
- real(wp), parameter pi = 3.14159265358979323846264338327950288419 7_wp

  Circle constant.
- real(wp), parameter emconstant = 0.57721566490153286060651209008240243104215933593992_wp

  Euler Mascheroni constant.
- real(wp), parameter gravityconstant = 9.80665_wp

  Gravity constant for the Netherlands.
- integer, parameter shortmax = 32
- integer, parameter longmax = 255

7.25.1 Detailed Description

Module with parameters for KINDs and mathematical constants.

7.25.2 Variable Documentation

7.25.2.1 real(wp), parameter precision::almostinf = huge(1.0d0)

Very large value as limit.

Definition at line 30 of file precision.f90.

7.25.2.2 real(wp), parameter precision::almostzero = 1.0d-30

Definition of zero for the machine.

Definition at line 29 of file precision.f90.

7.25.2.3 real(wp), parameter precision::betalimit = 0.99999999d0

Limit value for the beta ratio in combining DO NOT CHANGE ONLY USED FOR EXTRA WIND STOCHAST.

Definition at line 32 of file precision.f90.

7.25.2.4 integer, parameter precision::dp = kind(1.0d0)

Double precision.

Definition at line 20 of file precision.f90.

7.25.2.5 real(wp), parameter precision::emconstant = 0.577215664901532860606512090082402431042159335939 92_wp

Euler Mascheroni constant.

Definition at line 38 of file precision.f90.

7.25.2.6 real(wp), parameter precision::gravityconstant = 9.80665_wp

Gravity constant for the Netherlands.

Definition at line 39 of file precision.f90.

7.25.2.7 integer, parameter precision::longmax = 255

Definition at line 45 of file precision.f90.

7.25.2.8 real(wp), parameter precision::pi = 3.14159265358979323846264338327950288419 7_wp

Circle constant.

Definition at line 37 of file precision.f90.

7.25.2.9 real(wp), parameter precision::rholimit = 0.99999d0

Limit value for the correlation coefficient DO NOT CHANGE THIS VALUE WITHOUT SUFFICIENT RESEARCH.

Definition at line 31 of file precision.f90.

7.25.2.10 integer, parameter precision::shortmax = 32

Definition at line 44 of file precision.f90.

7.25.2.11 integer, parameter precision::sp = kind(1.0)

Single precision.

Definition at line 19 of file precision.f90.

7.25.2.12 integer, parameter precision::wp = dp

Working precision.

Definition at line 25 of file precision.f90.

7.26 typedefinitionsrtoovertopping Module Reference

Data Types

- type tpgeometry

  tpGeometry: structure with geometry data
- type tpload

  tpLoad: structure with load parameters
- type tpovertopping

  tpOvertopping: structure with overtopping results
- type tpovertoppinginput

  OvertoppingModelFactors: C-structure with model factors.

Variables

- real(wp), parameter xdiff_min = 2.0d-2

  minimal value distance between x-coordinates (m)
- real(wp), parameter margindiff = 1.0d-14

  margin for minimal distance (m)
- real(wp), parameter berm_min = 0.0d0

  minimal value gradient berm segment
- real(wp), parameter berm_max = 1.0d0/15

  maximal value gradient berm segment
- real(wp), parameter slope_min = 1.0d0/8

  minimal value gradient slope segment
- real(wp), parameter slope_max = 1.0d0

  maximal value gradient slope segment
- real(wp), parameter margingrad = 0.0025d0

  margin for minimal and maximal gradients
- real(wp), parameter rfactor_min = 0.5d0

  minimal value roughness factor dike segments
- real(wp), parameter rfactor_max = 1.0d0

  maximal value roughness factor dike segments
- real(wp), parameter mz2_min = 0.0d0

  minimal value model factor of 2% runup height
- real(wp), parameter mz2_max = huge(mz2_max)

  maximal value model factor of 2% runup height
- real(wp), parameter frunup1_min = 0.0d0

  minimal value model factor 1 for wave run-up
- real(wp), parameter frunup1_max = huge(fRunup1_max)

  maximal value model factor 1 for wave run-up
- real(wp), parameter frunup2_min = 0.0d0

  minimal value model factor 2 for wave run-up
- real(wp), parameter frunup2_max = huge(fRunup2_max)

  maximal value model factor 2 for wave run-up
- real(wp), parameter frunup3_min = 0.0d0

  minimal value model factor 3 for wave run-up
- real(wp), parameter frunup3_max = huge(fRunup3_max)

  maximal value model factor 3 for wave run-up
- real(wp), parameter fb_min = 0.0d0

minimal value model factor for breaking waves

- real(wp), parameter fb_max = huge(fB_max)

  maximal value model factor for breaking waves
- real(wp), parameter fn_min = 0.0d0

  minimal value model factor for non-breaking waves
- real(wp), parameter fn_max = huge(fN_max)

  maximal value model factor for non-breaking waves
- real(wp), parameter fs_min = 0.0d0

  minimal value model factor for shallow waves
- real(wp), parameter fs_max = huge(fS_max)

  maximal value model factor for shallow waves
- real(wp), parameter foreshore_min = 0.3d0

  minimal value reduction factor foreshore
- real(wp), parameter foreshore_max = 1.0d0

  maximal value reduction factor foreshore
- integer, parameter z2_iter_max = 100

  maximal number of iterations for calculation z2
- real(wp), parameter z2_margin = 0.001d0

  margin for convergence criterium calculation z2

7.26.1 Variable Documentation

7.26.1.1 real(wp), parameter typedefinitionsrtoovertopping::berm_max = 1.0d0/15

maximal value gradient berm segment

Definition at line 67 of file typeDefinitionsRTOovertopping.f90.

7.26.1.2 real(wp), parameter typedefinitionsrtoovertopping::berm_min = 0.0d0

minimal value gradient berm segment

Definition at line 66 of file typeDefinitionsRTOovertopping.f90.

7.26.1.3 real(wp), parameter typedefinitionsrtoovertopping::fb_max = huge(fB_max)

maximal value model factor for breaking waves

Definition at line 82 of file typeDefinitionsRTOovertopping.f90.

7.26.1.4 real(wp), parameter typedefinitionsrtoovertopping::fb_min = 0.0d0

minimal value model factor for breaking waves

Definition at line 81 of file typeDefinitionsRTOovertopping.f90.

7.26.1.5 real(wp), parameter typedefinitionsrtoovertopping::fn_max = huge(fN_max)

maximal value model factor for non-breaking waves

Definition at line 84 of file typeDefinitionsRTOovertopping.f90.

7.26.1.6 real(wp), parameter typedefinitionsrtoovertopping::fn_min = 0.0d0

minimal value model factor for non-breaking waves

Definition at line 83 of file typeDefinitionsRTOovertopping.f90.

7.26.1.7 real(wp), parameter typedefinitionsrtoovertopping::foreshore_max = 1.0d0

maximal value reduction factor foreshore

Definition at line 88 of file typeDefinitionsRTOovertopping.f90.

7.26.1.8 real(wp), parameter typedefinitionsrtoovertopping::foreshore_min = 0.3d0

minimal value reduction factor foreshore

Definition at line 87 of file typeDefinitionsRTOovertopping.f90.

7.26.1.9 real(wp), parameter typedefinitionsrtoovertopping::frunup1_max = huge(fRunup1_max)

maximal value model factor 1 for wave run-up

Definition at line 76 of file typeDefinitionsRTOovertopping.f90.

7.26.1.10 real(wp), parameter typedefinitionsrtoovertopping::frunup1_min = 0.0d0

minimal value model factor 1 for wave run-up

Definition at line 75 of file typeDefinitionsRTOovertopping.f90.

7.26.1.11 real(wp), parameter typedefinitionsrtoovertopping::frunup2_max = huge(fRunup2_max)

maximal value model factor 2 for wave run-up

Definition at line 78 of file typeDefinitionsRTOovertopping.f90.

7.26.1.12 real(wp), parameter typedefinitionsrtoovertopping::frunup2_min = 0.0d0

minimal value model factor 2 for wave run-up

Definition at line 77 of file typeDefinitionsRTOovertopping.f90.

7.26.1.13 real(wp), parameter typedefinitionsrtoovertopping::frunup3_max = huge(fRunup3_max)

maximal value model factor 3 for wave run-up

Definition at line 80 of file typeDefinitionsRTOovertopping.f90.

7.26.1.14 real(wp), parameter typedefinitionsrtoovertopping::frunup3_min = 0.0d0

minimal value model factor 3 for wave run-up

Definition at line 79 of file typeDefinitionsRTOovertopping.f90.

7.26.1.15 real(wp), parameter typedefinitionsrtoovertopping::fs_max = huge(fS_max)

maximal value model factor for shallow waves

Definition at line 86 of file typeDefinitionsRTOovertopping.f90.

7.26.1.16 real(wp), parameter typedefinitionsrtoovertopping::fs_min = 0.0d0

minimal value model factor for shallow waves

Definition at line 85 of file typeDefinitionsRTOovertopping.f90.

7.26.1.17 real(wp), parameter typedefinitionsrtoovertopping::margindiff = 1.0d-14

margin for minimal distance (m)

Definition at line 65 of file typeDefinitionsRTOovertopping.f90.

7.26.1.18 real(wp), parameter typedefinitionsrtoovertopping::margingrad = 0.0025d0

margin for minimal and maximal gradients

Definition at line 70 of file typeDefinitionsRTOovertopping.f90.

7.26.1.19 real(wp), parameter typedefinitionsrtoovertopping::mz2_max = huge(mz2_max)

maximal value model factor of 2% runup height

Definition at line 74 of file typeDefinitionsRTOovertopping.f90.

7.26.1.20 real(wp), parameter typedefinitionsrtoovertopping::mz2_min = 0.0d0

minimal value model factor of 2% runup height

Definition at line 73 of file typeDefinitionsRTOovertopping.f90.

7.26.1.21 real(wp), parameter typedefinitionsrtoovertopping::rfactor_max = 1.0d0

maximal value roughness factor dike segments

Definition at line 72 of file typeDefinitionsRTOovertopping.f90.

7.26.1.22 real(wp), parameter typedefinitionsrtoovertopping::rfactor_min = 0.5d0

minimal value roughness factor dike segments

Definition at line 71 of file typeDefinitionsRTOovertopping.f90.

7.26.1.23 real(wp), parameter typedefinitionsrtoovertopping::slope_max = 1.0d0

maximal value gradient slope segment

Definition at line 69 of file typeDefinitionsRTOovertopping.f90.

7.26.1.24 real(wp), parameter typedefinitionsrtoovertopping::slope_min = 1.0d0/8

minimal value gradient slope segment

Definition at line 68 of file typeDefinitionsRTOovertopping.f90.

7.26.1.25 real(wp), parameter typedefinitionsrtoovertopping::xdiff_min = 2.0d-2

minimal value distance between x-coordinates (m)

Definition at line 64 of file typeDefinitionsRTOovertopping.f90.

7.26.1.26 integer, parameter typedefinitionsrtoovertopping::z2_iter_max = 100

maximal number of iterations for calculation z2

Definition at line 89 of file typeDefinitionsRTOovertopping.f90.

7.26.1.27 real(wp), parameter typedefinitionsrtoovertopping::z2_margin = 0.001d0

margin for convergence criterium calculation z2

Definition at line 90 of file typeDefinitionsRTOovertopping.f90.

7.27 utilities Module Reference

Module with general utility routines.

Functions/Subroutines

- subroutine getfreelunumber (lun)

  getFreeLuNumber Routine to find a free LU-number
- pure character(len(str)) function to_lower (str)

  to_lower Changes a string to lower case
- pure character(len(str)) function to_upper (str)

  to_upper Changes a string to upper case

Variables

- character(26), parameter, private cap = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
- character(26), parameter, private low = 'abcdefghijklmnopqrstuvwxyz'

7.27.1 Detailed Description

Module with general utility routines.

7.27.2 Function/Subroutine Documentation

7.27.2.1 subroutine utilities::getfreelunumber (integer, intent(out) *lun*)

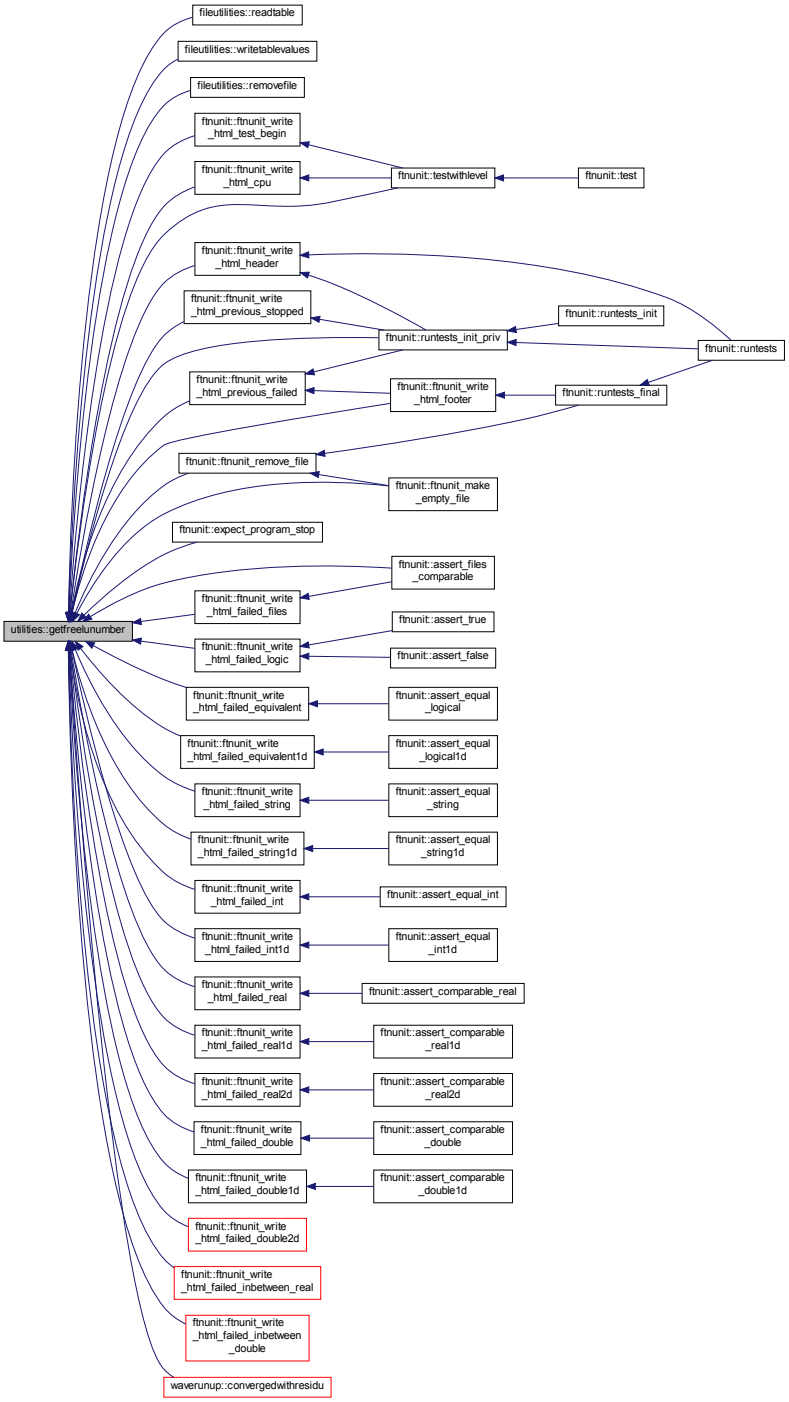
getFreeLuNumber Routine to find a free LU-number

Parameters

out	*lun*	LU-number to use

Definition at line 28 of file utilities.f90.

Here is the caller graph for this function:



7.27.2.2 pure character(len(str)) function utilities::to_lower (character(∗), intent(in) *str*)

to_lower Changes a string to lower case

Definition at line 58 of file utilities.f90.

7.27.2.3 pure character(len(str)) function utilities::to_upper (character(∗), intent(in) *str*)

to_upper Changes a string to upper case

Definition at line 78 of file utilities.f90.

7.27.3 Variable Documentation

7.27.3.1 character(26), parameter, private utilities::cap = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

Definition at line 15 of file utilities.f90.

7.27.3.2 character(26), parameter, private utilities::low = 'abcdefghijklmnopqrstuvwxyz'

Definition at line 16 of file utilities.f90.

7.28 vectorutilities Module Reference

Module with vector utility functions.

Functions/Subroutines

- real(kind=wp) function, dimension(size(x)) normalize (x)

  Function for normalizing a vector If the vector is the null vector, a unit vector with uniform components is returned instead.
- real(kind=wp) function interpolateline (x1, x2, f1, f2, x)

  Function to interpolate and extrapolate linear on a line.
- real(kind=wp) function linearinterpolate (X, Y, xx)

  Function for linearly interpolating and extrapolating.
- real(kind=wp) function loglinearinterpolate (X, Y, xx)

  Function for log-linearly interpolating and extrapolating.
- real(kind=wp) function linearinterpolatecyclic (X, Y, xx)

  Function for linearly interpolating cyclic data.
- real(kind=wp) function loglinearinterpolatecyclic (X, Y, xx)

  Function for log-linearly interpolating and extrapolating cyclic data.
- real(kind=wp) function linearinterpolateangles (X, Y, xx)

  Function for linearly interpolating and extrapolating angles.
- subroutine sortandlocate (X, IDXsorted, xx, locID, locIDm1, Cyclic)

  Subroutine to bracket a given value xx within two elements of an array after sort in descending order.
- real(kind=wp) function stepsize (i, x)

  Function for the Trapezium rule.
- real(kind=wp) function trapeziumrule (x, f)

  Function for the Trapezium rule.
- real(kind=wp) function intgratewithexponentcomponent (x, f)

  Function for the Trapezium rule.

- subroutine ssort (x, iy, n, kflag)

  Subroutine to sort a vector in either ascending or descending order.

7.28.1 Detailed Description

Module with vector utility functions.

7.28.2 Function/Subroutine Documentation

7.28.2.1 real (kind=wp) function vectorutilities::interpolateline (real (kind=wp), intent(in) *x1,* real (kind=wp), intent(in) *x2,* real (kind=wp), intent(in) *f1,* real (kind=wp), intent(in) *f2,* real (kind=wp), intent(in) *x*)

Function to interpolate and extrapolate linear on a line.

Parameters

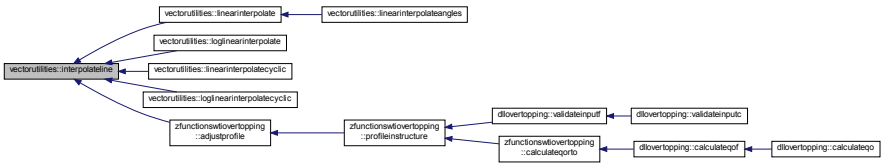
in	*x1*	first x-value
in	*x2*	second x-value
in	*f1*	first function-value
in	*f2*	second function-value
in	*x*	x-value for which the function value is desired

Returns

value of the dependent variable associated with xx

Definition at line 44 of file vectorUtilities.f90.

Here is the caller graph for this function:



7.28.2.2 real (kind=wp) function vectorutilities::intgratewithexponentcomponent (real (kind=wp), dimension(:), intent(in) *x,* real (kind=wp), dimension(:), intent(in) *f*)

Function for the Trapezium rule.

Parameters

in	*x*	vector of x-values
in	*f*	vector of function-values

Returns

value after the Trapizium Rule

Definition at line 452 of file vectorUtilities.f90.

7.28.2.3 real (kind=wp) function vectorutilities::linearinterpolate (real (kind=wp), dimension(:), intent(in) *X,* real (kind=wp), dimension(:), intent(in) *Y,* real (kind=wp), intent(in) *xx*)

Function for linearly interpolating and extrapolating.

Parameters

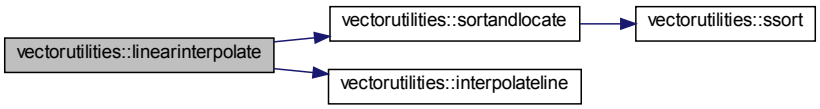
in	*x*	vector with independent elements X - in increasing order
in	*y*	vector with dependent elements Y (depending on X)
in	*xx*	x-value for which the dependent variable is desired

Returns

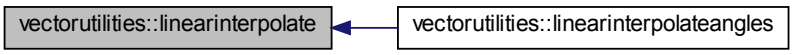
value of the dependent variable associated with xx

Definition at line 59 of file vectorUtilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.28.2.4 real (kind=wp) function vectorutilities::linearinterpolateangles (real (kind=wp), dimension(:), intent(in) *X,* real (kind=wp), dimension(:), intent(in) *Y,* real (kind=wp), intent(in) *xx*)

Function for linearly interpolating and extrapolating angles.

Parameters

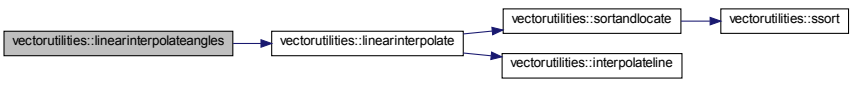
in	*x*	vector with independent elements X
in	*y*	vector with dependent elements Y (depending on X)
in	*xx*	x-value for which the dependent variable is desired

Returns

value of the dependent variable associated with xx

Definition at line 292 of file vectorUtilities.f90.

Here is the call graph for this function:



7.28.2.5 real (kind=wp) function vectorutilities::linearinterpolatecyclic (real (kind=wp), dimension(:), intent(in) *X*, real (kind=wp), dimension(:), intent(in) *Y*, real (kind=wp), intent(in) *xx*)

Function for linearly interpolating cyclic data.

Parameters

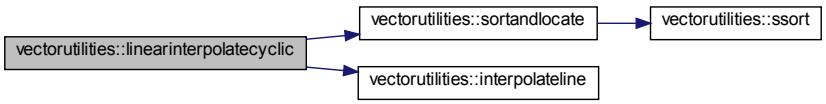
in	*x*	vector with independent elements X - in increasing order
in	*y*	vector with dependent elements Y (depending on X)
in	*xx*	x-value for which the dependent variable is desired

Returns

value of the dependent variable associated with xx

Definition at line 178 of file vectorUtilities.f90.

Here is the call graph for this function:



7.28.2.6 real (kind=wp) function vectorutilities::loglinearinterpolate (real (kind=wp), dimension(:), intent(in) *X,* real (kind=wp), dimension(:), intent(in) *Y,* real (kind=wp), intent(in) *xx*)

Function for log-linearly interpolating and extrapolating.

Parameters

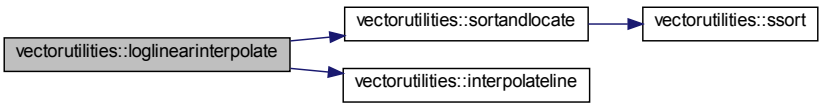
in	*x*	vector with independent elements X - in decreasing order
in	*y*	vector with dependent elements Y (depending on X)
in	*xx*	x-value for which the dependent variable is desired

Returns

value of the dependent variable associated with xx

Definition at line 109 of file vectorUtilities.f90.

Here is the call graph for this function:



7.28.2.7 real (kind=wp) function vectorutilities::loglinearinterpolatecyclic (real (kind=wp), dimension(:), intent(in) *X,* real (kind=wp), dimension(:), intent(in) *Y,* real (kind=wp), intent(in) *xx*)

Function for log-linearly interpolating and extrapolating cyclic data.

Parameters

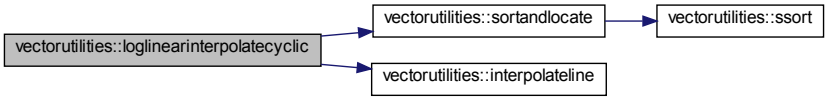
in	*x*	vector with independent elements X - in decreasing order
in	*y*	vector with dependent elements Y (depending on X)
in	*xx*	x-value for which the dependent variable is desired

Returns

value of the dependent variable associated with xx

Definition at line 227 of file vectorUtilities.f90.

Here is the call graph for this function:



7.28.2.8 real (kind=wp) function, dimension(size(x)) vectorutilities::normalize (real (kind=wp), dimension(:), intent(in) *x*)

Function for normalizing a vector If the vector is the null vector, a unit vector with uniform components is returned instead.

Parameters

in	*x*	vector to be normalized

Returns

output vector

Definition at line 23 of file vectorUtilities.f90.

7.28.2.9 subroutine vectorutilities::sortandlocate (real (kind=wp), dimension(:), intent(inout) *X,* integer, dimension(:) *IDXsorted,* real (kind=wp), intent(in) *xx,* integer *locID,* integer *locIDm1,* logical *Cyclic*)

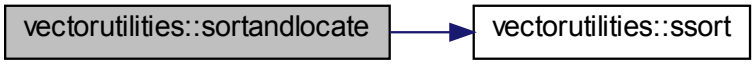
Subroutine to bracket a given value xx within two elements of an array after sort in descending order.

Parameters

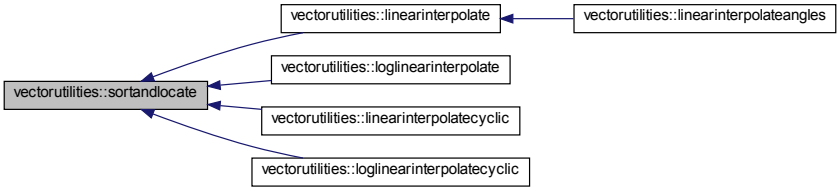
in,out	*x*	vector with independent elements X - in decreasing order
	idxsorted	sorted indices of x vector
in	*xx*	x-value for which the dependent variable is desired
	locid	sought index
	locidm1	sought index - 1
	cyclic	a flag to be true if data are cyclic; false if not

Definition at line 338 of file vectorUtilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.28.2.10 subroutine vectorutilities::ssort (real (kind=wp), dimension(n) *x,* integer, dimension(n), intent(inout) *iy,* integer, intent(in) *n,* integer, intent(in) *kflag*)

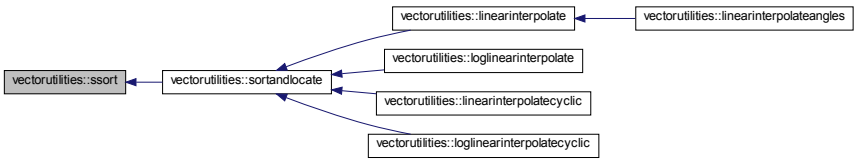
Subroutine to sort a vector in either ascending or descending order.

Parameters

	x	vector of x-values
in,out	*iy*	vector of indices to contain sorted indices of x
in	*n*	size of x and iy
in	*kflag*	a flag to indicate ascending(positive) or descending (negative) sorting

Definition at line 483 of file vectorUtilities.f90.

Here is the caller graph for this function:



7.28.2.11 real (kind=wp) function vectorutilities::stepsize (integer, intent(in) *i,* real (kind=wp), dimension(:), intent(in) *x*)

Function for the Trapezium rule.

Parameters

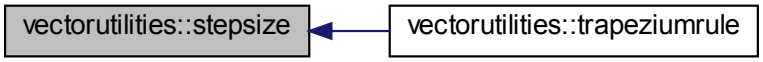
in	*i*	index of the vector x
in	*x*	vector of x-values

Returns

Step size

Definition at line 395 of file vectorUtilities.f90.

Here is the caller graph for this function:



7.28.2.12 real (kind=wp) function vectorutilities::trapeziumrule (real (kind=wp), dimension(:), intent(in) *x*, real (kind=wp), dimension(:), intent(in) *f*)

Function for the Trapezium rule.

Parameters

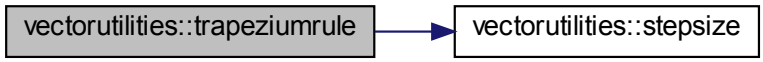
in	*x*	vector of x-values
in	*f*	vector of function-values

Returns

value after the Trapizium Rule

Definition at line 423 of file vectorUtilities.f90.

Here is the call graph for this function:



7.29 waveparametersutilities Module Reference

Module to calculate the wave steepness or the wave period.

Functions/Subroutines

- real(wp) function computewavesteepness (waveHeight, wavePeriod)

Module to calculate the the wave steepness from the wave height and the wave period.

- real(wp) function computewaveperiod (waveHeight, waveSteepness)

  Module to calculate the the wave period from the wave height and the wave steepness.

7.29.1 Detailed Description

Module to calculate the wave steepness or the wave period.

7.29.2 Function/Subroutine Documentation

7.29.2.1 real(wp) function waveparametersutilities::computewaveperiod (real(wp), intent(in) *waveHeight,* real(wp), intent(in) *waveSteepness*)

Module to calculate the the wave period from the wave height and the wave steepness.

Parameters

in	*waveheight*	wave height Hs
in	*wavesteepness*	wave steepness

Returns

wave period Tm-1,0

Definition at line 59 of file waveParametersUtilities.f90.

7.29.2.2 real(wp) function waveparametersutilities::computewavesteepness (real(wp), intent(in) *waveHeight,* real(wp), intent(in) *wavePeriod*)

Module to calculate the the wave steepness from the wave height and the wave period.

Parameters

in	*waveheight*	wave height Hs
in	*waveperiod*	wave period Tm-1,0

Returns

wave steepness

Definition at line 25 of file waveParametersUtilities.f90.

7.30 waverunup Module Reference

Functions/Subroutines

- subroutine, public iterationwaverunup (geometry, h, Hm0, Tm_10, gammaBeta_z, modelFactors, z2, succes, errorMessage)

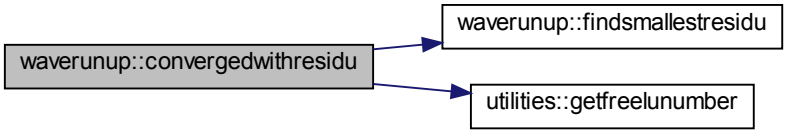
  iterationWaveRunup: iteration for the wave runup
- real(kind=wp) function determinestartingvalue (i, relaxationFactor, z2_start, z2_end, Hm0)
- integer function findsmallestresidu (z2_start, z2_end)
- logical function convergedwithresidu (i, z2_start, z2_end)

7.30.1 Function/Subroutine Documentation

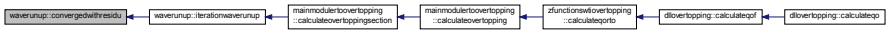
7.30.1.1 logical function waverunup::convergedwithresidu (integer, intent(in) *i,* real(kind=wp), dimension(:), intent(in) *z2_start,* real(kind=wp), dimension(:), intent(inout) *z2_end*) [private]

Definition at line 284 of file waveRunup.f90.

Here is the call graph for this function:



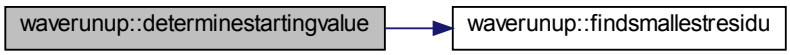
Here is the caller graph for this function:



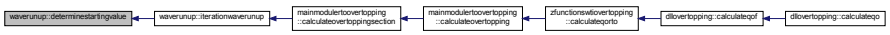
7.30.1.2 real(kind=wp) function waverunup::determinestartingvalue (integer, intent(in) *i,* real(kind=wp), intent(in) *relaxationFactor,* real(kind=wp), dimension(:), intent(in) *z2_start,* real(kind=wp), dimension(:), intent(in) *z2_end,* real(kind=wp), intent(in) *Hm0*) [private]

Definition at line 231 of file waveRunup.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.30.1.3 integer function waverunup::findsmallestresidu (real(kind=wp), dimension(:), intent(in) *z2_start,* real(kind=wp), dimension(:), intent(in) *z2_end*) [private]

Definition at line 260 of file waveRunup.f90.

Here is the caller graph for this function:

7.30.1.4 subroutine, public waverunup::iterationwaverunup (type (tpgeometry), intent(in) *geometry,* real(wp), intent(in) *h,* real(wp), intent(in) *Hm0,* real(wp), intent(in) *Tm_10,* real(wp), intent(inout) *gammaBeta_z,* type (tpovertoppinginput), intent(in) *modelFactors,* real(wp), intent(out) *z2,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

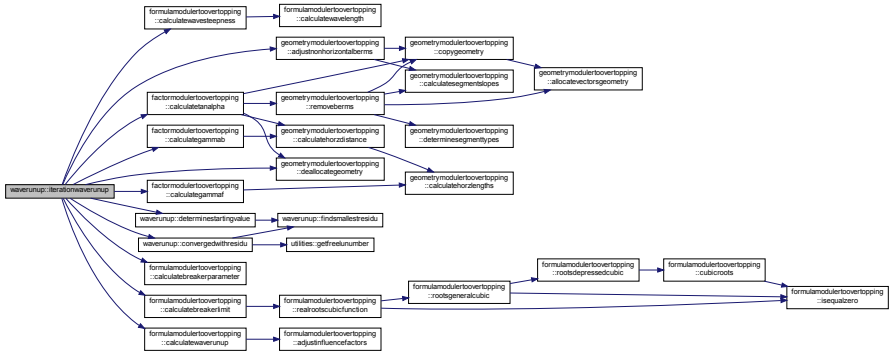
iterationWaveRunup: iteration for the wave runup

Parameters

in	*geometry*	structure with geometry data
in	*h*	local water level (m+NAP)
in	*hm0*	significant wave height (m)
in	*tm_10*	spectral wave period (s)
in,out	*gammabeta_z*	influence factor angle wave attack 2% run-up
in	*modelfactors*	structure with model factors
out	*z2*	2% wave run-up (m)
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 33 of file waveRunup.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

7.31 writeutilities Module Reference

Data Types

- interface writetodevice

Functions/Subroutines

- subroutine writetodevice_line (fileId, line)

  Write to device.
- subroutine writetodevice_text_real (fileId, formatText, textValue, realValue)

  Write to device.
- subroutine, public writetodevicechars (fileId, char, count)

  Write count times the same character to output device.
- character(len=5) function, public inttostr (intValue)

  Convert integer to string.

7.31.1 Function/Subroutine Documentation

7.31.1.1 character(len=5) function, public writeutilities::inttostr (integer, intent(in) *intValue*)

Convert integer to string.

Parameters

in	*intvalue*	integer number to convert

Returns

return text

Definition at line 71 of file writeUtilities.f90.

7.31.1.2 subroutine writeutilities::writetodevice_line (integer, intent(in) *fileId,* character(len=∗), intent(in) *line*) [private]

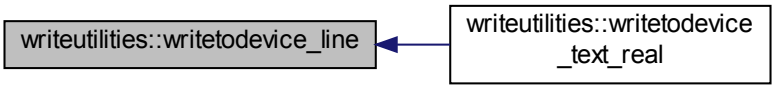
Write to device.

Parameters

in	*fileid*	device number to write to, 0 = screen
in	*line*	text to write

Definition at line 28 of file writeUtilities.f90.

Here is the caller graph for this function:



7.31.1.3 subroutine writeutilities::writetodevice_text_real (integer, intent(in) *fileId,* character(len=∗), intent(in) *formatText,* character(len=∗), intent(in) *textValue,* real (kind=wp), intent(in) *realValue*) [private]

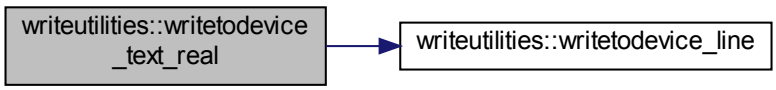
Write to device.

Parameters

in	*fileid*	device number to write to, 0 = screen
in	*formattext*	format specifier
in	*textvalue*	text to write
in	*realvalue*	real value to print

Definition at line 42 of file writeUtilities.f90.

Here is the call graph for this function:



7.31.1.4 subroutine, public writeutilities::writetodevicechars (integer, intent(in) *fileId,* character(len=1), intent(in) *char,* integer, intent(in) *count*)

Write count times the same character to output device.

Parameters

in	*fileid*	device number to write to, 0 = screen
in	*char*	character to be printed
in	*count*	number of times to output the character

Definition at line 57 of file writeUtilities.f90.

7.32 zfunctionswtiovertopping Module Reference

Module for the Limit State Functions (Z-functions) for wave overtopping.

Functions/Subroutines

- subroutine, public calculateqorto (dikeHeight, modelFactors, overtopping, load, geometry, succes, error↩ Message)

  Subroutine to calculate the overtopping discharge with the RTO-overtopping dll.
- subroutine, public profileinstructure (nrCoordinates, xcoordinates, ycoordinates, dikeHeight, nrCoords↩ Adjusted, xCoordsAdjusted, zCoordsAdjusted, succes, errorMessage)

  Subroutine to fill the profile in a structure and call the adjustment function of the profile due to a desired dike height.
- subroutine adjustprofile (nrCoordinates, coordinates, dikeHeight, nrCoordsAdjusted, xCoordsAdjusted, z↩ CoordsAdjusted, succes, errorMessage)

  Subroutine adjust the profile due to a desired dike height.
- real(kind=wp) function, public zfunclogratios (qo, qc, mqo, mqc, success, errorMessage)

  Routine to compute the limit state value by using the logs of the overtopping discharges (computed and desired)

7.32.1 Detailed Description

Module for the Limit State Functions (Z-functions) for wave overtopping.

7.32.2 Function/Subroutine Documentation

7.32.2.1 subroutine zfunctionswtiovertopping::adjustprofile (integer, intent(in) *nrCoordinates,* type(tpprofilecoordinate), dimension(nrcoordinates), intent(in) *coordinates,* real(kind=wp), intent(in) *dikeHeight,* integer, intent(out) *nrCoordsAdjusted,* real(kind=wp), dimension(:), pointer *xCoordsAdjusted,* real(kind=wp), dimension(:), pointer *zCoordsAdjusted,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*) [private]

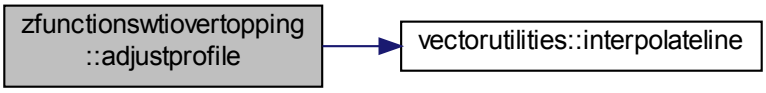
Subroutine adjust the profile due to a desired dike height.

Parameters

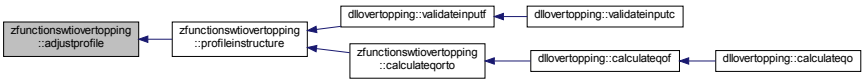
in	*nrcoordinates*	number of coordinates of the profile
in	*coordinates*	structure for the profile
in	*dikeheight*	dike height
out	*nrcoordsadjusted*	number of coordinates in the adjusted profile
	xcoordsadjusted	vector with x-coordinates of the adjusted profile
	zcoordsadjusted	vector with y-coordinates of the adjusted profile
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 109 of file zFunctionsWTIOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.32.2.2 subroutine, public zfunctionswtiovertopping::calculateqorto (real(kind=wp), intent(in) *dikeHeight,* type(tpovertoppinginput), intent(inout) *modelFactors,* type (tpovertopping), intent(out) *overtopping,* type (tpload), intent(in) *load,* type (tpgeometry), intent(in) *geometry,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

Subroutine to calculate the overtopping discharge with the RTO-overtopping dll.

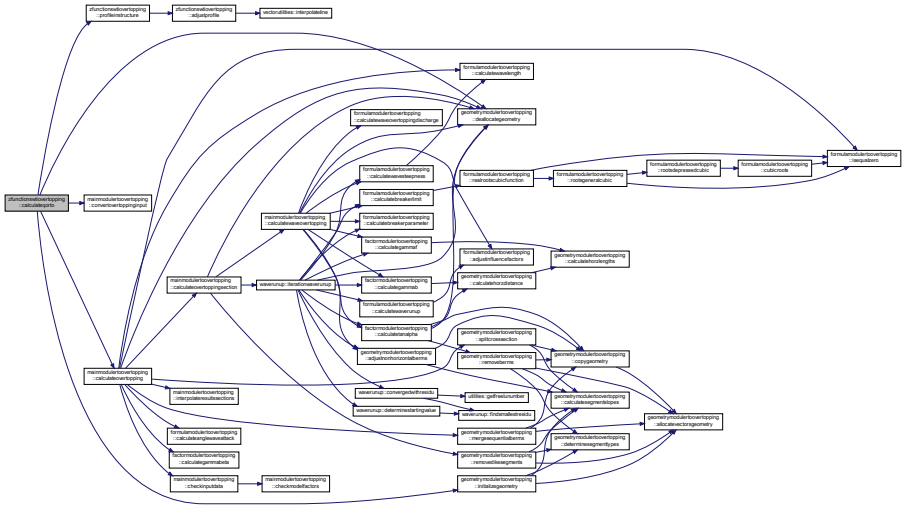
Parameters

in	*dikeheight*	dike height

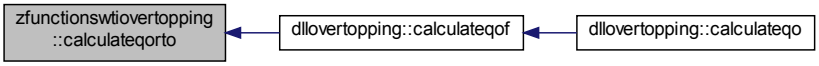
in,out	*modelfactors*	struct with model factors
out	*overtopping*	structure with overtopping results
in	*geometry*	structure with geometry data
in	*load*	structure with load parameters
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 32 of file zFunctionsWTIOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.32.2.3 subroutine, public zfunctionswtiovertopping::profileinstructure (integer, intent(in) *nrCoordinates,* real(kind=wp), dimension(nrcoordinates), intent(in) *xcoordinates,* real(kind=wp), dimension(nrcoordinates), intent(in) *ycoordinates,* real(kind=wp), intent(in) *dikeHeight,* integer, intent(out) *nrCoordsAdjusted,* real(kind=wp), dimension(:), pointer *xCoordsAdjusted,* real(kind=wp), dimension(:), pointer *zCoordsAdjusted,* logical, intent(out) *succes,* character(len=∗), intent(out) *errorMessage*)

Subroutine to fill the profile in a structure and call the adjustment function of the profile due to a desired dike height.

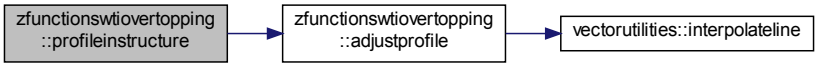
Parameters

in	*nrcoordinates*	number of coordinates of the profile
in	*xcoordinates*	vector with x-coordinates of the profile
in	*ycoordinates*	vector with y-coordinates of the profile

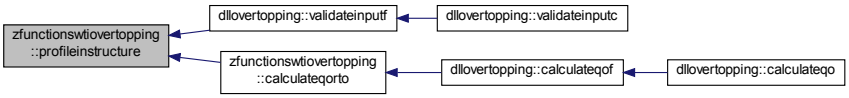
in	*dikeheight*	dike height
out	*nrcoordsadjusted*	number of coordinates in the adjusted profile
	xcoordsadjusted	vector with x-coordinates of the adjusted profile
	zcoordsadjusted	vector with y-coordinates of the adjusted profile
out	*succes*	flag for succes
out	*errormessage*	error message

Definition at line 84 of file zFunctionsWTIOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.32.2.4 real (kind=wp) function, public zfunctionswtiovertopping::zfunclogratios (real (kind=wp), intent(in) *qo,* real (kind=wp), intent(in) *qc,* real (kind=wp), intent(in) *mqo,* real (kind=wp), intent(in) *mqc,* logical, intent(out) *success,* character(len=∗), intent(out) *errorMessage*)

Routine to compute the limit state value by using the logs of the overtopping discharges (computed and desired)

Parameters

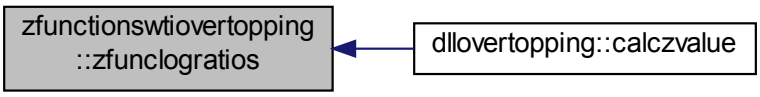
in	*qo*	computed overtopping discharge
in	*qc*	Critical overtopping discharge
in	*mqo*	Model factor computed overtopping discharge
in	*mqc*	Model factor Critical overtopping discharge
out	*success*	Flag for succes
out	*errormessage*	error message, only set if not successful

Returns

Value z-function

Definition at line 198 of file zFunctionsWTIOvertopping.f90.

Here is the caller graph for this function:

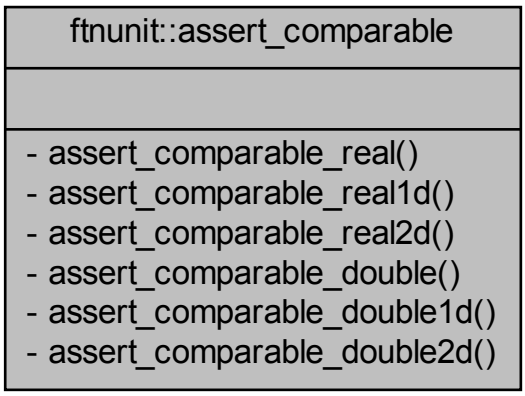


Chapter 8

Data Type Documentation

8.1 ftnunit::assert_comparable Interface Reference

Collaboration diagram for ftnunit::assert_comparable:



Private Member Functions

- subroutine assert_comparable_real (value1, value2, margin, text)
- subroutine assert_comparable_real1d (array1, array2, margin, text)
- subroutine assert_comparable_real2d (array1, array2, margin, text)
- subroutine assert_comparable_double (value1, value2, margin, text)
- subroutine assert_comparable_double1d (array1, array2, margin, text)
- subroutine assert_comparable_double2d (array1, array2, margin, text)

8.1.1 Detailed Description

Definition at line 90 of file ftnunit.f90.

8.1.2 Member Function/Subroutine Documentation

8.1.2.1 subroutine ftnunit::assert_comparable::assert_comparable_double (real(kind=dp), intent(in) *value1,* real(kind=dp), intent(in) *value2,* real(kind=dp), intent(in) *margin,* character(len=∗), intent(in) *text*) [private]

Definition at line 966 of file ftnunit.f90.

8.1.2.2 subroutine ftnunit::assert_comparable::assert_comparable_double1d (real(kind=dp), dimension(:), intent(in) *array1,* real(kind=dp), dimension(:), intent(in) *array2,* real(kind=dp), intent(in) *margin,* character(len=∗), intent(in) *text*) [private]

Definition at line 997 of file ftnunit.f90.

8.1.2.3 subroutine ftnunit::assert_comparable::assert_comparable_double2d (real(kind=dp), dimension(:,:), intent(in) *array1,* real(kind=dp), dimension(:,:), intent(in) *array2,* real(kind=dp), intent(in) *margin,* character(len=∗), intent(in) *text*) [private]

Definition at line 1061 of file ftnunit.f90.

8.1.2.4 subroutine ftnunit::assert_comparable::assert_comparable_real (real, intent(in) *value1,* real, intent(in) *value2,* real, intent(in) *margin,* character(len=∗), intent(in) *text*) [private]

Definition at line 802 of file ftnunit.f90.

8.1.2.5 subroutine ftnunit::assert_comparable::assert_comparable_real1d (real, dimension(:), intent(in) *array1,* real, dimension(:), intent(in) *array2,* real, intent(in) *margin,* character(len=∗), intent(in) *text*) [private]

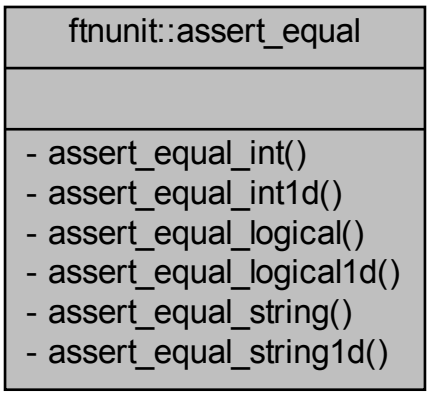
Definition at line 834 of file ftnunit.f90.

8.1.2.6 subroutine ftnunit::assert_comparable::assert_comparable_real2d (real, dimension(:,:), intent(in) *array1,* real, dimension(:,:), intent(in) *array2,* real, intent(in) *margin,* character(len=∗), intent(in) *text*) [private]

Definition at line 898 of file ftnunit.f90.

8.2 ftnunit::assert_equal Interface Reference

Collaboration diagram for ftnunit::assert_equal:



Private Member Functions

- subroutine assert_equal_int (value1, value2, text)
- subroutine assert_equal_int1d (array1, array2, text)
- subroutine assert_equal_logical (value1, value2, text)
- subroutine assert_equal_logical1d (array1, array2, text)
- subroutine assert_equal_string (value1, value2, text)
- subroutine assert_equal_string1d (array1, array2, text)

8.2.1 Detailed Description

Definition at line 81 of file ftnunit.f90.

8.2.2 Member Function/Subroutine Documentation

8.2.2.1 subroutine ftnunit::assert_equal::assert_equal_int (integer, intent(in) *value1,* integer, intent(in) *value2,* character(len=∗), intent(in) *text*) [private]

Definition at line 724 of file ftnunit.f90.

8.2.2.2 subroutine ftnunit::assert_equal::assert_equal_int1d (integer, dimension(:), intent(in) *array1,* integer, dimension(:), intent(in) *array2,* character(len=∗), intent(in) *text*) [private]

Definition at line 748 of file ftnunit.f90.

8.2.2.3 subroutine ftnunit::assert_equal::assert_equal_logical (logical, intent(in) *value1,* logical, intent(in) *value2,* character(len=∗), intent(in) *text*) [private]

Definition at line 575 of file ftnunit.f90.

8.2.2.4 subroutine ftnunit::assert_equal::assert_equal_logical1d (logical, dimension(:), intent(in) *array1,* logical, dimension(:), intent(in) *array2,* character(len=∗), intent(in) *text*) [private]

Definition at line 599 of file ftnunit.f90.

8.2.2.5 subroutine ftnunit::assert_equal::assert_equal_string (character(len=∗), intent(in) *value1,* character(len=∗), intent(in) *value2,* character(len=∗), intent(in) *text*) [private]

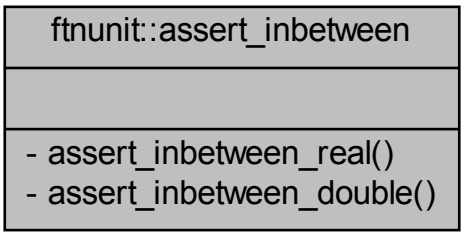
Definition at line 649 of file ftnunit.f90.

8.2.2.6 subroutine ftnunit::assert_equal::assert_equal_string1d (character(len=∗), dimension(:), intent(in) *array1,* character(len=∗), dimension(:), intent(in) *array2,* character(len=∗), intent(in) *text*) [private]

Definition at line 675 of file ftnunit.f90.

8.3 ftnunit::assert_inbetween Interface Reference

Collaboration diagram for ftnunit::assert_inbetween:



Private Member Functions

- subroutine assert_inbetween_real (value, vmin, vmax, text)
- subroutine assert_inbetween_double (value, vmin, vmax, text)

8.3.1 Detailed Description

Definition at line 98 of file ftnunit.f90.

8.3.2 Member Function/Subroutine Documentation

8.3.2.1 subroutine ftnunit::assert_inbetween::assert_inbetween_double (real(kind=dp), intent(in) *value,* real(kind=dp), intent(in) *vmin,* real(kind=dp), intent(in) *vmax,* character(len=∗), intent(in) *text*) [private]

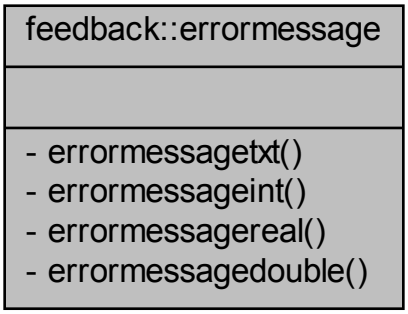
Definition at line 1168 of file ftnunit.f90.

8.3.2.2 subroutine ftnunit::assert_inbetween::assert_inbetween_real (real, intent(in) *value,* real, intent(in) *vmin,* real, intent(in) *vmax,* character(len=*), intent(in) *text*) [private]

Definition at line 1129 of file ftnunit.f90.

8.4 feedback::errormessage Interface Reference

Collaboration diagram for feedback::errormessage:



Private Member Functions

- subroutine errormessagetxt (msgtext)

  Write an error message.
- subroutine errormessageint (msgtext, value)

  Write an error message with an integer value.
- subroutine errormessagereal (msgtext, value)

  Write an error message with a real value.
- subroutine errormessagedouble (msgtext, value)

  Write an error message with a double precision real value.

8.4.1 Detailed Description

Definition at line 149 of file feedback.f90.

8.4.2 Member Function/Subroutine Documentation

8.4.2.1 subroutine feedback::errormessage::errormessagedouble (character(len=*), intent(in) *msgtext,* real(kind=dp), intent(in) *value*) [private]

Write an error message with a double precision real value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 661 of file feedback.f90.

8.4.2.2 subroutine feedback::errormessage::errormessageint (character(len=∗), intent(in) *msgtext,* integer, intent(in) *value*) [private]

Write an error message with an integer value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 631 of file feedback.f90.

8.4.2.3 subroutine feedback::errormessage::errormessagereal (character(len=∗), intent(in) *msgtext,* real, intent(in) *value*) [private]

Write an error message with a real value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 646 of file feedback.f90.

8.4.2.4 subroutine feedback::errormessage::errormessagetxt (character(len=∗), intent(in) *msgtext*) [private]

Write an error message.

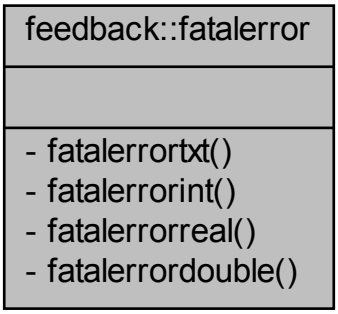
Parameters

in	*msgtext*	Error message to write

Definition at line 608 of file feedback.f90.

8.5 feedback::fatalerror Interface Reference

Collaboration diagram for feedback::fatalerror:



Private Member Functions

- subroutine fatalerrortxt (msgtext)

  Write a fatal error message.
- subroutine fatalerrorint (msgtext, value)

  Write a fatal error message with an integer value.
- subroutine fatalerrorreal (msgtext, value)

  Write a fatal error message with a real value.
- subroutine fatalerrordouble (msgtext, value)

  Write a fatal error message with a double precision real value.

8.5.1 Detailed Description

Definition at line 163 of file feedback.f90.

8.5.2 Member Function/Subroutine Documentation

8.5.2.1 subroutine feedback::fatalerror::fatalerrordouble (character(len=∗), intent(in) *msgtext,* real(kind=dp), intent(in) *value*) [private]

Write a fatal error message with a double precision real value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 815 of file feedback.f90.

8.5.2.2 subroutine feedback::fatalerror::fatalerrorint (character(len=∗), intent(in) *msgtext,* integer, intent(in) *value*) [private]

Write a fatal error message with an integer value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 785 of file feedback.f90.

8.5.2.3 subroutine feedback::fatalerror::fatalerrorreal (character(len=∗), intent(in) *msgtext,* real, intent(in) *value*) [private]

Write a fatal error message with a real value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 800 of file feedback.f90.

8.5.2.4 subroutine feedback::fatalerror::fatalerrortxt (character(len=∗), intent(in) *msgtext*) [private]

Write a fatal error message.

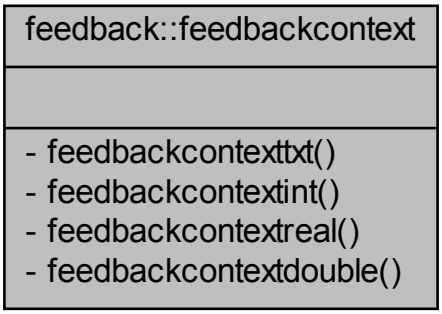
Parameters

in	*msgtext*	Error message to write

Definition at line 744 of file feedback.f90.

8.6 feedback::feedbackcontext Interface Reference

Collaboration diagram for feedback::feedbackcontext:



Private Member Functions

- subroutine feedbackcontexttxt (text)

  Register a context for error messages.
- subroutine feedbackcontextint (text, value)

Register a context for error messages - with integer value.

- subroutine feedbackcontextreal (text, value)

  Register a context for error messages - with real value.

- subroutine feedbackcontextdouble (text, value)

  Register a context for error messages - with double precision real value.

8.6.1 Detailed Description

Definition at line 142 of file feedback.f90.

8.6.2 Member Function/Subroutine Documentation

8.6.2.1 subroutine feedback::feedbackcontext::feedbackcontextdouble (character(len=∗), intent(in) *text,* real(kind=dp), intent(in) *value*) [private]

Register a context for error messages - with double precision real value.

Parameters

in	*text*	text to be added to the context stack
in	*value*	number to be added to the text

Definition at line 900 of file feedback.f90.

8.6.2.2 subroutine feedback::feedbackcontext::feedbackcontextint (character(len=∗), intent(in) *text,* integer, intent(in) *value*) [private]

Register a context for error messages - with integer value.

Parameters

in	*text*	text to be added to the context stack
in	*value*	number to be added to the text

Definition at line 870 of file feedback.f90.

8.6.2.3 subroutine feedback::feedbackcontext::feedbackcontextreal (character(len=∗), intent(in) *text,* real(kind=sp), intent(in) *value*) [private]

Register a context for error messages - with real value.

Parameters

in	*text*	text to be added to the context stack
in	*value*	number to be added to the text

Definition at line 885 of file feedback.f90.

8.6.2.4 subroutine feedback::feedbackcontext::feedbackcontexttxt (character(len=∗), intent(in) *text*) [private]

Register a context for error messages.

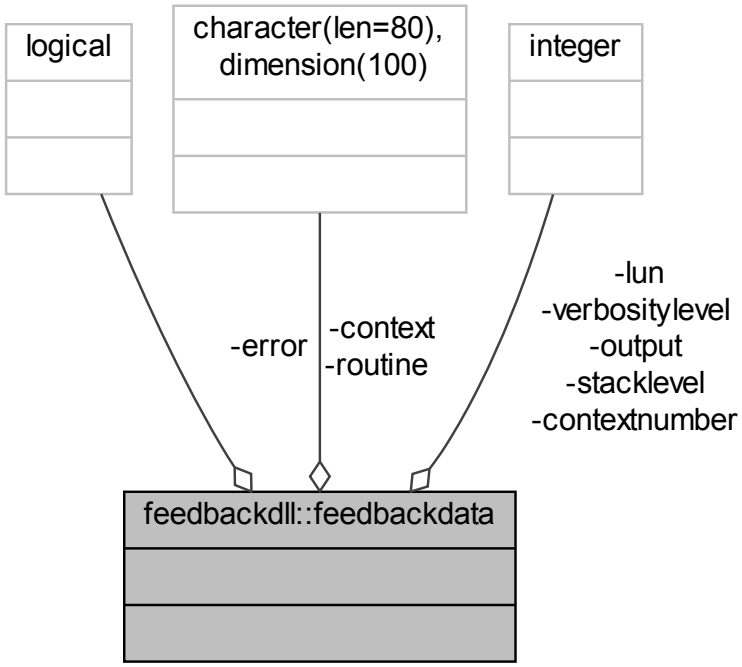
Parameters

in	*text*	text to be added to the context stack

Definition at line 852 of file feedback.f90.

8.7 feedbackdll::feedbackdata Type Reference

Collaboration diagram for feedbackdll::feedbackdata:



Private Attributes

- character(len=80), dimension(100) routine
- character(len=80), dimension(100) context
- integer output
- integer contextnumber
- integer stacklevel
- integer verbositylevel
- integer lun = -1
- logical error = .true.

8.7.1 Detailed Description

Definition at line 24 of file feedbackDLL.f90.

8.7.2 Member Data Documentation

8.7.2.1 character(len=80), dimension(100) feedbackdll::feedbackdata::context [private]

Definition at line 26 of file feedbackDLL.f90.

8.7.2.2 integer feedbackdll::feedbackdata::contextnumber [private]

Definition at line 28 of file feedbackDLL.f90.

8.7.2.3 logical feedbackdll::feedbackdata::error = .true. [private]

Definition at line 32 of file feedbackDLL.f90.

8.7.2.4 integer feedbackdll::feedbackdata::lun = -1 [private]

Definition at line 31 of file feedbackDLL.f90.

8.7.2.5 integer feedbackdll::feedbackdata::output [private]

Definition at line 27 of file feedbackDLL.f90.

8.7.2.6 character(len=80), dimension(100) feedbackdll::feedbackdata::routine [private]

Definition at line 25 of file feedbackDLL.f90.

8.7.2.7 integer feedbackdll::feedbackdata::stacklevel [private]

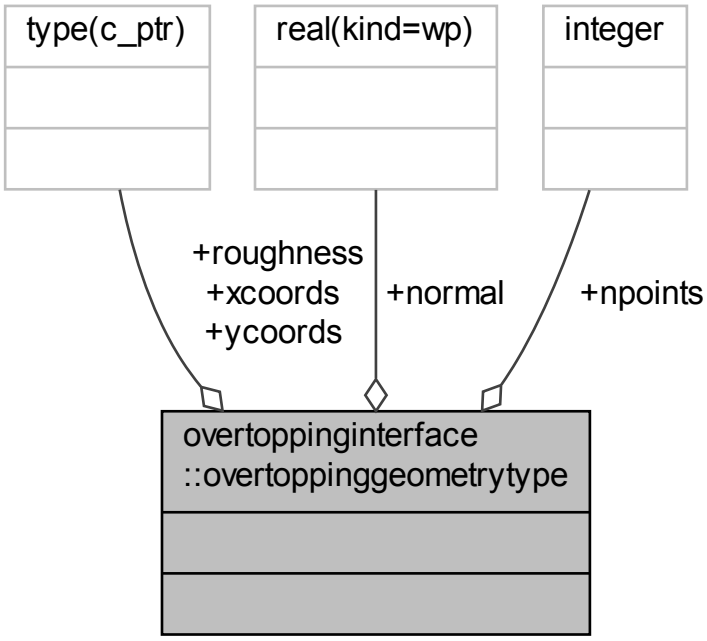
Definition at line 29 of file feedbackDLL.f90.

8.7.2.8 integer feedbackdll::feedbackdata::verbositylevel [private]

Definition at line 30 of file feedbackDLL.f90.

8.8 overtoppinginterface::overtoppinggeometrytype Type Reference

Collaboration diagram for overtoppinginterface::overtoppinggeometrytype:



Public Attributes

- real(kind=wp) normal
- integer npoints
- type(c_ptr) xcoords
- type(c_ptr) ycoords
- type(c_ptr) roughness

8.8.1 Detailed Description

Definition at line 25 of file overtoppingInterface.f90.

8.8.2 Member Data Documentation

8.8.2.1 real(kind=wp) overtoppinginterface::overtoppinggeometrytype::normal

Definition at line 26 of file overtoppingInterface.f90.

8.8.2.2 integer overtoppinginterface::overtoppinggeometrytype::npoints

Definition at line 27 of file overtoppingInterface.f90.

8.8.2.3 type(c_ptr) overtoppinginterface::overtoppinggeometrytype::roughness

Definition at line 30 of file overtoppingInterface.f90.

8.8.2.4 type(c_ptr) overtoppinginterface::overtoppinggeometrytype::xcoords

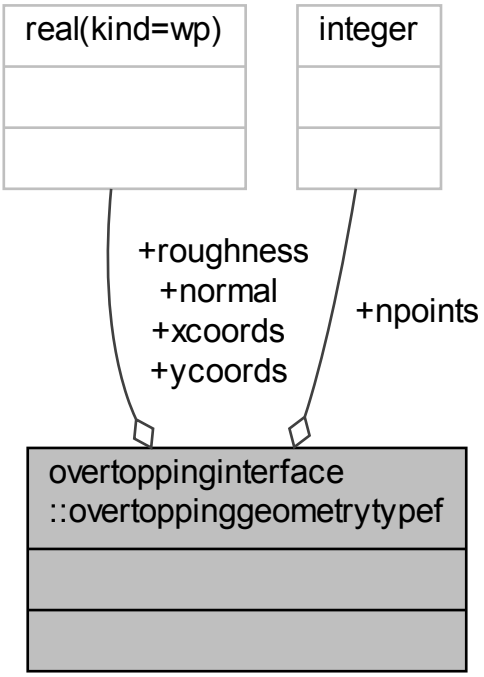
Definition at line 28 of file overtoppingInterface.f90.

8.8.2.5 type(c_ptr) overtoppinginterface::overtoppinggeometrytype::ycoords

Definition at line 29 of file overtoppingInterface.f90.

8.9 overtoppinginterface::overtoppinggeometrytypef Type Reference

Collaboration diagram for overtoppinginterface::overtoppinggeometrytypef:



Public Attributes

- real(kind=wp) normal
- integer npoints
- real(kind=wp), dimension(:), pointer xcoords
- real(kind=wp), dimension(:), pointer ycoords
- real(kind=wp), dimension(:), pointer roughness

8.9.1 Detailed Description

Definition at line 33 of file overtoppingInterface.f90.

8.9.2 Member Data Documentation

8.9.2.1 real(kind=wp) overtoppinginterface::overtoppinggeometrytypef::normal

Definition at line 34 of file overtoppingInterface.f90.

8.9.2.2 integer overtoppinginterface::overtoppinggeometrytypef::npoints

Definition at line 35 of file overtoppingInterface.f90.

8.9.2.3 real(kind=wp), dimension(:), pointer overtoppinginterface::overtoppinggeometrytypef::roughness

Definition at line 38 of file overtoppingInterface.f90.

8.9.2.4 real(kind=wp), dimension(:), pointer overtoppinginterface::overtoppinggeometrytypef::xcoords

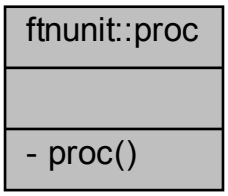
Definition at line 36 of file overtoppingInterface.f90.

8.9.2.5 real(kind=wp), dimension(:), pointer overtoppinginterface::overtoppinggeometrytypef::ycoords

Definition at line 37 of file overtoppingInterface.f90.

8.10 ftnunit::proc Interface Reference

Collaboration diagram for ftnunit::proc:



Private Member Functions

- subroutine proc

8.10.1 Detailed Description

Definition at line 40 of file ftnunit.f90.

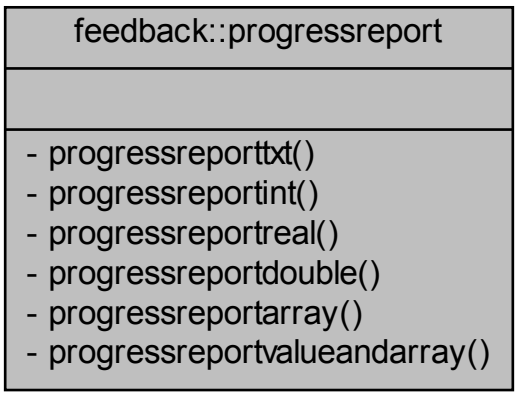
8.10.2 Constructor & Destructor Documentation

8.10.2.1 subroutine ftnunit::proc::proc ( ) [private]

Definition at line 41 of file ftnunit.f90.

8.11 feedback::progressreport Interface Reference

Collaboration diagram for feedback::progressreport:



Private Member Functions

- subroutine progressreporttxt (level, msgtext)

  Write a single message to the output for the benefit of the user.
- subroutine progressreportint (level, msgtext, value)

  Write a single message with an integer value.
- subroutine progressreportreal (level, msgtext, value)

  Write a single message with a real value.
- subroutine progressreportdouble (level, msgtext, value)

  Write a single message with a double precision real value.
- subroutine progressreportarray (level, msgText, x, size)

  Print array values to report file.
- subroutine progressreportvalueandarray (level, msgText, z, x, size)

  Print z-value and array values to report file.

8.11.1 Detailed Description

Definition at line 170 of file feedback.f90.

8.11.2 Member Function/Subroutine Documentation

8.11.2.1 subroutine feedback::progressreport::progressreportarray (integer, intent(in) *level,* character(len=*), intent(in) *msgText,* real(kind=wp), dimension(:), intent(in) *x,* integer, intent(in) *size*) [private]

Print array values to report file.

Parameters

in	*level*	Level of verbosity
in	*msgtext*	Error message to write
in	*x*	x vector to write
in	*size*	the length of x

Definition at line 489 of file feedback.f90.

8.11.2.2 subroutine feedback::progressreport::progressreportdouble (integer, intent(in) *level,* character(len=∗), intent(in) *msgtext,* real(kind=dp), intent(in) *value*) [private]

Write a single message with a double precision real value.

Parameters

in	*level*	Level of verbosity
in	*msgtext*	Error message to write
in	*value*	Double value to add

Definition at line 472 of file feedback.f90.

8.11.2.3 subroutine feedback::progressreport::progressreportint (integer, intent(in) *level,* character(len=∗), intent(in) *msgtext,* integer, intent(in) *value*) [private]

Write a single message with an integer value.

Parameters

in	*level*	Level of verbosity
in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 440 of file feedback.f90.

8.11.2.4 subroutine feedback::progressreport::progressreportreal (integer, intent(in) *level,* character(len=∗), intent(in) *msgtext,* real, intent(in) *value*) [private]

Write a single message with a real value.

Parameters

in	*level*	Level of verbosity
in	*msgtext*	Error message to write
in	*value*	Real value to add

Definition at line 456 of file feedback.f90.

8.11.2.5 subroutine feedback::progressreport::progressreporttxt (integer, intent(in) *level,* character(len=∗), intent(in) *msgtext*) [private]

Write a single message to the output for the benefit of the user.

Parameters

in	*level*	Level of verbosity

in	*msgtext*	Text to write

Definition at line 400 of file feedback.f90.

8.11.2.6 subroutine feedback::progressreport::progressreportvalueandarray (integer, intent(in) *level,* character(len=∗), intent(in) *msgText,* real(kind=wp), intent(in) *z,* real(kind=wp), dimension(:), intent(in) *x,* integer, intent(in) *size*) [private]

Print z-value and array values to report file.

Parameters

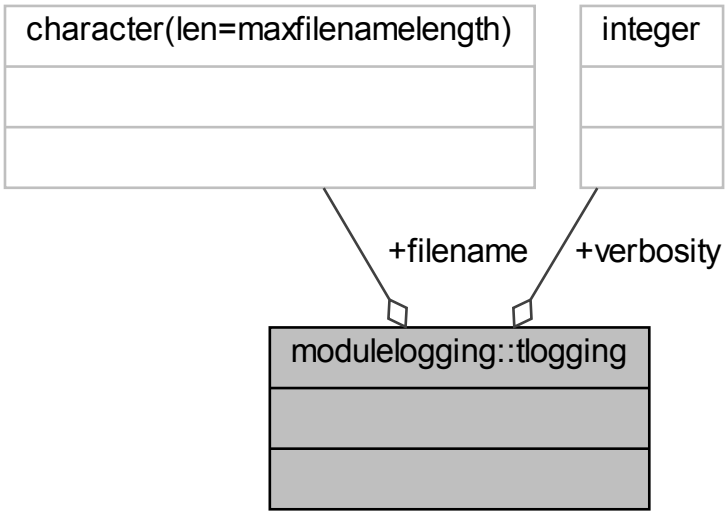
in	*level*	Level of verbosity
in	*msgtext*	Error message to write
in	*z*	z value to write
in	*x*	x vector to write
in	*size*	the length of x

Definition at line 525 of file feedback.f90.

8.12 modulelogging::tlogging Type Reference

TLogging: structure for steering the logging.

Collaboration diagram for modulelogging::tlogging:



Public Attributes

- integer verbosity = verboseNone

  level of verbosity: one of verboseNone, verboseBasic, verboseDetailed, verboseDebugging
- character(len=maxfilenamelength) filename = ' '

  filename of logging

8.12.1 Detailed Description

TLogging: structure for steering the logging.

Definition at line 16 of file ModuleLogging.f90.

8.12.2 Member Data Documentation

8.12.2.1 character(len=maxfilenamelength) modulelogging::tlogging::filename = ' '

filename of logging

Definition at line 18 of file ModuleLogging.f90.

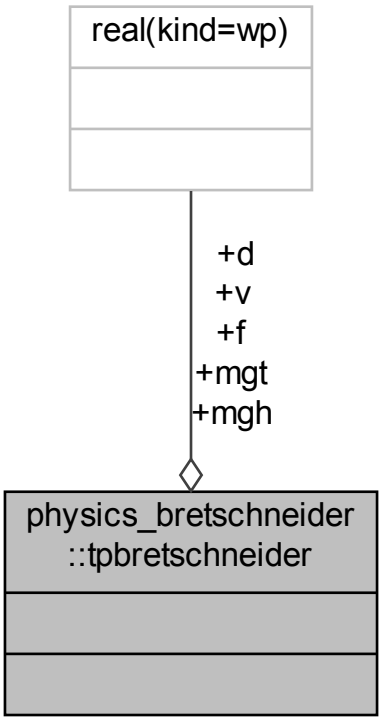
8.12.2.2 integer modulelogging::tlogging::verbosity = verboseNone

level of verbosity: one of verboseNone, verboseBasic, verboseDetailed, verboseDebugging

Definition at line 17 of file ModuleLogging.f90.

8.13 physics_bretschneider::tpbretschneider Type Reference

Collaboration diagram for physics_bretschneider::tpbretschneider:



Public Attributes

- real(kind=wp) v

  Wind velocity.
- real(kind=wp) d

  Depth of water for the section.
- real(kind=wp) f

  Fetch length for the section.
- real(kind=wp) mgh

  Model factor wave height.
- real(kind=wp) mgt

  Model factor wave period.

8.13.1 Detailed Description

Definition at line 20 of file bretschneider.f90.

8.13.2 Member Data Documentation

8.13.2.1 real (kind=wp) physics_bretschneider::tpbretschneider::d

Depth of water for the section.

Definition at line 22 of file bretschneider.f90.

8.13.2.2 real (kind=wp) physics_bretschneider::tpbretschneider::f

Fetch length for the section.

Definition at line 23 of file bretschneider.f90.

8.13.2.3 real (kind=wp) physics_bretschneider::tpbretschneider::mgh

Model factor wave height.

Definition at line 24 of file bretschneider.f90.

8.13.2.4 real (kind=wp) physics_bretschneider::tpbretschneider::mgt

Model factor wave period.

Definition at line 25 of file bretschneider.f90.

8.13.2.5 real (kind=wp) physics_bretschneider::tpbretschneider::v

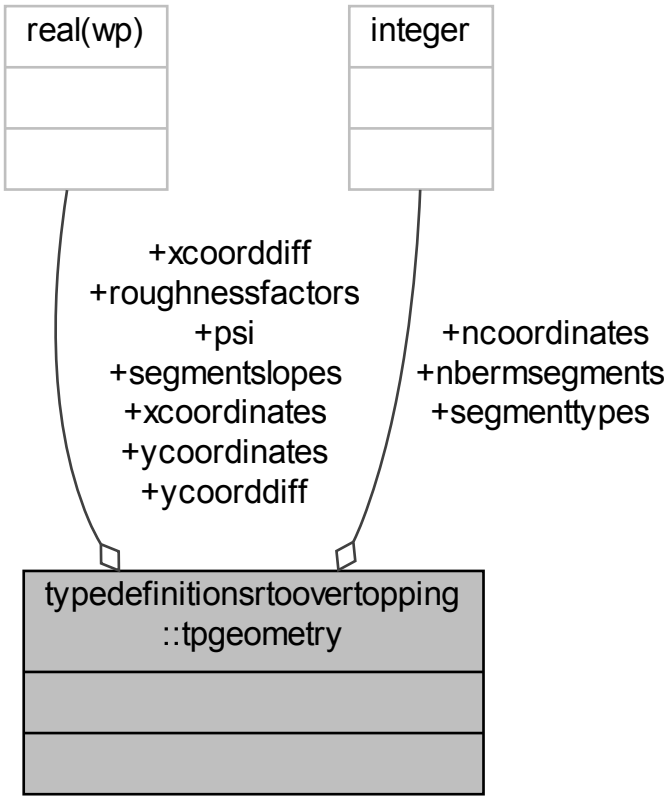
Wind velocity.

Definition at line 21 of file bretschneider.f90.

8.14 typedefinitionsrtoovertopping::tpgeometry Type Reference

tpGeometry: structure with geometry data

Collaboration diagram for typedefinitionsrtoovertopping::tpgeometry:



Public Attributes

- real(wp) psi
  dike normal (degrees)
- integer ncoordinates
  number of coordinates cross section
- real(wp), dimension(:), pointer xcoordinates
  vector with x-coordinates cross section (m)
- real(wp), dimension(:), pointer ycoordinates
  vector with y-coordinates cross section (m+NAP)
- real(wp), dimension(:), pointer roughnessfactors
  vector with roughness factors cross section
- real(wp), dimension(:), pointer xcoorddiff
  vector with differences in x-coordinates (m)
- real(wp), dimension(:), pointer ycoorddiff
  vector with differences in y-coordinates (m)

- real(wp), dimension(:), pointer segmentslopes
  vector with slopes dike segments
- integer, dimension(:), pointer segmenttypes
  vector with segment types (1=slope,2=berm,3=other)
- integer nbermsegments
  number of berm segments

8.14.1 Detailed Description

tpGeometry: structure with geometry data

Definition at line 18 of file typeDefinitionsRTOovertopping.f90.

8.14.2 Member Data Documentation

8.14.2.1 integer typedefinitionsrtoovertopping::tpgeometry::nbermsegments

number of berm segments

Definition at line 28 of file typeDefinitionsRTOovertopping.f90.

8.14.2.2 integer typedefinitionsrtoovertopping::tpgeometry::ncoordinates

number of coordinates cross section

Definition at line 20 of file typeDefinitionsRTOovertopping.f90.

8.14.2.3 real(wp) typedefinitionsrtoovertopping::tpgeometry::psi

dike normal (degrees)

Definition at line 19 of file typeDefinitionsRTOovertopping.f90.

8.14.2.4 real(wp), dimension(:), pointer typedefinitionsrtoovertopping::tpgeometry::roughnessfactors

vector with roughness factors cross section

Definition at line 23 of file typeDefinitionsRTOovertopping.f90.

8.14.2.5 real(wp), dimension(:), pointer typedefinitionsrtoovertopping::tpgeometry::segmentslopes

vector with slopes dike segments

Definition at line 26 of file typeDefinitionsRTOovertopping.f90.

8.14.2.6 integer, dimension(:), pointer typedefinitionsrtoovertopping::tpgeometry::segmenttypes

vector with segment types (1=slope,2=berm,3=other)

Definition at line 27 of file typeDefinitionsRTOovertopping.f90.

8.14.2.7 real(wp), dimension(:), pointer typedefinitionsrtoovertopping::tpgeometry::xcoorddiff

vector with differences in x-coordinates (m)

Definition at line 24 of file typeDefinitionsRTOovertopping.f90.

8.14.2.8 real(wp), dimension(:), pointer typedefinitionsrtoovertopping::tpgeometry::xcoordinates

vector with x-coordinates cross section (m)

Definition at line 21 of file typeDefinitionsRTOovertopping.f90.

8.14.2.9 real(wp), dimension(:), pointer typedefinitionsrtoovertopping::tpgeometry::ycorddiff

vector with differences in y-coordinates (m)

Definition at line 25 of file typeDefinitionsRTOovertopping.f90.

8.14.2.10 real(wp), dimension(:), pointer typedefinitionsrtoovertopping::tpgeometry::ycoordinates

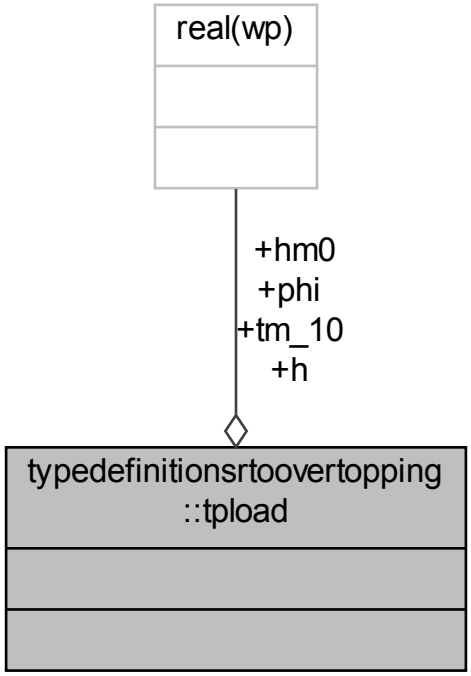
vector with y-coordinates cross section (m+NAP)

Definition at line 22 of file typeDefinitionsRTOovertopping.f90.

8.15 typedefinitionsrtoovertopping::tpload Type Reference

tpLoad: structure with load parameters

Collaboration diagram for typedefinitionsrtoovertopping::tpload:



Public Attributes

- real(wp) h

local water level (m+NAP)

- real(wp) hm0

  significant wave height (m)

- real(wp) tm_10

  spectral wave period (s)

- real(wp) phi

  wave direction (degrees)

8.15.1 Detailed Description

tpLoad: structure with load parameters

Definition at line 32 of file typeDefinitionsRTOovertopping.f90.

8.15.2 Member Data Documentation

8.15.2.1 real(wp) typedefinitionsrtoovertopping::tpload::h

local water level (m+NAP)

Definition at line 33 of file typeDefinitionsRTOovertopping.f90.

8.15.2.2 real(wp) typedefinitionsrtoovertopping::tpload::hm0

significant wave height (m)

Definition at line 34 of file typeDefinitionsRTOovertopping.f90.

8.15.2.3 real(wp) typedefinitionsrtoovertopping::tpload::phi

wave direction (degrees)

Definition at line 36 of file typeDefinitionsRTOovertopping.f90.

8.15.2.4 real(wp) typedefinitionsrtoovertopping::tpload::tm_10

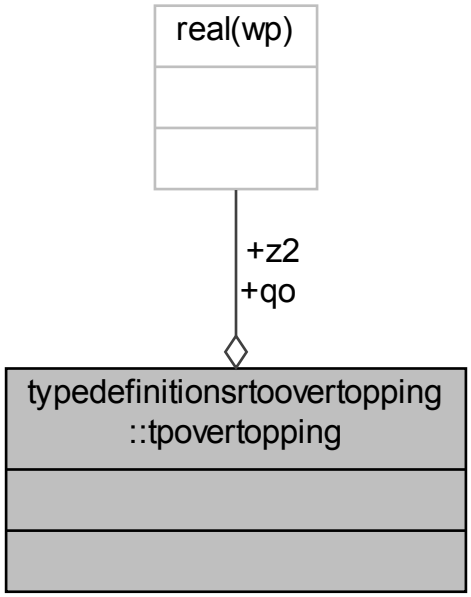
spectral wave period (s)

Definition at line 35 of file typeDefinitionsRTOovertopping.f90.

8.16 typedefinitionsrtoovertopping::tpovertopping Type Reference

tpOvertopping: structure with overtopping results

Collaboration diagram for typedefinitionsrtoovertopping::tpovertopping:



Public Attributes

- real(wp) z2

  2% wave run-up (m)
- real(wp) qo

  wave overtopping discharge (m3/m per s)

8.16.1 Detailed Description

tpOvertopping: structure with overtopping results

Definition at line 56 of file typeDefinitionsRTOovertopping.f90.

8.16.2 Member Data Documentation

8.16.2.1 real(wp) typedefinitionsrtoovertopping::tpovertopping::qo

wave overtopping discharge (m3/m per s)

Definition at line 58 of file typeDefinitionsRTOovertopping.f90.

8.16.2.2 real(wp) typedefinitionsrtoovertopping::tpovertopping::z2

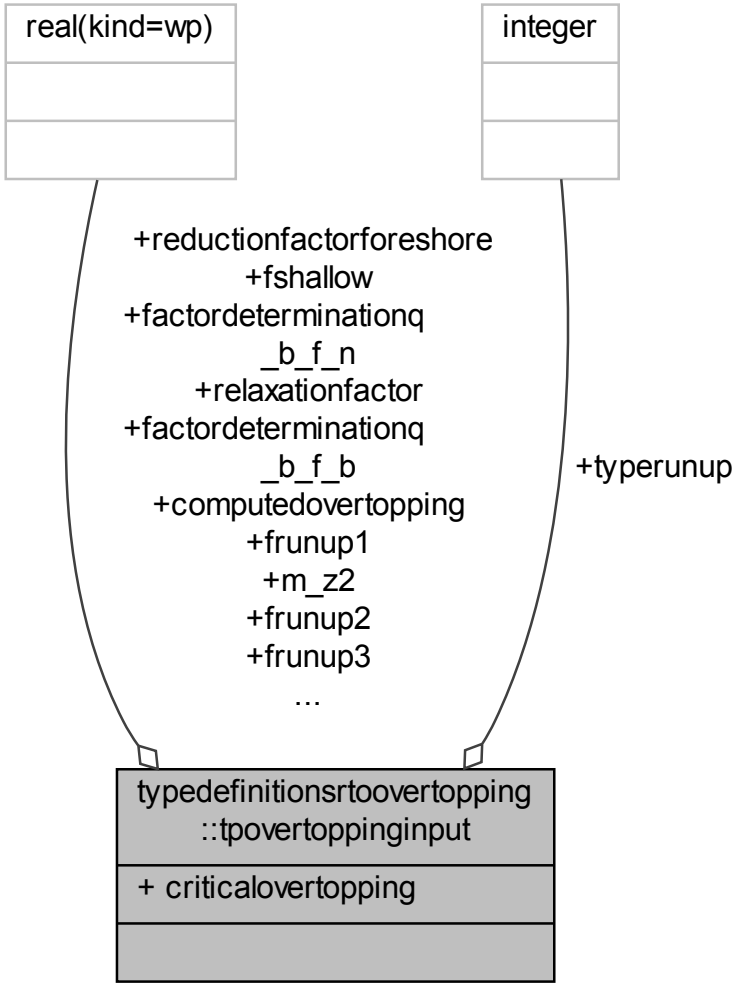
2% wave run-up (m)

Definition at line 57 of file typeDefinitionsRTOovertopping.f90.

8.17 typedefinitionsrtoovertopping::tpovertoppinginput Type Reference

OvertoppingModelFactors: C-structure with model factors.

Collaboration diagram for typedefinitionsrtoovertopping::tpovertoppinginput:



Public Attributes

- real(kind=wp) factordeterminationq_b_f_n

  model factor for non-breaking waves
- real(kind=wp) factordeterminationq_b_f_b

  model factor for breaking waves
- real(kind=wp) m_z2

  model factor describing the uncertainty of 2% runup height
- real(kind=wp) frunup1

  model factor 1 for wave run-up (for backwards compatability)
- real(kind=wp) frunup2

model factor 2 for wave run-up (idem)

- real(kind=wp) frunup3

  model factor 3 for wave run-up (idem)
- real(kind=wp) fshallow

  model factor for shallow waves
- real(kind=wp) computedovertopping

  model factor computed overtopping
- real(kind=wp) criticalovertopping

  model factor critical overtopping
- integer typerunup

  0: fRunup1, 2 and 3 are given; 1: m_z2 is given
- real(kind=wp) relaxationfactor

  relaxation factor iteration procedure wave runup
- real(kind=wp) reductionfactorforeshore = 0.5_wp

  reduction factor foreshore

8.17.1 Detailed Description

OvertoppingModelFactors: C-structure with model factors.

Definition at line 40 of file typeDefinitionsRTOovertopping.f90.

8.17.2 Member Data Documentation

8.17.2.1 real(kind=wp) typedefinitionsrtoovertopping::tpovertoppinginput::computedovertopping

model factor computed overtopping

Definition at line 48 of file typeDefinitionsRTOovertopping.f90.

8.17.2.2 real(kind=wp) typedefinitionsrtoovertopping::tpovertoppinginput::criticalovertopping

model factor critical overtopping

Definition at line 49 of file typeDefinitionsRTOovertopping.f90.

8.17.2.3 real(kind=wp) typedefinitionsrtoovertopping::tpovertoppinginput::factordeterminationq_b_f_b

model factor for breaking waves

Definition at line 42 of file typeDefinitionsRTOovertopping.f90.

8.17.2.4 real(kind=wp) typedefinitionsrtoovertopping::tpovertoppinginput::factordeterminationq_b_f_n

model factor for non-breaking waves

Definition at line 41 of file typeDefinitionsRTOovertopping.f90.

8.17.2.5 real(kind=wp) typedefinitionsrtoovertopping::tpovertoppinginput::frunup1

model factor 1 for wave run-up (for backwards compatability)

Definition at line 44 of file typeDefinitionsRTOovertopping.f90.

8.17.2.6 real(kind=wp) typedefinitionsrtoovertopping::tpovertoppinginput::frunup2

model factor 2 for wave run-up (idem)

Definition at line 45 of file typeDefinitionsRTOovertopping.f90.

8.17.2.7 real(kind=wp) typedefinitionsrtoovertopping::tpovertoppinginput::frunup3

model factor 3 for wave run-up (idem)

Definition at line 46 of file typeDefinitionsRTOovertopping.f90.

8.17.2.8 real(kind=wp) typedefinitionsrtoovertopping::tpovertoppinginput::fshallow

model factor for shallow waves

Definition at line 47 of file typeDefinitionsRTOovertopping.f90.

8.17.2.9 real(kind=wp) typedefinitionsrtoovertopping::tpovertoppinginput::m_z2

model factor describing the uncertainty of 2% runup height

Definition at line 43 of file typeDefinitionsRTOovertopping.f90.

8.17.2.10 real(kind=wp) typedefinitionsrtoovertopping::tpovertoppinginput::reductionfactorforeshore = 0.5_wp

reduction factor foreshore

Definition at line 52 of file typeDefinitionsRTOovertopping.f90.

8.17.2.11 real(kind=wp) typedefinitionsrtoovertopping::tpovertoppinginput::relaxationfactor

relaxation factor iteration procedure wave runup

Definition at line 51 of file typeDefinitionsRTOovertopping.f90.

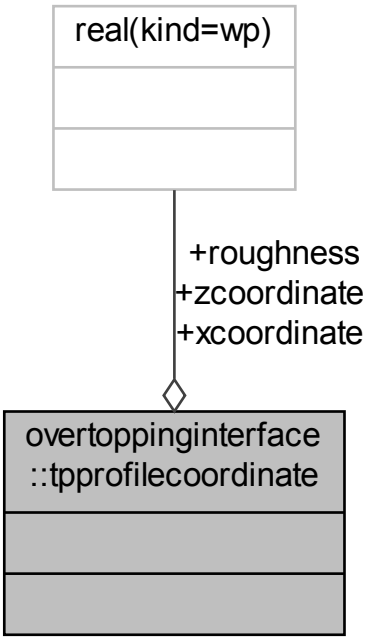
8.17.2.12 integer typedefinitionsrtoovertopping::tpovertoppinginput::typerunup

0: fRunup1, 2 and 3 are given; 1: m_z2 is given

Definition at line 50 of file typeDefinitionsRTOovertopping.f90.

8.18 overtoppinginterface::tpprofilecoordinate Type Reference

Collaboration diagram for overtoppinginterface::tpprofilecoordinate:



Public Attributes

- real(kind=wp) xcoordinate

  X-coordinate foreland profile.
- real(kind=wp) zcoordinate

  Z-coordinate foreland profile.
- real(kind=wp) roughness

  Roughness of the area between two points.

8.18.1 Detailed Description

Definition at line 19 of file overtoppingInterface.f90.

8.18.2 Member Data Documentation

8.18.2.1 real(kind=wp) overtoppinginterface::tpprofilecoordinate::roughness

Roughness of the area between two points.

Definition at line 22 of file overtoppingInterface.f90.

8.18.2.2 real(kind=wp) overtoppinginterface::tpprofilecoordinate::xcoordinate

X-coordinate foreland profile.

Definition at line 20 of file overtoppingInterface.f90.

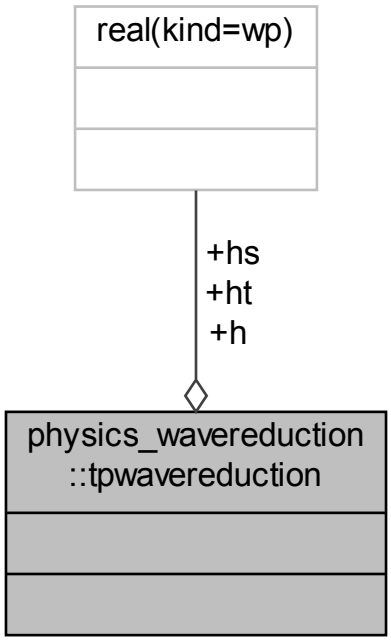
8.18.2.3 real(kind=wp) overtoppinginterface::tpprofilecoordinate::zcoordinate

Z-coordinate foreland profile.

Definition at line 21 of file overtoppingInterface.f90.

8.19 physics_wavereduction::tpwavereduction Type Reference

Collaboration diagram for physics_wavereduction::tpwavereduction:



Public Attributes

- real(kind=wp) hs

  wave height

- real(kind=wp) h

  local waterlevel

- real(kind=wp) ht

  level (height) of the toe of the dike

8.19.1 Detailed Description

Definition at line 22 of file waveReduction.f90.

8.19.2 Member Data Documentation

8.19.2.1 real (kind=wp) physics_wavereduction::tpwavereduction::h

local waterlevel

Definition at line 24 of file waveReduction.f90.

8.19.2.2 real (kind=wp) physics_wavereduction::tpwavereduction::hs

wave height

Definition at line 23 of file waveReduction.f90.

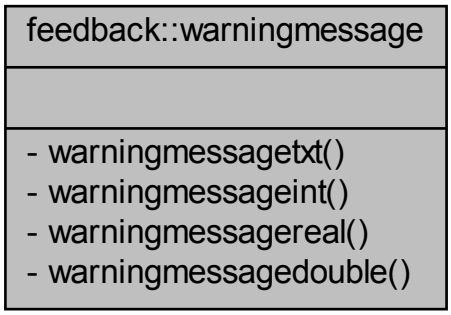
8.19.2.3 real (kind=wp) physics_wavereduction::tpwavereduction::ht

level (height) of the toe of the dike

Definition at line 25 of file waveReduction.f90.

8.20 feedback::warningmessage Interface Reference

Collaboration diagram for feedback::warningmessage:



Private Member Functions

- subroutine warningmessagetxt (msgtext)

  Write a warning message.
- subroutine warningmessageint (msgtext, value)

  Write a warning message with an integer value.
- subroutine warningmessagereal (msgtext, value)

Write a warning message with a real value.

- subroutine warningmessagedouble (msgtext, value)

  Write a warning message with a double precision real value.

8.20.1 Detailed Description

Definition at line 156 of file feedback.f90.

8.20.2 Member Function/Subroutine Documentation

8.20.2.1 subroutine feedback::warningmessage::warningmessagedouble (character(len=∗), intent(in) *msgtext,* real(kind=dp), intent(in) *value*) [private]

Write a warning message with a double precision real value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Real (double) value to add

Definition at line 729 of file feedback.f90.

8.20.2.2 subroutine feedback::warningmessage::warningmessageint (character(len=∗), intent(in) *msgtext,* integer, intent(in) *value*) [private]

Write a warning message with an integer value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Integer value to add

Definition at line 699 of file feedback.f90.

8.20.2.3 subroutine feedback::warningmessage::warningmessagereal (character(len=∗), intent(in) *msgtext,* real, intent(in) *value*) [private]

Write a warning message with a real value.

Parameters

in	*msgtext*	Error message to write
in	*value*	Real value to add

Definition at line 714 of file feedback.f90.

8.20.2.4 subroutine feedback::warningmessage::warningmessagetxt (character(len=∗), intent(in) *msgtext*) [private]

Write a warning message.

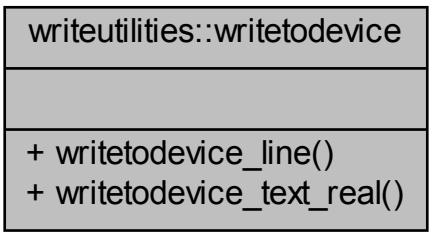
Parameters

in	*msgtext*	Error message to write

Definition at line 676 of file feedback.f90.

8.21 writeutilities::writetodevice Interface Reference

Collaboration diagram for writeutilities::writetodevice:



Public Member Functions

- subroutine writetodevice_line (fileId, line)

  Write to device.
- subroutine writetodevice_text_real (fileId, formatText, textValue, realValue)

  Write to device.

8.21.1 Detailed Description

Definition at line 14 of file writeUtilities.f90.

8.21.2 Member Function/Subroutine Documentation

8.21.2.1 subroutine writeutilities::writetodevice::writetodevice_line (integer, intent(in) *fileId,* character(len=∗), intent(in) *line*)

Write to device.

Parameters

in	*fileid*	device number to write to, 0 = screen
in	*line*	text to write

Definition at line 28 of file writeUtilities.f90.

8.21.2.2 subroutine writeutilities::writetodevice::writetodevice_text_real (integer, intent(in) *fileId,* character(len=∗), intent(in) *formatText,* character(len=∗), intent(in) *textValue,* real (kind=wp), intent(in) *realValue*)

Write to device.

Parameters

in	*fileid*	device number to write to, 0 = screen

in	*formattext*	format specifier
in	*textvalue*	text to write
in	*realvalue*	real value to print

Definition at line 42 of file writeUtilities.f90.

Chapter 9

File Documentation

9.1 angleUtilities.f90 File Reference

Implement various utility functions dealing with directions.

Modules

- module angleutilities

  Module with direction utility functions.

Functions/Subroutines

- real(kind=wp) function angleutilities::anglebetween2directions (direction1, direction2)

  Function angleBetween2Directions() for the computation of the angle between two directions.
  The two directions must be given in degrees and within [0, 360]
  The computed angle between these two directions is also in degrees, and within [0, 180]
  .

9.1.1 Detailed Description

Implement various utility functions dealing with directions.

9.2 breakwater.f90 File Reference

Breakwater formulas.

Modules

- module physics_breakwater

  Module with routines for breakwater.

Functions/Subroutines

- subroutine, public physics_breakwater::calculatebreakwater (breakwaterType, heightBreakwater, waterLevel, Hs)

  Function to transform the wave height for the influence of an occupant breakwater.

9.2.1 Detailed Description

Breakwater formulas.

9.3 breakwaterEnumerations.f90 File Reference

Module for enumerating breakwater types.

Modules

- module physics_breakwatertypesenumerations

  Enumeration for breakwater types WHEN CHANGING THESE VALUES. ALSO CHANGE THEM IN HydraRing↩ Enumerations.CS.

Variables

- integer, parameter physics_breakwatertypesenumerations::breakwaternotpresent = 0

  no breakwater
- integer, parameter physics_breakwatertypesenumerations::breakwatercaisson = 1

  breakwater type is caisson
- integer, parameter physics_breakwatertypesenumerations::breakwaterverticalwall = 2

  breakwater type is vertical wall
- integer, parameter physics_breakwatertypesenumerations::breakwaterrubblemound = 3

  breakwater type is rubble mound 1:1.5

9.3.1 Detailed Description

Module for enumerating breakwater types.

9.4 bretschneider.f90 File Reference

Compute wave parameters using Bretschneider formulas.

Data Types

- type physics_bretschneider::tpbretschneider

Modules

- module physics_bretschneider

  Module with routines for bretschneider.

Functions/Subroutines

- subroutine, public physics_bretschneider::calculatewavebretschneider (bretschneider, Hs, Ts)

  Function to compute wave parameters according to Bretschneider formula: wave height (Hs) wave period (Ts)

9.4.1 Detailed Description

Compute wave parameters using Bretschneider formulas.

9.5 conversionFunctions.f90 File Reference

Modules

- module conversionfunctions

Functions/Subroutines

- subroutine, public conversionfunctions::pqfrombeta (beta, p, q)

  Subroutine for computing the exceedance and non-exceedance probabilities for a given reliability index (beta) assuming a standard normal distribution.
- real(kind=wp) function, public conversionfunctions::pfrombeta (beta)

  Function for computing the non-exceedance probability (P) for a given reliability index (beta), assuming a standard normal distribution.

  .
- real(kind=wp) function, public conversionfunctions::qfrombeta (beta)

  Function for computing the exceedance probability (Q) for a given reliability index (beta), assuming a standard normal distribution.

  .
- subroutine, public conversionfunctions::returntimefrombeta (beta, returnTime)

  Subroutine for computing the return time for a given reliability index (beta) assuming a standard normal distribution.
- subroutine, public conversionfunctions::freqfrombeta (beta, freq)

  Subroutine for computing the frequency for a given reliability index (beta) assuming a standard normal distribution.
- subroutine, public conversionfunctions::logqfrombeta (beta, logQ)

  Subroutine for computing the (negative) logarithm of the non-exceedance probability for a given reliability index (beta) assuming a standard normal distribution.
- subroutine, public conversionfunctions::betafromq (q, beta)

  Subroutine for computing the reliability index (beta) for a given exceedance probability. The reliability index is corrected with the subroutine PQfromBeta.

Variables

- real(kind=wp), parameter, private conversionfunctions::qmin = 1.0d-300
- real(kind=wp), parameter, private conversionfunctions::upperlog = 700.0d0
- real(kind=wp), parameter, private conversionfunctions::ulimit = 5.6d0

9.6 dllOvertopping.f90 File Reference

Main entry for the dll DikesOvertopping FUNCTIONS/SUBROUTINES exported from dllOvertopping.dll:

Modules

- module dllovertopping

  Calculate one type of overtopping.

Functions/Subroutines

- subroutine, public dllovertopping::calculateqo (load, geometryInput, dikeHeight, modelFactors, overtopping, success, errorText, verbosity, logFile)

  Subroutine that calculates the discharge needed for the Z-function DikesOvertopping Wrapper for calculateQoF: convert C-like input structures to Fortran input structures.

- subroutine, public dllovertopping::calculateqof (load, geometryF, dikeHeight, modelFactors, overtopping, success, errorText, logging)

  Subroutine that calculates the discharge needed for the Z-function DikesOvertopping.

- subroutine, public dllovertopping::calczvalue (criticalOvertoppingRate, modelFactors, Qo, z, success, errorMessage)

  Subroutine that calculates the Z-function DikesOvertopping based on the discharge calculated with calculateQoF.

- subroutine, public dllovertopping::validateinputc (geometryInput, dikeHeight, modelFactors, success, errorText)

  Subroutine that validates the geometry Wrapper for ValidateInputF: convert C-like input structures to Fortran input structures.

- subroutine, public dllovertopping::validateinputf (geometryF, dikeHeight, modelFactors, success, errorText)

  Subroutine that validates the geometry.

- subroutine, public dllovertopping::versionnumber (version)

  Subroutine that delivers the version number.

- type(overtoppinggeometrytypef) function dllovertopping::geometry_c_f (geometryInput)

  Private subroutine that converts geometry from c-pointer to fortran struct.

9.6.1 Detailed Description

Main entry for the dll DikesOvertopping FUNCTIONS/SUBROUTINES exported from dllOvertopping.dll:

- zFuncOvertopping
- calculateQo
- calculateQoF
- ValidateInputC
- ValidateInputF
- versionNumber

9.7 equalReals.f90 File Reference

This file contains a module with functions to test quality of reals.

Modules

- module equalreals

  Module with functions to test quality of reals.

Functions/Subroutines

- logical function equalreals::equalrealsrelative (x, y, margin)

  *equalRealsRelative Function to test for equality of two reals with a relative value for the margin equalRealsRelative = |x-y| <= margin * (|x|+|y|)/2*
- logical function equalreals::equalrealsabsolute (x, y, margin)

  equalRealsAbsolute Function to test for equality of two reals with an absolute value for the margin equalRealsAbsolute = |x-y| <= margin
- logical function equalreals::equalrealsinvector (n, vector, margin)

  Function to test for equality of reals in a vector with a relative value of the margin.
- logical function equalreals::equalvectorswithreals (n, vector1, vector2, margin, equality)

  Function to test for equality of two vectors with reals Two vectors are equal if all elements in the vector are equal.
- logical function equalreals::equaltableswithreals (matrix1, matrix2, margin)

  Function to test for equality of two vectors with reals Two vectors are equal if all elements in the vector are equal.
- logical function equalreals::equalvectorswithrealsweightedmargin (n, vector1, vector2, margin, equality)

  Function to test for equality of two vectors with reals. Two vectors are equal if all elements in the vector are almost equal Here, margin is weighted according to tested numbers.
- logical function equalreals::equalvectorswithintegers (n, vector1, vector2, equality)

9.7.1 Detailed Description

This file contains a module with functions to test quality of reals.

9.8 factorModuleRTOovertopping.f90 File Reference

This file contains a module with functions for the slope angle and influence factors.

Modules

- module factormodulertoovertopping

Functions/Subroutines

- subroutine, public factormodulertoovertopping::calculatetanalpha (h, Hm0, z2, geometry, tanAlpha, succes, errorMessage)

  calculateTanAlpha representative slope angle
- subroutine, public factormodulertoovertopping::calculategammabeta (Hm0, Tm_10, beta, gammaBeta_z, gammaBeta_o)

  calculateGammaBeta influence factor angle of wave attack
- subroutine, public factormodulertoovertopping::calculategammaf (h, ksi0, ksi0Limit, gammaB, z2, geometry, gammaF, succes, errorMessage)

  calculateGammaF influence factor roughness
- subroutine, public factormodulertoovertopping::calculategammab (h, Hm0, z2, geometry, NbermSegments, gammaB, succes, errorMessage)

  calculateGammaB influence factor berms

9.8.1 Detailed Description

This file contains a module with functions for the slope angle and influence factors.

9.9 feedback.f90 File Reference

Feedback library: error messaging and progress reporting.

Data Types

- interface feedback::feedbackcontext
- interface feedback::errormessage
- interface feedback::warningmessage
- interface feedback::fatalerror
- interface feedback::progressreport

Modules

- module feedback

Functions/Subroutines

- integer function feedback::p ()

  Returns the pointer to the feedback dll, avoiding many loadlibrary calls.
- logical function, public feedback::todll ()

  Is feedback writing to the LogMessage function in HydraRing.IO.Native.dll ?
- logical function, public feedback::fatalerroroccured (value)

  Has a fatal error occured? Mainly used in unit tests.
- subroutine, public feedback::resetfatalerroroccured ()

  Reset fatal error occurance Mainly used in unit tests.
- subroutine feedback::setfatalerroroccured (value)

  Set fatal error occurance Mainly used in unit tests.
- logical function, public feedback::getfatalerrorexpected (value)

  Get value of FatalErrorExpected Mainly used in unit tests.
- subroutine, public feedback::setfatalerrorexpected (value)

  Set the value of FatalErrorExpected Mainly used in unit tests.
- logical function, public feedback::allow_log ()

  Is logging active?
- subroutine, public feedback::setallowlogging (value)

  Set logging on/off.
- subroutine, public feedback::getthreadid (threadId)

  Get the tread ID Note that multi threading is not implemented, so always the value 0 is returned.
- subroutine, public feedback::setfatalerroraction (action)

  Set the value of FatalErrorAction possible values: onfatalerrorStop = 1 : Action on fatal error: exit the program onfatalerrorThrowException = 2 : Action on fatal error: throw exception (for use as dll) onfatalerrorUnittest = 3 : Action on fatal error: handle as unit test.
- subroutine, public feedback::getfatalerrormessage (message)

  Get the last of FatalErrorMessage.
- subroutine, public feedback::feedbackerror (error)

  Return the error status of the current thread If an error has been reported (via the errorMessage routines), this routine will return true, otherwise false.
- subroutine, public feedback::feedbackclose

  Close the log file.
- subroutine, public feedback::feedbackinitialise (output, logfile, verbosity)

  Initialise the feedback subsystem.

- subroutine feedback::progressreporttxt (level, msgtext)

  Write a single message to the output for the benefit of the user.
- subroutine feedback::progressreportint (level, msgtext, value)

  Write a single message with an integer value.
- subroutine feedback::progressreportreal (level, msgtext, value)

  Write a single message with a real value.
- subroutine feedback::progressreportdouble (level, msgtext, value)

  Write a single message with a double precision real value.
- subroutine feedback::progressreportarray (level, msgText, x, size)

  Print array values to report file.
- subroutine feedback::progressreportvalueandarray (level, msgText, z, x, size)

  Print z-value and array values to report file.
- subroutine, public feedback::feedbackenter (routine)

  Register a new routine level.
- subroutine, public feedback::feedbackleave

  Drop a routine level from the stack trace.
- subroutine, public feedback::feedbackprintstack

  Print the stack trace as known via feedbackEnter and feedbackLeave.
- subroutine feedback::errormessagetxt (msgtext)

  Write an error message.
- subroutine feedback::errormessageint (msgtext, value)

  Write an error message with an integer value.
- subroutine feedback::errormessagereal (msgtext, value)

  Write an error message with a real value.
- subroutine feedback::errormessagedouble (msgtext, value)

  Write an error message with a double precision real value.
- subroutine feedback::warningmessagetxt (msgtext)

  Write a warning message.
- subroutine feedback::warningmessageint (msgtext, value)

  Write a warning message with an integer value.
- subroutine feedback::warningmessagereal (msgtext, value)

  Write a warning message with a real value.
- subroutine feedback::warningmessagedouble (msgtext, value)

  Write a warning message with a double precision real value.
- subroutine feedback::fatalerrortxt (msgtext)

  Write a fatal error message.
- subroutine feedback::fatalerrorint (msgtext, value)

  Write a fatal error message with an integer value.
- subroutine feedback::fatalerrorreal (msgtext, value)

  Write a fatal error message with a real value.
- subroutine feedback::fatalerrordouble (msgtext, value)

  Write a fatal error message with a double precision real value.
- subroutine, public feedback::feedbackdropcontext

  Drop the entire context.
- subroutine, public feedback::feedbackprintcontext

  Register a context for error messages.
- subroutine feedback::feedbackcontexttxt (text)

  Register a context for error messages.
- subroutine feedback::feedbackcontextint (text, value)

  Register a context for error messages - with integer value.
- subroutine feedback::feedbackcontextreal (text, value)

Register a context for error messages - with real value.

- subroutine feedback::feedbackcontextdouble (text, value)

  Register a context for error messages - with double precision real value.

- subroutine, public feedback::throw (msgText)

  Stop execution by throwing an exception Useful if running in a dll and main program is a GUI in e.g. C++ or C#.

Variables

- integer, parameter feedback::id = 0

  Thread ID; multi threading is not supported yet!

- integer, parameter feedback::sp = kind(1.0)
- integer, parameter feedback::dp = kind(1.0d0)
- integer, parameter feedback::wp = kind(1.0d0)

9.9.1 Detailed Description

Feedback library: error messaging and progress reporting.

9.10 feedback_parameters.f90 File Reference

Feedback parameters: parameters needed bij feedback and feedback_dll.

Modules

- module feedback_parameters

Variables

- integer, parameter feedback_parameters::onfatalerrorstop = 1

  Action on fatal error: exit the program.

- integer, parameter feedback_parameters::onfatalerrorthrowexception = 2

  Action on fatal error: throw exception (for use as dll)

- integer, parameter feedback_parameters::onfatalerrorunittest = 3

  Action on fatal error: handle as unit test.

- integer, parameter feedback_parameters::onlyscreen = 0

  Mode: write to screen only.

- integer, parameter feedback_parameters::onlyfile = 1

  Mode: write to file only.

- integer, parameter feedback_parameters::screenandfile = 2

  Mode: write to both screen and file.

- integer, parameter feedback_parameters::dll = 3

  Mode: write to dll.

- integer, parameter feedback_parameters::verbosenone = -1

  Reporting: NO messages.

- integer, parameter feedback_parameters::verbosebasic = 0

  Reporting: basic messages only.

- integer, parameter feedback_parameters::verbosedetailed = 1

  Reporting: detailed messages (including traceback)

- integer, parameter feedback_parameters::verbosedebugging = 2

  Reporting: all messages.

9.10.1 Detailed Description

Feedback parameters: parameters needed bij feedback and feedback_dll.

9.11 feedbackDLL.f90 File Reference

Feedback dll: error messaging and progress reporting.

Data Types

- type feedbackdll::feedbackdata

Modules

- module feedbackdll

Functions/Subroutines

- subroutine feedbackdll::initialise (output, logfile, verbosity)

  Initialise the feedback dll.
- subroutine feedbackdll::printmessage (msgtext)

  Write a message to output device (as selected in Initialise)
- subroutine feedbackdll::outputtodll (active)

  Is output is to another dll?
- subroutine feedbackdll::getallowlogging (value)

  Is logging active.
- subroutine feedbackdll::setallowlogging (value)

  Set logging on/off.
- subroutine feedbackdll::getverbosity (level)

  Get the current verbosity level.
- subroutine feedbackdll::setonfatalerroraction (action)

  Set the OnFatalErrorAction.
- subroutine feedbackdll::getonfatalerroraction (onfatalerrorAction)

  Get the current OnFatalErrorAction.
- subroutine feedbackdll::getfeedbackerror (error)

  Get the error status (true/false)
- subroutine feedbackdll::setfatalerrormessage (message)

  Set the errormessage.
- subroutine feedbackdll::getfatalerrormsg (message)

  Get the last error message.
- subroutine feedbackdll::getverbositylevel (verbosityLevel)

  Get the verbosity level.
- subroutine feedbackdll::pushroutine (routine, success)

  Add a routine name to the call stack.
- subroutine feedbackdll::poproutine (success)

  Removes and get a routine name from the call stack.
- subroutine feedbackdll::printstack

  Print the call stack.
- subroutine feedbackdll::getstacklevel (level)

Get the number of routines in the call stack.

- subroutine feedbackdll::closefile

  Closes the output file.

- subroutine feedbackdll::getunitnumber (lun)

  Get the unit number of the logfile.

- subroutine feedbackdll::dropcontext

  Reset the context stack.

- subroutine feedbackdll::printcontext

  Print the context stack.

- subroutine feedbackdll::contexttxt (text, success)

  Add a line of text to the context stack.

- subroutine feedbackdll::getsetfatalerrorexpected (func, value)

  Get or Set the FatalErrorExpected if func=get: get the value if func=set: set the value.

- subroutine feedbackdll::getsetfatalerroroccured (func, value)

  Get or Set the FatalErrorOccured if func=get: get the value if func=set: set the value.

- subroutine feedbackdll::writetodll (msgText)

  Write message to another dll.

Variables

- type(feedbackdata) feedbackdll::fbdata

  Array holding the actual information per thread (multiple threads not yet implemented)

- integer feedbackdll::fatalerroraction = 1
- character(len=255) feedbackdll::fatalerrormessage
- integer, parameter feedbackdll::id = 0
- logical feedbackdll::allow_logging = .true.
- logical feedbackdll::isfatalerrorexpected = .false.

  if isFatalErrorExpected = .true. then log error message in fatalErrorMsgtext and set boolean isFatalErrorOccured to true if isFatalErrorExpected = .false. then same action as onfatalerrorStop

- logical feedbackdll::isfatalerroroccured = .false.

9.11.1 Detailed Description

Feedback dll: error messaging and progress reporting.

To be used in combination with the feedback module. feedback is the (fortran) interface to the programmer. This module make it possible to write to the same file from different dll's/exe's.

By putting all data and print statements in this dll, and let feedback initialise this dll and let feedback only print using this dll, all output ends up at the right location.

9.12 fileUtilities.f90 File Reference

Module for reading information from files.

Modules

- module fileutilities

Functions/Subroutines

- subroutine fileutilities::readtable (filename, nColumns, table)
- subroutine fileutilities::writetablevalues (filename, table)
- subroutine fileutilities::removefile (filename)

  Routine to throw away a file.

9.12.1 Detailed Description

Module for reading information from files.

9.13 foreland.f90 File Reference

Modules

- module physics_foreland

  wrapper for foreland dll !!

Functions/Subroutines

- subroutine, public physics_foreland::physics_calculateforeland (retval, hDamType, hDamHeight, hAlpha, hFc, hInvalid, hDimHm0, hHm0, hTp, hWlev, hIncomingWaveAngle, hDikeNormal, hDimX, hX, hMinStepSize, hBottomLevel, hRatioDepth, hLogging, hLoggingFileName, hHm0Dike, hTpDike, hRefractedWaveAngleDike, hMessage)

  wrapper for foreland calculation in DynamicLib-DaF.dll

Variables

- integer, parameter physics_foreland::message_length =1000

9.14 formulaModuleRTOovertopping.f90 File Reference

This file contains a module with the core computations for Dikes Overtopping.

Modules

- module formulamodulertoovertopping

Functions/Subroutines

- subroutine, public formulamodulertoovertopping::calculatewaverunup (Hm0, ksi0, ksi0Limit, gammaB, gammaF, gammaBeta, modelFactors, z2, succes, errorMessage)

  calculateWaveRunup: calculate wave runup
- subroutine, public formulamodulertoovertopping::calculatewaveovertoppingdischarge (h, Hm0, tanAlpha, gammaB, gammaF, gammaBeta, ksi0, hCrest, modelFactors, Qo, succes, errorMessage)

  calculateWaveOvertoppingDischarge: calculate the wave overtopping discharge
- subroutine, public formulamodulertoovertopping::calculatewavelength (Tm_10, L0)

  calculateWaveLength: calculate the wave length

- subroutine, public formulamodulertoovertopping::calculatewavesteepness (Hm0, Tm_10, s0, succes, error↩ Message)

  calculateWaveSteepness: calculate the wave steepness
- subroutine, public formulamodulertoovertopping::calculatebreakerparameter (tanAlpha, s0, ksi0, succes, errorMessage)

  calculateBreakerParameter: calculate the breaker parameter
- subroutine, public formulamodulertoovertopping::calculateanglewaveattack (phi, psi, beta)

  calculateAngleWaveAttack: calculate the angle of wave attack
- subroutine, public formulamodulertoovertopping::calculatebreakerlimit (modelFactors, gammaB, ksi0Limit, succes, errorMessage)

  calculateBreakerLimit: calculate the breaker limit
- subroutine, public formulamodulertoovertopping::adjustinfluencefactors (gammaB, gammaF, gammaBeta, gammaBetaType, ksi0, ksi0Limit, succes, errorMessage)

  adjustInfluenceFactors: adjust the influence factors
- subroutine, public formulamodulertoovertopping::realrootscubicfunction (a, b, c, d, N, x, succes, error↩ Message)

  realRootsCubicFunction: calculate the roots of a cubic function
- subroutine, public formulamodulertoovertopping::rootsgeneralcubic (a, b, c, d, z, succes, errorMessage)

  rootsGeneralCubic: calculate the roots of a generic cubic function
- subroutine, public formulamodulertoovertopping::rootsdepressedcubic (p, q, z)

  rootsDepressedCubic: calculate the roots of a depressed cubic function
- subroutine, public formulamodulertoovertopping::cubicroots (z, roots)

  cubicRoots: calculate the roots of a cubic function
- logical function, public formulamodulertoovertopping::isequalreal (x1, x2)

  isEqualReal: are two reals (almost) equal
- logical function, public formulamodulertoovertopping::isequalzero (x)

  isEqualZero: is a real (almost) zero

9.14.1 Detailed Description

This file contains a module with the core computations for Dikes Overtopping.

9.15 ftnunit.f90 File Reference

Data Types

- interface ftnunit::proc
- interface ftnunit::assert_equal
- interface ftnunit::assert_comparable
- interface ftnunit::assert_inbetween

Modules

- module ftnunit

Functions/Subroutines

- subroutine, public ftnunit::setruntestlevel (runtTestLevel)

  Set the run test level 0 = fast tests 1.. 2.. 3 = slow tests.
- subroutine, public ftnunit::settesttitle (message)

  Set the title that is displayed in the report of the test run.
- subroutine, public ftnunit::test (proc, text, ignore)
- subroutine, public ftnunit::testwithlevel (proc, text, testlevel, ignore)
- subroutine, public ftnunit::runtests_init
- subroutine ftnunit::runtests_init_priv
- subroutine, public ftnunit::runtests_final (stop)
- logical function ftnunit::issilent ()
- subroutine ftnunit::setworkingdir
- subroutine, public ftnunit::runtests (testproc)
- subroutine ftnunit::expect_program_stop
- subroutine, public ftnunit::assert_true (cond, text)
- subroutine, public ftnunit::assert_false (cond, text)
- subroutine ftnunit::assert_equal_logical (value1, value2, text)
- subroutine ftnunit::assert_equal_logical1d (array1, array2, text)
- subroutine ftnunit::assert_equal_string (value1, value2, text)
- subroutine ftnunit::assert_equal_string1d (array1, array2, text)
- subroutine ftnunit::assert_equal_int (value1, value2, text)
- subroutine ftnunit::assert_equal_int1d (array1, array2, text)
- subroutine ftnunit::assert_comparable_real (value1, value2, margin, text)
- subroutine ftnunit::assert_comparable_real1d (array1, array2, margin, text)
- subroutine ftnunit::assert_comparable_real2d (array1, array2, margin, text)
- subroutine ftnunit::assert_comparable_double (value1, value2, margin, text)
- subroutine ftnunit::assert_comparable_double1d (array1, array2, margin, text)
- subroutine ftnunit::assert_comparable_double2d (array1, array2, margin, text)
- subroutine ftnunit::assert_inbetween_real (value, vmin, vmax, text)
- subroutine ftnunit::assert_inbetween_double (value, vmin, vmax, text)
- subroutine, public ftnunit::assert_files_comparable (filename1, filename2, text, tolerance)
- subroutine write_header
- logical function ftnunit::ftnunit_file_exists (filename)
- subroutine ftnunit::ftnunit_remove_file (filename)
- subroutine ftnunit::ftnunit_make_empty_file (filename)
- subroutine ftnunit::ftnunit_write_html_header
- subroutine ftnunit::ftnunit_write_html_footer
- subroutine ftnunit::colour_number (number, coloured, colour1, colour2, allowed)
- subroutine ftnunit::ftnunit_write_html_test_begin (text)
- subroutine ftnunit::ftnunit_write_html_ignored (lun)
- subroutine ftnunit::ftnunit_write_html_cpu (cpu)
- subroutine ftnunit::ftnunit_write_html_previous_failed
- subroutine ftnunit::ftnunit_write_html_previous_stopped
- subroutine ftnunit::ftnunit_write_html_close_row (lun)
- subroutine ftnunit::ftnunit_write_html_failed_logic (text, expected)
- subroutine ftnunit::ftnunit_write_html_failed_equivalent (text)
- subroutine ftnunit::ftnunit_write_html_failed_equivalent1d (text, idx, value1, value2, addtext)
- subroutine ftnunit::ftnunit_write_html_failed_string (text, value1, value2)
- subroutine ftnunit::ftnunit_write_html_failed_string1d (text, idx, value1, value2, addtext)
- subroutine ftnunit::ftnunit_write_html_failed_int (text, value1, value2)
- subroutine ftnunit::ftnunit_write_html_failed_int1d (text, idx, value1, value2, addtext)
- subroutine ftnunit::ftnunit_write_html_failed_real (text, value1, value2)
- subroutine ftnunit::ftnunit_write_html_failed_real1d (text, idx, value1, value2, addtext)

- subroutine ftnunit::ftnunit_write_html_failed_real2d (text, idx1, idx2, value1, value2, addtext)
- subroutine ftnunit::ftnunit_write_html_failed_double (text, value1, value2)
- subroutine ftnunit::ftnunit_write_html_failed_double1d (text, idx, value1, value2, addtext)
- subroutine ftnunit::ftnunit_write_html_failed_double2d (text, idx1, idx2, value1, value2, addtext)
- subroutine ftnunit::ftnunit_write_html_failed_inbetween_real (text, value, vmin, vmax)
- subroutine ftnunit::ftnunit_write_html_failed_inbetween_double (text, value, vmin, vmax)
- subroutine ftnunit::ftnunit_write_html_failed_files (text, string1, string2, string3)

Variables

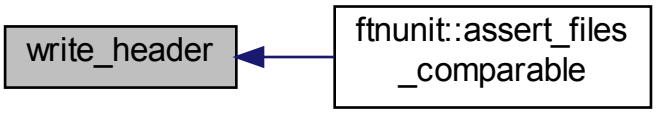
- procedure(proc), pointer, public ftnunit::subptrtestprograminit => null()
- procedure(proc), pointer, public ftnunit::subptrtestinit => null()
- integer, parameter ftnunit::mode_all = 0
- integer, parameter ftnunit::mode_single = 1
- integer, parameter ftnunit::mode_list = 2
- integer, parameter ftnunit::dp = kind(1.0d0)
- integer, save ftnunit::run_test_level = 99
- integer, save ftnunit::single_test = -1
- integer, save ftnunit::test_mode = mode_all
- integer, save ftnunit::last_test
- integer, save ftnunit::testno
- integer, save ftnunit::nofails
- integer, save ftnunit::nofails_prev
- integer, save ftnunit::notests_run
- integer, save ftnunit::notests_failed
- integer, save ftnunit::notests_ignored
- integer, save ftnunit::notests_work_in_progress
- integer, save ftnunit::noruns
- logical, save ftnunit::call_final = .true.
- logical, save ftnunit::previous = .false.
- integer, save ftnunit::failed_asserts = 0
- logical, save ftnunit::has_run = .false.
- logical, save ftnunit::ignore_test = .false.
- logical, save ftnunit::ignore_previous_test = .false.
- character(len=20), save ftnunit::html_file = 'ftnunit.html'
- character(len=80), save ftnunit::testname
- character(len=80), save ftnunit::testtitle = 'Result of unit tests'
- character(len=32), save ftnunit::ignore_reason
- character(len=32), save ftnunit::ignore_reason_previous_test = ' '
- character(len=*), parameter ftnunit::work_in_progress = 'work-in-progress'

9.15.1 Function/Subroutine Documentation

9.15.1.1 subroutine assert_files_comparable::write_header ( ) [private]

Definition at line 1407 of file ftnunit.f90.

Here is the caller graph for this function:



9.16 ftnunit_hooks.f90 File Reference

Modules

- module ftnunit_hooks

Functions/Subroutines

- subroutine ftnunit_hooks::ftnunit_hook_test_start (text)
- subroutine ftnunit_hooks::ftnunit_hook_test_stop (text)
- subroutine ftnunit_hooks::ftnunit_hook_test_assertion_failed (text, assert_text, failure_text)
- subroutine ftnunit_hooks::ftnunit_hook_test_completed

9.17 ftnunit_hooks_teamcity.f90 File Reference

Modules

- module ftnunit_hooks

Functions/Subroutines

- subroutine ftnunit_hooks::ftnunit_hook_test_start (text)
- subroutine ftnunit_hooks::ftnunit_hook_test_stop (text)
- subroutine ftnunit_hooks::ftnunit_hook_test_assertion_failed (text, assert_text, failure_text)
- subroutine ftnunit_hooks::ftnunit_hook_test_completed

9.18 general.f90 File Reference

Overall module for general library.

Modules

- module general

9.18.1 Detailed Description

Overall module for general library.

9.19 geometryModuleRTOovertopping.f90 File Reference

This file contains a module with the core computations for Dikes Overtopping related to the geometry.

Modules

- module geometrymodulertoovertopping

Functions/Subroutines

- subroutine, public geometrymodulertoovertopping::checkcrosssection (psi, nCoordinates, xCoordinates, y↩ Coordinates, roughnessFactors, succes, errorMessage)

  checkCrossSection: check cross section
- subroutine, public geometrymodulertoovertopping::initializegeometry (psi, nCoordinates, xCoordinates, y↩ Coordinates, roughnessFactors, geometry, succes, errorMessage)

  initializeGeometry: initialize the geometry
- subroutine, public geometrymodulertoovertopping::allocatevectorsgeometry (nCoordinates, geometry)

  allocateVectorsGeometry: allocate the geometry vectors
- subroutine, public geometrymodulertoovertopping::deallocategeometry (geometry)

  deallocateGeometry: deallocate the geometry vectors
- subroutine, public geometrymodulertoovertopping::calculatesegmentslopes (geometry, succes, error↩ Message)

  calculateSegmentSlopes: calculate the segment slopes
- subroutine, public geometrymodulertoovertopping::determinesegmenttypes (geometry)

  determineSegmentTypes: determine the segment types
- subroutine, public geometrymodulertoovertopping::copygeometry (geometry, geometryCopy)

  copyGeometry: copy a geometry structure
- subroutine, public geometrymodulertoovertopping::isequalgeometry (geometry1, geometry2, succes, error↩ Message)

  isEqualGeometry: are two geometries equal
- subroutine, public geometrymodulertoovertopping::mergesequentialberms (geometry, geometryMerged↩ Berms, succes, errorMessage)

  mergeSequentialBerms: merge sequential berms
- subroutine, public geometrymodulertoovertopping::adjustnonhorizontalberms (geometry, geometryFlat↩ Berms, succes, errorMessage)

  adjustNonHorizontalBerms: adjust non-horizontal berms
- subroutine, public geometrymodulertoovertopping::removeberms (geometry, geometryNoBerms, succes, errorMessage)

  removeBerms: remove berms
- subroutine, public geometrymodulertoovertopping::removedikesegments (geometry, index, geometry↩ Adjusted, succes, errorMessage)

  removeDikeSegments: remove dike segments
- subroutine, public geometrymodulertoovertopping::splitcrosssection (geometry, L0, NwideBerms, geometrysectionB, geometrysectionF, succes, errorMessage)

  splitCrossSection: split a cross section
- subroutine, public geometrymodulertoovertopping::calculatehorzlengths (geometry, yLower, yUpper, horz↩ Lengths, succes, errorMessage)

calculateHorzLengths: calculate horizontal lengths

- subroutine, public geometrymodulertoovertopping::calculatehorzdistance (geometry, yLower, yUpper, dx, succes, errorMessage)

  calculateHorzDistance: calculate horizontal distance
- subroutine, public geometrymodulertoovertopping::writecrosssection (geometry, geometryName)

  writeCrossSection: write a cross section

9.19.1 Detailed Description

This file contains a module with the core computations for Dikes Overtopping related to the geometry.

9.20 interpolationTriangular.f90 File Reference

Module containing the function for triangular interpolation.

Modules

- module interpolationtriangular

  Module with function for triangular interpolation.

Functions/Subroutines

- real(kind=wp) function interpolationtriangular::triangularinterpolation (x1, y1, h1, x2, y2, h2, x3, y3, h3, x, y)

  Function for for triangular interpolation Using three result values of three different stations on the Dutch RD grid a plane is spaned. The result values are in general water levels. For a fourth point on the Dutch RD grid the result value is calculated lying on this plane. In general a water level is calculated.
- real(kind=wp) function interpolationtriangular::datainterpolation (var1, var2, weight, iscyclic)

9.20.1 Detailed Description

Module containing the function for triangular interpolation.

9.21 mainModuleRTOovertopping.f90 File Reference

This file contains a module with the core computations for Dikes Overtopping.

Modules

- module mainmodulertoovertopping

Functions/Subroutines

- subroutine, public mainmodulertoovertopping::calculateovertopping (geometry, load, modelFactors, overtopping, succes, errorMessage)

  calculateOvertopping: calculate the overtopping
- subroutine, public mainmodulertoovertopping::calculateovertoppingsection (geometry, h, Hm0, Tm_10, L0, gammaBeta_z, gammaBeta_o, modelFactors, overtopping, succes, errorMessage)

  calculateOvertoppingSection: calculate the overtopping for a section

- subroutine, public mainmodulertoovertopping::calculatewaveovertopping (geometry, h, Hm0, Tm_10, z2, gammaBeta_o, modelFactors, Qo, succes, errorMessage)

  calculateWaveOvertopping: calculate wave overtopping
- subroutine mainmodulertoovertopping::calculateovertoppingnegativefreeboard (load, geometry, overtopping, succes, errorMessage)

  calculateOvertoppingNegativeFreeboard: calculate overtopping in case of negative freeboard
- subroutine, public mainmodulertoovertopping::interpolateresultssections (geometry, L0, NwideBerms, overtoppingB, overtoppingF, overtopping, succes, errorMessage)

  interpolateResultsSections: interpolate results for split cross sections
- subroutine, public mainmodulertoovertopping::checkinputdata (geometry, load, modelFactors, succes, errorMessage)

  checkInputdata: check the input data
- subroutine, public mainmodulertoovertopping::checkmodelfactors (modelFactors, succes, errorMessage)

  checkModelFactors: check the input data
- subroutine, public mainmodulertoovertopping::convertovertoppinginput (modelFactors, success, error↩ Message)

  convertOvertoppingInput: convert the model factors from C-like to Fortran

9.21.1 Detailed Description

This file contains a module with the core computations for Dikes Overtopping.

9.22 ModuleLogging.f90 File Reference

Module for steering the extra logging.

Data Types

- type modulelogging::tlogging

  TLogging: structure for steering the logging.

Modules

- module modulelogging

Variables

- integer, parameter modulelogging::maxfilenamelength = 256

  maximum length of filename
- type(tlogging) modulelogging::currentlogging

  copy of argument logging

9.22.1 Detailed Description

Module for steering the extra logging.

9.23 overtoppingInterface.f90 File Reference

This file contains the parameters and types (structs) as part of the interface to and from dllOvertopping.

Data Types

- type overtoppinginterface::tpprofilecoordinate
- type overtoppinginterface::overtoppinggeometrytype
- type overtoppinginterface::overtoppinggeometrytypef

Modules

- module overtoppinginterface

Variables

- integer, parameter, public overtoppinginterface::varmodelfactorcriticalovertopping = 8

  Model factor critical overtopping.

9.23.1 Detailed Description

This file contains the parameters and types (structs) as part of the interface to and from dllOvertopping.

9.24 physics.f90 File Reference

Overall module for physics library.

Modules

- module physics

9.24.1 Detailed Description

Overall module for physics library.

9.25 precision.f90 File Reference

Define KINDs for controlling the precision of the REAL variables and define parameters for mathematical constants.

Modules

- module precision

  Module with parameters for KINDs and mathematical constants.

Variables

- integer, parameter precision::sp = kind(1.0)

  Single precision.
- integer, parameter precision::dp = kind(1.0d0)

  Double precision.
- integer, parameter precision::wp = dp

Working precision.

- real(wp), parameter precision::almostzero = 1.0d-30

  Definition of zero for the machine.

- real(wp), parameter precision::almostinf = huge(1.0d0)

  Very large value as limit.

- real(wp), parameter precision::rholimit = 0.99999d0

  Limit value for the correlation coefficient DO NOT CHANGE THIS VALUE WITHOUT SUFFICIENT RESEARCH.

- real(wp), parameter precision::betalimit = 0.99999999d0

  Limit value for the beta ratio in combining DO NOT CHANGE ONLY USED FOR EXTRA WIND STOCHAST.

- real(wp), parameter precision::pi = 3.14159265358979323846264338327950288419 7_wp

  Circle constant.

- real(wp), parameter precision::emconstant = 0.57721566490153286060651209008240243104215933593992↩ _wp

  Euler Mascheroni constant.

- real(wp), parameter precision::gravityconstant = 9.80665_wp

  Gravity constant for the Netherlands.

- integer, parameter precision::shortmax = 32
- integer, parameter precision::longmax = 255

9.25.1 Detailed Description

Define KINDs for controlling the precision of the REAL variables and define parameters for mathematical constants.

Note: Use the parameter "wp" to set the working precision for using routines.

9.26 typeDefinitionsRTOovertopping.f90 File Reference

This file contains a module with the type definitions for Dikes Overtopping.

Data Types

- type typedefinitionsrtoovertopping::tpgeometry

  tpGeometry: structure with geometry data

- type typedefinitionsrtoovertopping::tpload

  tpLoad: structure with load parameters

- type typedefinitionsrtoovertopping::tpovertoppinginput

  OvertoppingModelFactors: C-structure with model factors.

- type typedefinitionsrtoovertopping::tpovertopping

  tpOvertopping: structure with overtopping results

Modules

- module typedefinitionsrtoovertopping

Variables

- real(wp), parameter typedefinitionsrtoovertopping::xdiff_min = 2.0d-2
  minimal value distance between x-coordinates (m)
- real(wp), parameter typedefinitionsrtoovertopping::margindiff = 1.0d-14
  margin for minimal distance (m)
- real(wp), parameter typedefinitionsrtoovertopping::berm_min = 0.0d0
  minimal value gradient berm segment
- real(wp), parameter typedefinitionsrtoovertopping::berm_max = 1.0d0/15
  maximal value gradient berm segment
- real(wp), parameter typedefinitionsrtoovertopping::slope_min = 1.0d0/8
  minimal value gradient slope segment
- real(wp), parameter typedefinitionsrtoovertopping::slope_max = 1.0d0
  maximal value gradient slope segment
- real(wp), parameter typedefinitionsrtoovertopping::margingrad = 0.0025d0
  margin for minimal and maximal gradients
- real(wp), parameter typedefinitionsrtoovertopping::rfactor_min = 0.5d0
  minimal value roughness factor dike segments
- real(wp), parameter typedefinitionsrtoovertopping::rfactor_max = 1.0d0
  maximal value roughness factor dike segments
- real(wp), parameter typedefinitionsrtoovertopping::mz2_min = 0.0d0
  minimal value model factor of 2% runup height
- real(wp), parameter typedefinitionsrtoovertopping::mz2_max = huge(mz2_max)
  maximal value model factor of 2% runup height
- real(wp), parameter typedefinitionsrtoovertopping::frunup1_min = 0.0d0
  minimal value model factor 1 for wave run-up
- real(wp), parameter typedefinitionsrtoovertopping::frunup1_max = huge(fRunup1_max)
  maximal value model factor 1 for wave run-up
- real(wp), parameter typedefinitionsrtoovertopping::frunup2_min = 0.0d0
  minimal value model factor 2 for wave run-up
- real(wp), parameter typedefinitionsrtoovertopping::frunup2_max = huge(fRunup2_max)
  maximal value model factor 2 for wave run-up
- real(wp), parameter typedefinitionsrtoovertopping::frunup3_min = 0.0d0
  minimal value model factor 3 for wave run-up
- real(wp), parameter typedefinitionsrtoovertopping::frunup3_max = huge(fRunup3_max)
  maximal value model factor 3 for wave run-up
- real(wp), parameter typedefinitionsrtoovertopping::fb_min = 0.0d0
  minimal value model factor for breaking waves
- real(wp), parameter typedefinitionsrtoovertopping::fb_max = huge(fB_max)
  maximal value model factor for breaking waves
- real(wp), parameter typedefinitionsrtoovertopping::fn_min = 0.0d0
  minimal value model factor for non-breaking waves
- real(wp), parameter typedefinitionsrtoovertopping::fn_max = huge(fN_max)
  maximal value model factor for non-breaking waves
- real(wp), parameter typedefinitionsrtoovertopping::fs_min = 0.0d0
  minimal value model factor for shallow waves
- real(wp), parameter typedefinitionsrtoovertopping::fs_max = huge(fS_max)
  maximal value model factor for shallow waves
- real(wp), parameter typedefinitionsrtoovertopping::foreshore_min = 0.3d0
  minimal value reduction factor foreshore
- real(wp), parameter typedefinitionsrtoovertopping::foreshore_max = 1.0d0

maximal value reduction factor foreshore

- integer, parameter typedefinitionsrtoovertopping::z2_iter_max = 100

  maximal number of iterations for calculation z2

- real(wp), parameter typedefinitionsrtoovertopping::z2_margin = 0.001d0

  margin for convergence criterium calculation z2

9.26.1 Detailed Description

This file contains a module with the type definitions for Dikes Overtopping.

9.27 utilities.f90 File Reference

General utility routines.

Modules

- module utilities

  Module with general utility routines.

Functions/Subroutines

- subroutine utilities::getfreelunumber (lun)

  getFreeLuNumber Routine to find a free LU-number

- pure character(len(str)) function utilities::to_lower (str)

  to_lower Changes a string to lower case

- pure character(len(str)) function utilities::to_upper (str)

  to_upper Changes a string to upper case

Variables

- character(26), parameter, private utilities::cap = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
- character(26), parameter, private utilities::low = 'abcdefghijklmnopqrstuvwxyz'

9.27.1 Detailed Description

General utility routines.

9.28 vectorUtilities.f90 File Reference

Implement various utility functions for vectors.

Modules

- module vectorutilities

  Module with vector utility functions.

Functions/Subroutines

- real(kind=wp) function, dimension(size(x)) vectorutilities::normalize (x)

  Function for normalizing a vector If the vector is the null vector, a unit vector with uniform components is returned instead.
- real(kind=wp) function vectorutilities::interpolateline (x1, x2, f1, f2, x)

  Function to interpolate and extrapolate linear on a line.
- real(kind=wp) function vectorutilities::linearinterpolate (X, Y, xx)

  Function for linearly interpolating and extrapolating.
- real(kind=wp) function vectorutilities::loglinearinterpolate (X, Y, xx)

  Function for log-linearly interpolating and extrapolating.
- real(kind=wp) function vectorutilities::linearinterpolatecyclic (X, Y, xx)

  Function for linearly interpolating cyclic data.
- real(kind=wp) function vectorutilities::loglinearinterpolatecyclic (X, Y, xx)

  Function for log-linearly interpolating and extrapolating cyclic data.
- real(kind=wp) function vectorutilities::linearinterpolateangles (X, Y, xx)

  Function for linearly interpolating and extrapolating angles.
- subroutine vectorutilities::sortandlocate (X, IDXsorted, xx, locID, locIDm1, Cyclic)

  Subroutine to bracket a given value xx within two elements of an array after sort in descending order.
- real(kind=wp) function vectorutilities::stepsize (i, x)

  Function for the Trapezium rule.
- real(kind=wp) function vectorutilities::trapeziumrule (x, f)

  Function for the Trapezium rule.
- real(kind=wp) function vectorutilities::intgratewithexponentcomponent (x, f)

  Function for the Trapezium rule.
- subroutine vectorutilities::ssort (x, iy, n, kflag)

  Subroutine to sort a vector in either ascending or descending order.

9.28.1 Detailed Description

Implement various utility functions for vectors.

9.29 waveParametersUtilities.f90 File Reference

Module waveParametersUtilities for wave parameter computations.

Modules

- module waveparametersutilities

  Module to calculate the wave steepness or the wave period.

Functions/Subroutines

- real(wp) function waveparametersutilities::computewavesteepness (waveHeight, wavePeriod)

  Module to calculate the the wave steepness from the wave height and the wave period.
- real(wp) function waveparametersutilities::computewaveperiod (waveHeight, waveSteepness)

  Module to calculate the the wave period from the wave height and the wave steepness.

9.29.1 Detailed Description

Module waveParametersUtilities for wave parameter computations.

9.30 waveReduction.f90 File Reference

Wave reduction routines.

Data Types

- type physics_wavereduction::tpwavereduction

Modules

- module physics_wavereduction

  Module with routines for wave reduction.

Functions/Subroutines

- subroutine, public physics_wavereduction::calculatewavereduction (wavereduction, errorCode, Hs)

  Computation of wave reduction due to shallow foreland (simple model) Wave reduction function to compute the wave reduction (simple model)

9.30.1 Detailed Description

Wave reduction routines.

9.31 waveRunup.f90 File Reference

This file contains a module with the core computations for Dikes Overtopping.

Modules

- module waverunup

Functions/Subroutines

- subroutine, public waverunup::iterationwaverunup (geometry, h, Hm0, Tm_10, gammaBeta_z, modelFactors, z2, succes, errorMessage)

  iterationWaveRunup: iteration for the wave runup
- real(kind=wp) function waverunup::determinestartingvalue (i, relaxationFactor, z2_start, z2_end, Hm0)
- integer function waverunup::findsmallestresidu (z2_start, z2_end)
- logical function waverunup::convergedwithresidu (i, z2_start, z2_end)

9.31.1 Detailed Description

This file contains a module with the core computations for Dikes Overtopping.

9.32 writeUtilities.f90 File Reference

General write utilities.

Data Types

- interface writeutilities::writetodevice

Modules

- module writeutilities

Functions/Subroutines

- subroutine writeutilities::writetodevice_line (fileId, line)

  Write to device.
- subroutine writeutilities::writetodevice_text_real (fileId, formatText, textValue, realValue)

  Write to device.
- subroutine, public writeutilities::writetodevicechars (fileId, char, count)

  Write count times the same character to output device.
- character(len=5) function, public writeutilities::inttostr (intValue)

  Convert integer to string.

9.32.1 Detailed Description

General write utilities.

9.33 zFunctionsWTIOvertopping.f90 File Reference

This file contains the limit state functions for wave overtopping within WTI.

Modules

- module zfunctionswtiovertopping

  Module for the Limit State Functions (Z-functions) for wave overtopping.

Functions/Subroutines

- subroutine, public zfunctionswtiovertopping::calculateqorto (dikeHeight, modelFactors, overtopping, load, geometry, succes, errorMessage)

  Subroutine to calculate the overtopping discharge with the RTO-overtopping dll.
- subroutine, public zfunctionswtiovertopping::profileinstructure (nrCoordinates, xcoordinates, ycoordinates, dikeHeight, nrCoordsAdjusted, xCoordsAdjusted, zCoordsAdjusted, succes, errorMessage)

  Subroutine to fill the profile in a structure and call the adjustment function of the profile due to a desired dike height.
- subroutine zfunctionswtiovertopping::adjustprofile (nrCoordinates, coordinates, dikeHeight, nrCoords↩Adjusted, xCoordsAdjusted, zCoordsAdjusted, succes, errorMessage)

  Subroutine adjust the profile due to a desired dike height.
- real(kind=wp) function, public zfunctionswtiovertopping::zfunclogratios (qo, qc, mqo, mqc, success, error↩Message)

  Routine to compute the limit state value by using the logs of the overtopping discharges (computed and desired)

9.33.1 Detailed Description

This file contains the limit state functions for wave overtopping within WTI.

Index