

**ferror**

1

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
1.1	Introduction . . . . .	1
<b>2</b>	<b>Modules Index</b>	<b>3</b>
2.1	Modules List . . . . .	3
<b>3</b>	<b>Data Type Index</b>	<b>5</b>
3.1	Data Types List . . . . .	5
<b>4</b>	<b>Module Documentation</b>	<b>7</b>
4.1	error Module Reference . . . . .	7
4.1.1	Detailed Description . . . . .	8
4.1.2	Function/Subroutine Documentation . . . . .	8
4.1.2.1	get_error_flag(this) . . . . .	8
4.1.2.2	get_exit_on_error(this) . . . . .	8
4.1.2.3	get_log_filename(this) . . . . .	8
4.1.2.4	get_warning_flag(this) . . . . .	9
4.1.2.5	has_error_occurred(this) . . . . .	9
4.1.2.6	has_warning_occurred(this) . . . . .	9
4.1.2.7	log_error(this, fcn, msg, flag) . . . . .	9
4.1.2.8	report_error(this, fcn, msg, flag) . . . . .	10
4.1.2.9	report_warning(this, fcn, msg, flag) . . . . .	10
4.1.2.10	reset_error_status(this) . . . . .	10
4.1.2.11	reset_warning_status(this) . . . . .	10
4.1.2.12	set_exit_on_error(this, x) . . . . .	11

4.1.2.13	<code>set_log_filename(this, str)</code>	11
4.2	<code>ferror_c_binding</code> Module Reference	11
4.2.1	Detailed Description	12
4.2.2	Function/Subroutine Documentation	12
4.2.2.1	<code>alloc_error_handler()</code>	12
4.2.2.2	<code>cstr_2_fstr(cstr)</code>	12
4.2.2.3	<code>error_occurred(err)</code>	13
4.2.2.4	<code>free_error_handler(ptr)</code>	13
4.2.2.5	<code>fstr_2_cstr(fstr, cstr, csize)</code>	13
4.2.2.6	<code>get_error_code(err)</code>	13
4.2.2.7	<code>get_error_log_fname(err, fname, nfname)</code>	14
4.2.2.8	<code>get_exit_behavior(err)</code>	14
4.2.2.9	<code>get_warning_code(err)</code>	14
4.2.2.10	<code>register_error(err, fcn, msg, flag)</code>	14
4.2.2.11	<code>register_warning(err, fcn, msg, flag)</code>	15
4.2.2.12	<code>reset_error(err)</code>	15
4.2.2.13	<code>reset_warning(err)</code>	15
4.2.2.14	<code>set_error_log_fname(err, fname)</code>	15
4.2.2.15	<code>set_exit_behavior(err, x)</code>	16
4.2.2.16	<code>warning_occurred(err)</code>	16
4.2.2.17	<code>write_error_log(err, fcn, msg, flag)</code>	16
5	<b>Data Type Documentation</b>	<b>17</b>
5.1	<code>ferror::errors</code> Type Reference	17
5.1.1	Detailed Description	18
	<b>Index</b>	<b>19</b>

# Chapter 1

## Main Page

### 1.1 Introduction

FERROR is a library to assist with error handling in Fortran projects.

#### Author

Jason Christopherson

#### Version

1.0



## Chapter 2

# Modules Index

### 2.1 Modules List

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

<a href="#">ferror</a>	<b>ferror</b> . . . . .	<a href="#">7</a>
<a href="#">ferror_c_binding</a>	<b>ferror</b> . . . . .	<a href="#">11</a>





## Chapter 3

# Data Type Index

### 3.1 Data Types List

Here are the data types with brief descriptions:

<code>feror::errors</code>	Defines a type for managing errors and warnings . . . . .	17
----------------------------	---	----



## Chapter 4

# Module Documentation

### 4.1 ferror Module Reference

#### ferror

##### Data Types

- type [errors](#)  
*Defines a type for managing errors and warnings.*

##### Functions/Subroutines

- pure character(len=:) function, allocatable [get\\_log\\_filename](#) (this)  
*Gets the name of the error log file.*
- subroutine [set\\_log\\_filename](#) (this, str)  
*Sets the name of the error log file.*
- subroutine [report\\_error](#) (this, fcn, msg, flag)  
*Reports an error condition to the user.*
- subroutine [report\\_warning](#) (this, fcn, msg, flag)  
*Reports a warning message to the user.*
- subroutine [log\\_error](#) (this, fcn, msg, flag)  
*Writes an error log file.*
- pure logical function [has\\_error\\_occurred](#) (this)  
*Tests to see if an error has been encountered.*
- subroutine [reset\\_error\\_status](#) (this)  
*Resets the error status flag to false, and the current error flag to zero.*
- pure logical function [has\\_warning\\_occurred](#) (this)  
*Tests to see if a warning has been encountered.*
- subroutine [reset\\_warning\\_status](#) (this)  
*Resets the warning status flag to false, and the current warning flag to zero.*
- pure integer function [get\\_error\\_flag](#) (this)  
*Gets the current error flag.*
- pure integer function [get\\_warning\\_flag](#) (this)  
*Gets the current warning flag.*
- pure logical function [get\\_exit\\_on\\_error](#) (this)  
*Gets a logical value determining if the application should be terminated when an error is encountered.*
- subroutine [set\\_exit\\_on\\_error](#) (this, x)  
*Sets a logical value determining if the application should be terminated when an error is encountered.*

### 4.1.1 Detailed Description

#### **ferro**

##### Purpose

Provides a series of error codes and error handling mechanisms.

### 4.1.2 Function/Subroutine Documentation

#### 4.1.2.1 pure integer function ferro::get\_error\_flag ( class(errors), intent(in) *this* ) [private]

Gets the current error flag.

##### Parameters

in	<i>this</i>	The errors object.
----	-------------	--------------------

##### Returns

The current error flag.

#### 4.1.2.2 pure logical function ferro::get\_exit\_on\_error ( class(errors), intent(in) *this* ) [private]

Gets a logical value determining if the application should be terminated when an error is encountered.

##### Parameters

in	<i>this</i>	The errors object.
----	-------------	--------------------

##### Returns

Returns true if the application should be terminated; else, false.

#### 4.1.2.3 pure character(len = :) function, allocatable ferro::get\_log\_filename ( class(errors), intent(in) *this* )

Gets the name of the error log file.

##### Parameters

in	<i>this</i>	The errors object.
----	-------------	--------------------

##### Returns

The filename.

#### 4.1.2.4 pure integer function `ferror::get_warning_flag ( class(errors), intent(in) this )` [private]

Gets the current warning flag.

##### Parameters

in	<i>this</i>	The errors object.
----	-------------	--------------------

##### Returns

The current warning flag.

#### 4.1.2.5 pure logical function `ferror::has_error_occurred ( class(errors), intent(in) this )` [private]

Tests to see if an error has been encountered.

##### Parameters

in	<i>this</i>	The errors object.
----	-------------	--------------------

##### Returns

Returns true if an error has been encountered; else, false.

#### 4.1.2.6 pure logical function `ferror::has_warning_occurred ( class(errors), intent(in) this )` [private]

Tests to see if a warning has been encountered.

##### Parameters

in	<i>this</i>	The errors object.
----	-------------	--------------------

##### Returns

Returns true if a warning has been encountered; else, false.

#### 4.1.2.7 subroutine `ferror::log_error ( class(errors), intent(in) this, character(len = *), intent(in) fcn, character(len = *), intent(in) msg, integer, intent(in) flag )` [private]

Writes an error log file.

##### Parameters

in	<i>this</i>	The errors object.
in	<i>fcn</i>	The name of the function or subroutine in which the error was encountered.
in	<i>msg</i>	The error message.
in	<i>flag</i>	The error flag.

**4.1.2.8** subroutine `error::report_error` ( `class(errors)`, `intent(inout) this`, `character(len = *)`, `intent(in) fcn`, `character(len = *)`, `intent(in) msg`, `integer`, `intent(in) flag` ) `[private]`

Reports an error condition to the user.

#### Parameters

<code>in, out</code>	<code>this</code>	The errors object.
<code>in</code>	<code>fcn</code>	The name of the function or subroutine in which the error was encountered.
<code>in</code>	<code>msg</code>	The error message.
<code>in</code>	<code>flag</code>	The error flag.

#### Remarks

The default behavior prints an error message, appends the supplied information to a log file, and terminates the program.

**4.1.2.9** subroutine `error::report_warning` ( `class(errors)`, `intent(inout) this`, `character(len = *)`, `intent(in) fcn`, `character(len = *)`, `intent(in) msg`, `integer`, `intent(in) flag` ) `[private]`

Reports a warning message to the user.

#### Parameters

<code>in, out</code>	<code>this</code>	The errors object.
<code>in</code>	<code>fcn</code>	The name of the function or subroutine from which the warning was issued.
<code>in</code>	<code>msg</code>	The warning message.
<code>in</code>	<code>flag</code>	The warning flag.

#### Remarks

The default behavior prints the warning message, and returns control back to the calling code.

**4.1.2.10** subroutine `error::reset_error_status` ( `class(errors)`, `intent(inout) this` ) `[private]`

Resets the error status flag to false, and the current error flag to zero.

#### Parameters

<code>in, out</code>	<code>this</code>	The errors object.
----------------------	-------------------	--------------------

**4.1.2.11** subroutine `error::reset_warning_status` ( `class(errors)`, `intent(inout) this` ) `[private]`

Resets the warning status flag to false, and the current warning flag to zero.

## Parameters

<code>in, out</code>	<code>this</code>	The errors object.
----------------------	-------------------	--------------------

4.1.2.12 `subroutine ferror::set_exit_on_error ( class(errors), intent(inout) this, logical, intent(in) x ) [private]`

Sets a logical value determining if the application should be terminated when an error is encountered.

## Parameters

<code>in, out</code>	<code>this</code>	The errors object.
<code>in</code>	<code>x</code>	Set to true if the application should be terminated when an error is reported; else, false.

4.1.2.13 `subroutine ferror::set_log_filename ( class(errors), intent(inout) this, character(len = :), allocatable str ) [private]`

Sets the name of the error log file.

## Parameters

<code>in, out</code>	<code>this</code>	The errors object.
<code>in</code>	<code>str</code>	The filename.

## 4.2 `error_c_binding` Module Reference

### `ferror`

#### Functions/Subroutines

- type(`c_ptr`) function [alloc\\_error\\_handler](#) ()  
*Initializes a pointer to a new error handler object.*
- subroutine [free\\_error\\_handler](#) (`ptr`)  
*Cleans up an error handler object.*
- subroutine [get\\_error\\_log\\_fname](#) (`err`, `fname`, `nfname`)  
*Gets the name of the error log file.*
- subroutine [set\\_error\\_log\\_fname](#) (`err`, `fname`)  
*Sets the error log filename.*
- subroutine [register\\_error](#) (`err`, `fcn`, `msg`, `flag`)  
*Reports an error condition to the user.*
- subroutine [register\\_warning](#) (`err`, `fcn`, `msg`, `flag`)  
*Reports a warning condition to the user.*
- subroutine [write\\_error\\_log](#) (`err`, `fcn`, `msg`, `flag`)  
*Writes an error log file.*
- logical(`c_bool`) function [error\\_occurred](#) (`err`)

- Tests to see if an error has been encountered.*

  - subroutine `reset_error` (err)
 

*Resets the error status flag to false, and the current error flag to zero.*
  - logical(c\_bool) function `warning_occurred` (err)
 

*Tests to see if a warning has been encountered.*
  - subroutine `reset_warning` (err)
 

*Resets the warning status flag to false, and the current warning flag to zero.*
  - integer(c\_int) function `get_error_code` (err)
 

*Gets the current error flag.*
  - integer(c\_int) function `get_warning_code` (err)
 

*Gets the current warning flag.*
  - logical(c\_bool) function `get_exit_behavior` (err)
 

*Gets a logical value determining if the application should be terminated when an error is encountered.*
  - subroutine `set_exit_behavior` (err, x)
 

*Sets a logical value determining if the application should be terminated when an error is encountered.*
  - character(len=:) function, allocatable `cstr_2_fstr` (cstr)
 

*Copies a C string (null terminated) to a Fortran string.*
  - subroutine `fstr_2_cstr` (fstr, cstr, csize)
 

*Copies a Fortran string into a C string.*

## 4.2.1 Detailed Description

### error

#### Purpose

Provides C bindings to the error library.

## 4.2.2 Function/Subroutine Documentation

### 4.2.2.1 type(c\_ptr) function `error_c_binding::alloc_error_handler` ( )

Initializes a pointer to a new error handler object.

#### Returns

The pointer to the newly created error handler object.

### 4.2.2.2 character(len = :) function, allocatable `error_c_binding::cstr_2_fstr` ( character(kind = c\_char), dimension(\*), intent(in) cstr )

Copies a C string (null terminated) to a Fortran string.

#### Parameters

in	cstr	The null-terminated C string.
----	------	-------------------------------



**Returns**

The Fortran copy.

**4.2.2.3** `logical(c_bool) function error_c_binding::error_occurred ( type(c_ptr), intent(in), value err )`

Tests to see if an error has been encountered.

**Parameters**

<code>in</code>	<code>err</code>	A pointer to the error handler object.
-----------------	------------------	--

**Returns**

Returns true if an error has been encountered; else, false.

**4.2.2.4** `subroutine error_c_binding::free_error_handler ( type(c_ptr), intent(in) ptr )`

Cleans up an error handler object.

**Parameters**

<code>in</code>	<code>ptr</code>	The pointer to the error handler object.
-----------------	------------------	--

**4.2.2.5** `subroutine error_c_binding::fstr_2_cstr ( character(len = *), intent(in) fstr, character(kind = c_char), dimension(*), intent(out) cstr, integer, intent(inout) csize )`

Copies a Fortran string into a C string.

**Parameters**

<code>in</code>	<code>fstr</code>	The Fortran string to copy.
<code>out</code>	<code>cstr</code>	The null-terminated C string.
<code>in, out</code>	<code>csize</code>	On input, the size of the character buffer <code>cstr</code> . On output, the actual number of characters (not including the null character) written to <code>cstr</code> .

**4.2.2.6** `integer(c_int) function error_c_binding::get_error_code ( type(c_ptr), intent(in), value err )`

Gets the current error flag.

**Parameters**

<code>in</code>	<code>err</code>	A pointer to the error handler object.
-----------------	------------------	--

**Returns**

The current error flag.

**4.2.2.7** subroutine `error_c_binding::get_error_log_fname` ( `type(c_ptr)`, `intent(in)`, `value err`, `character(kind = c_char)`, `dimension(*)`, `intent(out) fname`, `integer(c_int)`, `intent(inout) nfname` )

Gets the name of the error log file.

**Parameters**

<code>in</code>	<code>err</code>	A pointer to the error handler object.
<code>out</code>	<code>fname</code>	A character buffer where the filename will be written. It is recommended that this be in the neighborhood of 256 elements.
<code>in, out</code>	<code>nfname</code>	On input, the actual size of the buffer. Be sure to leave room for the null terminator character. On output, the actual numbers of characters written to <code>fname</code> (not including the null character).

**4.2.2.8** `logical(c_bool)` function `error_c_binding::get_exit_behavior` ( `type(c_ptr)`, `intent(in)`, `value err` )

Gets a logical value determining if the application should be terminated when an error is encountered.

**Parameters**

<code>in</code>	<code>err</code>	A pointer to the error handler object.
-----------------	------------------	--

**Returns**

Returns true if the application should be terminated; else, false.

**4.2.2.9** `integer(c_int)` function `error_c_binding::get_warning_code` ( `type(c_ptr)`, `intent(in)`, `value err` )

Gets the current warning flag.

**Parameters**

<code>in</code>	<code>err</code>	A pointer to the error handler object.
-----------------	------------------	--

**Returns**

The current warning flag.

**4.2.2.10** subroutine `error_c_binding::register_error` ( `type(c_ptr)`, `intent(in)`, `value err`, `character(kind = c_char)`, `intent(in) fcn`, `character(kind = c_char)`, `intent(in) msg`, `integer(c_int)`, `intent(in)`, `value flag` )

Reports an error condition to the user.

## Parameters

in	<i>err</i>	A pointer to the error handler object.
in	<i>fcn</i>	The name of the function or subroutine in which the error was encountered.
in	<i>msg</i>	The error message.
in	<i>flag</i>	The error flag.

4.2.2.11 subroutine `ferror_c_binding::register_warning` ( `type(c_ptr)`, intent(in), value *err*, `character(kind = c_char)`, intent(in) *fcn*, `character(kind = c_char)`, intent(in) *msg*, `integer(c_int)`, intent(in), value *flag* )

Reports a warning condition to the user.

## Parameters

in	<i>err</i>	A pointer to the error handler object.
in	<i>fcn</i>	The name of the function or subroutine in which the warning was encountered.
in	<i>msg</i>	The warning message.
in	<i>flag</i>	The warning flag.

4.2.2.12 subroutine `ferror_c_binding::reset_error` ( `type(c_ptr)`, intent(in), value *err* )

Resets the error status flag to false, and the current error flag to zero.

## Parameters

in	<i>err</i>	The error handler object.
----	------------	---------------------------

4.2.2.13 subroutine `ferror_c_binding::reset_warning` ( `type(c_ptr)`, intent(in), value *err* )

Resets the warning status flag to false, and the current warning flag to zero.

## Parameters

in	<i>err</i>	A pointer to the error handler object.
----	------------	--

4.2.2.14 subroutine `ferror_c_binding::set_error_log_fname` ( `type(c_ptr)`, intent(in), value *err*, `character(kind = c_char)`, dimension(\*), intent(in) *fname* )

Sets the error log filename.

## Parameters

in, out	<i>err</i>	A pointer to the error handler object.
in	<i>fname</i>	A null-terminated string containing the filename.

#### 4.2.2.15 subroutine `error_c_binding::set_exit_behavior` ( `type(c_ptr)`, `intent(in)`, `value err`, `logical(c_bool)`, `intent(in) x` )

Sets a logical value determining if the application should be terminated when an error is encountered.

##### Parameters

<code>in</code>	<code>err</code>	A pointer to the error handler object.
<code>in</code>	<code>x</code>	Set to true if the application should be terminated when an error is reported; else, false.

#### 4.2.2.16 `logical(c_bool)` function `error_c_binding::warning_occurred` ( `type(c_ptr)`, `intent(in)`, `value err` )

Tests to see if a warning has been encountered.

##### Parameters

<code>in</code>	<code>err</code>	A pointer to the error handler object.
-----------------	------------------	--

##### Returns

Returns true if a warning has been encountered; else, false.

#### 4.2.2.17 subroutine `error_c_binding::write_error_log` ( `type(c_ptr)`, `intent(in)`, `value err`, `character(kind = c_char)`, `intent(in) fcn`, `character(kind = c_char)`, `intent(in) msg`, `integer(c_int)`, `intent(in)`, `value flag` )

Writes an error log file.

##### Parameters

<code>in</code>	<code>err</code>	A pointer to the error handler object.
<code>in</code>	<code>fcn</code>	The name of the function or subroutine in which the error was encountered.
<code>in</code>	<code>msg</code>	The error message.
<code>in</code>	<code>flag</code>	The error flag.

## Chapter 5

# Data Type Documentation

### 5.1 `ferror::errors` Type Reference

Defines a type for managing errors and warnings.

#### Public Member Functions

- procedure, public [get\\_log\\_filename](#)  
*Gets the name of the error log file.*
- procedure, public [set\\_log\\_filename](#)  
*Sets the name of the error log file.*
- procedure, public [report\\_error](#)  
*Reports an error condition to the user.*
- procedure, public [report\\_warning](#)  
*Reports a warning message to the user.*
- procedure, public [log\\_error](#)  
*Writes an error log file.*
- procedure, public [has\\_error\\_occurred](#)  
*Tests to see if an error has been encountered.*
- procedure, public [reset\\_error\\_status](#)  
*Resets the error status flag to false.*
- procedure, public [has\\_warning\\_occurred](#)  
*Tests to see if a warning has been encountered.*
- procedure, public [reset\\_warning\\_status](#)  
*Resets the warning status flag to false.*
- procedure, public [get\\_error\\_flag](#)  
*Gets the current error flag.*
- procedure, public [get\\_warning\\_flag](#)  
*Gets the current warning flag.*
- procedure, public [get\\_exit\\_on\\_error](#)  
*Gets a logical value determining if the application should be terminated when an error is encountered.*
- procedure, public [set\\_exit\\_on\\_error](#)  
*Sets a logical value determining if the application should be terminated when an error is encountered.*

## Public Attributes

- character(len=256) `m_fname` = "error\_log.txt"  
*A maximum of 256 character error log filename.*
- logical `m_founderror` = .false.  
*Found an error.*
- logical `m_foundwarning` = .false.  
*Found a warning.*
- integer `m_errorflag` = 0  
*The error flag.*
- integer `m_warningflag` = 0  
*The warning flag.*
- logical `m_exitonerror` = .true.  
*Terminate the application on error.*

### 5.1.1 Detailed Description

Defines a type for managing errors and warnings.

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

- /home/jason/Documents/Code/ferror/src/ferror.f90

# Index

- alloc\_error\_handler
  - error\_c\_binding, 12
- cstr\_2\_fstr
  - error\_c\_binding, 12
- error\_occurred
  - error\_c\_binding, 13
- error, 7
  - get\_error\_flag, 8
  - get\_exit\_on\_error, 8
  - get\_log\_filename, 8
  - get\_warning\_flag, 8
  - has\_error\_occurred, 9
  - has\_warning\_occurred, 9
  - log\_error, 9
  - report\_error, 10
  - report\_warning, 10
  - reset\_error\_status, 10
  - reset\_warning\_status, 10
  - set\_exit\_on\_error, 11
  - set\_log\_filename, 11
- error::errors, 17
- error\_c\_binding, 11
  - alloc\_error\_handler, 12
  - cstr\_2\_fstr, 12
  - error\_occurred, 13
  - free\_error\_handler, 13
  - fstr\_2\_cstr, 13
  - get\_error\_code, 13
  - get\_error\_log\_fname, 14
  - get\_exit\_behavior, 14
  - get\_warning\_code, 14
  - register\_error, 14
  - register\_warning, 15
  - reset\_error, 15
  - reset\_warning, 15
  - set\_error\_log\_fname, 15
  - set\_exit\_behavior, 16
  - warning\_occurred, 16
  - write\_error\_log, 16
- free\_error\_handler
  - error\_c\_binding, 13
- fstr\_2\_cstr
  - error\_c\_binding, 13
- get\_error\_code
  - error\_c\_binding, 13
- get\_error\_flag
  - error, 8
- get\_error\_log\_fname
  - error\_c\_binding, 14
- get\_exit\_behavior
  - error\_c\_binding, 14
- get\_exit\_on\_error
  - error, 8
- get\_log\_filename
  - error, 8
- get\_warning\_code
  - error\_c\_binding, 14
- get\_warning\_flag
  - error, 8
- has\_error\_occurred
  - error, 9
- has\_warning\_occurred
  - error, 9
- log\_error
  - error, 9
- register\_error
  - error\_c\_binding, 14
- register\_warning
  - error\_c\_binding, 15
- report\_error
  - error, 10
- report\_warning
  - error, 10
- reset\_error
  - error\_c\_binding, 15
- reset\_error\_status
  - error, 10
- reset\_warning
  - error\_c\_binding, 15
- reset\_warning\_status
  - error, 10
- set\_error\_log\_fname
  - error\_c\_binding, 15
- set\_exit\_behavior
  - error\_c\_binding, 16
- set\_exit\_on\_error
  - error, 11
- set\_log\_filename
  - error, 11
- warning\_occurred
  - error\_c\_binding, 16
- write\_error\_log

`ferror_c_binding`, [16](#)