

tracefctxx

1.1.7

Generated by Doxygen 1.7.1

Sat Feb 11 2012 14:40:55

## Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
1.1	Copyright . . . . .	1
1.2	Introduction . . . . .	1
<b>2</b>	<b>Assertions</b>	<b>2</b>
<b>3</b>	<b>Basic Interface</b>	<b>2</b>
3.1	Function Stack . . . . .	2
3.2	Messages and Exit . . . . .	3
<b>4</b>	<b>Message Prefix</b>	<b>3</b>
<b>5</b>	<b>Output Streams</b>	<b>3</b>
<b>6</b>	<b>Directory Hierarchy</b>	<b>4</b>
6.1	Directories . . . . .	4
<b>7</b>	<b>Class Index</b>	<b>4</b>
7.1	Class List . . . . .	4
<b>8</b>	<b>File Index</b>	<b>4</b>
8.1	File List . . . . .	4
<b>9</b>	<b>Directory Documentation</b>	<b>5</b>
9.1	/data/pelf1/dj/hd0/axaf/ Directory Reference . . . . .	5
9.2	/data/pelf1/dj/ Directory Reference . . . . .	6
9.3	/data/pelf1/dj/hd0/ Directory Reference . . . . .	7
9.4	/data/pelf1/ Directory Reference . . . . .	8
9.5	/data/pelf1/dj/hd0/axaf/src/ Directory Reference . . . . .	9
9.6	/data/pelf1/dj/hd0/axaf/src/tracefctxx/tracefctxx/ Directory Reference	10
9.7	/data/pelf1/dj/hd0/axaf/src/tracefctxx/ Directory Reference . . . . .	11
<b>10</b>	<b>Class Documentation</b>	<b>11</b>

<b>1 Main Page</b>	<b>1</b>
--------------------	----------

---

10.1 TraceFct Class Reference . . . . .	11
10.1.1 Detailed Description . . . . .	13
10.1.2 Constructor & Destructor Documentation . . . . .	13
10.1.3 Member Function Documentation . . . . .	15
<b>11 File Documentation</b>	<b>20</b>
11.1 TraceFct.h File Reference . . . . .	20
11.1.1 Detailed Description . . . . .	22
11.1.2 Define Documentation . . . . .	22
<b>12 Example Documentation</b>	<b>24</b>
12.1 ex1.cc . . . . .	24
12.2 ex2.cc . . . . .	25

# 1 Main Page

## 1.1 Copyright

Copyright (C) 2006 Smithsonian Astrophysical Observatory

This file is part of tracefctxx

tracefctxx is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

tracefctxx is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor Boston, MA 02110-1301, USA

## 1.2 Introduction

**TraceFct** is a class which helps a program keep track of its calling sequence, allowing one to produce a "stack" dump upon error. It also provides a uniform mechanism for printing messages and errors, and provides `assert()` style macros which use these features.

**TraceFct** prefixes messages and errors with a string which uniquely identifies the process (useful for debugging when running multiple processes concurrently). It support switching the output stream for messages.

- [Basic Interface](#)
- [Output Streams](#)
- [Message Prefix](#)
- [Assertions](#)
- hints

## 2 Assertions

**TraceFct** provides two types of C style assertions, both implemented as pre-processor macros. Their main benefit is the uniform [prefix](#) string.

The first assertion is *always active* (i.e. you can't define it away with `NDEBUG`). It's available via the [tf\\_assert](#) macro.

The second type may be turned on and off, and comes in five levels. The macros are [tf\\_assert1](#), [tf\\_assert2](#), [tf\\_assert3](#), [tf\\_assert4](#), [tf\\_assert5](#), and are controlled by the value of the [TF\\_ASSERT\\_LEVEL](#) preprocessor macro. [TF\\_ASSERT\\_LEVEL](#) should be set to an integer; assertions with levels less than or equal to that will be made active.

## 3 Basic Interface

### 3.1 Function Stack

Tracking the calling sequence is done by creating a **TraceFct** object in each routine which should be tracked. The sequence is stored as class global -- **TraceFct** thus may not reliably be used in a multithreaded program.

The class must be initialized by calling the special, three argument version of the constructor. This is usually done in the `main` subroutine.

```
int main( int argc, char * argv[] )
{
    TraceFct tf( argv[0], 0, -1 );
```

Thereafter, use the one-argument constructor when entering a new subroutine to be traced.

```
void f2( void )
{
    static int level = 0;
    TraceFct tf( "f2" );
```

## 3.2 Messages and Exit

[TraceFct](#) provides several methods to output warning or diagnostic messages as well as produce a function stack dump upon an error exit. The benefit of using the output methods is that all output will have a common, unique prefix, which permits easier analysis of complicated log files. Messages may have embedded newlines.

```
tf.message( "message test: %d\n", level );
tf.message( "1. multiline\n2. message test\n" );
```

Currently [TraceFct](#) does *not* have an overloaded << operator; use the [TraceFct::message\(\)](#) and [TraceFct::vmessage\(\)](#) methods.

To generate a stack dump upon exit, use the [TraceFct::exit\(\)](#) method.

```
tf.exit( 1, "f1 exit: line1\nf1 exit: line2, extra newline\n" );
```

To generate just a stack dump, just call [TraceFct::dump\\_stack\(\)](#).

## 4 Message Prefix

Messages have the following appearance:

*#pid: exec-name: message*

where *pid* is the process id number, *exec-name* is the name of the executable, and *message* is a user provided message.

## 5 Output Streams

Normally, [TraceFct](#) produces output on the `stderr` stream. [TraceFct::open\(\)](#) causes it be redirected to an alternative stream. That stream may be closed with [TraceFct::close\(\)](#), after which the output will once again be sent to the `stderr` stream.

Output not written to the `stderr` stream will not have the standard prefix prepended. Messages written by [TraceFct::exit\(\)](#) will appear on both the `stderr` and alternative output streams.

## 6 Directory Hierarchy

### 6.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

<b>pelf1</b>	<b><a href="#">8</a></b>
<b>    dj</b>	<b><a href="#">6</a></b>
<b>        hd0</b>	<b><a href="#">7</a></b>
<b>            axaf</b>	<b><a href="#">5</a></b>
<b>                src</b>	<b><a href="#">9</a></b>
<b>                    tracefctxx</b>	<b><a href="#">11</a></b>
<b>                        tracefctxx</b>	<b><a href="#">10</a></b>

## 7 Class Index

### 7.1 Class List

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

<b><a href="#">TraceFct</a></b>	<b><a href="#">11</a></b>
---------------------------------	---------------------------

## 8 File Index

### 8.1 File List

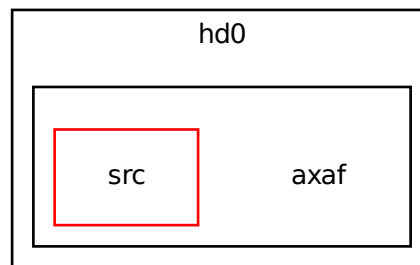
Here is a list of all documented files with brief descriptions:

<b>TraceFct.cc</b>	<b><a href="#">??</a></b>
<b><a href="#">TraceFct.h</a></b>	<b><a href="#">20</a></b>

## 9 Directory Documentation

### 9.1 /data/pelf1/dj/hd0/axaf/ Directory Reference

Directory dependency graph for /data/pelf1/dj/hd0/axaf/:

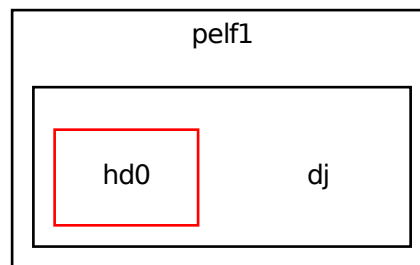


#### Directories

- directory [src](#)

## 9.2 /data/pelf1/dj/ Directory Reference

Directory dependency graph for /data/pelf1/dj/:



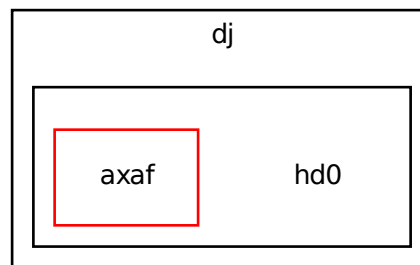
### Directories

- directory [hd0](#)



### 9.3 /data/pelf1/dj/hd0/ Directory Reference

Directory dependency graph for /data/pelf1/dj/hd0/:

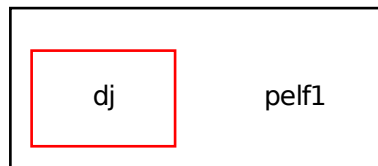


#### Directories

- directory [axaf](#)

## 9.4 /data/pelf1/ Directory Reference

Directory dependency graph for /data/pelf1/:

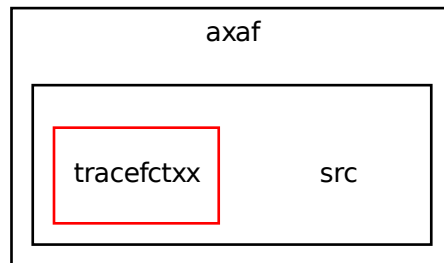


### Directories

- directory [dj](#)

## 9.5 /data/pelf1/dj/hd0/axaf/src/ Directory Reference

Directory dependency graph for /data/pelf1/dj/hd0/axaf/src/:

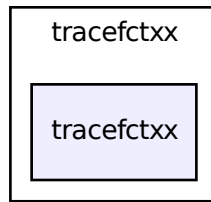


### Directories

- directory [tracefctxx](#)

## 9.6 /data/pelf1/dj/hd0/axaf/src/tracefctxx/tracefctxx/ Directory Reference

Directory dependency graph for /data/pelf1/dj/hd0/axaf/src/tracefctxx/tracefctxx/:

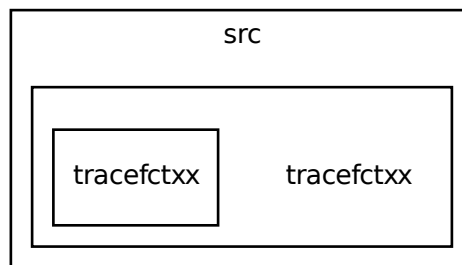


### Files

- file **TraceFct.cc**
- file [TraceFct.h](#)

## 9.7 /data/pelf1/dj/hd0/axaf/src/tracefctxx/ Directory Reference

Directory dependency graph for /data/pelf1/dj/hd0/axaf/src/tracefctxx/:



### Directories

- directory [tracefctxx](#)

## 10 Class Documentation

### 10.1 TraceFct Class Reference

```
#include <TraceFct.h>
```

#### Public Member Functions

- [TraceFct](#) (string name, bool print\_it, int num\_fct\_to\_print)  
*Class initializer.*
- [~TraceFct](#) ()
- [TraceFct](#) (const char \*name)  
*Function entry constructor.*

- [TraceFct](#) (string &name)  
*Function entry constructor.*

### Static Public Member Functions

- static void [dump\\_stack](#) (void)  
*Print the current function stack.*

### Exit

These functions print an error message to `stderr` as well as the **tracefct** output stream (if different from `stderr`) and then exits the program with the supplied error code. The error message is passed in the same fashion as the arguments to `vprintf` or `sprintf`. The message may contain multiple output lines (i.e., multiple newline characters). If a trailing newline character is not specified, it will be appended.

- static void [exit](#) (int exit\_code, const char \*format,...)  
*Exit a program, dumping the function stack.*
- static void [exit](#) (int exit\_code, Exception &ex)  
*Exit a program, dumping the function stack.*
- static void [exit](#) (int exit\_code, const string &msg)  
*Exit a program, dumping the function stack.*
- static void [die](#) (const char \*format,...)  
*Exit a program with error EXIT\_FAILURE, dumping the function stack.*
- static void [die](#) (Exception &ex)  
*Exit a program with error EXIT\_FAILURE, dumping the function stack.*
- static void [die](#) (const string &msg)  
*Exit a program with error EXIT\_FAILURE, dumping the function stack.*

### Messages

The message functions print a user supplied message to the **TraceFct** output stream, prefixed by the standard **TraceFct** style *prefix* string. Unlike **TraceFct::exit()**, a newline character is not appended to the message.

The `println` functions are similar, except that they ensure that a newline character is appended to the message.

- static void [message](#) (const char \*format,...)  
*Print a formatted message to the `tracefct` output stream.*
- static void [message](#) (Exception &ex)  
*Print a message to the `tracefct` output stream.*
- static void [message](#) (const string &msg)  
*Print a message to the `tracefct` output stream.*
- static void [vmessage](#) (const char \*format, va\_list args)  
*Print a formatted message to the `tracefct` output stream using a `stdargs` argument list.*
- static void [println](#) (Exception &ex)  
*Print a message to the `tracefct` output stream.*
- static void [println](#) (const string &msg)  
*Print a message to the `tracefct` output stream.*
- static void [println](#) (const char \*msg)  
*Print a message to the `tracefct` output stream.*

### Output Stream manipulation

- static void [open](#) (const string &filename)  
*redirect the `tracefct` output stream*
- static void [close](#) (void)  
*reset the `tracefct` output stream to `stderr`.*

#### 10.1.1 Detailed Description

The [TraceFct](#) Class

##### Examples:

[ex1.cc](#), and [ex2.cc](#).

Definition at line 152 of file `TraceFct.h`.

#### 10.1.2 Constructor & Destructor Documentation

##### 10.1.2.1 TraceFct::TraceFct ( const char \* *name* )

Function entry constructor.

#### Parameters

***name*** the name of the function

This constructor is used when entering a function which should be registered.

Definition at line 119 of file TraceFct.cc.

References `message()`.

#### 10.1.2.2 TraceFct::TraceFct ( string & *name* )

Function entry constructor.

#### Parameters

***name*** the name of the function

This constructor is used when entering a function which should be registered.

Definition at line 103 of file TraceFct.cc.

References `message()`.

#### 10.1.2.3 TraceFct::TraceFct ( string *prog\_name*, bool *print\_it*, int *num\_fct\_to\_print* )

Class initializer.

This initializer should be used to initialize the class, and should be called once in the program, before any subsequent calls to the per-function initializers

#### Parameters

***name*** the name of the program, preferably `argv[0]`. path information (if present) is not stripped.

***print\_it*** if true, a message will be output upon entry and exit of a function

***num\_fct\_to\_print*** if positive, only the requested number of function names will be printed. if non-positive, all of the function names will be printed.

The calling routine can also set up some output options which determine the depth of the function stack to print and whether a diagnostic should be printed at every entry and return of a function.

Definition at line 77 of file TraceFct.cc.



#### 10.1.2.4 TraceFct::~TraceFct ( )

The destructor.

Definition at line 135 of file TraceFct.cc.

References `message()`.

### 10.1.3 Member Function Documentation

#### 10.1.3.1 void TraceFct::close ( void ) [static]

reset the `tracefct` output stream to `stderr`.

`close()` resets the output stream to `stderr`, closing the file previously opened by `TraceFct::open()`.

Definition at line 652 of file TraceFct.cc.

#### 10.1.3.2 void TraceFct::die ( const char \* *format*, ... ) [static]

Exit a program with error `EXIT_FAILURE`, dumping the function stack.

##### Parameters

*format* a printf style format string

... additional arguments

Definition at line 452 of file TraceFct.cc.

#### 10.1.3.3 void TraceFct::die ( Exception & *ex* ) [inline, static]

Exit a program with error `EXIT_FAILURE`, dumping the function stack.

##### Parameters

*ex* The Exception Stack to print

This version of the exit function will print out the passed `Exception` stack.

Definition at line 169 of file TraceFct.h.

References `exit()`.

#### 10.1.3.4 void TraceFct::die ( const string & *msg* ) [inline, static]

Exit a program with error EXIT\_FAILURE, dumping the function stack.

##### Parameters

*msg* a message to print

Definition at line 172 of file TraceFct.h.

References `exit()`.

#### 10.1.3.5 void TraceFct::dump\_stack ( void ) [static]

Print the current function stack.

##### Parameters

*ostr* the output file stream

This prints the current function stack to the [TraceFct](#) output stream

##### Examples:

[ex1.cc](#).

Definition at line 278 of file TraceFct.cc.

References `message()`.

#### 10.1.3.6 void TraceFct::exit ( int *exit\_code*, Exception & *ex* ) [static]

Exit a program, dumping the function stack.

##### Parameters

*exit\_code* the exit code to be returned to the system

*ex* The Exception Stack to print

This version of the exit function will print out the passed `Exception` stack.

Definition at line 404 of file TraceFct.cc.

### 10.1.3.7 void TraceFct::exit ( int *exit\_code*, const char \* *format*, ... ) [static]

Exit a program, dumping the function stack.

#### Parameters

*exit\_code* the exit code to be returned to the system  
*format* a printf style format string  
... additional arguments

Definition at line 367 of file TraceFct.cc.

Referenced by die(), and open().

### 10.1.3.8 static void TraceFct::exit ( int *exit\_code*, const string & *msg* ) [inline, static]

Exit a program, dumping the function stack.

#### Parameters

*exit\_code* the exit code to be returned to the system  
*msg* a message to print

Definition at line 163 of file TraceFct.h.

References exit().

Referenced by exit().

### 10.1.3.9 static void TraceFct::message ( Exception & *ex* ) [inline, static]

Print a message to the `tracefct` output stream.

#### Parameters

*ex* The exception to print

Definition at line 176 of file TraceFct.h.

#### 10.1.3.10 static void TraceFct::message ( const string & *msg* ) [inline, static]

Print a message to the `tracefct` output stream.

##### Parameters

*msg* a message to print

Definition at line 178 of file TraceFct.h.

References `message()`.

Referenced by `message()`.

#### 10.1.3.11 void TraceFct::message ( const char \* *format*, ... ) [static]

Print a formatted message to the `tracefct` output stream.

##### Parameters

*format* a printf style format string

... additional arguments

The message is passed in the same fashion as the arguments to `printf`, allowing formatted output.

##### Examples:

[ex1.cc](#).

Definition at line 521 of file TraceFct.cc.

Referenced by `dump_stack()`, `TraceFct()`, and `~TraceFct()`.

#### 10.1.3.12 void TraceFct::open ( const string & *file* ) [static]

redirect the `tracefct` output stream

##### Parameters

*file* the file to which messages are to be written

serves to change the `TraceFct` output stream to the specified file. Normally `TraceFct` writes to `stderr`. `TraceFct::exit()` *always* writes to `stderr`. To reset the output stream, see `TraceFct::close()`. If the current output stream is not `stderr`, it is automatically closed.

If the passed filename is the string `stderr`, `open()` will use `stderr`.

Upon error, it writes and exit to `stderr` and exits.

Definition at line 604 of file `TraceFct.cc`.

References `exit()`.

#### 10.1.3.13 static void TraceFct::println ( const string & msg ) [inline, static]

Print a message to the `tracefct` output stream.

##### Parameters

*msg* a message to print

Definition at line 185 of file `TraceFct.h`.

#### 10.1.3.14 static void TraceFct::println ( Exception & ex ) [inline, static]

Print a message to the `tracefct` output stream.

##### Parameters

*ex* the exception to print

Definition at line 183 of file `TraceFct.h`.

#### 10.1.3.15 static void TraceFct::println ( const char \* msg ) [inline, static]

Print a message to the `tracefct` output stream.

##### Parameters

*msg* a message to print

Definition at line 187 of file TraceFct.h.

**10.1.3.16** `void TraceFct::vmessage ( const char * format, va_list args )`  
`[static]`

Print a formatted message to the `tracefct` output stream using a `stdargs` argument list.

#### Parameters

*format* a printf style string

*args* q The error message is passed in the same fashion as the arguments to `vprintf`. Note that `vmessage` expects a `va_list` to be passed, rather than a variable argument list.

Definition at line 549 of file TraceFct.cc.

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

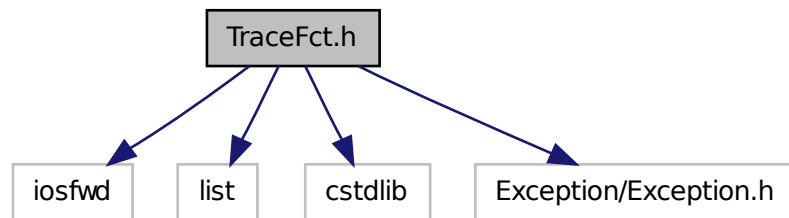
- [TraceFct.h](#)
- TraceFct.cc

## 11 File Documentation

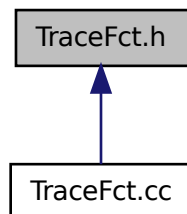
### 11.1 TraceFct.h File Reference

```
#include <iosfwd>
#include <list>
#include <cstdlib>
#include <Exception/Exception.h>
```

Include dependency graph for TraceFct.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [TraceFct](#)

### Defines

- #define [\\_tf\\_assert](#)(level, ex)
- #define [TF\\_ASSERT\\_LEVEL](#)

*This controls the level-based assertion macros. There are five levels; this macro should be assigned an integer corresponding to the maximum level for which assertions should be enabled. All assertions at levels less than or equal to this will be enabled. It is not defined by default.*

- `#define tf\_assert(ex) _tf_assert(990,ex)`  
*if assertion is not true, generate a message and exit with an error code of 990.*
- `#define tf\_assert1(ex) _tf_assert(991,ex)`  
*if assertion is not true and [TF\\_ASSERT\\_LEVEL](#) is greater or equal to 1, generate a message and exit with an error code of 991.*
- `#define tf\_assert2(ex) _tf_assert(992,ex)`  
*if assertion is not true and [TF\\_ASSERT\\_LEVEL](#) is greater or equal to than 1, generate a message and exit with an error code of 992.*
- `#define tf\_assert3(ex) _tf_assert(993,ex)`  
*if assertion is not true and [TF\\_ASSERT\\_LEVEL](#) is greater or equal to than 3, generate a message and exit with an error code of 993.*
- `#define tf\_assert4(ex) _tf_assert(994,ex)`  
*if assertion is not true and [TF\\_ASSERT\\_LEVEL](#) is greater or equal to than 4, generate a message and exit with an error code of 994.*
- `#define tf\_assert5(ex) _tf_assert(995,ex)`  
*if assertion is not true and [TF\\_ASSERT\\_LEVEL](#) is greater or equal to than 5, generate a message and exit with an error code of 995.*

### 11.1.1 Detailed Description

The class definitions

Definition in file [TraceFct.h](#).

### 11.1.2 Define Documentation

#### 11.1.2.1 `#define _tf_assert( level, ex )`

**Value:**

```
do
{
    if (!(ex))
        TraceFct::exit( level, "Assertion failed: file \"%s\", line %d\n%s\n", \

```



```
        __FILE__, __LINE__, #ex); \
    } while(0)
```

Definition at line 42 of file TraceFct.h.

#### 11.1.2.2 **#define tf\_assert( ex ) \_tf\_assert(990,ex)**

if *assertion* is not true, generate a message and exit with an error code of 990.

Definition at line 71 of file TraceFct.h.

#### 11.1.2.3 **#define tf\_assert1( ex ) \_tf\_assert(991,ex)**

if *assertion* is not true and [TF\\_ASSERT\\_LEVEL](#) is greater or equal to 1, generate a message and exit with an error code of 991.

Definition at line 82 of file TraceFct.h.

#### 11.1.2.4 **#define tf\_assert2( ex ) \_tf\_assert(992,ex)**

if *assertion* is not true and [TF\\_ASSERT\\_LEVEL](#) is greater or equal to than 1, generate a message and exit with an error code of 992.

Definition at line 91 of file TraceFct.h.

#### 11.1.2.5 **#define tf\_assert3( ex ) \_tf\_assert(993,ex)**

if *assertion* is not true and [TF\\_ASSERT\\_LEVEL](#) is greater or equal to than 3, generate a message and exit with an error code of 993.

Definition at line 100 of file TraceFct.h.

#### 11.1.2.6 **#define tf\_assert4( ex ) \_tf\_assert(994,ex)**

if *assertion* is not true and [TF\\_ASSERT\\_LEVEL](#) is greater or equal to than 4, generate a message and exit with an error code of 994.

Definition at line 109 of file TraceFct.h.

#### 11.1.2.7 `#define tf_assert5( ex ) _tf_assert(995,ex)`

if *assertion* is not true and `TF_ASSERT_LEVEL` is greater or equal to than 5, generate a message and exit with an error code of 995.

Definition at line 119 of file TraceFct.h.

#### 11.1.2.8 `#define TF_ASSERT_LEVEL`

This controls the level-based assertion macros. There are five levels; this macro should be assigned an integer corresponding to the maximum level for which assertions should be enabled. All assertions at levels less than or equal to this will be enabled. It is not defined by default.

Definition at line 61 of file TraceFct.h.

## 12 Example Documentation

### 12.1 ex1.cc

A simple example

```
using namespace std;
#include "TraceFct.h"

void f2( void )
{
    static int level = 0;
    TraceFct tf( "f2" );

    level++;
    tf.message( "message test: %d\n", level );
    tf.message( "1. multiline\n2. message test\n" );
    tf.message( "f2 stack dump:\n" );
    tf.dump_stack( );

    if ( level < 12 )
        f2();
}

void f1( bool die )
```

```
{
    TraceFct tf( "f1" );

    f2();

    if ( die )
        tf.exit( 1, "f1 exit: line1\nf1 exit: line2, extra newline\n" );
}

int main( int argc, char * argv[] )
{
    TraceFct tf( argv[0], 0, -1 );

    if ( argc > 1 )
        tf.open( argv[1] );

    f1( argc > 2 );

    tf.close( );

    tf.message( "main stack dump\n" );
    tf.dump_stack( );

    return EXIT_SUCCESS;
}
```

## 12.2 ex2.cc

An example using the Exception class

```
using namespace std;

#include <Exception.h>

#include "TraceFct.h"

void f2( )
{
    Exception ex( "My First Problem" );

    ex.set_message( "My Second Problem" );
    ex.set_message( "My Third Problem" );
    ex.set_message( "My Liney\nFourth Problem" );

    throw ex;
}

void f1( )
{
    TraceFct tf( "f1" );

    try {
        f2();
    } catch( Exception& e)
```

```
    {
        tf.exit( 1, e );
    }
}

int main( int argc, char * argv[] )
{
    TraceFct tf( argv[0], 0, -1 );

    fl( );

    tf.close( );

    tf.message( "main stack dump\n" );
    tf.dump_stack( );

    return EXIT_SUCCESS;
}
```

## Index

~TraceFct  
    TraceFct, [14](#)  
/data/pelf1/ Directory Reference, [8](#)  
/data/pelf1/dj/ Directory Reference, [6](#)  
/data/pelf1/dj/hd0/ Directory Reference, [7](#)  
/data/pelf1/dj/hd0/axaf/ Directory Reference, [5](#)  
/data/pelf1/dj/hd0/axaf/src/ Directory Reference, [9](#)  
/data/pelf1/dj/hd0/axaf/src/tracefctx/ Directory Reference, [11](#)  
/data/pelf1/dj/hd0/axaf/src/tracefctx/tracefctx/ Directory Reference, [10](#)  
\_tf\_assert  
    TraceFct.h, [22](#)  
close  
    TraceFct, [15](#)  
die  
    TraceFct, [15](#)  
dump\_stack  
    TraceFct, [16](#)  
exit  
    TraceFct, [16](#), [17](#)  
message  
    TraceFct, [17](#), [18](#)  
open  
    TraceFct, [18](#)  
println  
    TraceFct, [19](#)  
tf\_assert  
    TraceFct.h, [23](#)  
tf\_assert1  
    TraceFct.h, [23](#)  
tf\_assert2  
    TraceFct.h, [23](#)  
tf\_assert3  
    TraceFct.h, [23](#)  
tf\_assert4  
    TraceFct.h, [23](#)  
tf\_assert5  
    TraceFct.h, [24](#)  
TF\_ASSERT\_LEVEL  
    TraceFct.h, [24](#)  
TraceFct, [11](#)  
    ~TraceFct, [14](#)  
    close, [15](#)  
    die, [15](#)  
    dump\_stack, [16](#)  
    exit, [16](#), [17](#)  
    message, [17](#), [18](#)  
    open, [18](#)  
    println, [19](#)  
    TraceFct, [13](#), [14](#)  
    vmessage, [20](#)  
TraceFct.h, [20](#)  
    \_tf\_assert, [22](#)  
    tf\_assert, [23](#)  
    tf\_assert1, [23](#)  
    tf\_assert2, [23](#)  
    tf\_assert3, [23](#)  
    tf\_assert4, [23](#)  
    tf\_assert5, [24](#)  
    TF\_ASSERT\_LEVEL, [24](#)  
vmessage  
    TraceFct, [20](#)