# log4c

1.2.1

Generated by Doxygen 1.6.3

Mon Jul 5 22:53:07 2010

# Contents

# 1 Log4c : Logging for C Library

## 1.1 Summary

Log4c is a library of C for flexible logging to files, syslog and other destinations. It is modeled after the Log for Java library (`http://jakarta.apache.org/log4j/`), staying as close to their API as is reasonable. Here is a `short introduction` to Log4j which describes the API, and design rationale.

Mark Mendel started a parallel log4c projet with a different philosophy. The design is macro oriented, so much lighter and faster which perfect for kernel development.

Log4c is also available from SourceForge (`http://www.sourceforge.net/projects/log4c/`). This is work in progress.

## 1.2 Requirements

### 1.2.1 Platforms

log4c was successfully compiled and run on the following platforms :

- HP-UX release 11.00
- Tru 64 release 4.0F and 5.1
- Red Hat Linux Intel release 7.x, 8, 9
- Red Hat Enterprise Linux 3, 4
- Solaris Intel release 8, 9, 10
- FreeBSD 6.1-RELEASE
- AIX 5.3 (witk xlc compiler)
- Mac OS X
- Windows X

log4c should compile and run on the following platforms :

- The BSD family
- Other Linux distributions

### 1.2.2 Software

The following softwares are needed to generate log4c:

- `GCC` 3.0.1+, to generate log4c, but hopefully not to use it.
- `doxygen` 1.2.13+, a documentation system for C/C++ needed to generate the documentation.
- `graphviz`, the AT&T Graph Visualization Tools also needed to generate the documentation.

For the moment, log4c uses specific GCC extensions, like `__attribute__`, so you will need GCC to compile it. This will probably change one day.

## 1.3 Installation

### 1.3.1 Building from source tarballs

on SourceForge:

- `log4c-1.2.1.tar.gz`

The log4c package uses the GNU autotools compilation and installation framework. The following commands should build log4c on the supported platforms:

```
$ gzip -dc log4c-1.2.1.tar.gz | tar tvf -
$ cd log4c-1.2.1/
$ ./configure --prefix=/path/of/installation
$ make
$ make install
```

Checkout the `INSTALL` file for installation and the generated doxygen documentation for more information.

### 1.3.2 Using RPMs

FC7 RPMs on SourceForge:

- `log4c-1.2.1-1.i386.rpm`
- `log4c-devel-1.2.1-1.i386.rpm`
- `log4c-doc-1.2.1-1.i386.rpm`

The following command install the log4c RPMs :

```
$ sudo rpm -Uvh log4c-1.2.1.i386.rpm log4c-devel-1.2.1.i386.rpm
```

## 1.4 Configuration

### 1.4.1 Configuration files

log4c searches the folloging files to load its configuration:

- `${LOG4C_RCPATH}/log4crc`
- `${HOME}/`.log4crc
- ./log4crc

The environment variable `LOG4C_RCPATH` holds the prefix used for installation.

### 1.4.2 Configuration syntax

The `log4crc` configuration file uses an XML syntax. The root element is <log4c> and it can be used to control the configuration file version interface with the attribute `"version"`. The following 4 elements are supported: <config>, <category>, <appender> and <layout>.

- The <config> element controls the global log4c configuration. It has 3 sub elements. The <nocleanup> flag inhibits the log4c destructors routines. The <bufsize> element sets the buffer size used to format **log4c_logging_event_t** (p. 8) objects. If is set to 0, the allocation is dynamic (the <debug> element is currently unused).

- The <category> element has 3 possible attributes: the category `"name"`, the category `"priority"` and the category `"appender"`. Future versions will handle multple appenders per category.

- The <appender> element has 3 possible attributes: the appender `"name"`, the appender `"type"`, and the appender `"layout"`.

- The <layout> element has 2 possible attributes: the layout `"name"` and the layout `"type"`.

Here's the default `log4crc` configuration file:

This initial version of the log4c configuration file syntax is quite different from log4j. XML seemed the best choice to keep the log4j configuration power in a C API.

### 1.4.3 Environment variables

- `LOG4C_RCPATH` holds the path to the main `log4crc` configuration file
- `LOG4C_PRIORITY` holds the `"root"` category priority
- `LOG4C_APPENDER` holds the `"root"` category appender

## 1.5 Customization

This section will, one day, briefly describe how to define custom appenders and custom layouts. Be patient or checkout the source.

## 1.6 Thanks

Mark Mendel for his work on a previous version of log4c.

This project would not have existed without Ceki Gulcu, the creator and maintainer of `Log4j`, nor without Bastiaan Bakker, who intiated me with `Log4Cpp`.

Many thanks to

- Joel Schaubert for many contributions
- Robert Byrne for Windows port and also many contributions
- Olger Warnier for the Mac OS X port
- Jeff Smith for writing a primer on `how to use Log4c effectively`

## 1.7 Copyright



# 2 Todo List

**File appender.h** (p. **10**) the appender interface needs a better configuration system depending on the layout type. The udata field is a just a trick.

**File layout.h** (p. **39**) the layout interface needs a better configuration system depending on the layout type. The udata field is a just a trick.

a pattern layout would be welcomed !!

**Global log4c_category_get_chainedpriority** (p. **31**)**(const log4c_category_t *a_category)** the log4c_category_t is designed so that this method executes as quickly as possible. It could even be faster if the set priority was propagated through the children hierarchy of a category.

**Global log4c_category_set_appender** (p. **37**)**(log4c_category_t *a_category, struct __log4c_appe** need multiple appenders per category

**Class log4c_location_info_t** (p. **8**) this is not used

**Global log4c_logging_event_new** (p. **49**)**(const char *a_category, int a_priority, const char *a_mes** need to handle multi-threading (NDC)

# 3 Deprecated List

**Global log4c_appender_type_define (p. 12)(a_type)** This macro, and the static initialialization of appenders in general, is deprecated. Use rather the **log4c_appender_type_set()** (p. 16) function to initialize your appenders before calling **log4c_init()** (p. 39)

**Global log4c_layout_type_define (p. 41)(a_type)** This macro, and the static initialialization of layouts in general, is deprecated. Use rather the **log4c_layout_type_set()** (p. 44) function to initialize your appenders before calling **log4c_init()** (p. 39)

# 4 Bug List

**Global log4c_appender_append (p. 12)(log4c_appender_t *a_appender, log4c_logging_event_t (p** is this the right place to open an appender ?

**Global log4c_category_get (p. 30)(const char *a_name)** the root category name should be "" not "root". *

# 5 Data Structure Index

## 5.1 Data Structures

Here are the data structures with brief descriptions:



# 6 File Index

## 6.1 File List

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

# 7 Data Structure Documentation

## 7.1 log4c_appender_type Struct Reference

log4c appender type class

`#include <appender.h>`

### 7.1.1 Detailed Description

log4c appender type class Attributes description:

- `name` appender type name
- `open`
- `append`
- `close`

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

- **appender.h**

## 7.2 log4c_buffer_t Struct Reference

buffer object

`#include <buffer.h>`

### 7.2.1 Detailed Description

buffer object Attributes description:

- `size` current size of the buffer
- `maxsize` maximum size of the buffer. 0 means no limitation.
- `data` raw data

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

- **buffer.h**

## 7.3 log4c_layout_type Struct Reference

log4c layout type class

`#include <layout.h>`

### 7.3.1 Detailed Description

log4c layout type class Attributes description:

- `name` layout type name
- `format`

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

- **layout.h**

## 7.4 log4c_location_info_t Struct Reference

logging location information

`#include <location_info.h>`

### 7.4.1 Detailed Description

logging location information Attributes description:

- `loc_file` file name
- `loc_line` file line
- `loc_function` function name
- `loc_data` user data

**Todo**

this is not used

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

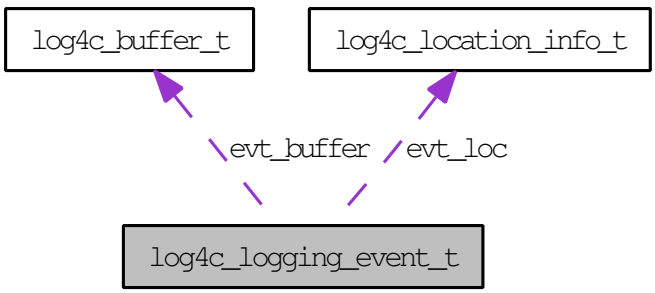
- **location_info.h**

## 7.5 log4c_logging_event_t Struct Reference

logging event object

`#include <logging_event.h>`

Collaboration diagram for log4c_logging_event_t:




### 7.5.1 Detailed Description

logging event object Attributes description:

- `evt_category` category name.
- `evt_priority` priority of logging event.
- `evt_msg` The application supplied message of logging event.

- `evt_buffer` a pre allocated buffer to be used by layouts to format in a multi-thread environment.
- `evt_rendered_msg` The application supplied message after layout format.
- `evt_timestamp` The number of seconds elapsed since the epoch (1/1/1970 00:00:00 UTC) until logging event was created.
- `evt_loc` The event's location information

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

- **logging_event.h**

## 7.6 log4c_rc_t Struct Reference

resource configuration object

`#include <rc.h>`

### 7.6.1 Detailed Description

resource configuration object Attributes description:

- `nocleanup` don't perform memory cleanup in log4c library destructor or in **log4c_fini()** (p. 38)
- `bufsize` maximum logging buffer size. 0 for no limits
- `debug` activate log4c debugging

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

- **rc.h**

## 7.7 log4c_rollingpolicy_type Struct Reference

log4c rollingpolicy type. Defines the interface a specific policy must provide to the rollingfile appender.

`#include <rollingpolicy.h>`

### 7.7.1 Detailed Description

log4c rollingpolicy type. Defines the interface a specific policy must provide to the rollingfile appender. Attributes description:

- `name` rollingpolicy type name
- `init()` init the rollingpolicy
- `is_triggering_event()`
- `rollover()`

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

- **rollingpolicy.h**

# 8 File Documentation

## 8.1 appender.h File Reference

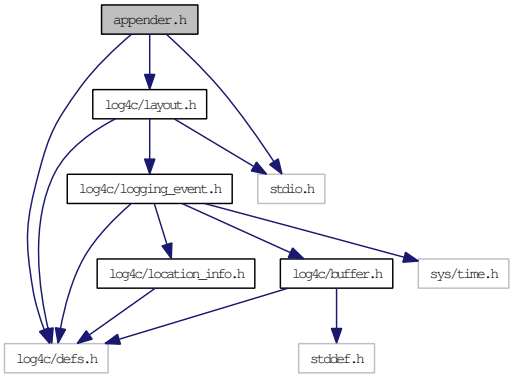
Implement this interface for your own strategies for printing log statements.

`#include <log4c/defs.h>`

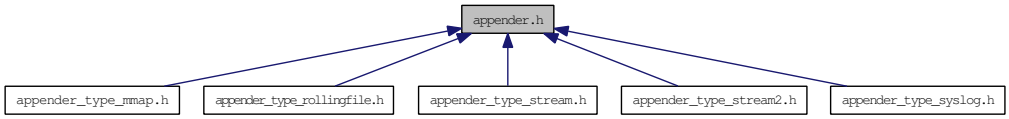
`#include <log4c/layout.h>`

`#include <stdio.h>`

Include dependency graph for appender.h:




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




### Data Structures

- struct **log4c_appender_type**
  *log4c appender type class*

### Defines

- #define **log4c_appender_type_define**(a_type)

### Typedefs

- typedef struct __log4c_appender **log4c_appender_t**
- typedef struct **log4c_appender_type log4c_appender_type_t**

  *log4c appender type class*

### Functions

- LOG4C_API const **log4c_appender_type_t** * **log4c_appender_type_get** (const char *a_name)
- LOG4C_API const **log4c_appender_type_t** * **log4c_appender_type_set** (const **log4c_appender_type_t** *a_type)
- LOG4C_API **log4c_appender_t** * **log4c_appender_get** (const char *a_name)
- LOG4C_API **log4c_appender_t** * **log4c_appender_new** (const char *a_name)
- LOG4C_API void **log4c_appender_delete** (**log4c_appender_t** *a_appender)
- LOG4C_API const char * **log4c_appender_get_name** (const **log4c_appender_t** *a_appender)
- LOG4C_API const **log4c_appender_type_t** * **log4c_appender_get_type** (const **log4c_appender_t** *a_appender)
- LOG4C_API const **log4c_layout_t** * **log4c_appender_get_layout** (const **log4c_appender_t** *a_appender)
- LOG4C_API void * **log4c_appender_get_udata** (const **log4c_appender_t** *a_appender)
- LOG4C_API const **log4c_appender_type_t** * **log4c_appender_set_type** (**log4c_appender_t** *a_appender, const **log4c_appender_type_t** *a_type)
- LOG4C_API void * **log4c_appender_set_udata** (**log4c_appender_t** *a_appender, void *a_udata)
- LOG4C_API const **log4c_layout_t** * **log4c_appender_set_layout** (**log4c_appender_t** *a_appender, const **log4c_layout_t** *a_layout)
- LOG4C_API int **log4c_appender_open** (**log4c_appender_t** *a_appender)
- LOG4C_API int **log4c_appender_append** (**log4c_appender_t** *a_appender, **log4c_logging_event_t** *a_event)
- LOG4C_API int **log4c_appender_close** (**log4c_appender_t** *a_appender)
- LOG4C_API void **log4c_appender_print** (const **log4c_appender_t** *a_appender, FILE *a_stream)
- LOG4C_API void **log4c_appender_types_print** (FILE *fp)

#### 8.1.1 Detailed Description

Implement this interface for your own strategies for printing log statements.

**Todo**

the appender interface needs a better configuration system depending on the layout type. The udata field is a just a trick.

### 8.1.2 Define Documentation

#### 8.1.2.1 #define log4c_appender_type_define(a_type)

Helper macro to define static appender types.

**Parameters**

***a_type*** the log4c_appender_type_t object to define

**Warning**

needs GCC support: otherwise this macro does nothing

**Deprecated**

This macro, and the static initialization of appenders in general, is deprecated. Use rather the **log4c_appender_type_set()** (p. 16) function to initialize your appenders before calling **log4c_init()** (p. 39)

### 8.1.3 Typedef Documentation

#### 8.1.3.1 typedef struct __log4c_appender log4c_appender_t

log4c appender class

#### 8.1.3.2 typedef struct log4c_appender_type log4c_appender_type_t

log4c appender type class

Attributes description:

- `name` appender type name
- `open`
- `append`
- `close`

### 8.1.4 Function Documentation

#### 8.1.4.1 LOG4C_API int log4c_appender_append (log4c_appender_t * *this*, log4c_logging_event_t * *a_event*)

log in appender specific way.

**Parameters**

***a_appender*** the log4c_appender object
***a_event*** the **log4c_logging_event_t** (p. 8) object to log.

**Bug**

is this the right place to open an appender ?

#### 8.1.4.2 LOG4C_API int log4c_appender_close (log4c_appender_t * *a_appender*)

closes the appender

**Parameters**

***a_appender*** the log4c_appender_t object

**Returns**

zero if successful, -1 otherwise

#### 8.1.4.3 LOG4C_API void log4c_appender_delete (log4c_appender_t * *a_appender*)

Destructor for log4c_appender_t.

#### 8.1.4.4 LOG4C_API log4c_appender_t* log4c_appender_get (const char * *a_name*)

Get a pointer to an existing appender.

**Parameters**

***a_name*** the name of the appender to return.

**Returns**

a pointer to an existing appender, or NULL if no appender with the specfied name exists.

#### 8.1.4.5 LOG4C_API const log4c_layout_t* log4c_appender_get_layout (const log4c_appender_t * *a_appender*)

**Parameters**

***a_appender*** the log4c_appender_t object

**Returns**

the appender layout

#### 8.1.4.6 LOG4C_API const char* log4c_appender_get_name (const log4c_appender_t * *a_appender*)

**Parameters**

***a_appender*** the log4c_appender_t object

**Returns**

the appender name

#### 8.1.4.7 LOG4C_API const log4c_appender_type_t* log4c_appender_get_type (const log4c_appender_t * *a_appender*)

**Parameters**

***a_appender*** the log4c_appender_t object

**Returns**

the appender operations

#### 8.1.4.8 LOG4C_API void* log4c_appender_get_udata (const log4c_appender_t * *a_appender*)

**Parameters**

***a_appender*** the log4c_appender_t object

**Returns**

the appender user data

#### 8.1.4.9 LOG4C_API log4c_appender_t* log4c_appender_new (const char * *a_name*)

Constructor for log4c_appender_t.

#### 8.1.4.10 LOG4C_API int log4c_appender_open (log4c_appender_t * *a_appender*)

opens the appender.

**Parameters**

***a_appender*** the log4c_appender_t object

#### 8.1.4.11 LOG4C_API void log4c_appender_print (const log4c_appender_t * *a_appender*, FILE * *a_stream*)

prints the appender on a stream

**Parameters**

***a_appender*** the log4c_appender_t object

***a_stream*** the stream

#### 8.1.4.12 LOG4C_API const log4c_layout_t* log4c_appender_set_layout (log4c_appender_t * *a_appender*, const log4c_layout_t * *a_layout*)

sets the appender layout

**Parameters**

***a_appender*** the log4c_appender_t object
***a_layout*** the new appender layout

**Returns**

the previous appender layout

#### 8.1.4.13 LOG4C_API const log4c_appender_type_t* log4c_appender_set_type (log4c_appender_t * *a_appender*, const log4c_appender_type_t * *a_type*)

sets the appender type

**Parameters**

***a_appender*** the log4c_appender_t object
***a_type*** the new appender type

**Returns**

the previous appender type

#### 8.1.4.14 LOG4C_API void* log4c_appender_set_udata (log4c_appender_t * *a_appender*, void * *a_udata*)

sets the appender user data

**Parameters**

***a_appender*** the log4c_appender_t object
***a_udata*** the new appender user data

**Returns**

the previous appender user data

#### 8.1.4.15 LOG4C_API const log4c_appender_type_t* log4c_appender_type_get (const char * *a_name*)

Get a pointer to an existing appender type.

**Parameters**

***a_name*** the name of the appender type to return.

**Returns**

a pointer to an existing appender type, or NULL if no appender type with the specified name exists.

#### 8.1.4.16 LOG4C_API const log4c_appender_type_t* log4c_appender_type_set (const log4c_appender_type_t * *a_type*)

Use this function to register an appender type with log4c. Once this is done you may refer to this type by name both programmatically and in the log4c configuration file.

**Parameters**

***a_type*** a pointer to the new appender type to set.

**Returns**

a pointer to the previous appender type of same name.

Example code fragment:

```
const log4c_appender_type_t log4c_appender_type_s13_file = {
  "s13_file",
  s13_file_open,
  s13_file_append,
  s13_file_close,
};

 log4c_appender_type_set(&log4c_appender_type_s13_file);
```

#### 8.1.4.17 LOG4C_API void log4c_appender_types_print (FILE * *fp*)

prints all the current registered appender types on a stream

**Parameters**

***fp*** the stream

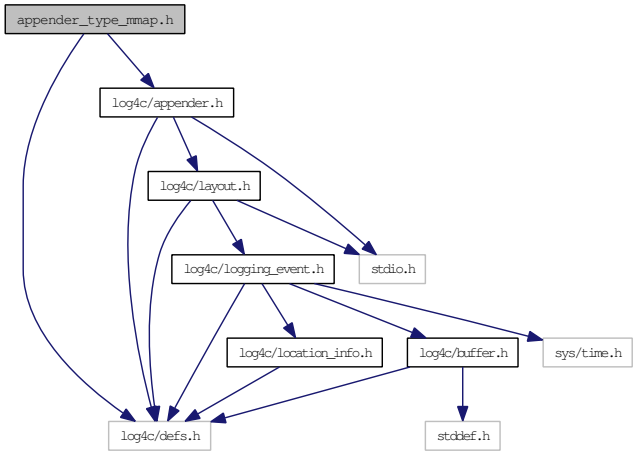
## 8.2 appender_type_mmap.h File Reference

Log4c mmap(2) appender interface.

`#include <log4c/defs.h>`

`#include <log4c/appender.h>`

Include dependency graph for appender_type_mmap.h:




### Variables

- __LOG4C_BEGIN_DECLS const **log4c_appender_type_t log4c_appender_type_mmap**

### 8.2.1 Detailed Description

Log4c mmap(2) appender interface. The mmap appender uses a fixed length memory mapped file for logging. The appender's name is used as the file name which will be opened and mapped to memory at first use. The memory mapped file is then used as a rotating buffer in which logging events are written.

The following examples shows how to define and use mmap appenders.

```
log4c_appender_t* myappender;

myappender = log4c_appender_get("myfile.log");
log4c_appender_set_type(myappender, &log4c_appender_type_mmap);
```

### Warning

the file is not created at first use. It should already exist and have a reasonable size, a mutilple of a page size.

### 8.2.2 Variable Documentation

#### 8.2.2.1 __LOG4C_BEGIN_DECLS const log4c_appender_type_t log4c_appender_type_mmap

Mmap appender type definition.

This should be used as a parameter to the **log4c_appender_set_type()** (p. 15) routine to set the type of the appender.

## 8.3 appender_type_rollingfile.h File Reference

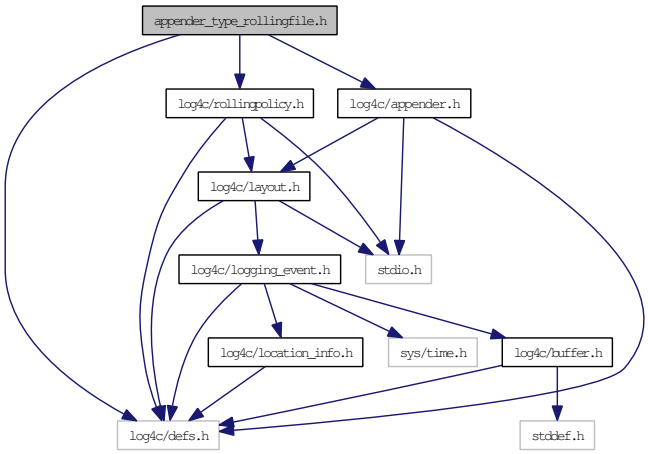
Log4c rolling file appender interface.

`#include <log4c/defs.h>`

`#include <log4c/appender.h>`

`#include <log4c/rollingpolicy.h>`

Include dependency graph for appender_type_rollingfile.h:




### Functions

- LOG4C_API rollingfile_udata_t * **rollingfile_make_udata** (void)
- LOG4C_API int **rollingfile_udata_set_logdir** (rollingfile_udata_t *rfudatap, char *logdir)
- LOG4C_API int **rollingfile_udata_set_files_prefix** (rollingfile_udata_t *rfudatap, char *prefix)
- LOG4C_API int **rollingfile_udata_set_policy** (rollingfile_udata_t *rfudatap, **log4c_rollingpolicy_t** *policyp)
- LOG4C_API const char * **rollingfile_udata_get_logdir** (rollingfile_udata_t *rfudatap)
- LOG4C_API const char * **rollingfile_udata_get_files_prefix** (rollingfile_udata_t *rfudatap)
- LOG4C_API long **rollingfile_get_current_file_size** (rollingfile_udata_t *rfudatap)

### Variables

- __LOG4C_BEGIN_DECLS LOG4C_API const **log4c_appender_type_t log4c_appender_type_rollingfile**

#### 8.3.1 Detailed Description

Log4c rolling file appender interface. The rolling file appender implements a logging mechanism of a list of files up to a maximum number.

The files are written by default to the current directory with logging names folowing the pattern log.1, log.2 etc. These parameters may be changed using the appropriate setter functions.

If the appender fails to open logfiles for writing then the messages are logged to stderr--it will continue to try to open the zero-th file for writing at rollover events so if it succeeds at some point to open that file the messages will start to appear therein and will no longer be sent to stderr.

Switching from logging from one file to the next is referred to as a 'rollover event'.

The policy that determines when a rollover event should happen is called a 'rolling policy'.

A mechanism is provided to allow different rolling policies to be defined.

Log4c ships with (and defaults to) the classic size-window rollover policy: this triggers rollover when files reach a maximum size. The first file in the list is always the current file; when a rollover event occurs files are shifted up by one position in the list--if the number of files in the list has already reached the max then the oldest file is rotated out of the window.

See the documentation in the **rollingpolicy_type_sizewin.h** (p. 58) file for more details on the size-win rollover policy.

### 8.3.2 Function Documentation

#### 8.3.2.1 LOG4C_API long rollingfile_get_current_file_size (rollingfile_udata_t * *rfudatap*)

Get the prefix string in this rolling file appender configuration.

**Parameters**

***rfudatap*** the rolling file appender configuration object.

**Returns**

the current size of the file being logged to.

#### 8.3.2.2 LOG4C_API rollingfile_udata_t* rollingfile_make_udata (void)

Get a new rolling file appender configuration object.

**Returns**

a new rolling file appender configuration object, otherwise NULL.

#### 8.3.2.3 LOG4C_API const char* rollingfile_udata_get_files_prefix (rollingfile_udata_t * *rfudatap*)

Get the prefix string in this rolling file appender configuration.

**Parameters**

***rfudatap*** the rolling file appender configuration object.

**Returns**

the prefix.

#### 8.3.2.4 LOG4C_API const char* rollingfile_udata_get_logdir (rollingfile_udata_t * *rfudatap*)

Get the logging directory in this rolling file appender configuration.

**Parameters**

***rfudatap*** the rolling file appender configuration object.

**Returns**

the logging directory.

#### 8.3.2.5 LOG4C_API int rollingfile_udata_set_files_prefix (rollingfile_udata_t * *rfudatap*, char * *prefix*)

Set the prefix string in this rolling file appender configuration.

**Parameters**

***rfudatap*** the rolling file appender configuration object.

***prefix*** the logging files prfix to use.

**Returns**

zero if successful, non-zero otherwise.

#### 8.3.2.6 LOG4C_API int rollingfile_udata_set_logdir (rollingfile_udata_t * *rfudatap*, char * *logdir*)

Set the logging directory in this rolling file appender configuration.

**Parameters**

***rfudatap*** the rolling file appender configuration object.

***logdir*** the logging directory to set.

**Returns**

zero if successful, non-zero otherwise.

#### 8.3.2.7 LOG4C_API int rollingfile_udata_set_policy (rollingfile_udata_t * *rfudatap*, log4c_rollingpolicy_t * *policyp*)

Set the rolling policy in this rolling file appender configuration.

**Parameters**

***rfudatap*** the rolling file appender configuration object.

***policyp*** the logging files prfix to use.

**Returns**

zero if successful, non-zero otherwise.

### 8.3.3 Variable Documentation

#### 8.3.3.1 __LOG4C_BEGIN_DECLS LOG4C_API const log4c_appender_type_t log4c_appender_type_rollingfile

rollingfile appender type definition.

This should be used as a parameter to the **log4c_appender_set_type()** (p. 15) routine to set the type of the appender.

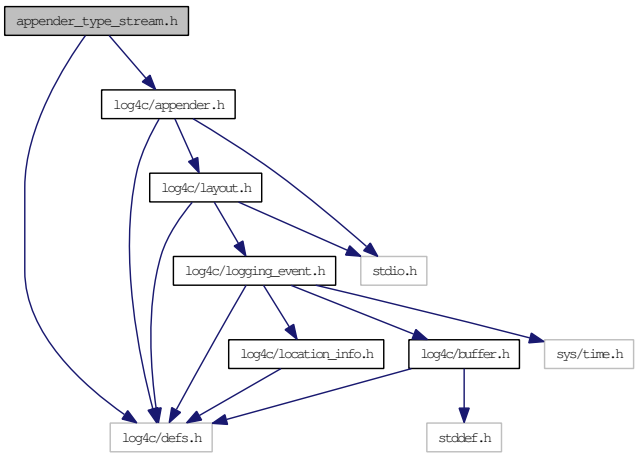
## 8.4 appender_type_stream.h File Reference

Log4c stream appender interface.

`#include <log4c/defs.h>`

`#include <log4c/appender.h>`

Include dependency graph for appender_type_stream.h:




### Variables

- __LOG4C_BEGIN_DECLS const **log4c_appender_type_t log4c_appender_type_stream**

### 8.4.1 Detailed Description

Log4c stream appender interface. The stream appender uses a file handle `FILE*` for logging. The appender's name is used as the file name which will be opened at first log. An appender can also be associated to an opened file handle using the **log4c_appender_set_udata()** (p. 15) method to update the appender user data field. In this last case, the appender name has no meaning. 2 default stream appenders are defined: `"stdout"` and `"stderr"`.

The following examples shows how to define and use stream appenders.

- the simple way

```
log4c_appender_t* myappender;
```

```
myappender = log4c_appender_get("myfile.log");
log4c_appender_set_type(myappender, &log4c_appender_type_stream);
```

- the sophisticated way

```
log4c_appender_t* myappender;

myappender = log4c_appender_get("myappender");

log4c_appender_set_type(myappender, &log4c_appender_type_stream);
log4c_appender_set_udata(myappender, fopen("myfile.log", "w"));
```

### 8.4.2 Variable Documentation

#### 8.4.2.1 __LOG4C_BEGIN_DECLS const log4c_appender_type_t log4c_appender_type_stream

Stream appender type definition.

This should be used as a parameter to the **log4c_appender_set_type()** (p. 15) routine to set the type of the appender.

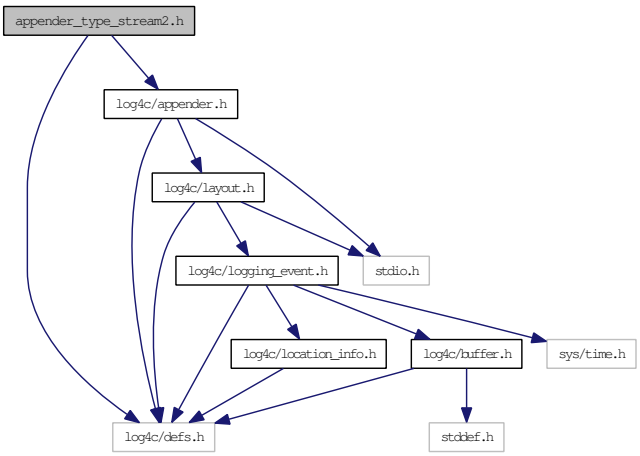
## 8.5 appender_type_stream2.h File Reference

Log4c stream2 appender interface.

`#include <log4c/defs.h>`

`#include <log4c/appender.h>`

Include dependency graph for appender_type_stream2.h:




### Functions

- LOG4C_API void **log4c_stream2_set_fp** (**log4c_appender_t** *a_this, FILE *fp)
- LOG4C_API FILE * **log4c_stream2_get_fp** (**log4c_appender_t** *a_this)
- LOG4C_API void **log4c_stream2_set_flags** (**log4c_appender_t** *a_this, int flags)
- LOG4C_API int **log4c_stream2_get_flags** (**log4c_appender_t** *a_this)

**Variables**

- __LOG4C_BEGIN_DECLS LOG4C_API const **log4c_appender_type_t log4c_appender_type_stream2**

### 8.5.1 Detailed Description

Log4c stream2 appender interface. The stream2 appender uses a file handle `FILE*` for logging. It can be used with `stdout`, `stderr` or a normal file. It is pretty primitive as it does not do file rotation, or have a maximum configurable file size etc. It improves on the stream appender in a few ways that make it a better starting point for new stream based appenders.

It enhances the stream appender by allowing the default file pointer to be used in buffered or unbuffered mode. Also when you set the file pointer stream2 will not attempt to close it on exit which avoids it fighting with the owner of the file pointer. stream2 is configured via setter functions--the udata is not exposed directly. This means that new options (eg. configure the open mode ) could be added to stream2 while maintaining backward compatability.

The appender can be used with default values, for example as follows:

```
log4c_appender_t* myappender;

myappender = log4c_appender_get("/var/logs/mylog.log");
log4c_appender_set_type(myappender,log4c_appender_type_get("stream2"));
```

In this case the appender will be configured automatically with default values:

- the filename is the same as the name of the appender, `"/var/logs/mymlog.log"`
- the file is opened in "w+" mode
- the default system buffer is used (cf; `setbuf()` ) in buffered mode

The stream2 appender can be configured by passing it a file pointer to use. In this case you manage the file pointer yourself--open, option setting, closing. If you set the file pointer log4c will not close the file on exiting--you must do this:

```
log4c_appender_t* myappender;
FILE * fp = fopen("myfile.log", "w");

myappender = log4c_appender_get("myappender");
log4c_appender_set_type(myappender, log4c_appender_type_get("stream2"));
log4c_stream2_set_fp(stream2_appender,myfp);
```

The default file pointer can be configured to use unbuffered mode. Buffered mode is typically 25-50% faster than unbuffered mode but unbuffered mode is useful if your preference is for a more synchronized log file:

```
log4c_appender_t* myappender;

myappender = log4c_appender_get("/var/logs/mylog.log");
log4c_appender_set_type(myappender,log4c_appender_type_get("stream2"));
log4c_stream2_set_flags(myappender, LOG4C_STREAM2_UNBUFFERED);
```

### 8.5.2 Function Documentation

#### 8.5.2.1 LOG4C_API int log4c_stream2_get_flags (log4c_appender_t * *a_this*)

Get the flags for this appender.

**Parameters**

***this*** a pointer to the appender

**Returns**

the flags for this appender. returns -1 if there was a problem.

#### 8.5.2.2 LOG4C_API FILE* log4c_stream2_get_fp (log4c_appender_t * *a_this*)

Get the file pointer for this appender.

**Parameters**

***this*** a pointer to the appender

**Returns**

the file pointer for this appender. If there's a problem returns NULL.

#### 8.5.2.3 LOG4C_API void log4c_stream2_set_flags (log4c_appender_t * *a_this*, int *flags*)

Set the flags for this appender.

**Parameters**

***this*** a pointer to the appender

***flags*** ar teh flags to set. These will overwrite the existing flags. Currently supported flags: LOG4C_STREAM2_UNBUFFERED

#### 8.5.2.4 LOG4C_API void log4c_stream2_set_fp (log4c_appender_t * *a_this*, FILE * *fp*)

Set the file pointer for this appender.

**Parameters**

***this*** a pointer to the appender

***fp*** the file pointer this appender will use. The caller is responsible for managing the file pointer (open, option setting, closing).

### 8.5.3 Variable Documentation

#### 8.5.3.1 __LOG4C_BEGIN_DECLS LOG4C_API const log4c_appender_type_t log4c_appender_type_stream2

Stream2 appender type definition.

This should be used as a parameter to the **log4c_appender_set_type()** (p. 15) routine to set the type of the appender.

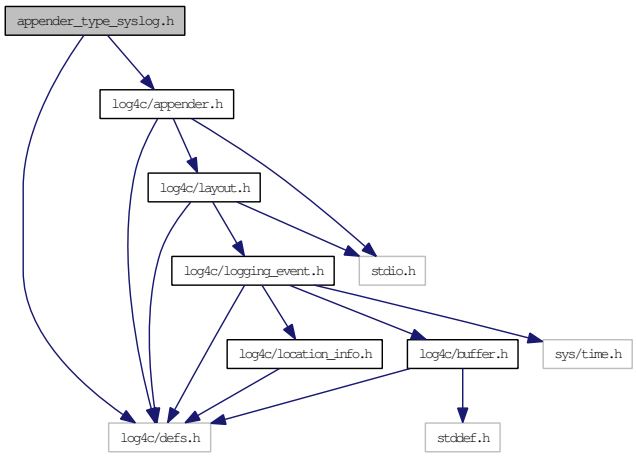
## 8.6 appender_type_syslog.h File Reference

Log4c syslog(3) appender interface.

`#include <log4c/defs.h>`

`#include <log4c/appender.h>`

Include dependency graph for appender_type_syslog.h:




### Variables

- __LOG4C_BEGIN_DECLS const **log4c_appender_type_t log4c_appender_type_syslog**

### 8.6.1 Detailed Description

Log4c syslog(3) appender interface. The syslog appender uses the syslog(3) interface for logging. The log4c priorities are mapped to the syslog priorities and the appender name is used as a syslog identifier. 1 default syslog appender is defined: `"syslog"`.

The following examples shows how to define and use syslog appenders.

```
log4c_appender_t* myappender;

myappender = log4c_appender_get("myappender");
log4c_appender_set_type(myappender, &log4c_appender_type_syslog);
```

### 8.6.2 Variable Documentation

#### 8.6.2.1 __LOG4C_BEGIN_DECLS const log4c_appender_type_t log4c_appender_type_syslog

Syslog appender type definition.

This should be used as a parameter to the **log4c_appender_set_type()** (p. 15) routine to set the type of the appender.

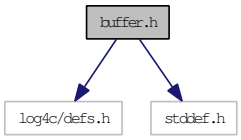
## 8.7 buffer.h File Reference

log4c buffer

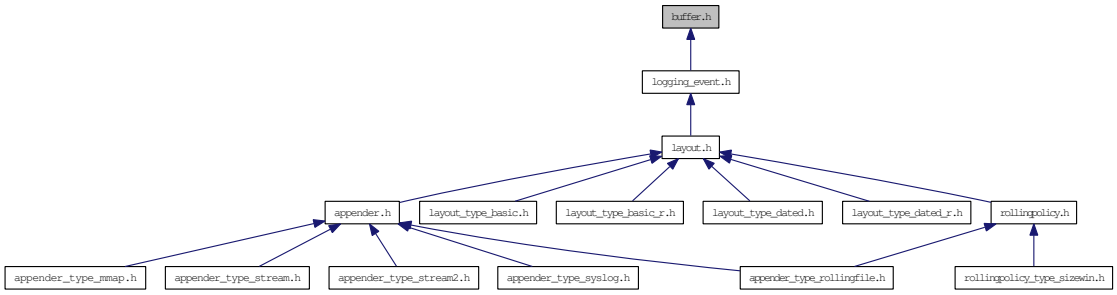
`#include <log4c/defs.h>`

`#include <stddef.h>`

Include dependency graph for buffer.h:




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




### Data Structures

- struct **log4c_buffer_t**
  *buffer object*

### 8.7.1 Detailed Description

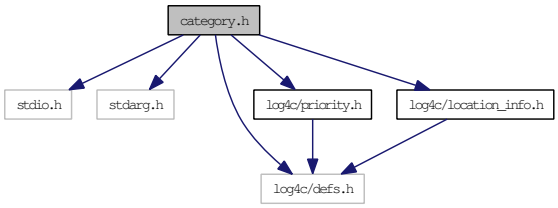
log4c buffer

## 8.8 category.h File Reference

central class in the log4c package.

```
#include <stdio.h>
#include <stdarg.h>
#include <log4c/defs.h>
#include <log4c/priority.h>
#include <log4c/location_info.h>
```

Include dependency graph for category.h:




### Defines

- #define **log4c_category_define**(a_category, a_name)

### Typedefs

- typedef struct __log4c_category **log4c_category_t**

### Functions

- LOG4C_API **log4c_category_t** * **log4c_category_get** (const char *a_name)
- LOG4C_API int **log4c_category_list** (**log4c_category_t** **a_cats, int a_ncats)
- LOG4C_API **log4c_category_t** * **log4c_category_new** (const char *a_name)
- LOG4C_API void **log4c_category_delete** (**log4c_category_t** *a_category)
- LOG4C_API const char * **log4c_category_get_name** (const **log4c_category_t** *a_category)
- LOG4C_API struct __log4c_appender * **log4c_category_get_appender** (const **log4c_category_t** *a_category)
- LOG4C_API int **log4c_category_get_additivity** (const **log4c_category_t** *a_category)
- LOG4C_API int **log4c_category_get_priority** (const **log4c_category_t** *a_category)
- LOG4C_API int **log4c_category_get_chainedpriority** (const **log4c_category_t** *a_category)
- LOG4C_API struct __log4c_appender * **log4c_category_set_appender** (**log4c_category_t** *a_category, struct __log4c_appender *a_appender)
- LOG4C_API int **log4c_category_set_priority** (**log4c_category_t** *a_category, int a_priority)
- LOG4C_API int **log4c_category_set_additivity** (**log4c_category_t** *a_category, int a_additivity)
- LOG4C_API void **log4c_category_print** (const **log4c_category_t** *a_category, FILE *a_stream)

- static int **log4c_category_is_priority_enabled** (const **log4c_category_t** *a_category, int a_priority)
- static int **log4c_category_is_fatal_enabled** (const **log4c_category_t** *a_category)
- static int **log4c_category_is_alert_enabled** (const **log4c_category_t** *a_category)
- static int **log4c_category_is_crit_enabled** (const **log4c_category_t** *a_category)
- static int **log4c_category_is_error_enabled** (const **log4c_category_t** *a_category)
- static int **log4c_category_is_warn_enabled** (const **log4c_category_t** *a_category)
- static int **log4c_category_is_notice_enabled** (const **log4c_category_t** *a_category)
- static int **log4c_category_is_info_enabled** (const **log4c_category_t** *a_category)
- static int **log4c_category_is_debug_enabled** (const **log4c_category_t** *a_category)
- static int **log4c_category_is_trace_enabled** (const **log4c_category_t** *a_category)
- static LOG4C_INLINE void **log4c_category_log** (const **log4c_category_t** *a_category, int a_priority, const char *a_format,...)
- static LOG4C_INLINE void **log4c_category_log_locinfo** (const **log4c_category_t** *a_category, const **log4c_location_info_t** *a_locinfo, int a_priority, const char *a_format,...)
- static LOG4C_INLINE void **log4c_category_fatal** (const **log4c_category_t** *a_category, const char *a_format,...)
- static LOG4C_INLINE void **log4c_category_alert** (const **log4c_category_t** *a_category, const char *a_format,...)
- static LOG4C_INLINE void **log4c_category_crit** (const **log4c_category_t** *a_category, const char *a_format,...)
- static LOG4C_INLINE void **log4c_category_error** (const **log4c_category_t** *a_category, const char *a_format,...)
- static LOG4C_INLINE void **log4c_category_warn** (const **log4c_category_t** *a_category, const char *a_format,...)
- static LOG4C_INLINE void **log4c_category_notice** (const **log4c_category_t** *a_category, const char *a_format,...)
- static LOG4C_INLINE void **log4c_category_info** (const **log4c_category_t** *a_category, const char *a_format,...)
- static LOG4C_INLINE void **log4c_category_debug** (const **log4c_category_t** *a_category, const char *a_format,...)
- static LOG4C_INLINE void **__log4c_category_trace** (const **log4c_category_t** *a_category, const char *a_format,...)

### 8.8.1 Detailed Description

central class in the log4c package. One of the distintive features of log4j (and hence log4c) are hierarchical categories and their evaluation.

### 8.8.2 Define Documentation

#### 8.8.2.1 #define log4c_category_define(a_category, a_name)

Helper macro to define static categories.

**Parameters**

***a_category*** the log4c_category_t pointer name

***a_name*** the category name

### 8.8.3 Typedef Documentation

#### 8.8.3.1 typedef struct __log4c_category log4c_category_t

log4c category class

### 8.8.4 Function Documentation

#### 8.8.4.1 static LOG4C_INLINE void __log4c_category_trace (const log4c_category_t * *a_category*, const char * *a_format*, ...) [static]

Log a message with trace priority.

**Parameters**

***a_category*** the log4c_category_t object

***a_format*** Format specifier for the string to write in the log file.

***...*** The arguments for a_format

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_TRACE.

#### 8.8.4.2 static LOG4C_INLINE void log4c_category_alert (const log4c_category_t * *a_category*, const char * *a_format*, ...) [static]

Log a message with alert priority.

**Parameters**

***a_category*** the log4c_category_t object

***a_format*** Format specifier for the string to write in the log file.

***...*** The arguments for a_format

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_ALERT.

#### 8.8.4.3 static LOG4C_INLINE void log4c_category_crit (const log4c_category_t * *a_category*, const char * *a_format*, ...) [static]

Log a message with crit priority.

**Parameters**

***a_category*** the log4c_category_t object

***a_format*** Format specifier for the string to write in the log file.

***...*** The arguments for a_format

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_CRIT.

#### 8.8.4.4 static LOG4C_INLINE void log4c_category_debug (const log4c_category_t * *a_category*, const char * *a_format*, ...) [static]

Log a message with debug priority.

**Parameters**

***a_category*** the log4c_category_t object

***a_format*** Format specifier for the string to write in the log file.

***...*** The arguments for a_format

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_DEBUG.

#### 8.8.4.5 LOG4C_API void log4c_category_delete (log4c_category_t * *a_category*)

Destructor for a log4c_category_t.

**Parameters**

***a_category*** the log4c_category_t object

#### 8.8.4.6 static LOG4C_INLINE void log4c_category_error (const log4c_category_t * *a_category*, const char * *a_format*, ...) [static]

Log a message with error priority.

**Parameters**

***a_category*** the log4c_category_t object

***a_format*** Format specifier for the string to write in the log file.

***...*** The arguments for a_format

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_ERROR.

#### 8.8.4.7 static LOG4C_INLINE void log4c_category_fatal (const log4c_category_t * *a_category*, const char * *a_format*, ...) [static]

Log a message with fatal priority.

**Parameters**

***a_category*** the log4c_category_t object

***a_format*** Format specifier for the string to write in the log file.

***...*** The arguments for a_format

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_FATAL.

#### 8.8.4.8 LOG4C_API log4c_category_t* log4c_category_get (const char * *a_name*)

Instantiate a log4c_category_t with name *name*. This method does not set priority of the category which is by default `LOG4C_PRIORITY_NOTSET`.

**Parameters**

***a_name*** The name of the category to retrieve.

**Bug**

the root category name should be "" not "root". *

#### 8.8.4.9 LOG4C_API int log4c_category_get_additivity (const log4c_category_t * *a_category*)

Get the additivity flag for this log4c_category_t..

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

the category additivity

#### 8.8.4.10 LOG4C_API struct __log4c_appender* log4c_category_get_appender (const log4c_category_t * *a_category*) [read]

Returns the Appender for this log4c_category_t, or NULL if no Appender has been set.

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

The Appender.

#### 8.8.4.11 LOG4C_API int log4c_category_get_chainedpriority (const log4c_category_t * *a_category*)

Starting from this category, search the category hierarchy for a set priority and return it. Otherwise, return the priority of the root category.

**Parameters**

***a_category*** the log4c_category_t object

**Todo**

the log4c_category_t is designed so that this method executes as quickly as possible. It could even be faster if the set priority was propagated through the children hierarchy of a category.

References LOG4C_PRIORITY_NOTSET, and LOG4C_PRIORITY_UNKNOWN.

#### 8.8.4.12 LOG4C_API const char* log4c_category_get_name (const log4c_category_t * *a_category*)

Return the category name.

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

the category name.

#### 8.8.4.13 LOG4C_API int log4c_category_get_priority (const log4c_category_t * *a_category*)

Returns the assigned Priority, if any, for this log4c_category_t.

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

Priority - the assigned Priority, can be LOG4C_PRIORITY_NOTSET

References LOG4C_PRIORITY_UNKNOWN.

#### 8.8.4.14 static LOG4C_INLINE void log4c_category_info (const log4c_category_t * *a_category*, const char * *a_format*, ...) [static]

Log a message with info priority.

**Parameters**

***a_category*** the log4c_category_t object

***a_format*** Format specifier for the string to write in the log file.

***...*** The arguments for a_format

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_INFO.

#### 8.8.4.15 static int log4c_category_is_alert_enabled (const log4c_category_t * *a_category*) [inline, static]

Return true if the category will log messages with priority `LOG4C_PRIORITY_ALERT`.

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

Whether the category will log.

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_ALERT.

#### 8.8.4.16 static int log4c_category_is_crit_enabled (const log4c_category_t * *a_category*) [inline, static]

Return true if the category will log messages with priority LOG4C_PRIORITY_CRIT.

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

Whether the category will log.

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_CRIT.

#### 8.8.4.17 static int log4c_category_is_debug_enabled (const log4c_category_t * *a_category*) [inline, static]

Return true if the category will log messages with priority LOG4C_PRIORITY_DEBUG.

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

Whether the category will log.

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_DEBUG.

#### 8.8.4.18 static int log4c_category_is_error_enabled (const log4c_category_t * *a_category*) [inline, static]

Return true if the category will log messages with priority LOG4C_PRIORITY_ERROR.

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

Whether the category will log.

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_ERROR.

#### 8.8.4.19 static int log4c_category_is_fatal_enabled (const log4c_category_t * *a_category*) [inline, static]

Return true if the category will log messages with priority LOG4C_PRIORITY_FATAL.

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

Whether the category will log.

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_FATAL.

#### 8.8.4.20 static int log4c_category_is_info_enabled (const log4c_category_t * *a_category*) [inline, static]

Return true if the category will log messages with priority LOG4C_PRIORITY_INFO.

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

Whether the category will log.

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_INFO.

#### 8.8.4.21 static int log4c_category_is_notice_enabled (const log4c_category_t * *a_category*) [inline, static]

Return true if the category will log messages with priority LOG4C_PRIORITY_NOTICE.

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

Whether the category will log.

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_NOTICE.

#### 8.8.4.22 static int log4c_category_is_priority_enabled (const log4c_category_t * *a_category*, int *a_priority*) [inline, static]

Returns true if the chained priority of the log4c_category_t is equal to or higher than given priority.

**Parameters**

***a_category*** the log4c_category_t object

***a_priority*** The priority to compare with.

**Returns**

whether logging is enable for this priority.

#### 8.8.4.23 static int log4c_category_is_trace_enabled (const log4c_category_t * *a_category*) [inline, static]

Return true if the category will log messages with priority LOG4C_PRIORITY_TRACE.

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

Whether the category will log.

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_TRACE.

#### 8.8.4.24 static int log4c_category_is_warn_enabled (const log4c_category_t * *a_category*) [inline, static]

Return true if the category will log messages with priority LOG4C_PRIORITY_WARN.

**Parameters**

***a_category*** the log4c_category_t object

**Returns**

Whether the category will log.

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_WARN.

#### 8.8.4.25 LOG4C_API int log4c_category_list (log4c_category_t ** *a_cats*, int *a_ncats*)

Fill in an array with the log4c categories.

**Parameters**

***a_cats*** array of categories that will be filled

***a_ncats*** number of categories in the array

**Returns**

-1 if it fails or the number of available categories in log4c.

#### 8.8.4.26 static LOG4C_INLINE void log4c_category_log (const log4c_category_t * *a_category*, int *a_priority*, const char * *a_format*, ...) [static]

Log a message with the specified priority.

**Parameters**

***a_category*** the log4c_category_t object

***a_priority*** The priority of this log message.

***a_format*** Format specifier for the string to write in the log file.

***...*** The arguments for a_format

References log4c_category_is_priority_enabled().

#### 8.8.4.27 static LOG4C_INLINE void log4c_category_log_locinfo (const log4c_category_t * *a_category*, const log4c_location_info_t * *a_locinfo*, int *a_priority*, const char * *a_format*, ...) [static]

Log a message with the specified priority and a user location info.

**Parameters**

***a_category*** the log4c_category_t object

***a_locinfo*** a user location info

***a_priority*** The priority of this log message.

***a_format*** Format specifier for the string to write in the log file.

***...*** The arguments for a_format

References log4c_category_is_priority_enabled().

#### 8.8.4.28 LOG4C_API log4c_category_t* log4c_category_new (const char * *a_name*)

Constructor for a log4c_category_t.

**Parameters**

***a_name*** the category name

**Returns**

a log4c_category object

**Warning**

this method should not be called directly. You should use the **log4c_category_get()** (p. 30) method in order to preserve the categories hierarchy.

References LOG4C_PRIORITY_NOTSET.

#### 8.8.4.29 static LOG4C_INLINE void log4c_category_notice (const log4c_category_t * *a_category*, const char * *a_format*, ...) [static]

Log a message with notice priority.

**Parameters**

***a_category*** the log4c_category_t object

***a_format*** Format specifier for the string to write in the log file.

***...*** The arguments for a_format

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_NOTICE.

#### 8.8.4.30 LOG4C_API void log4c_category_print (const log4c_category_t * *a_category*, FILE * *a_stream*)

prints the log4c_category_t object on a stream

**Parameters**

***a_category*** the log4c_category_t object

***a_stream*** The stream

#### 8.8.4.31 LOG4C_API int log4c_category_set_additivity (log4c_category_t * *a_category*, int *a_additivity*)

Sets a new additivity flag for this category.

**Parameters**

***a_category*** the log4c_category_t object

***a_additivity*** the new category additivity

**Returns**

the previous category additivity

#### 8.8.4.32 LOG4C_API struct __log4c_appender* log4c_category_set_appender (log4c_category_t * *this*, log4c_appender_t * *a_appender*) [read]

Sets a new appender for this category.

**Parameters**

***a_category*** the log4c_category_t object

***a_appender*** the new category appender

**Returns**

the previous category appender

**Todo**

need multiple appenders per category

#### 8.8.4.33 LOG4C_API int log4c_category_set_priority (log4c_category_t * *a_category*, int *a_priority*)

Sets a new priority of this category.

**Parameters**

***a_category*** the log4c_category_t object

***a_priority*** the new priority to set. Use LOG4C_PRIORITY_NOTSET to let the category use its parents priority as effective priority.

**Returns**

the previous category priority

References LOG4C_PRIORITY_UNKNOWN.

#### 8.8.4.34 static LOG4C_INLINE void log4c_category_warn (const log4c_category_t * *a_category*, const char * *a_format*, ...) [static]

Log a message with warn priority.

**Parameters**

***a_category*** the log4c_category_t object

***a_format*** Format specifier for the string to write in the log file.

***...*** The arguments for a_format

References log4c_category_is_priority_enabled(), and LOG4C_PRIORITY_WARN.

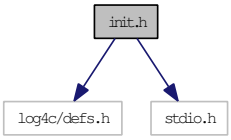
## 8.9 init.h File Reference

log4c constructors and destructors

`#include <log4c/defs.h>`

`#include <stdio.h>`

Include dependency graph for init.h:




**Functions**

- LOG4C_API int **log4c_init** (void)
- LOG4C_API int **log4c_fini** (void)

### 8.9.1 Detailed Description

log4c constructors and destructors

### 8.9.2 Function Documentation

#### 8.9.2.1 LOG4C_API int log4c_fini (void)

destructor

**Returns**

0 for success

References log4c_rc.

#### 8.9.2.2 LOG4C_API int log4c_init (void)

constructor

**Returns**

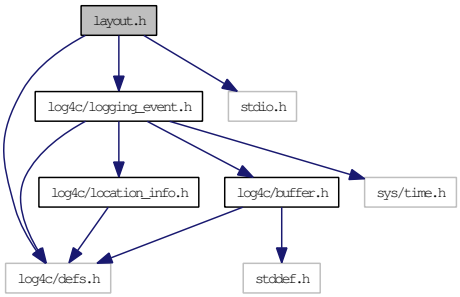
0 for success

## 8.10 layout.h File Reference

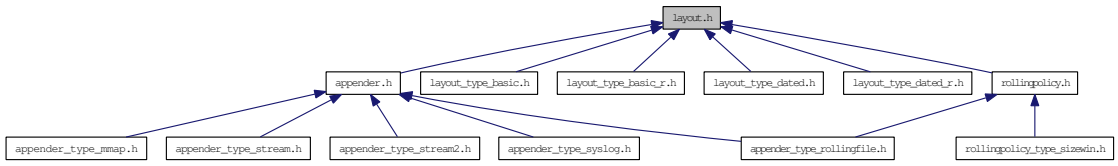
Interface for user specific layout format of log4c_logging_event events.

```
#include <log4c/defs.h>
#include <log4c/logging_event.h>
#include <stdio.h>
```

Include dependency graph for layout.h:




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




**Data Structures**

- struct **log4c_layout_type**

  *log4c layout type class*

### Defines

- #define **log4c_layout_type_define**(a_type)

### Typedefs

- typedef struct __log4c_layout **log4c_layout_t**
- typedef struct **log4c_layout_type** **log4c_layout_type_t**

  *log4c layout type class*

### Functions

- LOG4C_API const **log4c_layout_type_t** * **log4c_layout_type_get** (const char *a_name)
- LOG4C_API const **log4c_layout_type_t** * **log4c_layout_type_set** (const **log4c_layout_type_t** *a_type)
- LOG4C_API **log4c_layout_t** * **log4c_layout_get** (const char *a_name)
- LOG4C_API **log4c_layout_t** * **log4c_layout_new** (const char *a_name)
- LOG4C_API void **log4c_layout_delete** (**log4c_layout_t** *a_layout)
- LOG4C_API const char * **log4c_layout_get_name** (const **log4c_layout_t** *a_layout)
- LOG4C_API const **log4c_layout_type_t** * **log4c_layout_get_type** (const **log4c_layout_t** *a_layout)
- LOG4C_API const **log4c_layout_type_t** * **log4c_layout_set_type** (**log4c_layout_t** *a_layout, const **log4c_layout_type_t** *a_type)
- LOG4C_API void * **log4c_layout_get_udata** (const **log4c_layout_t** *a_layout)
- LOG4C_API void * **log4c_layout_set_udata** (**log4c_layout_t** *a_layout, void *a_udata)
- LOG4C_API const char * **log4c_layout_format** (const **log4c_layout_t** *a_layout, const **log4c_logging_event_t** *a_event)
- LOG4C_API void **log4c_layout_print** (const **log4c_layout_t** *a_layout, FILE *a_stream)
- LOG4C_API void **log4c_layout_types_print** (FILE *fp)

### 8.10.1 Detailed Description

Interface for user specific layout format of log4c_logging_event events.

**Todo**

the layout interface needs a better configuration system depending on the layout type. The udata field is a just a trick.

**Todo**

a pattern layout would be welcomed !!

### 8.10.2 Define Documentation

#### 8.10.2.1 #define log4c_layout_type_define(a_type)

Helper macro to define static layout types.

**Parameters**

***a_type*** the log4c_layout_type_t object to define

**Warning**

needs GCC support: otherwise this macro does nothing

**Deprecated**

This macro, and the static initialialization of layouts in general, is deprecated. Use rather the **log4c_layout_type_set()** (p. 44) function to initialize your appenders before calling **log4c_init()** (p. 39)

### 8.10.3 Typedef Documentation

#### 8.10.3.1 typedef struct __log4c_layout log4c_layout_t

log4c layout class

#### 8.10.3.2 typedef struct log4c_layout_type log4c_layout_type_t

log4c layout type class

Attributes description:

- `name` layout type name
- `format`

### 8.10.4 Function Documentation

#### 8.10.4.1 LOG4C_API void log4c_layout_delete (log4c_layout_t * *a_layout*)

Destructor for layout.

#### 8.10.4.2 LOG4C_API const char* log4c_layout_format (const log4c_layout_t * *a_layout*, const log4c_logging_event_t * *a_event*)

format a log4c_logging_event events to a string.

**Parameters**

***a_layout*** the log4c_layout_t object

***a_event*** a logging_event_t object

**Returns**

an appendable string.

#### 8.10.4.3 LOG4C_API log4c_layout_t* log4c_layout_get (const char * a_name)

Get a pointer to an existing layout.

**Parameters**

***a_name*** the name of the layout to return.

**Returns**

a pointer to an existing layout, or NULL if no layout with the specfied name exists.

#### 8.10.4.4 LOG4C_API const char* log4c_layout_get_name (const log4c_layout_t * a_layout)

**Parameters**

***a_layout*** the log4c_layout_t object

**Returns**

the layout name

#### 8.10.4.5 LOG4C_API const log4c_layout_type_t* log4c_layout_get_type (const log4c_layout_t * a_layout)

**Parameters**

***a_layout*** the log4c_layout_t object

**Returns**

a log4c_layout_type_t object

#### 8.10.4.6 LOG4C_API void* log4c_layout_get_udata (const log4c_layout_t * a_layout)

**Parameters**

***a_layout*** the log4c_layout_t object

**Returns**

the layout user data

#### 8.10.4.7 LOG4C_API log4c_layout_t* log4c_layout_new (const char * *a_name*)

Constructor for layout.

#### 8.10.4.8 LOG4C_API void log4c_layout_print (const log4c_layout_t * *a_layout*, FILE * *a_stream*)

prints the layout on a stream

**Parameters**

***a_layout*** the log4c_layout_t object

***a_stream*** the stream

#### 8.10.4.9 LOG4C_API const log4c_layout_type_t* log4c_layout_set_type (log4c_layout_t * *a_layout*, const log4c_layout_type_t * *a_type*)

sets the layout type

**Parameters**

***a_layout*** the log4c_layout_t object

***a_type*** the new layout type

**Returns**

the previous layout type

#### 8.10.4.10 LOG4C_API void* log4c_layout_set_udata (log4c_layout_t * *a_layout*, void * *a_udata*)

sets the layout user data

**Parameters**

***a_layout*** the log4c_layout_t object

***a_udata*** the new layout user data

**Returns**

the previous layout user data

#### 8.10.4.11 LOG4C_API const log4c_layout_type_t* log4c_layout_type_get (const char * *a_name*)

Get a pointer to an existing layout type.

**Parameters**

***a_name*** the name of the layout type to return.

**Returns**

a pointer to an existing layout type, or NULL if no layout type with the specified name exists.

#### 8.10.4.12 LOG4C_API const log4c_layout_type_t* log4c_layout_type_set (const log4c_layout_type_t * *a_type*)

Use this function to register a layout type with log4c. Once this is done you may refer to this type by name both programatically and in the log4c configuration file.

**Parameters**

***a_type*** a pointer to the new layout type to set.

**Returns**

a pointer to the previous layout type of same name.

Example code fragment:

```
const log4c_layout_type_t log4c_layout_type_xml = {
   "s13_xml",
   xml_format,
};

log4c_layout_type_set(&log4c_layout_type_xml);
```

#### 8.10.4.13 LOG4C_API void log4c_layout_types_print (FILE * *fp*)

prints all the current registered layout types on a stream

**Parameters**

***fp*** the stream

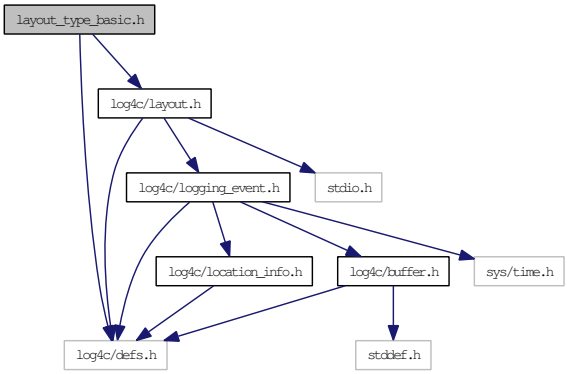
## 8.11 layout_type_basic.h File Reference

Implement a basic layout.

`#include <log4c/defs.h>`

`#include <log4c/layout.h>`

Include dependency graph for layout_type_basic.h:



### 8.11.1 Detailed Description

Implement a basic layout. In `log4j.PatternLayout` conventions, the basic layout has the following conversion pattern: `"%P %c - %m\n"`.

Where

- `"%P"` is the priority of the logging event
- `"%c"` is the category of the logging event
- `"%m"` is the application supplied message associated with the logging event

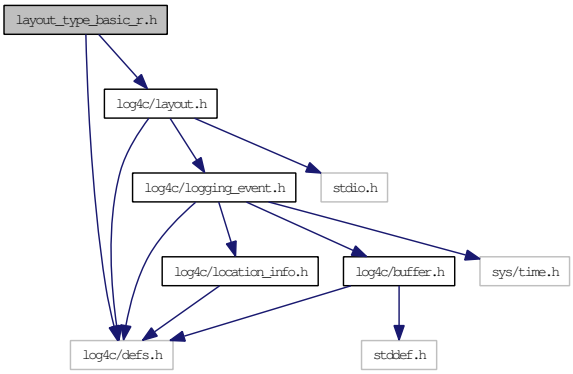
## 8.12 layout_type_basic_r.h File Reference

Implement a basic_r layout.

`#include <log4c/defs.h>`

`#include <log4c/layout.h>`

Include dependency graph for layout_type_basic_r.h:




### 8.12.1 Detailed Description

Implement a basic_r layout. In `log4j.PatternLayout` conventions, the basic_r layout has the following conversion pattern: `"%P %c - %m\n"`.

Where

- `"%P"` is the priority of the logging event
- `"%c"` is the category of the logging event
- `"%m"` is the application supplied message associated with the logging event

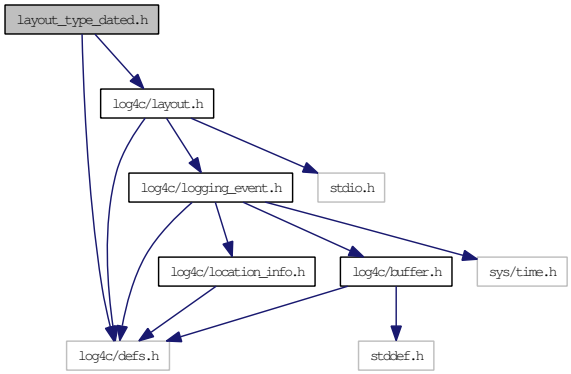
## 8.13 layout_type_dated.h File Reference

Implement a dated layout.

`#include <log4c/defs.h>`

`#include <log4c/layout.h>`

Include dependency graph for layout_type_dated.h:




### 8.13.1 Detailed Description

Implement a dated layout. In `log4j.PatternLayout` conventions, the dated layout has the following conversion pattern: `"%d %P %c - %m\n"`.

Where

- `"%d"` is the date of the logging event
- `"%P"` is the priority of the logging event
- `"%c"` is the category of the logging event
- `"%m"` is the application supplied message associated with the logging event

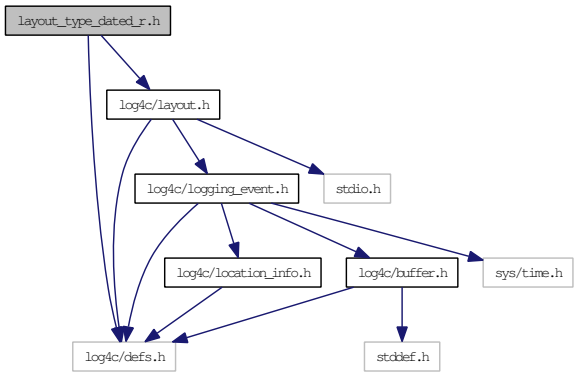
## 8.14 layout_type_dated_r.h File Reference

Implement a dated_r layout.

`#include <log4c/defs.h>`

`#include <log4c/layout.h>`

Include dependency graph for layout_type_dated_r.h:



### 8.14.1 Detailed Description

Implement a dated_r layout. In `log4j.PatternLayout` conventions, the dated_r layout has the following conversion pattern: `"%d %P %c - %m\n"`.

Where

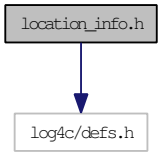
- `"%d"` is the date of the logging event
- `"%P"` is the priority of the logging event
- `"%c"` is the category of the logging event
- `"%m"` is the application supplied message associated with the logging event

## 8.15 location_info.h File Reference

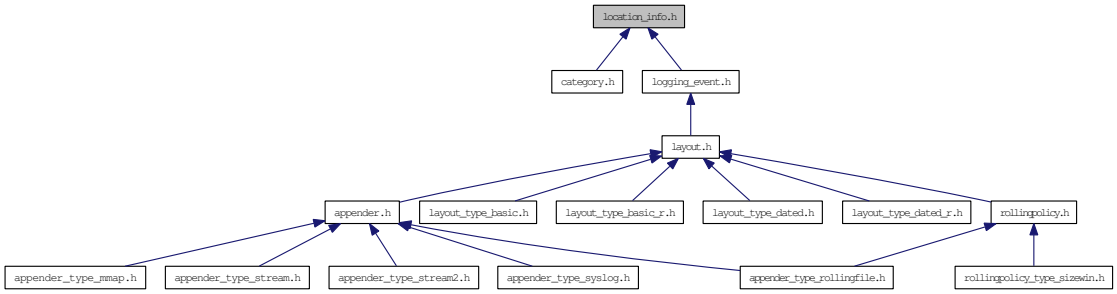
The internal representation of caller location information.

`#include <log4c/defs.h>`

Include dependency graph for location_info.h:




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




### Data Structures

- struct **log4c_location_info_t**
  *logging location information*

### Defines

- #define **LOG4C_LOCATION_INFO_INITIALIZER**(user_data) { __FILE__, __LINE__, "(nil)", user_data }
- #define **log4c_location** __log4c_location(__LINE__)

### 8.15.1 Detailed Description

The internal representation of caller location information. When a affirmative logging decision is made a **log4c_location_info_t** (p. 8) is created and is passed around the different log4c components.

### 8.15.2 Define Documentation

#### 8.15.2.1 #define log4c_location __log4c_location(__LINE__)

This macro returns the literal representation of a logging event location

#### 8.15.2.2 #define LOG4C_LOCATION_INFO_INITIALIZER(user_data) { __FILE__, __LINE__, "(nil)", user_data }

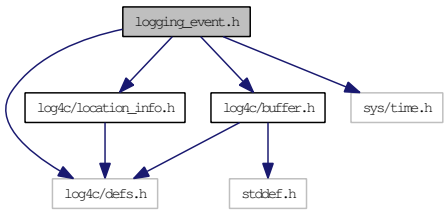
**log4c_location_info_t** (p. 8) initializer

## 8.16 logging_event.h File Reference

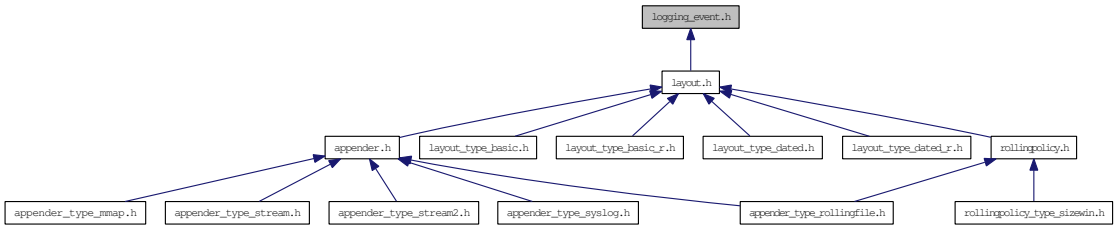
the internal representation of logging events.

```
#include <log4c/defs.h>
#include <log4c/buffer.h>
#include <log4c/location_info.h>
#include <sys/time.h>
```

Include dependency graph for logging_event.h:




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




### Data Structures

- struct **log4c_logging_event_t**

*logging event object*

**Functions**

- LOG4C_API **log4c_logging_event_t** ∗ **log4c_logging_event_new** (const char ∗a_category, int a_priority, const char ∗a_message)
- LOG4C_API void **log4c_logging_event_delete** (**log4c_logging_event_t** ∗a_event)

### 8.16.1 Detailed Description

the internal representation of logging events. When a affirmative logging decision is made a log4c_logging_event instance is created. This instance is passed around the different log4c components.

### 8.16.2 Function Documentation

#### 8.16.2.1 LOG4C_API void log4c_logging_event_delete (log4c_logging_event_t ∗ *a_event*)

Destructor for a logging event.

**Parameters**

***a_event*** the logging event object

#### 8.16.2.2 LOG4C_API log4c_logging_event_t∗ log4c_logging_event_new (const char ∗ *a_category*, int *a_priority*, const char ∗ *a_message*)

Constructor for a logging event.

**Parameters**

***a_category*** the category name

***a_priority*** the category initial priority

***a_message*** the message of this event

**Todo**

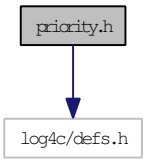
need to handle multi-threading (NDC)

## 8.17 priority.h File Reference

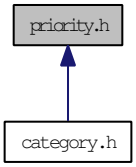
The priority class provides importance levels with which one can categorize log messages.

```
#include <log4c/defs.h>
```

Include dependency graph for priority.h:




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




**Enumerations**

- enum **log4c_priority_level_t** {
  **LOG4C_PRIORITY_FATAL** = 000,
  **LOG4C_PRIORITY_ALERT** = 100,
  **LOG4C_PRIORITY_CRIT** = 200,
  **LOG4C_PRIORITY_ERROR** = 300,
  **LOG4C_PRIORITY_WARN** = 400,
  **LOG4C_PRIORITY_NOTICE** = 500,
  **LOG4C_PRIORITY_INFO** = 600,
  **LOG4C_PRIORITY_DEBUG** = 700,
  **LOG4C_PRIORITY_TRACE** = 800,
  **LOG4C_PRIORITY_NOTSET** = 900,
  **LOG4C_PRIORITY_UNKNOWN** = 1000 }

**Functions**

- LOG4C_API const char * **log4c_priority_to_string** (int a_priority)
- LOG4C_API int **log4c_priority_to_int** (const char *a_priority_name)

### 8.17.1 Detailed Description

The priority class provides importance levels with which one can categorize log messages.

### 8.17.2 Enumeration Type Documentation

#### 8.17.2.1 enum log4c_priority_level_t

Predefined Levels of priorities. These correspond to the priority levels used by syslog(3).

**Enumerator:**

***LOG4C_PRIORITY_FATAL*** fatal
***LOG4C_PRIORITY_ALERT*** alert
***LOG4C_PRIORITY_CRIT*** crit
***LOG4C_PRIORITY_ERROR*** error
***LOG4C_PRIORITY_WARN*** warn
***LOG4C_PRIORITY_NOTICE*** notice
***LOG4C_PRIORITY_INFO*** info
***LOG4C_PRIORITY_DEBUG*** debug
***LOG4C_PRIORITY_TRACE*** trace
***LOG4C_PRIORITY_NOTSET*** notset
***LOG4C_PRIORITY_UNKNOWN*** unknown

### 8.17.3 Function Documentation

#### 8.17.3.1 LOG4C_API int log4c_priority_to_int (const char * *a_priority_name*)

**Parameters**

***a_priority_name*** a priority string name.

**Returns**

the given numeric value of the priority.

References LOG4C_PRIORITY_UNKNOWN.

#### 8.17.3.2 LOG4C_API const char* log4c_priority_to_string (int *a_priority*)

**Parameters**

***a_priority*** a numeric value of the priority.

**Returns**

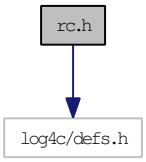
the given priority string name.

## 8.18 rc.h File Reference

log4c resource configuration

`#include <log4c/defs.h>`

Include dependency graph for rc.h:




### Data Structures

- struct **log4c_rc_t**
  *resource configuration object*

### Functions

- LOG4C_API int **log4c_load** (const char *a_filename)

### Variables

- LOG4C_API **log4c_rc_t** *const **log4c_rc**

#### 8.18.1 Detailed Description

log4c resource configuration

#### 8.18.2 Function Documentation

##### 8.18.2.1 LOG4C_API int log4c_load (const char * *a_filename*)

load log4c resource configuration file

**Parameters**

***a_filename*** name of file to load

#### 8.18.3 Variable Documentation

##### 8.18.3.1 LOG4C_API log4c_rc_t* const log4c_rc

default log4c resource configuration object

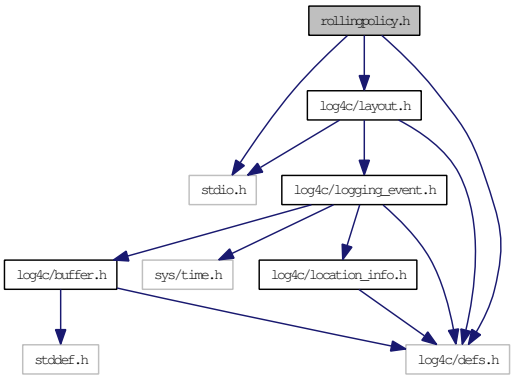
## 8.19 rollingpolicy.h File Reference

Log4c rolling policy interface. Defines the interface for managing and providing rolling policies.

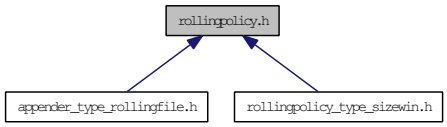
```
#include <stdio.h>
#include <log4c/defs.h>
```

```
#include <log4c/layout.h>
```

Include dependency graph for rollingpolicy.h:




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




### Data Structures

- struct **log4c_rollingpolicy_type**

  *log4c rollingpolicy type. Defines the interface a specific policy must provide to the rollingfile appender.*

### Defines

- #define **ROLLINGPOLICY_ROLLOVER_ERR_CAN_LOG** 0x05

### Typedefs

- typedef struct __log4c_rollingpolicy **log4c_rollingpolicy_t**
- typedef struct **log4c_rollingpolicy_type** **log4c_rollingpolicy_type_t**

  *log4c rollingpolicy type. Defines the interface a specific policy must provide to the rollingfile appender.*

### Functions

- LOG4C_API **log4c_rollingpolicy_t** * **log4c_rollingpolicy_get** (const char *policy_name)
- LOG4C_API const **log4c_rollingpolicy_type_t** * **log4c_rollingpolicy_type_set** (const **log4c_rollingpolicy_type_t** *a_type)

- LOG4C_API void **log4c_rollingpolicy_set_udata** (**log4c_rollingpolicy_t** *policyp, void *udatap)
- LOG4C_API int **log4c_rollingpolicy_init** (**log4c_rollingpolicy_t** *policyp, rollingfile_udata_t *rfup)
- LOG4C_API int **log4c_rollingpolicy_fini** (**log4c_rollingpolicy_t** *a_this)
- LOG4C_API int **log4c_rollingpolicy_is_triggering_event** (**log4c_rollingpolicy_t** *policyp, const **log4c_logging_event_t** *evtp, long current_file_size)
- LOG4C_API const **log4c_rollingpolicy_type_t** * **log4c_rollingpolicy_set_type** (**log4c_rollingpolicy_t** *a_rollingpolicy, const **log4c_rollingpolicy_type_t** *a_type)
- LOG4C_API const **log4c_rollingpolicy_type_t** * **log4c_rollingpolicy_type_get** (const char *a_name)
- LOG4C_API void * **log4c_rollingpolicy_get_udata** (const **log4c_rollingpolicy_t** *policyp)
- LOG4C_API rollingfile_udata_t * **log4c_rollingpolicy_get_rfudata** (const **log4c_rollingpolicy_t** *policyp)

### 8.19.1 Detailed Description

Log4c rolling policy interface. Defines the interface for managing and providing rolling policies. A rolling policy is used to confogure a rollingfile appender to tell it when to trigger a rolover event.

### 8.19.2 Define Documentation

#### 8.19.2.1 #define ROLLINGPOLICY_ROLLOVER_ERR_CAN_LOG 0x05

Effect a rollover according to policyp on the given file stream.

**Parameters**

***policyp*** pointer to the rolling policy

***fp*** filestream to rollover.

**Returns**

zero if successful, non-zero otherwise. The policy can return an indication that something went wrong but that the rollingfile appender can stull go ahead and log by returning an error code <= ROLLINGPOLICY_ROLLOVER_ERR_CAN_LOG. Anything greater than means that the rolling file appender will not try to log it's message.

### 8.19.3 Typedef Documentation

#### 8.19.3.1 typedef struct __log4c_rollingpolicy log4c_rollingpolicy_t

log4c rollingpolicy type

#### 8.19.3.2 typedef struct log4c_rollingpolicy_type log4c_rollingpolicy_type_t

log4c rollingpolicy type. Defines the interface a specific policy must provide to the rollingfile appender.

Attributes description:

- `name` rollingpolicy type name
- `init()` init the rollingpolicy
- `is_triggering_event()`
- `rollover()`

### 8.19.4 Function Documentation

#### 8.19.4.1 LOG4C_API int log4c_rollingpolicy_fini (log4c_rollingpolicy_t * *a_this*)

Call the un initialization code of a rolling policy. This will call the fini routine of the particular rollingpolicy type to allow it to free up resources. If the call to fini in the rollingpolicy type fails then the rollingpolicy is not uninitialized. Try again later model...

**Parameters**

***policyp*** pointer to the rolling policy

**Returns**

zero if successful, non-zero otherwise.

#### 8.19.4.2 LOG4C_API log4c_rollingpolicy_t* log4c_rollingpolicy_get (const char * *policy_name*)

Get a new rolling policy

**Parameters**

***policy_name*** a name for the policy

**Returns**

a new rolling policy, otherwise NULL.

#### 8.19.4.3 LOG4C_API rollingfile_udata_t* log4c_rollingpolicy_get_rfudata (const log4c_rollingpolicy_t * *policyp*)

Get the rollingfile appender associated with this policy.

**Parameters**

***policyp*** pointer to the rolling policy

**Returns**

pointer to the rolling file appender associated with this policy

#### 8.19.4.4 LOG4C_API void* log4c_rollingpolicy_get_udata (const log4c_rollingpolicy_t * *policyp*)

Get the rolling policy configuration.

**Parameters**

***policyp*** pointer to the rolling policy

**Returns**

pointer to the rolling policy configuration.

#### 8.19.4.5 LOG4C_API int log4c_rollingpolicy_init (log4c_rollingpolicy_t * *policyp*, rollingfile_udata_t * *rfup*)

Call the initialization code of a rolling policy.

**Parameters**

***policyp*** pointer to the rolling policy

***app*** the rolling appender this policy is used with

**Returns**

zero if successful, non-zero otherwise.

#### 8.19.4.6 LOG4C_API int log4c_rollingpolicy_is_triggering_event (log4c_rollingpolicy_t * *policyp*, const log4c_logging_event_t * *evtp*, long *current_file_size*)

Determine if a logging event should trigger a rollover according to the given policy.

**Parameters**

***policyp*** pointer to the rolling policy

***evtp*** the logging event pointer.

***current_file_size*** the size of the current file being logged to.

**Returns**

non-zero if rollover required, zero otherwise.

#### 8.19.4.7 LOG4C_API const log4c_rollingpolicy_type_t* log4c_rollingpolicy_set_type (log4c_rollingpolicy_t * *a_rollingpolicy*, const log4c_rollingpolicy_type_t * *a_type*)

sets the rolling policy type

**Parameters**

***a_rollingpolicy*** the log4c_rollingpolicy_t object

***a_type*** the new rollingpolicy type

**Returns**

the previous appender type

#### 8.19.4.8 LOG4C_API void log4c_rollingpolicy_set_udata (log4c_rollingpolicy_t * *policyp*, void * *udatap*)

Configure a rolling policy with a specific policy.

**Parameters**

***policyp*** pointer to the rolling policy

***udatap*** a specific policy type, for example sizewin.

**Returns**

zero if successful, non-zero otherwise.

#### 8.19.4.9 LOG4C_API const log4c_rollingpolicy_type_t* log4c_rollingpolicy_type_get (const char * *a_name*)

Get a pointer to an existing rollingpolicy type.

**Parameters**

***a_name*** the name of the rollingpolicy type to return.

**Returns**

a pointer to an existing rollingpolicy type, or NULL if no rollingpolicy type with the specified name exists.

#### 8.19.4.10 LOG4C_API const log4c_rollingpolicy_type_t* log4c_rollingpolicy_type_set (const log4c_rollingpolicy_type_t * *a_type*)

Use this function to register a rollingpolicy type with log4c. Once this is done you may refer to this type by name both programmatically and in the log4c configuration file.

**Parameters**

***a_type*** a pointer to the new rollingpolicy type to register.

**Returns**

a pointer to the previous rollingpolicy type of same name.

Example code fragment:

```
const log4c_rollingpolicy_type_t log4c_rollingpolicy_type_sizewin = {
  "sizewin",
  sizewin_init,
  sizewin_is_triggering_event,
  sizewin_rollover
};

log4c_rollingpolicy_type_set(&log4c_rollingpolicy_type_sizewin);
```

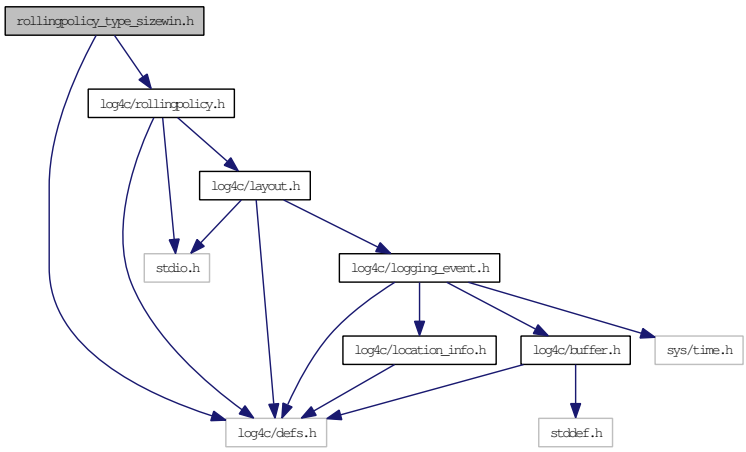
## 8.20 rollingpolicy_type_sizewin.h File Reference

Log4c rolling file size-win interface. Log4c ships with (and defaults to) the classic size-window rollover policy: this triggers rollover when files reach a maximum size. The first file in the list is always the current file; when a rollover event occurs files are shifted up by one position in the list--if the number of files in the list has already reached the max then the oldest file is rotated out of the window.

`#include <log4c/defs.h>`

`#include <log4c/rollingpolicy.h>`

Include dependency graph for rollingpolicy_type_sizewin.h:




### Typedefs

- typedef struct __sizewin_udata **rollingpolicy_sizewin_udata_t**

### Functions

- LOG4C_API **rollingpolicy_sizewin_udata_t** * **sizewin_make_udata** (void)
- LOG4C_API int **sizewin_udata_set_file_maxsize** (**rollingpolicy_sizewin_udata_t** *swup, long max_size)
- LOG4C_API int **sizewin_udata_set_max_num_files** (**rollingpolicy_sizewin_udata_t** *swup, long max_num)
- LOG4C_API int **sizewin_udata_set_appender** (**rollingpolicy_sizewin_udata_t** *swup, **log4c_appender_t** *app)

### 8.20.1 Detailed Description

Log4c rolling file size-win interface. Log4c ships with (and defaults to) the classic size-window rollover policy: this triggers rollover when files reach a maximum size. The first file in the list is always the current file; when a rollover event occurs files are shifted up by one position in the list--if the number of files in the list has already reached the max then the oldest file is rotated out of the window. If the max file size is set to zero, this means 'no-limit'.

The default parameters for the size-win policy are 5 files of maximum size of 20kilobytes each. These parameters may be changed using the appropriate setter functions.

### 8.20.2 Typedef Documentation

#### 8.20.2.1 typedef struct __sizewin_udata rollingpolicy_sizewin_udata_t

log4c size-win rolling policy type

### 8.20.3 Function Documentation

#### 8.20.3.1 LOG4C_API rollingpolicy_sizewin_udata_t* sizewin_make_udata (void)

Get a new size-win rolling policy

**Returns**

a new size-win rolling policy, otherwise NULL.

#### 8.20.3.2 LOG4C_API int sizewin_udata_set_appender (rollingpolicy_sizewin_udata_t * *swup*, log4c_appender_t * *app*)

Set the rolling file appender in this rolling policy configuration.

**Parameters**

***swup*** the size-win configuration object.

***app*** the rolling file appender to set.

**Returns**

zero if successful, non-zero otherwise.

#### 8.20.3.3 LOG4C_API int sizewin_udata_set_file_maxsize (rollingpolicy_sizewin_udata_t * *swup*, long *max_size*)

Set the maximum file size in this rolling policy configuration.

**Parameters**

***swup*** the size-win configuration object.

***max_size*** the approximate maximum size any logging file will attain. If you set zero then it means 'no-limit' and so only one file of unlimited size will be used for logging.

**Returns**

zero if successful, non-zero otherwise.

#### 8.20.3.4 LOG4C_API int sizewin_udata_set_max_num_files (rollingpolicy_sizewin_udata_t * *swup*, long *max_num*)

Set the maximum number of filesin this rolling policy configuration.

**Parameters**

***swup*** the size-win configuration object.

***max_num*** the maximum number of files in the list.

**Returns**

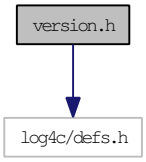
zero if successful, non-zero otherwise.

## 8.21 version.h File Reference

log4c version information

`#include <log4c/defs.h>`

Include dependency graph for version.h:




**Defines**

- #define **LOG4C_MAJOR_VERSION** 1
- #define **LOG4C_MINOR_VERSION** 2
- #define **LOG4C_MICRO_VERSION** 1

**Functions**

- const char * **log4c_version** (void)

**Variables**

- const int **log4c_major_version**
- const int **log4c_minor_version**
- const int **log4c_micro_version**

### 8.21.1 Detailed Description

log4c version information

### 8.21.2 Define Documentation

#### 8.21.2.1 #define LOG4C_MAJOR_VERSION 1

constant macro holding the major version of log4c

#### 8.21.2.2 #define LOG4C_MICRO_VERSION 1

constant macro holding the micro version of log4c

#### 8.21.2.3 #define LOG4C_MINOR_VERSION 2

constant macro holding the minor version of log4c

### 8.21.3 Function Documentation

#### 8.21.3.1 const char* log4c_version (void)

**Returns**

a string containing the full log4c version

### 8.21.4 Variable Documentation

#### 8.21.4.1 const int log4c_major_version

constant variable holding the major version of log4c

#### 8.21.4.2 const int log4c_micro_version

constant variable holding the micro version of log4c

#### 8.21.4.3 const int log4c_minor_version

constant variable holding the minor version of log4c

# Index