rlog

1.3

Generated by Doxygen 1.5.5

Mon Mar 17 08:21:51 2008

Contents

Chapter 1

RLog - a C++ logging library

1.1 Introduction

RLog provides a flexible message logging facility for C++ programs and libraries. It is meant to be fast enough to leave in production code.

RLog provides macros which are similar to Qt's debug macros, which are similar to simple printf() statements:

```
void func(int foo)
{
    rDebug("foo = %i", foo);
    int ans = 6 * 9;
    if(ans != 42)
        rWarning("ans = %i, expecting 42", ans);
    rError("I'm sorry %s, I can't do that (error code %i)", name, errno);
}
```

The difference to Qt's macros is that the log messages are considered *publishers* and there can be any number of *subscribers* to log messages. Subscribers may choose which messages they want to receive in a number of different ways:

- subscribe to messages to a particular *channel*. Channels are hierarchical can be easily created. See RLog Channels.
- subscribe to anything from a particular *component*. See RLog Components.
- subscribe to messages from a particular file name within a component.

If there are no subscribers to a particular logging statement, that statement can be said to be *dormant*. RLog is optimized to minimize overhead of dormant logging statements, with the goal of allowing logging to be left in release versions of software. This way if problems show up in production code, it is possible to activate logging statements in real time to aid debugging.

As an indication of just how cheap a dormant logging statement is, on a Pentium-4 class CPU with g++ 3.3.1, a dormant log in a tight loop adds on the order of 2-6 (two to six) clock cycles of overhead (1). By comparison a simple logging function such as Qt's qDebug() adds about 1000 (a thousand) clock cycles of overhead - even when messages are being thrown away.

In addition, logging statements in RLog can be individually activated at run-time without affecting any other statements, allowing targeted log reporting.

(1) The first time a logging statement is encountered, it must be registered in order to determine if there are any subscribers. So there is additional overhead the first time a statement is encountered.

1.2 Using RLog

In order to begin using RLog in your code, you should do the following:

- define RLOG_COMPONENT in your build environment. Eg: librlog is built with -DRLOG_COMPONENT="rlog". You should use a unique name for your program or library (do not use "rlog"). If your program is made up of separate components, then you can define RLOG_COMPONENT as a different name for each component.

- (optional) add a call to RLogInit() in your main program startup code. This is not a requirement, however not including it may reduce functionality of external rlog modules.

- link with librlog

- add subscribers (rlog::StdioNode , rlog::SyslogNode , or your own) to catch any messages you are interested in.

1.3 Requirements

RLog has been tested on the following systems (all releases may not have been tested on all systems):

Platform	Operating System	Compiler	Notes
ix86	SuSE 9.2	GNU G++ 3.3.4	binary RPM available
	SuSE 9.0	Intel ICC 8.0	last test was prior to RLog 1.3.4
	RedHat 7.3	GNU G++ 2.96	binary RPM available
	OpenBSD 3.4	GNU G++ 2.95.3	Tested with 1.3.5
	FreeBSD 4.10-beta	GNU G++ 2.95.4	Support added in 1.3.6 release
sparc	Solaris 5.9	GNU G++ 3.3.2	
PowerPC	Darwin 5.5	gcc-932.1	Support added in 1.3.6 release

To build development versions, you will also need the GNU autoconf tools (with automake and libtool). Documentation is built using Doxygen.

1.4 Downloads

RLog is available in source code and RPM packaged binaries for some systems.

RLog Version 1.3.7 - Oct 5, 2005 release.

- Tarball: `rlog-1.3.7.tgz` + `tarball GPG signature`
- Source RPM: `rlog-1.3.7-1.src.rpm`

Binary packages:

- SuSE 9.2 i586 RPM: `rlog-1.3.7-1suse92.i586.rpm`
- RedHat 7.3 i386 RPM: `rlog-1.3.7-1rh73.i386.rpm`

To check the signature, you can download my public key from a public key server, or from the link at the top of `my homepage`.

If you wish to be notified automatically of new releases, you can subscribe to new release notifications on the `Freshmeat page`.

Chapter 2

RLog Channels

An RLog Channel is a naming method for logging messages.

All logs are associated with a single channel, however there a variety of ways of subscribing to a log message.

2.1 Channel Hierarchy

Channels are hierarchical. For example, if a log message is published on the "debug" channel:

```
rDebug("hi");
// same as
static RLogChannel *myChannel = DEF_CHANNEL("debug", Log_Debug);
rLog(myChannel, "hi");
```

In the example above, all subscribers to the "debug" channel receive the messages, but *not* subscribers to "debug/foo" or other sub-channels.

If a log is published under "debug/foo/bar":

```
static RLogChannel *myChannel = DEF_CHANNEL("debug/foo/bar", Log_Debug);
rLog(myChannel, "hi");
```

In that example, all subscribers to "debug/foo/bar", "debug/foo", and "debug" will receive the message.

All channels are considered to be derived from a root channel. It doesn't have a true name and is referenced as the empty string "". So, to capture *all* messages:

```
// capture all messages and log them to stderr
StdioNode stdLog( STDERR_FILENO );
stdLog.subscribeTo( GetGlobalChannel("") ); // empty string is root channel
```

2.2 Channel Components

Or in mathematical terms, the cross product of channels and components.

Channels are componentized. By default, all log messages using one of the rLog type macros is actually published on the component-specific version of the channel (the component being the value of RLOG_COMPONENT at compile time). So, instead of just saying a message was published on "debug" channel, we need to also say which component it was part of, which we could represent as a pair (< COMPONENT, CHANNEL >) – eg <"rlog", "debug">.

This means that two separate components, both using rDebug() (for example) could be subscribed to separately, or together.

There is a way to subscribe to channels in the following ways:

- <COMPONENT, CHANNEL> : subscribe to a particular channel from a component
- <COMPONENT, *> : subscribe to all channels from a component
- <*, CHANNEL> : subscribe to a channel from *all* components
- <*, *> : subscribe to all channels from all components

```
{
    // this is published on the channel "debug", and the component
    // [RLOG_COMPONENT]
    rDebug("hi");

    StdioNode stdLog( STDERR_FILENO );

    // subscribe to a particular channel, from the current component
    // ([RLOG_COMPONENT])
    stdLog.subscribeTo( RLOG_CHANNEL("debug/foo") );

    // subscribe to all channels from the current component ([RLOG_COMPONENT])
    // (the root channel is the empty string "")
    stdLog.subscribeTo( RLOG_CHANNEL("") );

    // subscribe to a particular channel from all components
    stdLog.subscribeTo( GetGlobalChannel("debug/foo") );

    // subscribe to all channels from all components
    stdLog.subscribeTo( GetGlobalChannel("") );
}
```

As you can see from the pattern above, using the RLOG_CHANNEL() macro limits the selection to the current component. If you want to specify a component other then the current component, use GetComponentChannel() which takes the component name as the first argument.

Chapter 3

RLog Components

An RLog Component is typically a group of files with some shared purpose.

When programs are built with RLog, the value of RLOG_COMPONENT is used as the component name. If RLOG_COMPONENT is not set at compile time, you may receive a compiler warning, and the component will be set to "[unknown]".

For example when rlog is built, it specifies **-DRLOG_COMPONENT="rlog"**.

For more detail on how to use components in subscriptions, see RLog Channels.

Chapter 4

Module Index

4.1 Modules

Here is a list of all modules:

Chapter 5

Class Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 6

Class Index

6.1 Class List

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

Chapter 7

File Index

7.1 File List

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

Chapter 8

Module Documentation

8.1 RLogMacros

Defines

- #define rDebug(...) _rMessage(LOGID, rlog::_RLDebugChannel, ##__VA_-ARGS__)

  Log a message to the "debug" channel. Takes printf style arguments.

- #define rInfo(...) _rMessage(LOGID, rlog::_RLInfoChannel, ##__VA_ARGS_-_)

  Log a message to the "debug" channel. Takes printf style arguments.

- #define rWarning(...) _rMessage(LOGID, rlog::_RLWarningChannel, ##__-VA_ARGS__)

  Log a message to the "warning" channel. Takes printf style arguments.

- #define rError(...) _rMessage(LOGID, rlog::_RLErrorChannel, ##__VA_-ARGS__)

  Log a message to the "error" channel. Takes printf style arguments.

- #define rLog(channel,...) _rMessage(LOGID, channel, ##__VA_ARGS__)

  Log a message to a user defined channel. Takes a channel and printf style arguments.

8.1.1 Detailed Description

These macros are the primary interface for logging messages:

- rDebug(format, ...)
- rInfo(format, ...)
- rWarning(format, ...)
- rError(format, ...)
- rLog(channel, format, ...)
- rAssert(condition)

8.1.2 Define Documentation

8.1.2.1 #define rDebug(...) _rMessage(LOGID, rlog::_RLDebugChannel, ##__VA_ARGS__)

Log a message to the "debug" channel. Takes printf style arguments.

Format is ala printf, eg:

```
rDebug("I'm sorry %s, I can't do %s", name, request);
```

When using a recent GNU compiler, it should automatically detect format string / argument mismatch just like it would with printf.

Note that unless there are subscribers to this message, it will do nothing.

8.1.2.2 #define rError(...) _rMessage(LOGID, rlog::_RLErrorChannel, ##__VA_ARGS__)

Log a message to the "error" channel. Takes printf style arguments.

An error indicates that something has definately gone wrong.

Format is ala printf, eg:

```
rError("bad input %s, aborting request", input);
```

When using a recent GNU compiler, it should automatically detect format string / argument mismatch just like it would with printf.

Note that unless there are subscribers to this message, it will do nothing.

8.1.2.3 #define rInfo(...) _rMessage(LOGID, rlog::_RLInfoChannel, ##__VA_ARGS__)

Log a message to the "debug" channel. Takes printf style arguments.

Format is ala printf, eg:

```
rInfo("I'm sorry %s, I can't do %s", name, request);
```

When using a recent GNU compiler, it should automatically detect format string / argument mismatch just like it would with printf.

Note that unless there are subscribers to this message, it will do nothing.

8.1.2.4 #define rLog(channel, ...) _rMessage(LOGID, channel, ##__VA_ARGS__)

Log a message to a user defined channel. Takes a channel and printf style arguments.

An error indicates that something has definately gone wrong.

Format is ala printf, eg:

```
static RLogChannel * MyChannel = RLOG_CHANNEL( "debug/mine" );
rLog(MyChannel, "happy happy, joy joy");
```

When using a recent GNU compiler, it should automatically detect format string / argument mismatch just like it would with printf.

Note that unless there are subscribers to this message, it will do nothing.

8.1.2.5 #define rWarning(...) _rMessage(LOGID, rlog::_RLWarningChannel, ##__VA_ARGS__)

Log a message to the "warning" channel. Takes printf style arguments.

Output a warning message - meant to indicate that something doesn't seem right.

Format is ala printf, eg:

```
rWarning("passed %i, expected %i, continuing", foo, bar);
```

When using a recent GNU compiler, it should automatically detect format string / argument mismatch just like it would with printf.

Note that unless there are subscribers to this message, it will do nothing.

Chapter 9

Class Documentation

9.1 rlog::Error Class Reference

Error Used as exception thrown from rAssert() on failure.

`#include <rlog/Error.h>`

Public Member Functions

- **Error** (const char *component, const char *file, const char *function, int line, const char *msg)
- **Error** (const char *component, const char *file, const char *function, int line, const std::string &msg)
- **Error** (const Error &src)
- Error & **operator=** (const Error &src)
- void log (RLogChannel *channel) const
- const char * component () const
- const char * **file** () const
- const char * **function** () const
- int **line** () const
- const char * **message** () const

9.1.1 Detailed Description

Error Used as exception thrown from rAssert() on failure.

Error is derived from std::runtime_error exception and is thrown from rAssert() to indicate the reason and location of the failure.

```
void testFunc()
{
    bool testAssert = true;
    rAssert( testAssert == false ); // fails..
}

void A()
{
    try
    {
        testFunc();
    } catch( Error &err )
    {
        rDebug("caught assert failure from %s line %i ( %s )",
            err.file(), err.line(), err.function() );
    }
}
```

Author:

Valient Gough

See also:

rAssert()

9.1.2 Member Function Documentation

9.1.2.1 void Error::log (RLogChannel * *channel*) const

Log the error to the given channel

9.1.2.2 const char * Error::component () const

return component string. If this was due to an rAssert() then this will be the definition of RLOG_COMPONENT at the point of the rAssert.

References rlog::ErrorData::component.

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

- Error.h
- Error.cpp

9.2 rlog::ErrorData Struct Reference

Internal RLog structure - holds internal data for rlog::Error.

`#include <rlog/Error.h>`

Public Attributes

- int usageCount
  for reference counting
- string component
  component where error occured
- string file
  file where error occured
- string function
  function name where error occured
- int line
  line number where error occured
- string msg
  condition string (in case of rAssert) at error location

9.2.1 Detailed Description

Internal RLog structure - holds internal data for rlog::Error.

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

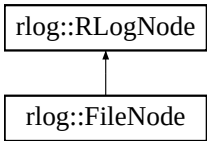
- Error.cpp

9.3 rlog::FileNode Class Reference

Provides filename based logging nodes.

`#include <rlog/rloglocation.h>`

Inheritance diagram for rlog::FileNode::



Public Member Functions

- **FileNode** (const char *componentName, const char *fileName)
- **FileNode** (const char *fileName)
- **RLOG_NO_COPY** (FileNode)

Static Public Member Functions

- static FileNode * **Lookup** (const char *componentName, const char *fileName)
- static FileNode * **Lookup** (const char *fileName)

Public Attributes

- std::string **componentName**
- std::string **fileName**

9.3.1 Detailed Description

Provides filename based logging nodes.

This allows subscribing to messages only from particular files. For example,

```
int main()
{
    // display some messages to stderr
    StdioNode std( STDERR_FILENO );

    // subscribe to everything from this file
    std.subscribeTo( FileNode::Lookup( __FILE__ ) );
```

```
    // and everything from "important.cpp"
    std.subscribeTo( FileNode::Lookup( "important.cpp" ));
}
```

Note that file names are not required to be unique across the entire program. Different components may contain the same filename, which is why there is a second Lookup function which also takes the component name.

See also:

RLogChannel

Author:

Valient Gough

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

- rloglocation.h
- rloglocation.cpp

9.4 rlog::Lock Class Reference

`#include <Lock.h>`

Public Member Functions

- **Lock** (Mutex *mutex)

Public Attributes

- Mutex * **_mutex**

9.4.1 Detailed Description

Simple helper class or doing locking, so we can be sure the lock is released when our scope ends for whatever reason..

Usage

```
func()
{
    // mutex released when lock goes out of scope
    Lock lock( &mutex );

    ...
}
```

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

- Lock.h

9.5 rlog::Mutex Class Reference

Class encapsulating a critical section under Win32 or a mutex under Unix.

`#include <Mutex.h>`

Public Member Functions

- void **Lock** ()
- void **Unlock** ()

9.5.1 Detailed Description

Class encapsulating a critical section under Win32 or a mutex under Unix.

This class should never be used standalone but always passed to Lock, see the example there.

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

- Mutex.h

9.6 rlog::PublishLoc Struct Reference

Internal RLog structure - static struct for each rLog() statement.

`#include <rlog/rlog.h>`

Public Attributes

- void(∗ publish)(PublishLoc ∗, RLogChannel ∗, const char ∗format,...) PRINTF_FP(3

  function to call when we reach the log statement.

- void(∗) RLogNode **pub**)
- const char ∗ **component**
- const char ∗ **fileName**
- const char ∗ **functionName**
- int **lineNum**
- RLogChannel ∗ **channel**

9.6.1 Detailed Description

Internal RLog structure - static struct for each rLog() statement.

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

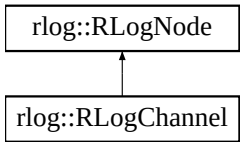
- rlog.h

9.7 rlog::RLogChannel Class Reference

Implements a hierarchical logging channel.

`#include <rlog/RLogChannel.h>`

Inheritance diagram for rlog::RLogChannel::



Public Member Functions

- **RLogChannel** (const std::string &name, LogLevel level)
- virtual void publish (const RLogData &data)

  Publish data.

- const std::string & **name** () const
- LogLevel **logLevel** () const
- void **setLogLevel** (LogLevel level)

Protected Member Functions

- RLogChannel * **getComponent** (RLogChannel *componentParent, const char *component)

Friends

- RLogChannel * GetComponentChannel (const char *component, const char *path, LogLevel level)

  Return the named channel for a particular component.

Related Functions

(Note that these are not member functions.)

- RLogChannel * GetGlobalChannel (const char *path, LogLevel level)

  Return the named channel across all components.

9.7.1 Detailed Description

Implements a hierarchical logging channel.

You should not need to use RLogChannel directly under normal circumstances. See RLOG_CHANNEL() macro, GetComponentChannel() and GetGlobalChannel()

RLogChannel implements channel logging support. A channel is a named logging location which is global to the program. Channels are hierarchically related.

For example, if somewhere in your program a message is logged to "debug/foo/bar", then it will be delived to any subscribers to "debug/foo/bar", or subscribers to "debug/foo", or subscribers to "debug". Subscribing to a channel means you will receive anything published on that channel or sub-channels.

As a special case, subscribing to the channel "" means you will receive all messages - as every message has a channel and the empty string "" is considered to mean the root of the channel tree.

In addition, componentized channels are all considered sub channels of the global channel hierarchy. All rDebug(), rWarning(), and rError() macros publish to the componentized channels (component defined by RLOG_COMPONENT).

```
// get the "debug" channel for our component.  This is the same as
// what rDebug() publishes to.
RLogChannel *node = RLOG_CHANNEL( "debug", Log_Debug );
// equivalent to
RLogChannel *node = GetComponentChannel( RLOG_COMPONENT, "debug" );

// Or, get the global "debug" channel, which will have messages from
// *all* component's "debug" channels.
RLogChannel *node = GetGlobalChannel( "debug", Log_Debug );
```

Author:

Valient Gough

See also:

RLOG_CHANNEL()
GetComponentChannel()
GetGlobalChannel()

9.7.2 Member Function Documentation

9.7.2.1 void RLogChannel::publish (const RLogData & *data*) [virtual]

Publish data.

This iterates over the list of subscribers which have stated interest and sends them the data.

Reimplemented from rlog::RLogNode.

References rlog::RLogNode::publish(), and rlog::RLogData::seen.

9.7.3 Friends And Related Function Documentation

9.7.3.1 RLogChannel * GetComponentChannel (const char * *component*, const char * *path*, LogLevel *level* = `Log_Undef`) `[friend]`

Return the named channel for a particular component.

Author:

Valient Gough

Referenced by GetGlobalChannel().

9.7.3.2 RLogChannel * GetGlobalChannel (const char * *path*, LogLevel *level*) `[related]`

Return the named channel across all components.

Channels are hierarchical. See RLogChannel for more detail. The global channel contains messages for all component channels.

For example, subscribing to the global "debug" means the subscriber would also get messages from <Component , "debug">, and <Component-B, "debug">, and <Component-C, "debug/foo">, etc.

Author:

Valient Gough

References GetComponentChannel.

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

- RLogChannel.h
- RLogChannel.cpp

9.8 rlog::RLogData Struct Reference

Data published through RLogNode.

`#include <rlog/RLogNode.h>`

Public Attributes

- struct PublishLoc * **publisher**
- time_t time

  time of publication

- const char * msg

  formatted msg - gets destroyed when publish() call returns.

- std::set< RLogNode * > **seen**

9.8.1 Detailed Description

Data published through RLogNode.

RLogData is the data which is published from rDebug(), rWarning(), rError() and rLog() macros. It contains a link to the publisher, the (approximate) time of the publication, and the formatted message.

Note that none of the data in the publication is considered to be static. Once control has returned to the publisher, the data may be destroyed. If it is necessary to hang on to any of the data, a deep copy must be made.

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

- RLogNode.h

9.9 rlog::RLogModule Class Reference

Allows registration of external modules to rlog.

`#include <rlog/rloginit.h>`

Public Member Functions

- virtual void init (int &argv, char **argc)
- virtual const char * moduleName () const =0

Related Functions

(Note that these are not member functions.)

- RLogModule * RegisterModule (RLogModule *module)

9.9.1 Detailed Description

Allows registration of external modules to rlog.

Currently this only allows for initialization callbacks. When RLogInit() is called, init() is called on all modules which have been registered.

Author:

Valient Gough

9.9.2 Member Function Documentation

9.9.2.1 void RLogModule::init (int & *argv*, char ** *argc*) [virtual]

Called by RLogInit() to give the modules the command-line arguments

Referenced by RegisterModule().

9.9.2.2 virtual const char* rlog::RLogModule::moduleName () const [pure virtual]

Must be implemented to return the name of the module.

9.9.3 Friends And Related Function Documentation

9.9.3.1 RLogModule * RegisterModule (RLogModule * *module*) [related]

Registers the module - which will have init() called when RLogInit is called. Returns the module so that it can be used easily as a static initializer.

```
class MyModule : public rlog::RLogModule
{
public:
    virtual const char *moduleName() const {return "MyModule";}
};
static RLogModule * testModule = rlog::RegisterModule( new MyModule() );
```

References init().

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

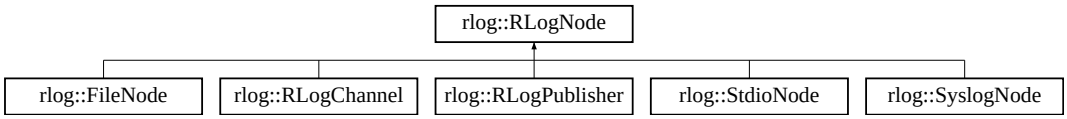
- rloginit.h
- rloginit.cpp

9.10 rlog::RLogNode Class Reference

Core of publication system, forms activation network.

`#include <rlog/RLogNode.h>`

Inheritance diagram for rlog::RLogNode::



Public Member Functions

- RLogNode ()

  Instantiate an empty RLogNode. No subscribers or publishers..

- virtual ~RLogNode ()

  Disconnects from any remaining subscribers and publishers.

- virtual void clear ()

  Force disconnection from any subscribers or publishers.

- virtual void publish (const RLogData &data)

  Publish data.

- virtual void addPublisher (RLogNode *)

  Have this node subscribe to a new publisher.

- virtual void dropPublisher (RLogNode *, bool callbacks=true)

  Drop our subscription to a publisher.

- bool enabled () const

  Returns true *if this node is active.*

- virtual void addSubscriber (RLogNode *)

  Add a subscriber.

- virtual void dropSubscriber (RLogNode *)

  Remove a subscriber.

- virtual void isInterested (RLogNode *node, bool isInterested)

  Change the state of one of our subscribers.

Protected Member Functions

- virtual void setEnabled (bool newState)

  For derived classes to get notified of activation status change.

Protected Attributes

- std::list< RLogNode * > publishers

  list of nodes we are subscribed to

- std::list< RLogNode * > subscribers

  list of nodes we publish to

- std::list< RLogNode * > interestList

  list of subscribers that are interested in receiving publications..

- Mutex **mutex**

9.10.1 Detailed Description

Core of publication system, forms activation network.

RLogNode formes the core of the publication system. It has two primary purposes :

- link publications with subscribers
- transfer meta-data in the form or node activations

Both publishers (eg RLogPublisher) and subscribers (eg StdioNode) are derived from RLogNode, although RLogNode can be used entirely unmodified.

An RLogNode contains a list of publishers which it is linked with. It also contains a list of subscribers which it is linked with. Publications always flow from publishers to subscribers, and activation information flows the opposite direction from subscribers to publishers.

An RLogNode by default acts as both a subscriber and publisher – when it has been subscribed to another node and receives a publication it simply repeats the information to all of its subscribers.

More specifically, it only publishes to subscribers who have also voiced an interest in receiving the publication. If a node has no subscribers which are also interested (or no subscribers at all), then it can be said to be dormant and it tells the publishers that it is subscribed to that it is no longer interested. This propogates all the way up to RLogPublishers which will disable the logging statement completely if there are no interested parties.

Author:

Valient Gough

9.10.2 Member Function Documentation

9.10.2.1 void RLogNode::publish (const RLogData & *data*) [virtual]

Publish data.

This iterates over the list of subscribers which have stated interest and sends them the data.

Reimplemented in rlog::RLogChannel, rlog::StdioNode, and rlog::SyslogNode.

References interestList.

Referenced by rlog::RLogChannel::publish().

9.10.2.2 void RLogNode::addPublisher (RLogNode * *publisher*) [virtual]

Have this node subscribe to a new publisher.

We become a subscriber of the publisher. The publisher's addSubscriber() function is called to complete the link.

If our node is active then we also tell the publisher that we want publications.

References addSubscriber(), interestList, isInterested(), and publishers.

Referenced by rlog::StdioNode::subscribeTo().

9.10.2.3 void RLogNode::dropPublisher (RLogNode * *publisher*, bool *callback* = true) [virtual]

Drop our subscription to a publisher.

A callback parameter is provided to help avoid loops in the code which may affect the thread locking code.

Parameters:

callback If True, then we call publisher->dropSubscriber() to make sure the publisher also drops us as a subscriber.

References dropSubscriber(), interestList, isInterested(), and publishers.

9.10.2.4 bool RLogNode::enabled () const

Returns *true* if this node is active.

Returns:

true if we have one or more interested subscribers, otherwise false

References interestList.

9.10.2.5 void RLogNode::addSubscriber (RLogNode * *subscriber*) [virtual]

Add a subscriber.

Normally a subscriber calls this for itself when it's addPublisher() method is called.

References subscribers.

Referenced by addPublisher().

9.10.2.6 void RLogNode::dropSubscriber (RLogNode * *subscriber*) [virtual]

Remove a subscriber.

Normally a subscriber calls this for itself when it's dropPublisher() method is called.

Note that the subscriber list is kept separate from the interest list. If the subscriber is active, then you must undo that by calling isInterested(subscriber, false) in addition to dropSubscriber

References subscribers.

Referenced by dropPublisher().

9.10.2.7 void RLogNode::isInterested (RLogNode * *node*, bool *interest*) [virtual]

Change the state of one of our subscribers.

This allows a subscriber to say that it wants to be notified of publications or not. The *node* should already be registered as a subscriber.

If we previously had no interested parties and now we do, then we need to notify the publishers in our publishers list that we are now interested.

If we previously had interested parties and we remove the last one, then we can notify the publishers that we are no longer interested..

References interestList, publishers, and setEnabled().

Referenced by addPublisher(), dropPublisher(), and rlog::StdioNode::subscribeTo().

9.10.2.8 void RLogNode::setEnabled (bool *newState*) [protected, virtual]

For derived classes to get notified of activation status change.

This is called by isInterested() when our state changes. If *true* is passed, then this node has become active. If *false* is passed, then this node has just become dormant.

Reimplemented in rlog::RLogPublisher.

Referenced by clear(), and isInterested().

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

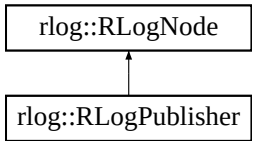
- RLogNode.h
- RLogNode.cpp

9.11 rlog::RLogPublisher Class Reference

RLog publisher used by rLog macros.

`#include <rlog/RLogPublisher.h>`

Inheritance diagram for rlog::RLogPublisher::



Public Member Functions

- **RLogPublisher** (PublishLoc *src)

Static Public Member Functions

- static void **Publish** (PublishLoc *, RLogChannel *, const char *format,...)
- static void **PublishVA** (PublishLoc *, RLogChannel *, const char *format, va_list args)

Public Attributes

- PublishLoc * **src**

Protected Member Functions

- virtual void setEnabled (bool newState)

  For derived classes to get notified of activation status change.

- **RLogPublisher** (const RLogPublisher &)
- RLogPublisher & **operator=** (const RLogPublisher &)

9.11.1 Detailed Description

RLog publisher used by rLog macros.

This derives from RLogNode and interfaces to the static PublishLoc logging data allowing them to be enabled or disabled depending on subscriber status.

An instance of this class is created for every error message location. Normally this class is not used directly.

For example, this

```
rDebug( "hello world" );
```

is turned approximatly into this:

```
static PublishLoc _rl = {
    publish:  & rlog::RLog_Register ,
    pub: 0,
    component: "component",
    fileName: "myfile.cpp",
    functionName: "functionName()",
    lineNum: __LINE__,
    channel: 0
};
if(_rL.publish != 0)
    (*_rl.publish)( &_rL, _RLDebugChannel, "hello world" );
```

The RLogPublisher instance manages the contents of the static structure _rL. When someone subscribes to it's message, then _rL.publish is set to point to the publishing function, and when there are no subscribers then it is set to 0.

The code produced contains one if statement, and with optimization comes out to about 3 instructions on an x86 computer (not including the function call). If there are no subscribers to this message then that is all the overhead, plus the memory usage for the structures involved and the initial registration when the statement is first encountered..

See also:

RLogChannel

Author:

Valient Gough

9.11.2 Member Function Documentation

9.11.2.1 void RLogPublisher::setEnabled (bool *newState*) [protected, virtual]

For derived classes to get notified of activation status change.

This is called by isInterested() when our state changes. If *true* is passed, then this node has become active. If *false* is passed, then this node has just become dormant.

Reimplemented from rlog::RLogNode.

References rlog::PublishLoc::publish.

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

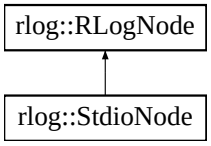
- RLogPublisher.h
- RLogPublisher.cpp

9.12 rlog::StdioNode Class Reference

Logs subscribed messages to a file descriptor.

`#include <rlog/StdioNode.h>`

Inheritance diagram for rlog::StdioNode::



Public Types

- enum **StdioFlags** {

  DefaultOutput = 0, **OutputColor** = 1, **OutputThreadId** = 2, **OutputContext** = 4,

  OutputChannel = 8 }

Public Member Functions

- StdioNode (int fdOut=2, int flags=(int) DefaultOutput)
- **StdioNode** (int fdOut, bool colorize_if_tty)
- void subscribeTo (RLogNode *node)

Protected Member Functions

- virtual void publish (const RLogData &data)

  Publish data.

- **StdioNode** (const StdioNode &)
- StdioNode & **operator=** (const StdioNode &)

Protected Attributes

- bool **colorize**
- bool **outputThreadId**
- bool **outputContext**
- bool **outputChannel**
- int **fdOut**

9.12.1 Detailed Description

Logs subscribed messages to a file descriptor.

This displays all subscribed messages to a file descriptor. If the output is a terminal, then the messages are colorized (yellow for warnings, red for errors).

For example, to log to stderr:

```
int main(int argc, char **argv)
{
    // tell RLog the program name..
    RLog_Init( argv[0] );

    // log to standard error
    StdioNode stdlog( STDERR_FILENO );

    // show all warning and error messages, no matter what component they
    // come from.
    stdlog.subscribeTo( GetGlobalChannel( "warning" ));
    stdlog.subscribeTo( GetGlobalChannel( "error" ));
}
```

See also:

RLogChannel
RLOG_CHANNEL()

Author:

Valient Gough

9.12.2 Constructor & Destructor Documentation

9.12.2.1 StdioNode::StdioNode (int *_fdOut* = 2, int *flags* = (int)DefaultOutput)

Parameters:

_fdOut File descriptor to send output

flags bitmask of the following options:

```
OutputChannel - Includes the channel name in the output. e.g. [debug]
OutputContext - Includes the filename and line number in the output. e.g. OpsecA
OutputColor   - Output to a TTY is colored based on log level (not supported by
OutputThreadId - Includes the thread id in the output e.g. [tid:37333936] (Not s
```

9.12.3 Member Function Documentation

9.12.3.1 void StdioNode::subscribeTo (RLogNode * *node*)

Subscribe to log messages.

Example:

```
StdioNode log( STDERR_FILENO );
// subscribe to error and warning messages
log.subscribeTo( GetGlobalChannel("error" ));
log.subscribeTo( RLOG_CHANNEL("warning" ));
```

References rlog::RLogNode::addPublisher(), and rlog::RLogNode::isInterested().

9.12.3.2 void StdioNode::publish (const RLogData & *data*) [protected, virtual]

Publish data.

This iterates over the list of subscribers which have stated interest and sends them the data.

Reimplemented from rlog::RLogNode.

References rlog::PublishLoc::channel, rlog::PublishLoc::fileName, rlog::PublishLoc::lineNum, rlog::RLogChannel::logLevel(), rlog::RLogData::msg, rlog::RLogChannel::name(), rlog::RLogData::publisher, and rlog::RLogData::time.

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

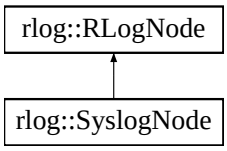
- StdioNode.h
- StdioNode.cpp

9.13 rlog::SyslogNode Class Reference

Logs subscribed messages using syslog.

`#include <rlog/SyslogNode.h>`

Inheritance diagram for rlog::SyslogNode::



Public Member Functions

- **SyslogNode** (const char *ident)
- SyslogNode (const char *ident, int option, int facility)

  See syslog(3) for possible values of option and facility arguments.

- void **subscribeTo** (RLogNode *node)

Protected Member Functions

- virtual void publish (const RLogData &data)

  Publish data.

Protected Attributes

- const char * **ident**
- int **option**
- int **facility**

9.13.1 Detailed Description

Logs subscribed messages using syslog.

This logs all subscribed messages using syslog.

For example:

```
int main(int argc, char **argv)
{
    // tell RLog the program name..
    RLog_Init( argv[0] );

    // log to syslog, prepending "rlog-test" to every line
    SyslogNode slog( "rlog-test" );

    // show all warning and error messages, no matter what component they
    // come from.
    stdlog.subscribeTo( GetGlobalChannel( "warning" ));
    stdlog.subscribeTo( GetGlobalChannel( "error" ));
}
```

See also:

RLogChannel
RLOG_CHANNEL()

Author:

Valient Gough

9.13.2 Member Function Documentation

9.13.2.1 void SyslogNode::publish (const RLogData & *data*) [protected, virtual]

Publish data.

This iterates over the list of subscribers which have stated interest and sends them the data.

Reimplemented from rlog::RLogNode.

References rlog::PublishLoc::channel, rlog::RLogChannel::logLevel(), rlog::RLogData::msg, and rlog::RLogData::publisher.

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

- SyslogNode.h
- SyslogNode.cpp

Chapter 10

File Documentation

10.1 rlog.h File Reference

Defines macros for debug, warning, and error messages.

```
#include <rlog/common.h>
#include <rlog/rlog-c99.h>
```

Namespaces

- namespace **rlog**

Classes

- struct rlog::PublishLoc

  Internal RLog structure - static struct for each rLog() statement.

Defines

- #define **CONCAT2**(A, B) A##B
- #define **CONCAT**(A, B) CONCAT2(A,B)
- #define **STR**(X) #X
- #define **CURRENT_RLOG_VERSION** 20040503
- #define DEF_CHANNEL(path, level) RLOG_CHANNEL_IMPL(RLOG_COMPONENT, path, level)

  Returns pointer to RLogChannel struct for the given path.

- #define **RLOG_CHANNEL**(path) RLOG_CHANNEL_IMPL(RLOG_COMPONENT, path, rlog::Log_Undef)
- #define **RLOG_CHANNEL_IMPL**(COMPONENT, path, level) rlog::GetComponentChannel(STR(COMPONENT),path,level)
- #define **_rAssertFailed**(COMPONENT, COND) rlog::rAssertFailed(STR(COMPONENT),__FILE__,__FUNCTION__,__LINE__, COND)
- #define rAssert(cond)

  Assert condition - throws error if cond *evaluates to false.*

- #define rAssertSilent(cond)

  Assert condition - throws error if cond *evaluates to false, but does not display error message.*

- #define **RLOG_NO_COPY**(CNAME)

Enumerations

- enum **LogLevel** {

  rlog::Log_Undef = 0, rlog::Log_Critical, rlog::Log_Error, rlog::Log_Warning, rlog::Log_Notice, rlog::Log_Info, rlog::Log_Debug }

Functions

- int **RLogVersion** ()
- void RLOG_DECL rlog::RLogInit (int &argc, char **argv)

  Initializer for external rlog modules.

- RLOG_DECL RLogChannel * **rlog::GetComponentChannel** (const char *component, const char *path, LogLevel level=Log_Undef)
- RLOG_DECL RLogChannel * **rlog::GetGlobalChannel** (const char *path, LogLevel level=Log_Undef)
- void **rlog::RLog_Register** (PublishLoc *loc, RLogChannel *channel, const char *format,...)
- void **rlog::rAssertFailed** (const char *component, const char *file, const char *function, int line, const char *conditionStr)

Variables

- RLOG_DECL RLogChannel * **rlog::_RLDebugChannel** = GetGlobalChannel("debug", Log_Debug)

- RLOG_DECL RLogChannel * **rlog::_RLInfoChannel** = GetGlobalChannel("info", Log_Info)
- RLOG_DECL RLogChannel * **rlog::_RLWarningChannel** = GetGlobalChannel("warning", Log_Warning)
- RLOG_DECL RLogChannel * **rlog::_RLErrorChannel** = GetGlobalChannel("error", Log_Error)

10.1.1 Detailed Description

Defines macros for debug, warning, and error messages.

10.1.2 Define Documentation

10.1.2.1 #define DEF_CHANNEL(path, level) RLOG_CHANNEL_IMPL(RLOG_COMPONENT, path, level)

Returns pointer to RLogChannel struct for the given path.

Parameters:

path The hierarchical path to the channel. Elements in the path are separated by '/'.

DEF_CHANNEL gets an existing (or defines a new) log type. For example "debug", "warning", "error" are predefined types. You might define completely new types, like "timing", or perhaps sub-types like "debug/timing/foo", depending on your needs.

Reporting paths do not need to be unique within a project (or even a file).

Channels form a hierarchy. If one subscribes to "debug", then you also get messages posted to more specific types such as "debug/foo". But if you subscribe to a more specific type, such as "debug/foo", then you will not receive more general messages such as to "debug".

Example:

```
#include <rlog/rlog.h>
#include <rlog/RLogChannel.h>

static RLogChannel *MyChannel = DEF_CHANNEL("me/mine/allmine",Log_Info);

func()
{
    rLog( MyChannel, "this is being sent to my own channel" );
    rLog( MyChannel, "%s %s", "hello", "world" );
}
```

```
main()
{
    // log all messages to the "me" channel to stderr
    StdioNode stdLog( STDERR_FILENO );
    stdLog.subscribeTo( RLOG_CHANNEL ("me") );

    func();
}
```

See also:

test.cpp

10.1.2.2 #define rAssert(cond)

Value:

```
do { \
        if( unlikely((cond) == false) ) \
        { rError( "Assert failed: " STR(cond) ); \
          _rAssertFailed(RLOG_COMPONENT, STR(cond)); \
        } \
    } while(0)
```

Assert condition - throws error if *cond* evaluates to false.

Assert error condition. Displays error message if condition does not evaluate to TRUE.

We throw an error from rAssertFailed. It isn't done inline so that we don't have to include the STL exception code here and bloat callers that don't use it or don't care.

10.1.2.3 #define rAssertSilent(cond)

Value:

```
do { \
        if( unlikely((cond) == false) ) \
        { _rAssertFailed(RLOG_COMPONENT, STR(cond)); } \
    } while(0)
```

Assert condition - throws error if *cond* evaluates to false, but does not display error message.

Assert error condition. Similar to rAssert except that it does not display an error.

10.1.2.4 #define RLOG_NO_COPY(CNAME)

Value:

```
private: \
        CNAME(const CNAME&); \
        CNAME & operator = (const CNAME &)
```

Index