

ERS

Generated by Doxygen 1.8.5

Tue Apr 14 2020 15:27:55

Contents

1	Error Reporting Service (ERS)	1
2	Namespace Index	9
2.1	Namespace List	9
3	Hierarchical Index	9
3.1	Class Hierarchy	9
4	Class Index	10
4.1	Class List	10
5	File Index	12
5.1	File List	12
6	Namespace Documentation	13
6.1	ers Namespace Reference	13
6.1.1	Detailed Description	15
6.1.2	Function Documentation	16
6.2	ers::format Namespace Reference	20
6.2.1	Detailed Description	20
6.3	thread Namespace Reference	20
6.3.1	Detailed Description	20
7	Class Documentation	20
7.1	ers::AbortStream Struct Reference	20
7.1.1	Detailed Description	21
7.2	ers::AnyIssue Class Reference	21
7.3	ers::Configuration Class Reference	21
7.3.1	Detailed Description	22
7.3.2	Member Function Documentation	22
7.4	ers::Context Class Reference	23
7.4.1	Detailed Description	24
7.4.2	Member Function Documentation	24
7.5	ers::ExitStream Struct Reference	26
7.5.1	Detailed Description	26
7.5.2	Constructor & Destructor Documentation	27
7.6	ers::FilterStream Class Reference	27
7.6.1	Detailed Description	27
7.6.2	Constructor & Destructor Documentation	27
7.6.3	Member Function Documentation	28
7.7	ers::FormattedStandardStream< Device > Struct Template Reference	28

7.7.1	Constructor & Destructor Documentation	28
7.8	ers::GlobalLockStream Struct Reference	29
7.8.1	Detailed Description	29
7.9	ers::InputStream Class Reference	29
7.9.1	Detailed Description	29
7.10	ers::Issue Class Reference	30
7.10.1	Detailed Description	31
7.10.2	Constructor & Destructor Documentation	31
7.10.3	Member Function Documentation	32
7.11	IssueCatcher Struct Reference	33
7.12	ers::IssueCatcherHandler Class Reference	33
7.12.1	Detailed Description	34
7.13	ers::IssueFactory Class Reference	34
7.13.1	Detailed Description	34
7.13.2	Member Function Documentation	35
7.14	ers::IssueReceiver Class Reference	35
7.14.1	Detailed Description	36
7.15	ers::IssueRegistrar< T > Class Template Reference	36
7.16	ers::LocalContext Class Reference	36
7.16.1	Constructor & Destructor Documentation	37
7.16.2	Member Function Documentation	37
7.17	ers::LocalProcessContext Struct Reference	39
7.17.1	Member Data Documentation	40
7.18	ers::LockStream Struct Reference	40
7.18.1	Detailed Description	40
7.19	MyIssueReceiver Struct Reference	41
7.20	ers::NullStream Struct Reference	41
7.20.1	Detailed Description	41
7.21	ers::OutputStream Class Reference	42
7.21.1	Detailed Description	42
7.22	ers::PluginException Class Reference	43
7.23	ers::PluginManager Class Reference	43
7.23.1	Constructor & Destructor Documentation	43
7.24	ers::RemoteContext Class Reference	43
7.24.1	Constructor & Destructor Documentation	44
7.24.2	Member Function Documentation	44
7.25	ers::RemoteProcessContext Struct Reference	47
7.25.1	Member Data Documentation	47
7.26	ers::RFilterStream Class Reference	47
7.26.1	Detailed Description	48

7.26.2	Constructor & Destructor Documentation	48
7.26.3	Member Function Documentation	48
7.27	ers::Severity Struct Reference	49
7.28	ers::SingletonCreator< class > Class Template Reference	49
7.29	ers::StandardStream< Device > Struct Template Reference	49
7.29.1	Detailed Description	49
7.30	ers::StandardStreamOutput Struct Reference	50
7.30.1	Detailed Description	50
7.31	ers::StreamInitializer Class Reference	50
7.32	ers::StreamManager Class Reference	51
7.32.1	Detailed Description	52
7.32.2	Constructor & Destructor Documentation	52
7.32.3	Member Function Documentation	52
7.33	Test Struct Reference	53
7.34	ers::ThrottleStream Class Reference	54
7.34.1	Detailed Description	54
7.34.2	Member Function Documentation	54
7.35	ers::ThrowStream Struct Reference	55
7.35.1	Detailed Description	55
8	File Documentation	55
8.1	bin/config.cxx File Reference	55
8.1.1	Detailed Description	56
8.2	ers/AnyIssue.h File Reference	56
8.2.1	Detailed Description	56
8.3	ers/Assertion.h File Reference	56
8.3.1	Detailed Description	56
8.3.2	Macro Definition Documentation	57
8.4	ers/Configuration.h File Reference	58
8.4.1	Detailed Description	58
8.5	ers/Context.h File Reference	58
8.5.1	Detailed Description	58
8.6	ers/ers.h File Reference	59
8.6.1	Detailed Description	59
8.6.2	Macro Definition Documentation	60
8.7	ers/InputStream.h File Reference	60
8.7.1	Detailed Description	61
8.7.2	Macro Definition Documentation	61
8.8	ers/internal/AbortStream.h File Reference	61
8.8.1	Detailed Description	61

8.9	ers/internal/ExitStream.h File Reference	62
8.9.1	Detailed Description	62
8.10	ers/internal/FilterStream.h File Reference	62
8.10.1	Detailed Description	62
8.11	ers/internal/FormattedStandardStream.h File Reference	62
8.11.1	Detailed Description	63
8.12	ers/internal/LockStream.h File Reference	63
8.12.1	Detailed Description	63
8.13	ers/internal/macro.h File Reference	64
8.13.1	Detailed Description	64
8.13.2	Macro Definition Documentation	64
8.14	ers/internal/NullStream.h File Reference	65
8.14.1	Detailed Description	66
8.15	ers/internal/PluginManager.h File Reference	66
8.15.1	Detailed Description	66
8.16	ers/internal/RFilterStream.h File Reference	66
8.16.1	Detailed Description	67
8.17	ers/internal/SingletonCreator.h File Reference	67
8.17.1	Detailed Description	67
8.18	ers/internal/StandardStream.h File Reference	67
8.18.1	Detailed Description	67
8.19	ers/internal/ThrowStream.h File Reference	68
8.19.1	Detailed Description	68
8.20	ers/internal/Util.h File Reference	68
8.20.1	Detailed Description	68
8.21	ers/Issue.h File Reference	69
8.21.1	Detailed Description	69
8.22	ers/IssueCatcherHandler.h File Reference	70
8.22.1	Detailed Description	70
8.23	ers/IssueFactory.h File Reference	70
8.23.1	Detailed Description	71
8.24	ers/IssueReceiver.h File Reference	71
8.24.1	Detailed Description	71
8.25	ers/LocalContext.h File Reference	71
8.25.1	Detailed Description	72
8.26	ers/LocalStream.h File Reference	72
8.26.1	Detailed Description	72
8.26.2	Function Documentation	72
8.27	ers/OutputStream.h File Reference	72
8.27.1	Detailed Description	73

8.28	ers/RemoteContext.h File Reference	73
8.28.1	Detailed Description	73
8.29	ers/SampleIssues.h File Reference	73
8.29.1	Detailed Description	73
8.30	ers/Severity.h File Reference	74
8.30.1	Detailed Description	74
8.31	ers/StandardStreamOutput.h File Reference	75
8.31.1	Detailed Description	75
8.32	ers/StreamFactory.h File Reference	75
8.32.1	Detailed Description	75
8.32.2	Function Documentation	75
8.33	ers/StreamManager.h File Reference	76
8.33.1	Detailed Description	76

1 Error Reporting Service (ERS)

The Error Reporting System (ERS) software package provides a common API for error reporting in the ATLAS TDAQ system. ERS offers several macro that can be used for reporting pre-defined errors if some conditions are violated at the level of C++ code. ERS also provides tools for defining custom classes that can be used for reporting high-level issues.

The Error Reporting System (ERS) software package provides a common API for error reporting in the ATLAS TDAQ system. ERS offers several macro that can be used for reporting pre-defined errors if some conditions are violated at the level of C++ code. ERS also provides tools for defining custom classes that can be used for reporting high-level issues.

Header File

In order to use ERS functionality an application has to include a single header file [ers/ers.h](#)

Assertion Macro

ERS provides several convenience macro for checking conditions in a C++ code. If a certain condition is violated a corresponding macro creates a new instance of `ers::Assertion` class and sends it to the `ers::fatal` stream. Further processing depends on the `ers::fatal` stream configuration. By default the issue is printed to the standard error stream.

Here is a list of available macro:

- `ERS_ASSERT(expression)` generic macro that checks whether a given expression is valid.
- `ERS_PRECONDITION(expression)` the same as `ERS_ASSERT` but uses a message that is adopted for reporting an invalid input parameter for a function.
- `ERS_RANGE_CHECK(min, val, max)` is a special type of pre-condition, which checks that a value is in a range between min and max values.
- `ERS_STRICT_RANGE_CHECK(min, val, max)` is similar to the `ERS_RANGE_CHECK` but does not allow the checked value to be equal to either min or max values.

Note: These macro are defined to empty statements if `ERS_NO_DEBUG` macro is defined at compilation time.

Logging Macro

ERS package offers a set of macro for information logging:

- **ERS_DEBUG(level, message)** - sends ers::Message issue to the `ers::debug` stream
- **ERS_LOG(message)** - sends ers::Message issue to the `ers::log` stream
- **ERS_INFO(message)** - sends ers::Message issue to the `ers::information` stream

Note: ERS_DEBUG macro is defined to empty statement if **ERS_NO_DEBUG** macro is defined at compilation time.

Each of these macro constructs an new issue of ers::Message time and sends it to an appropriate stream. The **message** argument of these macro can be any value, for which the standard C++ output stream operator (`**operator<<*`) is defined. This means that the message can be a single value of a certain type as well as a sequence of output operations of any supported types. For example:

```
ERS_DEBUG( 1, "simple debug output " << 123 << " that shows how to use debug macro" )
```

The actual behavior of these macro depends on the configuration of a respective stream. Debug macro can be disabled at run-time by defining the **TDAQ_ERS_DEBUG_LEVEL** environment variable to the highest possible debug level. For instance, if **TDAQ_ERS_DEBUG_LEVEL** is set to **N**, then **ERS_DEBUG(M, ...)** where **M > N** will not produce any output. The default value for the **TDAQ_ERS_DEBUG_LEVEL** is 0. Negative debug levels are also allowed.

The amount of information, which is printed for an issue depends on the actual ERS verbosity level, which can be controlled via the **TDAQ_ERS_VERBOSITY_LEVEL** macro. Default verbosity level is zero. In this case the following information is reported for any issue:

- severity (DEBUG, LOG, INFO, WARNING, ERROR, FATAL)
- the time of the issue occurrence
- the issue's context, which includes package name, file name, function name and line number
- the issue's message

One can control the current verbosity level via the **TDAQ_ERS_VERBOSITY_LEVEL** macro:

```
export TDAQ_ERS_VERBOSITY_LEVEL=N
```

where **N** must be an integer number.

- For **N > 0** the issue attributes names and values are reported in addition to 0-level data
- For **N > 1** the following information is added to the issue:
 - host name
 - user name
 - process id
 - process current working directory
- For **N > 2** a stack trace is added to each issue if the code was compiled without **ERS_NO_DEBUG** macro.

Using Custom Issue Classes

ERS assumes that user functions should throw exceptions in case of errors. If such exceptions are instances of classes, which inherit the `ers::Issue` one, ERS offers a number of advantages with respect to their handling:

- ERS issues can be reported to any ERS stream
- One can create chains of issues to preserve the original cause of the problem as well as the error handling sequence
- ERS issues can be printed to a standard C++ output stream using the output operator provided by ERS

In order to define a custom issue one has to do the following steps:

- Declare a class, which inherits `ers::Issue`
- Implement 3 pure virtual functions, declared in the `ers::Issue` class
- Register new class using the `ers::IssueFactory::register_issue` function

ERS defines two helper macro, which implement all these steps. The macro are called `ERS_DECLARE_ISSUE` and `ERS_DECLARE_ISSUE_BASE`. The first one should be used to declare an issue class that inherits from the `ers::Issue` as it is shown on the following example:

```
ERS_DECLARE_ISSUE(
ers,
    Assertion,
    "Assertion ( " << condition << " ) failed because " << reason,
    ((const char *)condition )
    ((const char *)reason )
)
// namespace
// issue name
// message
// first attribute
// second attribute
```

Note that attribute names may appear in the message expression. Also note a special syntax of the attributes declaration, which must always be declared using a list of `**((attribute_type)attribute_name)**` tokens. All the brackets in this expression are essential. Do not use commas to separate attributes. The only requirement for the type of an issue attribute is that for this type must be defined the output operator to the standard C++ output stream and the input operator from the standard C++ input stream. It is important to note that these operators must unambiguously match each other, i.e. the input operator must be able to unambiguously restore the state of the attribute from a stream, which had been used to save the object's state with the output operator. Evidently all the built-in C++ types satisfy this criteria. The result of the `ERS_DECLARE_ISSUE` macro expansion would look like:

```
namespace ers {
class Assertion : public ers::Issue {
    template <class> friend class ers::IssueRegistrar;
    Assertion() { ; }

    static const char * get_uid() { return "ers::Assertion"; }

    virtual void raise() const throw( std::exception ) { throw *this; }
    virtual const char * get_class_name() const { return get_uid(); }
    virtual ers::Issue * clone() const { return new ers::Assertion( *this ); }

public:
    Assertion( const ers::Context & context , const char * condition , const char * reason
    )
        : ers::Issue( context ) {
            set_value( "condition", condition );
            set_value( "reason", reason );
            std::ostream out;
            out << "Assertion ( " << condition << " ) failed because " << reason;
            set_message( out.str() );
        }

    Assertion( const ers::Context & context, const std::string & msg , const char *
    condition , const char * reason )
        : ers::Issue( context, msg ) {
            set_value( "condition", condition );
            set_value( "reason", reason );
        }

    Assertion( const ers::Context & context , const char * condition , const char * reason
```

```

, const std::exception & cause )
: ers::Issue( context, cause ) {
    set_value( "condition", condition );
    set_value( "reason", reason );
    std::ostringstream out;
    out << "Assertion (" << condition << ") failed because " << reason;
    set_message( out.str() );
}

const char * get_condition () {
    const char * val;
    get_value( "condition", val );
    return val;
}

const char * get_reason () {
    const char * val;
    get_value( "reason", val );
    return val;
}
};
}

```

The macro **ERS_DECLARE_ISSUE_BASE** has to be used if one wants to declare a new issue class, which inherits from one of the other custom ERS issue classes. For example, the following class inherits from the **ers::Assertion** class defined above:

```

ERS_DECLARE_ISSUE_BASE(ers, // namespace name
    Precondition, // issue name
    ers::Assertion, // base issue name
    "Precondition (" << condition << ") located in " << location
    << " failed because " << reason, // message
    ((const char *)condition) ((const char *)reason ), // base class attributes
    ((const char *)location) // this class attributes
)

```

The result of the **ERS_DECLARE_ISSUE_BASE** macro expansion looks like:

```

namespace ers {
class Precondition : public ers::Assertion {
    template <class> friend class ers::IssueRegistrator;
    Precondition() { ; }

    static const bool registered =
    ers::IssueRegistrator< ers::Precondition >::instance.
done;
    static const char * get_uid() { return "ers::Precondition"; }

    virtual void raise() const throw( std::exception ) { throw *this; }
    virtual const char * get_class_name() const { return get_uid(); }
    virtual ers::Issue * clone() const { return new ers::Precondition( *this ); }

public:
    Precondition( const ers::Context & context , const char * condition , const char *
reason, const char * location )
        : ers::Assertion( context, condition, reason ) {
            set_value( "location", location );
            std::ostringstream out;
            out << "Precondition (" << condition << ") located in " << location << ") failed because " <<
reason;
            set_message( out.str() );
        }

    Precondition( const ers::Context & context, const std::string & msg , const char *
condition , const char * reason, const char * location )
        : ers::Assertion( context, msg, condition, reason ) {
            set_value( "location", location );
        }

    Precondition( const ers::Context & context , const char * condition , const char *
reason , const char * location, const std::exception & cause )
        : ers::Assertion( context, condition, reason, cause ) {
            set_value( "location", location );
            std::ostringstream out;
            out << "Precondition (" << condition << ") located in " << location << ") failed because " <<
reason;
            set_message( out.str() );
        }

    const char * get_location () {
        const char * val;
        get_value( "location", val );
    }
}

```

```

        return val;
    }
};
}

```

ERS_HERE macro

The macro `ERS_HERE` is a convenience macro that is used to add the context information, like the file name, the line number and the signature of the function where the issue was constructed, to the new issue object. This means that when a new issue is created one shall always use `ERS_HERE` macro as the first parameter of the issue constructor.

Exception Handling

Functions, which can throw exceptions must be invoked inside **try...catch** statement. The following example shows a typical use case of handling ERS exceptions.

```

#include <ers/SampleIssues.h>

try {
    foo( );
}
catch ( ers::PermissionDenied & ex ) {
    ers::CantOpenFile issue( ERS_HERE, ex.get_file_name(), ex );
    ers::warning( issue );
}
catch ( ers::FileDoesNotExist & ex ) {
    ers::CantOpenFile issue( ERS_HERE, ex.get_file_name(), ex );
    ers::warning( issue );
}
catch ( ers::Issue & ex ) {
    ERS_DEBUG( 0, "Unknown issue caught: " << ex )
    ers::error( ex );
}
catch ( std::exception & ex ) {
    ers::CantOpenFile issue( ERS_HERE, "unknown", ex );
    ers::warning( issue );
}

```

This example demonstrates the main features of the ERS API:

- An issue does not have severity by itself. Severity of the issue is defined by the stream to which this issue is reported.
- An issue can be send to one of the existing ERS streams using one of the following functions: [ers::debug](#), [ers::info](#), [ers::warning](#), [ers::error](#), [ers::fatal](#)
- Any ERS issue has a constructor, which accepts another issue as its last parameter. If this constructor is used the new issue will hold the copy of the original one and will report it as its cause.
- Any ERS issue has a constructor, which accepts `std::exception` issue as its last parameter. If it is used the new issue will hold the copy of the given `std::exception` one and will report it as its cause.

Configuring ERS Streams

The ERS system provides multiple instances of the stream API, one per severity level, to report issues. The issues which are sent to different streams may be forwarded to different destinations depending on a particular stream configuration. By default the ERS streams are configured in the following way:

- [ers::debug](#) - "Istdout" - prints issues to the standard C++ output stream
- [ers::log](#) - "Istdout" - prints issues to the standard C++ output stream
- [ers::info](#) - "throttle,Istdout" - prints throttled issues to the standard C++ output stream
- [ers::warning](#) - "throttle,Istderr" - prints throttled issues to the standard C++ error stream

- `ers::error` - "throttle,lstderr" - prints throttled issues to the standard C++ error stream
- `ers::fatal` - "lstderr" - prints issues to the standard C++ error stream

Note: the letter "l" at the beginning of "lstdout" and "lstderr" names indicates that these stream implementations are thread-safe and can be safely used in multi-threaded applications so that issues reported from different threads will not be mixed up in the output.

In order to change the default configuration for an ERS stream one should use the `TDAQ_ERS_<SEVERITY>` environment variable. For example the following command:

```
export TDAQ_ERS_ERROR="lstderr,throw"
```

will cause all the issues, which are sent to the `ers::error` stream, been printed to the standard C++ error stream and then been thrown using the C++ throw operator.

A filter stream can also be associated with any severity level. For example:

```
export TDAQ_ERS_ERROR="lstderr,filter(ipc),throw"
```

The difference with the previous configuration is that only errors, which have "ipc" qualifier will be passed to the "throw" stream. Users can add any qualifiers to their specific issues by using the `Issue::add_qualifier` function. By default every issue has one qualifier associated with it - the name of the TDAQ software package, which builds the binary (a library or an application) where the issue object is constructed.

One can also define complex and reversed filters like in the following example:

```
<blockquote>
export TDAQ_ERS_ERROR="lstderr,filter(!ipc,!is),throw"
</blockquote>
```

This configuration will throw all the errors, which come neither from "ipc" nor from "is" TDAQ packages.

Existing Stream Implementations

ERS provides several stream implementations which can be used in any combination in ERS streams configurations. Here is the list of available stream implementations:

- "stdout" - prints issues to the standard C++ output stream. It is not thread-safe.
- "stderr" - prints issues to the standard C++ error stream. It is not thread-safe.
- "lstdout" - prints issues to the standard C++ output stream. It is thread-safe.
- "lstderr" - prints issues to the standard C++ error stream. It is thread-safe.
- "lock" - locks a global mutex for the duration of reporting an issue to the next streams in the given configuration. This stream can be used for adding thread-safety to an arbitrary non-thread-safe stream implementation. For example "lock,stdout" configuration is equivalent to "lstdout".
- "null" - silently drop any reported issue.
- "throw" - apply the C++ throw operator to the reported issue
- "abort" - calls abort() function for any issue reported
- "exit" - calls exit() function for any issue reported
- "filter(A,B,!C,...)" - pass through only issues, which have either A or B and don't have C qualifier
- "rfilter(RA,RB,!RC,...)" - the same as "filter" stream but treats all the given parameters as regular expressions.
- "throttle(initial_threshold, time_interval)" - rejects the same issues reported within the **time_interval** after passing through the **initial_threshold** number of them.

Custom Stream Implementation

While ERS provides a set of basic stream implementations one can also implement a custom one if this is required. Custom streams can be plugged into any existing application which is using ERS without recompiling this application.

Implementing a Custom Stream

In order to provide a new custom stream implementation one has to declare a sub-class of the `ers::OutputStream` class and implement its pure virtual method called `write`. Here is an example of how this is done by the `FilterStream` stream implementation:

```
void
ers::FilterStream::write( const ers::Issue & issue )
{
    if ( is_accepted( issue ) ) {
        chained().write( issue );
    }
}
```

An implementation of the `ers::OutputStream::write` function must decide whether to pass the given issue to the next stream in the chain or not. If a custom stream does not provide any filtering functionality then it shall always pass the input message to the next stream by using the `chained().write(issue)` code.

Registering a Custom Stream

In order to register and use a custom ERS stream implementation one can use a dedicated macro called `ERS_REGISTER_STREAM` in the following way:

```
ERS_REGISTER_OUTPUT_STREAM( ers::FilterStream, "filter", format )
```

The first parameter of this macro is the name of the class which implements the new stream; the second one gives a new stream name to be used in ERS stream configurations (this is the name which one can put to the `TDAQ_ERS_<SEVERITY>` environment variables); and the last parameter is a placeholder for the stream class constructor parameters. If the constructor of the new custom stream does not require parameters then last field of this macro should be left empty.

Using Custom Stream

In order to use a custom stream one has to build a new shared library from the class that implements this stream and pass this library to ERS by setting its name to the `TDAQ_ERS_STREAM_LIBS` environment variable. For example if this macro is set to the following value:

```
export TDAQ_ERS_STREAM_LIBS=MyCustomFilter
```

then ERS will be looking for the `libMyCustomFilter.so` library in all the directories which appear in the `LD_LIBRARY_PATH` environment variable.

Error Reporting in Multi-threaded Applications

ERS can be used for error reporting in multi-threaded applications. As C++ language does not provide a way of passing exceptions across thread boundaries, ERS provides the `ers::set_issue_catcher` function to overcome this limitation. When one of the threads of a software application catches an issue it can send it to one of the ERS streams using `ers::error`, `ers::fatal` or `ers::warning` functions. If no error catcher thread is installed in this application the new issue will be forwarded to the respective ERS stream implementations according to the stream configuration. Otherwise if a custom issue catcher is installed the issue will be passed to the dedicated thread which will call the custom error catcher function.

Setting up an Error Catcher

An error catcher should be installed by calling the `ers::set_issue_catcher` function and passing it a function object as parameter. This function object will be executed in the context of a dedicated thread (created by the `ers::set_issue_catcher` function) for every issue which is reported by the current application to `ers::fatal`, `ers::error` and `ers::warning` streams. The parameter of the `ers::set_issue_catcher` is of the `std::function<void (const ers::Issue &)>` type which allows to use plain C-style functions as well as C++ member functions for implementing a custom error catcher. For example one can define an error catcher as a member function of a certain class:

```
struct IssueCatcher {
    void handler( const ers::Issue & issue ) {
        std::cout << "IssueCatcher has been called: " << issue << std::endl;
    }
};
```

This error catcher can be registered with ERS in the following way:

```
IssueCatcher * catcher = new IssueCatcher();
ers::IssueCatcherHandler * handler;
try {
    handler = ers::set_issue_catcher( std::bind( &IssueCatcher::handler, catcher,
        std::placeholders::_1 ) );
}
catch(ers::IssueCatcherAlreadySet & ex){
    ...
}
```

Note that the error handling catcher can be set only once for the lifetime of an application. An attempt to set it again will fail and the `ers::IssueCatcherAlreadySet` exception will be thrown.

To unregister a previously installed issue catcher one need to destroy the handler that is returned by the `ers::set_issue_catcher` function using `delete` operator:

```
delete handler;
```

Receiving Issues Across Application Boundaries

There is a specific implementation of ERS input and output streams which allows to exchange issue across application boundaries, i.e. one process may receive ERS issues produces by another processes. The following example shows how to do that:

```
#include <ers/InputStream.h>
#include <ers/ers.h>

struct MyIssueReceiver : public ers::IssueReceiver {
    void receive( const ers::Issue & issue ) {
        std::cout << issue << std::endl;
    }
};

MyIssueReceiver * receiver = new MyIssueReceiver;
try {
    ers::StreamManager::instance().add_receiver( "mts", "*", receiver );
}
catch( ers::Issue & ex ) {
    ers::fatal( ex );
}
```

The `MyIssueReceiver` instance will be getting all messages, which are sent to the "mts" stream implementation by all applications working in the current TDAQ partition whose name will be taken from the `TDAQ_PARTITION` environment variable. Alternatively one may pass partition name explicitly via the "mts" stream parameters list:

```
MyIssueReceiver * receiver = new MyIssueReceiver;
try {
    ers::StreamManager::instance().add_receiver( "mts", {"my partition name", "*" }, receiver );
}
catch( ers::Issue & ex ) {
    ers::fatal( ex );
}
```

To cancel a previously made subscription one should use the `ers::StreamManager::remove_receiver` function and giving it a pointer to the corresponding receiver object, e.g.:

```
try {
    ers::StreamManager::instance().remove_receiver( receiver );
}
catch( ers::Issue & ex ) {
    ers::error( ex );
}
```

2 Namespace Index

2.1 Namespace List

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

ers	13
ers::format	20
thread	20

3 Hierarchical Index

3.1 Class Hierarchy

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

ers::Configuration	21
ers::Context	23
ers::LocalContext	36
ers::RemoteContext	43
Device	
ers::FormattedStandardStream< Device >	28
ers::StandardStream< Device >	49
exception	
ers::Issue	30
ers::AnyIssue	21
ers::InputStream	29
IssueCatcher	33
ers::IssueCatcherHandler	33
ers::IssueFactory	34
ers::IssueReceiver	35
MyIssueReceiver	41
ers::IssueRegistrar< T >	36

ers::LocalProcessContext	39
ers::OutputStream	42
ers::AbortStream	20
ers::ExitStream	26
ers::FilterStream	27
ers::FormattedStandardStream< Device >	28
ers::GlobalLockStream	29
ers::LockStream	40
ers::NullStream	41
ers::RFilterStream	47
ers::StandardStream< Device >	49
ers::StreamInitializer	50
ers::ThrottleStream	54
ers::ThrowStream	55
ers::PluginException	43
ers::PluginManager	43
ers::RemoteProcessContext	47
ers::Severity	49
ers::SingletonCreator< class >	49
ers::StandardStreamOutput	50
ers::StandardStream< Device >	49
ers::StreamManager	51
Test	53

4 Class Index

4.1 Class List

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

ers::AbortStream	
Aborts the current application	20
ers::AnyIssue	21
ers::Configuration	
Manager of ERS streams configuration	21

ers::Context	An abstract interface to access an Issue context	23
ers::ExitStream	Terminates the current application	26
ers::FilterStream	Filtering stream implementation	27
ers::FormattedStandardStream< Device >		28
ers::GlobalLockStream	Lock for ERS streams	29
ers::InputStream	ERS Issue input stream interface	29
ers::Issue	Base class for any user define issue	30
IssueCatcher		33
ers::IssueCatcherHandler	Implements issue catcher lifetime management	33
ers::IssueFactory	Implements factory pattern for user defined Issues	34
ers::IssueReceiver	ERS Issue receiver interface	35
ers::IssueRegistrar< T >		36
ers::LocalContext		36
ers::LocalProcessContext		39
ers::LockStream	Lock implementation for an ERS stream	40
MyIssueReceiver		41
ers::NullStream	Null stream	41
ers::OutputStream	ERS abstract output stream interface	42
ers::PluginException		43
ers::PluginManager		43
ers::RemoteContext		43
ers::RemoteProcessContext		47
ers::RFilterStream	Filtering stream implementation	47
ers::Severity		49
ers::SingletonCreator< class >		49

ers::StandardStream< Device >	49
Single line, human readable format stream	
ers::StandardStreamOutput	50
ers::StreamInitializer	50
ers::StreamManager	51
This class manages and provides access to ERS streams	
Test	53
ers::ThrottleStream	54
Throws issues as exceptions	
ers::ThrowStream	55
Throws issues as exceptions	

5 File Index

5.1 File List

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

bin/config.cxx	55
ers/AnyIssue.h	56
ers/Assertion.h	56
ers/Configuration.h	58
Ers header and documentation file	
ers/Context.h	58
ers/ers.h	59
Ers main header and documentation file	
ers/InputStream.h	60
Ers header and documentation file	
ers/Issue.h	69
Ers header and documentation file	
ers/IssueCatcherHandler.h	70
Ers header and documentation file	
ers/IssueFactory.h	70
Ers header and documentation file	
ers/IssueReceiver.h	71
Ers header and documentation file	
ers/LocalContext.h	71
ers/LocalStream.h	72
Ers header and documentation file	
ers/OutputStream.h	72
Ers header and documentation file	

ers/ RemoteContext.h	73
ers/ SampleIssues.h Sample ERS issues	73
ers/ Severity.h Ers header and documentation file	74
ers/ StandardStreamOutput.h Ers header file	75
ers/ StreamFactory.h Ers header and documentation file	75
ers/ StreamManager.h Ers header and documentation file	76
ers/internal/ AbortStream.h Ers header file	61
ers/internal/ ExitStream.h Ers header file	62
ers/internal/ FilterStream.h Ers header file	62
ers/internal/ FormattedStandardStream.h Ers header file	62
ers/internal/ GlobalLockStream.h	??
ers/internal/ IssueDeclarationMacro.h	??
ers/internal/ LockStream.h Ers header file	63
ers/internal/ macro.h Ers header file	64
ers/internal/ NullStream.h Ers header file	65
ers/internal/ PluginManager.h Ers header file	66
ers/internal/ RFilterStream.h Ers header file	66
ers/internal/ SingletonCreator.h Ers header file	67
ers/internal/ StandardStream.h Ers header file	67
ers/internal/ ThrottleStream.h	??
ers/internal/ ThrowStream.h Ers header file	68
ers/internal/ Util.h Ers header file	68

6 Namespace Documentation

6.1 ERS Namespace Reference

Namespaces

- [format](#)

Classes

- class [AnyIssue](#)
- class [SingletonCreator](#)
- class [Configuration](#)
Manager of ERS streams configuration.
- class [Context](#)
An abstract interface to access an [Issue](#) context.
- class [InputStream](#)
ERS [Issue](#) input stream interface.
- struct [AbortStream](#)
Aborts the current application.
- struct [ExitStream](#)
Terminates the current application.
- class [FilterStream](#)
Filtering stream implementation.
- struct [FormattedStandardStream](#)
- struct [GlobalLockStream](#)
Lock for ERS streams.
- struct [LockStream](#)
Lock implementation for an ERS stream.
- struct [NullStream](#)
Null stream.
- class [PluginException](#)
- class [PluginManager](#)
- class [RFilterStream](#)
Filtering stream implementation.
- struct [StandardStream](#)
Single line, human readable format stream.
- class [ThrottleStream](#)
Throws issues as exceptions.
- struct [ThrowStream](#)
Throws issues as exceptions.
- class [IssueRegistrar](#)
- class [Issue](#)
Base class for any user define issue.
- class [IssueCatcherHandler](#)
Implements issue catcher lifetime management.
- class [IssueFactory](#)
Implements factory pattern for user defined Issues.
- class [IssueReceiver](#)
ERS [Issue](#) receiver interface.
- struct [LocalProcessContext](#)

- class [LocalContext](#)
- class [OutputStream](#)
 - ERS abstract output stream interface.*
- struct [RemoteProcessContext](#)
- class [RemoteContext](#)
- struct [Severity](#)
- struct [StandardStreamOutput](#)
- class [StreamManager](#)
 - This class manages and provides access to ERS streams.*
- class [StreamInitializer](#)

Typedefs

- typedef [Issue](#) **Exception**
- typedef `std::map< std::string, std::string >` **string_map**

Enumerations

- enum **severity** {
Debug, Log, Information, Warning, Error, Fatal }

Functions

- `std::ostream & operator<<` (`std::ostream &`, `const ers::Configuration &`)
- `IssueCatcherHandler * set_issue_catcher` (`const std::function< void(const ers::Issue &)> &catcher`)
- `int debug_level` ()
- `void debug` (`const Issue &issue`, `int level=debug_level()`)
- `void error` (`const Issue &issue`)
- `void fatal` (`const Issue &issue`)
- `void info` (`const Issue &issue`)
- `void log` (`const Issue &issue`)
- `int verbosity_level` ()
- `void warning` (`const Issue &issue`)
- `int enable_core_dump` ()
- `void tokenize` (`const std::string &text`, `const std::string &separators`, `std::vector< std::string > &tokens`)
- `int read_from_environment` (`const char *name`, `int default_value`)
- `const char * read_from_environment` (`const char *name`, `const char *default_value`)
- `std::ostream & operator<<` (`std::ostream &`, `const ers::Issue &`)
- `std::ostream & operator<<` (`std::ostream &`, `const IssueFactory &factory`)
- *streaming operator*
- `severity parse` (`const std::string &s`, `severity &`)
- `Severity parse` (`const std::string &s`, `Severity &`)
- `std::string to_string` (`severity s`)
- `std::string to_string` (`Severity s`)
- *Transforms a severity type into the corresponding string.*
- `std::ostream & operator<<` (`std::ostream &out`, `ers::severity severity`)
- `std::ostream & operator<<` (`std::ostream &out`, `const ers::Severity &severity`)
- `std::istream & operator>>` (`std::istream &in`, `ers::severity &severity`)
- `std::istream & operator>>` (`std::istream &in`, `ers::Severity &severity`)
- `std::ostream & operator<<` (`std::ostream &`, `const ers::StreamManager &`)

6.1.1 Detailed Description

This is a wrapping namespace for all ERS classes and global functions.

6.1.2 Function Documentation

6.1.2.1 `void ers::debug (const Issue & issue, int level = debug_level()) [inline]`

This function sends the issue to the ERS DEBUG stream which corresponds to the given debug level.

Parameters

<i>issue</i>	the issue to be reported debug level which will be associated with the reported issue
--------------	---

6.1.2.2 `int ers::debug_level () [inline]`

This function returns the current debug level for ERS.

6.1.2.3 `void ers::error (const Issue & issue) [inline]`

This function sends the issue to the ERS ERROR stream.

Parameters

<i>issue</i>	the issue to be reported
--------------	--------------------------

6.1.2.4 `void ers::fatal (const Issue & issue) [inline]`

This function sends the issue to the ERS FATAL stream.

Parameters

<i>issue</i>	the issue to be reported
--------------	--------------------------

6.1.2.5 `void ers::info (const Issue & issue) [inline]`

This function sends the issue to the ERS INFO stream.

Parameters

<i>issue</i>	the issue to be reported
--------------	--------------------------

6.1.2.6 `void ers::log (const Issue & issue) [inline]`

This function sends the issue to the ERS LOG stream.

Parameters

<i>issue</i>	the issue to be reported
--------------	--------------------------

6.1.2.7 `std::ostream & ers::operator<< (std::ostream & out, const ers::Issue & issue)`

Standard streaming operator - puts the issue in human readable format into the standard out stream.

Parameters

<i>out</i>	the destination out stream
<i>issue</i>	the Issue to be printed

6.1.2.8 `ers::severity` `ers::parse (const std::string & string, ers::severity & s)`

Parses a string and extracts a severity

Parameters

<i>s</i>	the string to parse
----------	---------------------

Returns

a severity value

6.1.2.9 `ers::Severity ers::parse (const std::string & string, ers::Severity & s)`

Parses a string and extracts a severity

Parameters

<i>s</i>	the string to parse
----------	---------------------

Returns

a severity value

6.1.2.10 `ers::IssueCatcherHandler * ers::LocalStream::set_issue_catcher (const std::function< void(const ers::Issue &)> & catcher) [inline]`

This function sets up the local issue handler function. This function will be executed in the context of dedicated thread which will be created as a result of this call. All the issues which are reported via the [ers::error](#), [ers::fatal](#) and [ers::warning](#) functions will be forwarded to this thread.

Parameters

<i>issue</i>	the issue to be reported
--------------	--------------------------

Returns

pointer to the handler object, which allows to remove the catcher by just destroying this object. If an application ignores this return value there will no way of de installing the issue catcher.

Exceptions

<code>ers::IssueCatcherAlready-Set</code>	for safety reasons local issue handler can be set only once
---	---

See Also

[ers::error\(\)](#)
[ers::fatal\(\)](#)
[ers::warning\(\)](#)

6.1.2.11 `std::string ers::to_string (ers::Severity severity)`

Transforms a severity type into the corresponding string.

Parameters

<i>s</i>	severity
----------	----------

Returns

pointer to string with associated text

6.1.2.12 `int ers::verbosity_level () [inline]`

This function returns the current verbosity level for ERS.

6.1.2.13 `void ers::warning (const Issue & issue) [inline]`

This function sends the issue to the ERS WARNING stream.

Parameters

<i>issue</i>	the issue to be reported
--------------	--------------------------

6.2 ers::format Namespace Reference

Enumerations

- enum **Token** {
Severity, Time, Position, Context,
Host, PID, TID, User,
CWD, Function, Line, Text,
Stack, Cause, Parameters, Qualifiers }

6.2.1 Detailed Description

This is a helper class that provides implementation of an output stream that can be used to customise the output format of the streamed issues.

Author

Serguei Kolos

6.3 thread Namespace Reference

6.3.1 Detailed Description

This is a wrapping namespace for ERS classes and global functions which can be used for the local inter-thread error reporting.

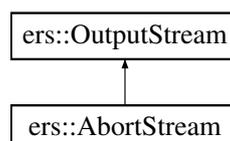
7 Class Documentation

7.1 ers::AbortStream Struct Reference

Aborts the current application.

```
#include <AbortStream.h>
```

Inheritance diagram for ers::AbortStream:



Public Member Functions

- void **write** (const **Issue** &issue) override

Additional Inherited Members

7.1.1 Detailed Description

Aborts the current application.

This class implements a stream, which aborts the application whenever it receives an issue. In order to employ this implementation in a stream configuration the name to be used is "abort". E.g. the following configuration will print a first issue that is passed to the FATAL ERS stream to the standard output and then aborts the application:

```
export TDAQ_ERS_FATAL="stdout, abort"
```

Author

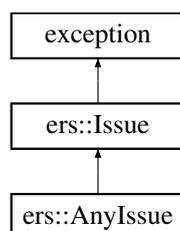
Serguei Kolos

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

- [ers/internal/AbortStream.h](#)

7.2 ers::AnyIssue Class Reference

Inheritance diagram for ers::AnyIssue:



Public Member Functions

- **AnyIssue** (const std::string &type, const [ers::Context](#) &context, const std::string &message="")
- **AnyIssue** (const std::string &type, [Severity severity](#), const [ers::Context](#) &context, const system_clock::time_point &time, const std::string &message, const std::vector< std::string > &qualifiers, const std::map< std::string, std::string > ¶meters, const [ers::Issue](#) *cause=0)
- virtual [ers::Issue](#) * **clone** () const
- virtual const char * **get_class_name** () const
Get key for class (used for serialisation)
- virtual void **raise** () const
throws a copy of this issue preserving the real issue type

Additional Inherited Members

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

- [ers/AnyIssue.h](#)

7.3 ers::Configuration Class Reference

Manager of ERS streams configuration.

```
#include <Configuration.h>
```

Public Member Functions

- int [debug_level](#) () const
returns current debug level
- int [verbosity_level](#) () const
returns current verbosity level
- void [debug_level](#) (int debug_level)
- void [verbosity_level](#) (int verbosity_level)
can be used to set the current verbosity level

Static Public Member Functions

- static [Configuration](#) & [instance](#) ()
return the singleton

Friends

- template<class >
class **SingletonCreator**
- std::ostream & **operator**<< (std::ostream &, const [ers::Configuration](#) &)

7.3.1 Detailed Description

Manager of ERS streams configuration.

The [Configuration](#) class provides API for configuring ERS output streams.

Author

Serguei Kolos

Version

1.2

See Also

[ers::debug](#)
[ers::error](#)
[ers::fatal](#)
[ers::information](#)
[ers::log](#)
[ers::warning](#)

7.3.2 Member Function Documentation

7.3.2.1 int [ers::Configuration::debug_level](#) () const [inline]

returns current debug level

<

7.3.2.2 void [ers::Configuration::debug_level](#) (int *debug_level*) [inline]

Parameters

<code>debug_level</code>	can be used to set the current debug level
--------------------------	--

7.3.2.3 ers::Configuration & ers::Configuration::instance () [static]

return the singleton

This method returns the singleton instance. It should be used for every operation on the factory.

Returns

a reference to the singleton instance

Singleton instance

7.3.2.4 int ers::Configuration::verbosity_level () const [inline]

returns current verbosity level

<

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

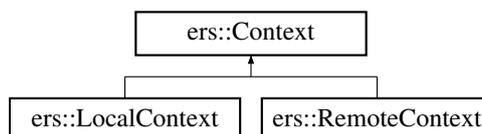
- [ers/Configuration.h](#)
- [src/Configuration.cxx](#)

7.4 ers::Context Class Reference

An abstract interface to access an [Issue](#) context.

```
#include <Context.h>
```

Inheritance diagram for ers::Context:



Public Member Functions

- `std::string position (int verbosity=ers::Configuration::instance().verbosity_level()) const`
- `std::vector< std::string > stack () const`
- `virtual Context * clone () const =0`
- `virtual const char * cwd () const =0`
- `virtual const char * file_name () const =0`
- `virtual const char * function_name () const =0`
- `virtual const char * host_name () const =0`
- `virtual int line_number () const =0`
- `virtual const char * package_name () const =0`
- `virtual pid_t process_id () const =0`
- `virtual pid_t thread_id () const =0`
- `virtual void *const * stack_symbols () const =0`
- `virtual int stack_size () const =0`
- `virtual int user_id () const =0`
- `virtual const char * user_name () const =0`
- `virtual const char * application_name () const =0`

7.4.1 Detailed Description

An abstract interface to access an [Issue](#) context.

This class provides an abstract interface to access the context of an issue.

Author

Serguei Kolos

7.4.2 Member Function Documentation

7.4.2.1 `virtual const char* ers::Context::application_name () const` [pure virtual]

Returns

application name

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.2 `virtual Context* ers::Context::clone () const` [pure virtual]

Returns

copy of the current context

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.3 `virtual const char* ers::Context::cwd () const` [pure virtual]

Returns

current working directory of the process

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.4 `virtual const char* ers::Context::file_name () const` [pure virtual]

Returns

name of the file which created the issue

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.5 `virtual const char* ers::Context::function_name () const` [pure virtual]

Returns

name of the function which created the issue

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.6 `virtual const char* ers::Context::host_name () const` [pure virtual]

Returns

host where the process is running

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.7 `virtual int ers::Context::line_number () const` [pure virtual]

Returns

line number, in which the issue has been created

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.8 `virtual const char* ers::Context::package_name () const` [pure virtual]

Returns

CMT package name

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.9 `std::string ers::Context::position (int verbosity = ers::Configuration::instance () . verbosity_level ()) const`

Returns

position in the code

Pretty printed code position format: package_name/file_name:line_number <function_name>

Returns

reference to string containing format

7.4.2.10 `virtual pid_t ers::Context::process_id () const` [pure virtual]

Returns

process id

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.11 `std::vector< std::string > ers::Context::stack () const`

Returns

stack frames vector

7.4.2.12 `virtual int ers::Context::stack_size () const` [pure virtual]

Returns

number of frames in stack

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.13 `virtual void* const* ers::Context::stack_symbols () const` [pure virtual]

Returns

stack frames

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.14 `virtual pid_t ers::Context::thread_id () const` [pure virtual]

Returns

thread id

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.15 `virtual int ers::Context::user_id() const [pure virtual]`

Returns

user id

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

7.4.2.16 `virtual const char* ers::Context::user_name() const [pure virtual]`

Returns

user name

Implemented in [ers::RemoteContext](#), and [ers::LocalContext](#).

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

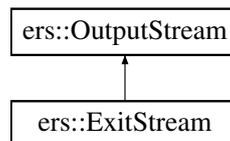
- [ers/Context.h](#)
- [src/Context.cxx](#)

7.5 ers::ExitStream Struct Reference

Terminates the current application.

```
#include <ExitStream.h>
```

Inheritance diagram for `ers::ExitStream`:



Public Member Functions

- [ExitStream](#) (const std::string &exit_code="1")
- void **write** (const [Issue](#) &issue) override

Additional Inherited Members

7.5.1 Detailed Description

Terminates the current application.

This class implements a stream, which exits the application whenever it receives an issue. In order to employ this implementation in a stream configuration the name to be used is "exit". E.g. the following configuration will print a first issue that is passed to the FATAL ERS stream to the standard output and then terminates the application:

```
export TDAQ_ERS_FATAL="stdout,exit"
```

Author

Serguei Kolos

Version

1.0

7.5.2 Constructor & Destructor Documentation

7.5.2.1 ers::ExitStream::ExitStream (const std::string & *exit_code* = "1") [explicit]

Creates a new instance of Exit stream.

Parameters

<i>exit_code</i>	This sting shall contain a number that will be used as the application exit status.
------------------	---

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

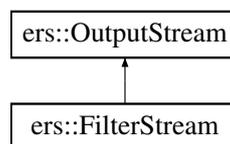
- [ers/internal/ExitStream.h](#)
- [src/streams/ExitStream.cxx](#)

7.6 ers::FilterStream Class Reference

Filtering stream implementation.

```
#include <FilterStream.h>
```

Inheritance diagram for ers::FilterStream:



Public Member Functions

- [FilterStream](#) (const std::string &format)
- void [write](#) (const [Issue](#) &issue) override

Additional Inherited Members

7.6.1 Detailed Description

Filtering stream implementation.

This stream offers basic filtering capability. It hooks up in front of another stream and filters the messages that are passed to it with respect to the given configuration. Filtering is based on using plain string comparison of the issue's qualifiers with the given configuration tokens. A stream configuration is composed of the stream name, that is "filter", followed by brackets with a comma separated list of string tokens, where any token can be preceded by an exclamation mark. For example:

- filter(internal,test) - this stream will pass messages that have either "internal" or "test" qualifier.
- filter(!internal,!test) this stream will pass messages that have neither "internal" nor "test" qualifier.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 ers::FilterStream::FilterStream (const std::string & *format*) [explicit]

Constructor that creates a new instance of the filter stream with the given configuration. Only messages that pass the given filter will go through, the others will be filtered out.

Parameters

<i>format</i>	filter expression.
---------------	--------------------

7.6.3 Member Function Documentation

7.6.3.1 void ers::FilterStream::write (const Issue & issue) [override],[virtual]

Write method basically calls `is_accept` to check if the issue is accepted. If this is the case, the `write` method on the chained stream is called with `issue`.

Parameters

<i>issue</i>	issue to be sent.
--------------	-------------------

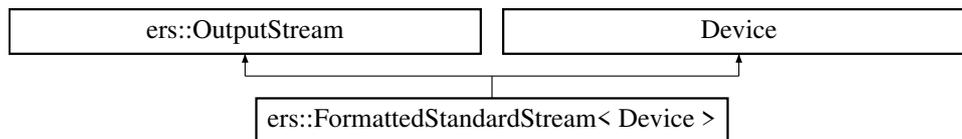
Implements [ers::OutputStream](#).

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

- [ers/internal/FilterStream.h](#)
- [src/streams/FilterStream.cxx](#)

7.7 ers::FormattedStandardStream< Device > Struct Template Reference

Inheritance diagram for `ers::FormattedStandardStream< Device >`:



Public Member Functions

- [FormattedStandardStream](#) (const std::string &format)
- void **write** (const [Issue](#) &issue) override

Additional Inherited Members

7.7.1 Constructor & Destructor Documentation

7.7.1.1 template<class Device > ers::FormattedStandardStream< Device >::FormattedStandardStream (const std::string & format) [explicit]

This constructor creates a new formatted output stream. The format parameter is a comma separated list of tokens that defines the which attributes of the issues will be printed as well as their order. Here is the list of supported tokens: "severity, time, position, context, pid, tid, cwd, function, line, cause, stack, parameters, qualifiers, user"

Parameters

<i>format</i>	a list of issue attributes that have to be printed for each issue
---------------	---

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

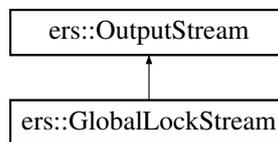
- [ers/internal/FormattedStandardStream.h](#)

7.8 ers::GlobalLockStream Struct Reference

Lock for ERS streams.

```
#include <GlobalLockStream.h>
```

Inheritance diagram for ers::GlobalLockStream:



Public Member Functions

- void **write** (const [Issue](#) &issue) override

Additional Inherited Members

7.8.1 Detailed Description

Lock for ERS streams.

This class can be used to protect output produced by distinct ERS streams, e.g. INFO and LOG, from been mixed up when originated from concurrent threads. The name to be used for this stream in stream configurations is "glock".

Author

Serguei Kolos

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

- ers/internal/GlobalLockStream.h

7.9 ers::InputStream Class Reference

ERS [Issue](#) input stream interface.

```
#include <InputStream.h>
```

Protected Member Functions

- [InputStream](#) ()
Will be called when a new issue is received.
- void **receive** (const [Issue](#) &issue)

Friends

- class **StreamManager**

7.9.1 Detailed Description

ERS [Issue](#) input stream interface.

ERS [Issue](#) input stream interface.

Author

Serguei Kolos

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

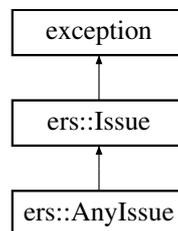
- [ers/InputStream.h](#)
- [src/InputStream.cxx](#)

7.10 ers::Issue Class Reference

Base class for any user define issue.

```
#include <Issue.h>
```

Inheritance diagram for ers::Issue:

**Public Member Functions**

- [Issue](#) (const [Context](#) &context, const std::string &message=std::string())
- [Issue](#) (const [Context](#) &context, const std::exception &cause)
- [Issue](#) (const [Context](#) &context, const std::string &message, const std::exception &cause)
- [Issue](#) (const [Issue](#) &other)
- virtual [Issue](#) * [clone](#) () const =0
- virtual const char * [get_class_name](#) () const =0
Get key for class (used for serialisation)
- virtual void [raise](#) () const =0
throws a copy of this issue preserving the real issue type
- void [add_qualifier](#) (const std::string &qualif)
adds a qualifier to the issue
- const [Issue](#) * [cause](#) () const
return the cause Issue of this Issue
- const [Context](#) & [context](#) () const
Context of the issue.
- const std::string & [message](#) () const
General cause of the issue.
- const std::vector< std::string > & [qualifiers](#) () const
return array of qualifiers
- const string_map & [parameters](#) () const
return array of parameters
- [Severity](#) [severity](#) () const
severity of the issue
- template<class Precision = std::chrono::seconds>
std::string [time](#) (const std::string &format="%Y-%b-%d %H:%M:%S", bool isUTC=false) const
string representation of local time of the issue

- `template<class Precision >`
`std::string localtime (const std::string &format="%Y-%b-%d %H:%M:%S") const`
string representation of UTC time of the issue
- `template<class Precision >`
`std::string gmtime (const std::string &format="%Y-%b-%d %H:%M:%S") const`
- `std::time_t time_t () const`
seconds since 1 Jan 1970
- `const system_clock::time_point & ptime () const`
original time point of the issue
- `const char * what () const noexcept`
General cause of the issue.
- `ers::Severity set_severity (ers::Severity severity) const`
- `void wrap_message (const std::string &begin, const std::string &end)`

Protected Member Functions

- `Issue (Severity severity, const system_clock::time_point &time, const ers::Context &context, const std::string &message, const std::vector< std::string > &qualifiers, const std::map< std::string, std::string > ¶meters, const ers::Issue *cause=0)`
Gets a value of any type that has an input operator for the standard stream defined.
- `template<typename T >`
`void get_value (const std::string &key, T &value) const`
- `void get_value (const std::string &key, const char *&value) const`
- `void get_value (const std::string &key, std::string &value) const`
Sets a value of any type that has an output operator for the standard stream defined.
- `template<typename T >`
`void set_value (const std::string &key, T value)`
- `void set_message (const std::string &message)`
- `void prepend_message (const std::string &message)`

Friends

- class **IssueFactory**

7.10.1 Detailed Description

Base class for any user define issue.

This is a base class for any user define issue. The class stores all attributes declared in a user define descendant class in a hashmap as sting key/value pairs. The object defines a number of methods for providing access to this map. For an example of how to define a custom subclass of the [Issue](#) have a look at the [SampleIssues.h](#) file.

See Also

[ers::IssueFactory](#)
[SampleIssues.h](#)

7.10.2 Constructor & Destructor Documentation

7.10.2.1 Issue::Issue (const Context & context, const std::string & message = std::string())

This constructor create a new issue with the given message.

Parameters

<i>context</i>	the context of the Issue , e.g where in the code the issue appeared
<i>message</i>	the user message associated with this issue

7.10.2.2 `Issue::Issue (const Context & context, const std::exception & cause)`

This constructor takes another exceptions as its cause.

Parameters

<i>context</i>	the context of the Issue , e.g where in the code the issue appeared
<i>cause</i>	the other exception that has caused this one

7.10.2.3 `Issue::Issue (const Context & context, const std::string & message, const std::exception & cause)`

This constructor takes another exceptions as its cause.

Parameters

<i>context</i>	the context of the Issue , e.g where in the code did the issue appear
<i>message</i>	the user message associated with this issue
<i>cause</i>	exception that caused the current issue

7.10.3 Member Function Documentation

7.10.3.1 `void Issue::add_qualifier (const std::string & qualifier)`

adds a qualifier to the issue

Add a new qualifier to the qualifiers list of this issue

Parameters

<i>qualifier</i>	the qualifier to add
------------------	----------------------

7.10.3.2 `const Issue* ers::Issue::cause () const [inline]`

return the cause [Issue](#) of this [Issue](#)

<

7.10.3.3 `const Context& ers::Issue::context () const [inline]`

[Context](#) of the issue.

<

7.10.3.4 `const std::string& ers::Issue::message () const [inline]`

General cause of the issue.

<

7.10.3.5 `const string_map& ers::Issue::parameters () const [inline]`

return array of parameters

<

7.10.3.6 `void Issue::prepend_message (const std::string & msg) [protected]`

Adds the given text to the beginning of the issue's message

Parameters

<i>msg</i>	text to be prepended
------------	----------------------

7.10.3.7 `const system_clock::time_point& ers::Issue::ptime () const` [inline]

original time point of the issue

<

7.10.3.8 `const std::vector<std::string>& ers::Issue::qualifiers () const` [inline]

return array of qualifiers

<

7.10.3.9 `ers::Severity ers::Issue::severity () const` [inline]

severity of the issue

<

7.10.3.10 `const char* ers::Issue::what () const` [inline], [noexcept]

General cause of the issue.

<

7.10.3.11 `void Issue::wrap_message (const std::string & begin, const std::string & end)`

Adds the given text strings to the beginning and to the end of the issue's message

Parameters

<i>begin</i>	text to be prepended
<i>end</i>	text to be appended

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

- [ers/Issue.h](#)
- [src/Issue.cxx](#)

7.11 IssueCatcher Struct Reference

Public Member Functions

- void **handler** (const [ers::Issue](#) &issue)

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

- [test/test.cxx](#)

7.12 ers::IssueCatcherHandler Class Reference

Implements issue catcher lifetime management.

```
#include <IssueCatcherHandler.h>
```

Friends

- class **LocalStream**

7.12.1 Detailed Description

Implements issue catcher lifetime management.

This is a helper class that is used to support issue catcher management. An instance of this class holds a reference to the last successfully registered issue catcher. When this instance is destroyed the issue catcher is unregistered.

Author

Serguei Kolos

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

- [ers/IssueCatcherHandler.h](#)
- [src/IssueCatcherHandler.cxx](#)

7.13 ers::IssueFactory Class Reference

Implements factory pattern for user defined Issues.

```
#include <IssueFactory.h>
```

Public Member Functions

- [Issue](#) * [create](#) (const std::string &name, const [Context](#) &context) const
build an empty issue for a given name
- [Issue](#) * [create](#) (const std::string &name, const [Context](#) &context, [Severity](#) severity, const system_clock::time_point &time, const std::string &message, const std::vector< std::string > &qualifiers, const std::map< std::string, std::string > ¶meters, const [Issue](#) *cause=0) const
build issue out of all the given parameters
- void [register_issue](#) (const std::string &name, IssueCreator creator)
register an issue factory

Static Public Member Functions

- static [IssueFactory](#) & [instance](#) ()
method to access singleton

Friends

- template<class >
class [SingletonCreator](#)

7.13.1 Detailed Description

Implements factory pattern for user defined Issues.

This class implements factory pattern for Issues. The main responsibility of this class is to keep track of the existing types of Issues. Each user defined issue class should register one factory method for creating instances of this class. This is required for reconstructing issues produced in the context of a different process.

Author

Serguei Kolos

7.13.2 Member Function Documentation

7.13.2.1 ers::Issue * ers::IssueFactory::create (const std::string & name, const Context & context) const

build an empty issue for a given name

Builds an issue out of the name it was registered with

Parameters

<i>name</i>	the name used to indentify the class
-------------	--------------------------------------

Returns

an newly allocated instance of type *name* or [AnyIssue](#)

Note

If the requested type cannot be resolved an instance of type [AnyIssue](#)

7.13.2.2 ers::IssueFactory & ers::IssueFactory::instance () [static]

method to access singleton

Returns the singleton instance of the factory.

Returns

a reference to the singleton instance

7.13.2.3 void ers::IssueFactory::register_issue (const std::string & name, IssueCreator creator)

register an issue factory

Register an issue type with the factory

Parameters

<i>name</i>	the name that will be used to lookup new instances
<i>creator</i>	a pointer to the function used to create new instance for that particular type of function

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

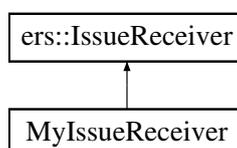
- [ers/IssueFactory.h](#)
- [src/IssueFactory.cxx](#)

7.14 ers::IssueReceiver Class Reference

ERS [Issue](#) receiver interface.

```
#include <IssueReceiver.h>
```

Inheritance diagram for ers::IssueReceiver:



Public Member Functions

- virtual void `receive` (const `Issue` &issue)=0
Is called when a new issue is received.

7.14.1 Detailed Description

ERS `Issue` receiver interface.

ERS `Issue` receiver abstract interface. User must create a subclass of this class in order to receive issues.

Author

Serguei Kolos

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

- [ers/IssueReceiver.h](#)

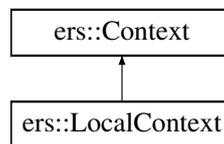
7.15 `ers::IssueRegistrator< T >` Class Template Reference

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

- [ers/Issue.h](#)

7.16 `ers::LocalContext` Class Reference

Inheritance diagram for `ers::LocalContext`:



Public Member Functions

- `LocalContext` (const char *`package_name`, const char *`filename`, int `line_number`, const char *`function_name`, bool `debug`=false)
- virtual `Context` * `clone` () const
- const char * `cwd` () const
- const char * `file_name` () const
- const char * `function_name` () const
- const char * `host_name` () const
- int `line_number` () const
- const char * `package_name` () const
- pid_t `process_id` () const
- pid_t `thread_id` () const
- void *const * `stack_symbols` () const
- int `stack_size` () const
- int `user_id` () const
- const char * `user_name` () const
- const char * `application_name` () const

Static Public Member Functions

- static void `resetProcessContext` ()

7.16.1 Constructor & Destructor Documentation

7.16.1.1 `ers::LocalContext::LocalContext (const char * package_name, const char * filename, int line_number, const char * function_name, bool debug = false)`

creates a new instance of a local context for an issue. This constructor should not be called directly, instead one should use the `ERS_HERE` macro.

Parameters

<i>package_name</i>	name of the current sw package
<i>filename</i>	name of the source code file
<i>line_number</i>	line_number in the source code
<i>function_name</i>	name of the current function

7.16.2 Member Function Documentation

7.16.2.1 `const char * ers::LocalContext::application_name () const` `[virtual]`

Returns

application name

Implements [ers::Context](#).

7.16.2.2 `virtual Context* ers::LocalContext::clone () const` `[inline],[virtual]`

<

Returns

copy of the current context

Implements [ers::Context](#).

7.16.2.3 `const char* ers::LocalContext::cwd () const` `[inline],[virtual]`

<

Returns

current working directory of the process

Implements [ers::Context](#).

7.16.2.4 `const char* ers::LocalContext::file_name () const` `[inline],[virtual]`

<

Returns

name of the file which created the issue

Implements [ers::Context](#).

7.16.2.5 `const char* ers::LocalContext::function_name () const` `[inline],[virtual]`

<

Returns

name of the function which created the issue

Implements [ers::Context](#).

7.16.2.6 `const char* ers::LocalContext::host_name () const` `[inline],[virtual]`

<

Returns

host where the process is running

Implements [ers::Context](#).

7.16.2.7 `int ers::LocalContext::line_number () const` `[inline],[virtual]`

<

Returns

line number, in which the issue has been created

Implements [ers::Context](#).

7.16.2.8 `const char* ers::LocalContext::package_name () const` `[inline],[virtual]`

<

Returns

CMT package name

Implements [ers::Context](#).

7.16.2.9 `pid_t ers::LocalContext::process_id () const` `[inline],[virtual]`

<

Returns

process id

Implements [ers::Context](#).

7.16.2.10 `int ers::LocalContext::stack_size () const` `[inline],[virtual]`

<

Returns

number of frames in stack

Implements [ers::Context](#).

7.16.2.11 `void* const* ers::LocalContext::stack_symbols () const` `[inline],[virtual]`

<

Returns

stack frames

Implements [ers::Context](#).

7.16.2.12 `pid_t ers::LocalContext::thread_id () const` `[inline],[virtual]`

<

Returns

thread id

Implements [ers::Context](#).

7.16.2.13 `int ers::LocalContext::user_id () const` `[inline],[virtual]`

<

Returns

user id

Implements [ers::Context](#).

7.16.2.14 `const char* ers::LocalContext::user_name () const` `[inline],[virtual]`

<

Returns

user name

Implements [ers::Context](#).

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

- [ers/LocalContext.h](#)
- [src/LocalContext.cxx](#)

7.17 `ers::LocalProcessContext` Struct Reference

Public Member Functions

- **LocalProcessContext** (`const char *const host_name`, `const char *const cwd`, `int uid`, `const char *const uname`)

Public Attributes

- `const char *const` [m_host_name](#)
- `const char *const` [m_cwd](#)
- `const int` [m_uid](#)
- `const char *const` [m_uname](#)

7.17.1 Member Data Documentation

7.17.1.1 `const char* const ers::LocalProcessContext::m_cwd`

process cwd

7.17.1.2 `const char* const ers::LocalProcessContext::m_host_name`

host name

7.17.1.3 `const int ers::LocalProcessContext::m_uid`

user id

7.17.1.4 `const char* const ers::LocalProcessContext::m_uname`

user name

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

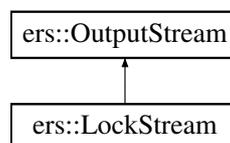
- [ers/LocalContext.h](#)

7.18 `ers::LockStream` Struct Reference

Lock implementation for an ERS stream.

```
#include <LockStream.h>
```

Inheritance diagram for `ers::LockStream`:



Public Member Functions

- void **write** (const [Issue](#) &issue) override

Additional Inherited Members

7.18.1 Detailed Description

Lock implementation for an ERS stream.

This class can be used to lock a particular ERS output streams to prevent output to this stream produced by concurrent threads from been mixed up. In order to employ this implementation in a stream configuration the name to be used is "lock". E.g. the following configuration will assure that the output sent to the LOG stream by concurrent threads is never mixed up:

```
export TDAQ_ERS_LOG="lock, stdout"
```

Author

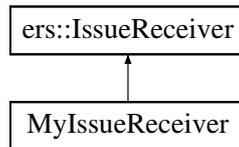
Serguei Kolos

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

- [ers/internal/LockStream.h](#)

7.19 MyIssueReceiver Struct Reference

Inheritance diagram for MyIssueReceiver:



Public Member Functions

- void **receive** (const [ers::Issue](#) &issue)
Is called when a new issue is received.

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

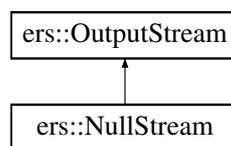
- test/receiver.cxx

7.20 ers::NullStream Struct Reference

Null stream.

```
#include <NullStream.h>
```

Inheritance diagram for ers::NullStream:



Public Member Functions

- void **write** (const [Issue](#) &) override
- bool **isNull** () const override

Additional Inherited Members

7.20.1 Detailed Description

Null stream.

This stream implementation silently discards any issue that is sent to it. In order to employ this implementation in a stream configuration the name to be used is "null". E.g. the following configuration will result in no output been produced for the ERROR stream:

```
export TDAQ_ERS_ERROR="null"
```

Author

Serguei Kolos

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

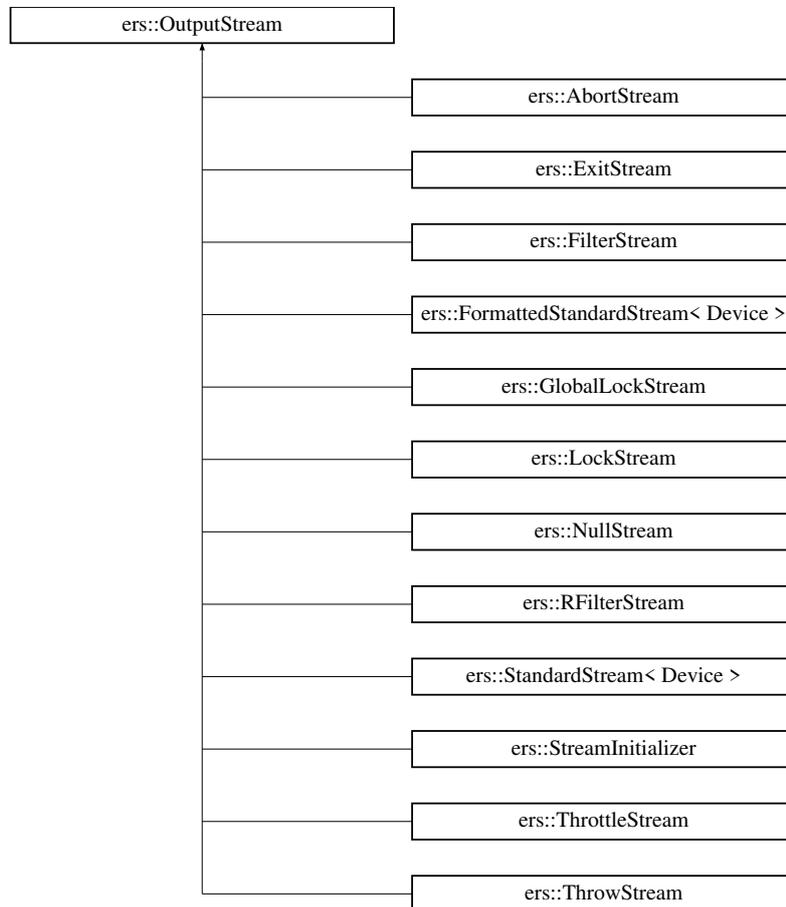
- ers/internal/[NullStream.h](#)

7.21 ers::OutputStream Class Reference

ERS abstract output stream interface.

```
#include <OutputStream.h>
```

Inheritance diagram for ers::OutputStream:



Public Member Functions

- virtual `~OutputStream ()`
Sends the issue into this stream.
- virtual void **write** (const `Issue` &issue)=0

Protected Member Functions

- `OutputStream & chained ()`
- virtual bool **isNull** () const

Friends

- class **StreamManager**

7.21.1 Detailed Description

ERS abstract output stream interface.

The abstract ERS output stream interface. This interface defines the pure virtual method to `write` issues to the stream. Any subclass must implement this method.

Author

Serguei Kolos

Version

1.0

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

- [ers/OutputStream.h](#)
- [src/OutputStream.cxx](#)

7.22 ers::PluginException Class Reference

Public Member Functions

- **PluginException** (const std::string &reason)
- const std::string & **reason** () const

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

- [ers/internal/PluginManager.h](#)

7.23 ers::PluginManager Class Reference

Public Member Functions

- [PluginManager](#) ()
- [~PluginManager](#) ()

7.23.1 Constructor & Destructor Documentation

7.23.1.1 ers::PluginManager::PluginManager ()

Constructor loads the plugins.

7.23.1.2 ers::PluginManager::~~PluginManager ()

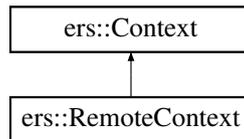
Destructor unloads the plugins.

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

- [ers/internal/PluginManager.h](#)
- [src/PluginManager.cxx](#)

7.24 ers::RemoteContext Class Reference

Inheritance diagram for ers::RemoteContext:



Public Member Functions

- [RemoteContext](#) (const std::string &package, const std::string &filename, int [line_number](#), const std::string &function_name, const [RemoteProcessContext](#) &pc)
- virtual [Context](#) * [clone](#) () const
- const char * [cwd](#) () const
- const char * [file_name](#) () const
- const char * [function_name](#) () const
- const char * [host_name](#) () const
- int [line_number](#) () const
- const char * [package_name](#) () const
- pid_t [process_id](#) () const
- pid_t [thread_id](#) () const
- void *const * [stack_symbols](#) () const
- int [stack_size](#) () const
- int [user_id](#) () const
- const char * [user_name](#) () const
- const char * [application_name](#) () const

7.24.1 Constructor & Destructor Documentation

7.24.1.1 `ers::RemoteContext::RemoteContext (const std::string & package, const std::string & filename, int line_number, const std::string & function_name, const RemoteProcessContext & pc) [inline]`

creates a new instance of a context for an [Issue](#) that has been originated from another process.

Parameters

<i>package_name</i>	name of the sw package
<i>filename</i>	name of the source code file
<i>line_number</i>	line_number in the source code
<i>function_name</i>	name of the function

7.24.2 Member Function Documentation

7.24.2.1 `const char* ers::RemoteContext::application_name () const [inline],[virtual]`

<

Returns

application name

Implements [ers::Context](#).

7.24.2.2 `virtual Context* ers::RemoteContext::clone () const [inline],[virtual]`

<

Returns

copy of the current context

Implements [ers::Context](#).

7.24.2.3 `const char* ers::RemoteContext::cwd () const [inline],[virtual]`

<

Returns

current working directory of the process

Implements [ers::Context](#).

7.24.2.4 `const char* ers::RemoteContext::file_name () const [inline],[virtual]`

<

Returns

name of the file which created the issue

Implements [ers::Context](#).

7.24.2.5 `const char* ers::RemoteContext::function_name () const [inline],[virtual]`

<

Returns

name of the function which created the issue

Implements [ers::Context](#).

7.24.2.6 `const char* ers::RemoteContext::host_name () const [inline],[virtual]`

<

Returns

host where the process is running

Implements [ers::Context](#).

7.24.2.7 `int ers::RemoteContext::line_number () const [inline],[virtual]`

<

Returns

line number, in which the issue has been created

Implements [ers::Context](#).

7.24.2.8 `const char* ers::RemoteContext::package_name () const [inline],[virtual]`

<

Returns

CMT package name

Implements [ers::Context](#).

7.24.2.9 `pid_t ers::RemoteContext::process_id() const [inline],[virtual]`

<

Returns

process id

Implements [ers::Context](#).

7.24.2.10 `int ers::RemoteContext::stack_size() const [inline],[virtual]`

<

Returns

number of frames in stack

Implements [ers::Context](#).

7.24.2.11 `void* const* ers::RemoteContext::stack_symbols() const [inline],[virtual]`

<

Returns

stack frames

Implements [ers::Context](#).

7.24.2.12 `pid_t ers::RemoteContext::thread_id() const [inline],[virtual]`

<

Returns

thread id

Implements [ers::Context](#).

7.24.2.13 `int ers::RemoteContext::user_id() const [inline],[virtual]`

<

Returns

user id

Implements [ers::Context](#).

7.24.2.14 `const char* ers::RemoteContext::user_name() const [inline],[virtual]`

<

Returns

user name

Implements [ers::Context](#).

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

- [ers/RemoteContext.h](#)

7.25 `ers::RemoteProcessContext` Struct Reference

Public Member Functions

- **RemoteProcessContext** (const std::string &host_name, int pid, int tid, const std::string &cwd, int uid, const std::string &uname, const std::string &app_name)

Public Attributes

- const std::string [m_host_name](#)
- const pid_t [m_pid](#)
- const pid_t [m_tid](#)
- const std::string [m_cwd](#)
- const int [m_uid](#)
- const std::string [m_uname](#)
- const std::string [m_app_name](#)

7.25.1 Member Data Documentation

7.25.1.1 const std::string `ers::RemoteProcessContext::m_app_name`

application name

7.25.1.2 const std::string `ers::RemoteProcessContext::m_cwd`

process cwd

7.25.1.3 const std::string `ers::RemoteProcessContext::m_host_name`

host name

7.25.1.4 const pid_t `ers::RemoteProcessContext::m_pid`

process id

7.25.1.5 const pid_t `ers::RemoteProcessContext::m_tid`

thread id

7.25.1.6 const int `ers::RemoteProcessContext::m_uid`

user id

7.25.1.7 const std::string `ers::RemoteProcessContext::m_uname`

user name

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

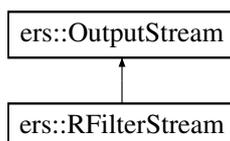
- [ers/RemoteContext.h](#)

7.26 `ers::RFilterStream` Class Reference

Filtering stream implementation.

```
#include <RFilterStream.h>
```

Inheritance diagram for `ers::RFilterStream`:



Public Member Functions

- [RFilterStream](#) (const std::string &format)
- void [write](#) (const [Issue](#) &issue) override

Additional Inherited Members

7.26.1 Detailed Description

Filtering stream implementation.

This stream offers basic filtering capability. It hooks up in front of another stream and filters the messages that are passed to it with respect to the given configuration. Filtering is based on using matching of the issue's function name and qualifiers with the given regular expressions. A stream configuration is composed of the stream name, that is "rfilter", followed by brackets with a comma separated list of regular expressions, where any expression can be preceded by an exclamation mark. For example:

- `rfilter(create.*,new.*)` - this stream will pass messages that have originated from a function that starts with either "create" or "new" string.
- `rfilter(!create.*,!new.*)` - this stream will pass messages that have originated from a function that starts with neither "create" nor "new" string.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 `ers::RFilterStream::RFilterStream (const std::string & format)`

Constructor

Parameters

<i>chained</i>	the chained stream, which will be filtered. Only messages, which pass the filter will go to the chained stream on the stack and owned by the current object, i.e it will be deleted upon destruction
<i>format</i>	describes filter expression.

7.26.3 Member Function Documentation

7.26.3.1 `void ers::RFilterStream::write (const Issue & issue) [override], [virtual]`

Write method basically calls `is_accept` to check if the issue is accepted. If this is the case, the `write` method on the chained stream is called with `issue`.

Parameters

<i>issue</i>	issue to be sent.
--------------	-------------------

Implements [ers::OutputStream](#).

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

- `ers/internal/RFilterStream.h`
- `src/streams/RFilterStream.cxx`

7.27 ers::Severity Struct Reference

Public Member Functions

- **Severity** (ers::severity severity, int level=0)
- **operator ers::severity** ()

Public Attributes

- enum severity **type**
- int **rank**

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

- [ers/Severity.h](#)

7.28 ers::SingletonCreator< class > Class Template Reference

Static Public Member Functions

- static T * **create** ()

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

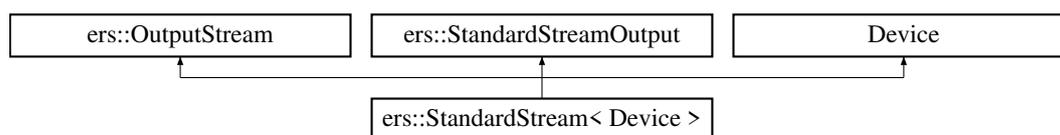
- [ers/Configuration.h](#)
- [ers/internal/SingletonCreator.h](#)

7.29 ers::StandardStream< Device > Struct Template Reference

Single line, human readable format stream.

```
#include <StandardStream.h>
```

Inheritance diagram for ers::StandardStream< Device >:



Public Member Functions

- **StandardStream** (const std::string &file_name)
- void **write** (const [Issue](#) &issue)

Additional Inherited Members

7.29.1 Detailed Description

```
template<class Device>struct ers::StandardStream< Device >
```

Single line, human readable format stream.

This class streams an issue into standard C++ output stream.

Author

Serguei Kolos

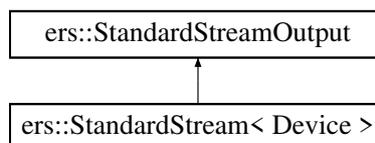
The documentation for this struct was generated from the following file:

- [ers/internal/StandardStream.h](#)

7.30 ers::StandardStreamOutput Struct Reference

```
#include <StandardStreamOutput.h>
```

Inheritance diagram for ers::StandardStreamOutput:

**Static Public Member Functions**

- static std::ostream & **print** (std::ostream &out, const [Issue](#) &issue, int verbosity)
- static std::ostream & **println** (std::ostream &out, const [Issue](#) &issue, int verbosity)

7.30.1 Detailed Description

This class provides a namespace for the functions that can be used to print ERS issues to a standard C++ output stream.

Author

Serguei Kolos

Version

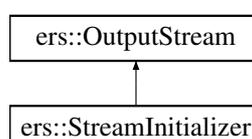
1.0

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

- [ers/StandardStreamOutput.h](#)
- [src/StandardStreamOutput.cxx](#)

7.31 ers::StreamInitializer Class Reference

Inheritance diagram for ers::StreamInitializer:



Public Member Functions

- **StreamInitializer** ([StreamManager](#) &manager)
- void **write** (const [Issue](#) &issue)

Additional Inherited Members

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

- src/StreamManager.cxx

7.32 ers::StreamManager Class Reference

This class manages and provides access to ERS streams.

```
#include <StreamManager.h>
```

Public Member Functions

- [~StreamManager](#) ()
- void [debug](#) (const [Issue](#) &issue, int level)
sends an [Issue](#) to the debug stream
- void [error](#) (const [Issue](#) &issue)
sends an issue to the error stream
- void [fatal](#) (const [Issue](#) &issue)
sends an issue to the fatal stream
- void [information](#) (const [Issue](#) &issue)
sends an issue to the information stream
- void [log](#) (const [Issue](#) &issue)
sends an issue to the log stream
- void [warning](#) (const [Issue](#) &issue)
sends an issue to the warning stream
- void **add_receiver** (const std::string &stream, const std::string &filter, [ers::IssueReceiver](#) *receiver)
- void **add_receiver** (const std::string &stream, const std::initializer_list< std::string > ¶ms, [ers::IssueReceiver](#) *receiver)
- void **remove_receiver** ([ers::IssueReceiver](#) *receiver)
- void **add_output_stream** ([ers::severity](#) severity, [ers::OutputStream](#) *new_stream)
- void [report_issue](#) ([ers::severity](#) type, const [Issue](#) &issue)

Static Public Member Functions

- static [StreamManager](#) & [instance](#) ()
return the singleton

Friends

- class **StreamInitializer**
- class **ers::LocalStream**
- class **ers::ErrorHandler**
- template<class >
class **SingletonCreator**

7.32.1 Detailed Description

This class manages and provides access to ERS streams.

The `StreamManager` class is responsible for creating and handling all the ERS streams used by an application. It implements the singleton pattern and handles a table of the different stream attached to each severity. When issues occur they are dispatched to an appropriate stream by the singleton instance of this class. Users should not use this class directly. In order to report issues users should use global functions declared in the `ers` namespace.

Author

Serguei Kolos

See Also

[ers::debug](#)
[ers::error](#)
[ers::fatal](#)
[ers::infomation](#)
[ers::log](#)
[ers::warning](#)

7.32.2 Constructor & Destructor Documentation

7.32.2.1 `ers::StreamManager::~~StreamManager ()`

Destructor - basic cleanup

7.32.3 Member Function Documentation

7.32.3.1 `void ers::StreamManager::debug (const Issue & issue, int level)`

sends an [Issue](#) to the debug stream

Sends an issue to the debug stream

Parameters

<i>issue</i>	the Issue to send
<i>level</i>	the debug level.

7.32.3.2 `void ers::StreamManager::error (const Issue & issue)`

sends an issue to the error stream

Sends an [Issue](#) to the error stream

Parameters

<i>issue</i>	
--------------	--

7.32.3.3 `void ers::StreamManager::fatal (const Issue & issue)`

sends an issue to the fatal stream

Sends an [Issue](#) to the fatal error stream

Parameters

<i>issue</i>	
--------------	--

7.32.3.4 void ers::StreamManager::information (const Issue & *issue*)

sends an issue to the information stream

Sends an issue to the info stream

Parameters

<i>issue</i>	the Issue to send
--------------	-----------------------------------

7.32.3.5 ers::StreamManager & ers::StreamManager::instance () [static]

return the singleton

This method returns the singleton instance. It should be used for every operation on the factory.

Returns

a reference to the singleton instance

Singleton instance

7.32.3.6 void ers::StreamManager::log (const Issue & *issue*)

sends an issue to the log stream

Sends an issue to the log stream

Parameters

<i>issue</i>	the Issue to send
--------------	-----------------------------------

7.32.3.7 void ers::StreamManager::report_issue (ers::severity *type*, const Issue & *issue*)

Sends an [Issue](#) to an appropriate stream

Parameters

<i>type</i>	
<i>issue</i>	

7.32.3.8 void ers::StreamManager::warning (const Issue & *issue*)

sends an issue to the warning stream

Sends an [Issue](#) to the warning stream

Parameters

<i>issue</i>	the issue to send
--------------	-------------------

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

- [ers/StreamManager.h](#)
- [src/StreamManager.cxx](#)

7.33 Test Struct Reference

Public Member Functions

- void **pass** (int step)

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

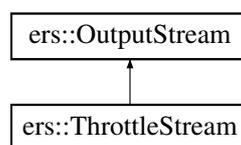
- test/test.cxx

7.34 ers::ThrottleStream Class Reference

Throws issues as exceptions.

```
#include <ThrottleStream.h>
```

Inheritance diagram for ers::ThrottleStream:



Public Member Functions

- **ThrottleStream** (const std::string &criteria)
- void **write** (const ers::Issue &issue) override

Additional Inherited Members

7.34.1 Detailed Description

Throws issues as exceptions.

This class implements a stream that swallows identical issues if they are sent to the given stream too often. In order to employ this implementation in a stream configuration the name to be used is "throttle". E.g. the following configuration will suppress identical issues from been printed to the standard output:

```
export TDAQ_ERS_FATAL="throttle(10, 20),stdout"
```

This stream has two configuration parameters:

- first parameter defines an initial number of identical messages after which the throttling shall be started
- second parameter defines a timeout in seconds after which the throttling is reset to its initial state if no issues of a given type have been reported in this period

Author

Serguei Kolos

7.34.2 Member Function Documentation

7.34.2.1 void ers::ThrottleStream::write (const ers::Issue & issue) [override],[virtual]

Write method basically calls `throttle` to check if the issue is accepted. If this is the case, the `write` method on the chained stream is called with `issue`.

Parameters

<i>issue</i>	issue to be sent.
--------------	-------------------

Implements [ers::OutputStream](#).

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

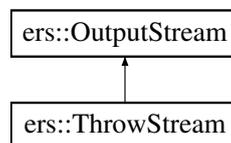
- [ers/internal/ThrottleStream.h](#)
- [src/streams/ThrottleStream.cxx](#)

7.35 ers::ThrowStream Struct Reference

Throws issues as exceptions.

```
#include <ThrowStream.h>
```

Inheritance diagram for `ers::ThrowStream`:



Public Member Functions

- void **write** (const [Issue](#) &issue) override

Additional Inherited Members

7.35.1 Detailed Description

Throws issues as exceptions.

This class implements a stream, which throws the issue written to it as a normal C++ exception. In order to employ this implementation in a stream configuration the name to be used is "throw". E.g. the following configuration will print a first issue that is passed to the FATAL ERS stream to the standard output and then throws this issue as a C++ exception:

```
export TDAQ_ERS_FATAL="stdout,throw"
```

Author

Serguei Kolos

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

- [ers/internal/ThrowStream.h](#)

8 File Documentation

8.1 bin/config.cxx File Reference

```
#include <ers/ers.h>
#include <string.h>
```

Functions

- void **print_description** ()
- void **print_usage** ()
- int **main** (int argc, char **argv)

8.1.1 Detailed Description

Prints current configuration of all ERS streams, taking into account environment variables.

8.2 ers/AnyIssue.h File Reference

```
#include <ers/Issue.h>
```

Classes

- class [ers::AnyIssue](#)

Namespaces

- [ers](#)

8.2.1 Detailed Description

This file defines the [ers::AnyIssue](#) class

8.3 ers/Assertion.h File Reference

```
#include <ers/Issue.h>
#include <ers/Severity.h>
#include <ers/StreamManager.h>
```

Macros

- #define **ERS_INTERNAL_ABORT**(_) ::abort()
- #define **ERS_ASSERT_MSG**(expression, message)
- #define **ERS_ASSERT**(expression) ERS_ASSERT_MSG(expression, "of bad internal state")
- #define **ERS_PRECONDITION**(expression) ERS_ASSERT_MSG(expression, "of bad external parameter")
- #define **ERS_RANGE_CHECK**(min, val, max)
- #define **ERS_STRICT_RANGE_CHECK**(min, val, max)

8.3.1 Detailed Description

This file defines the assertion class and the associated macros.

8.3.2 Macro Definition Documentation

8.3.2.1 #define ERS_ASSERT_MSG(*expression*, *message*)

Value:

```
{ \
if( !(expression) ) \
{ \
    std::ostringstream ers_report_impl_out_buffer; \
    ers_report_impl_out_buffer << BOOST_PP_IF( ERS_IS_EMPTY( message ), "of unknown reason", message ); \
    std::string reason = ers_report_impl_out_buffer.str(); \
    ers::Assertion __issue__( ERS_HERE, #expression, reason.c_str() ); \
    ers::StreamManager::instance().report_issue( ers::Fatal, \
        __issue__ ); \
    ERS_INTERNAL_ABORT(#expression); \
}}
```

8.3.2.2 #define ERS_PRECONDITION(*expression*) ERS_ASSERT_MSG(*expression*, "of bad external parameter")

If *expression* is not true, then an issue of type `ers::Assertion` is thrown. It should be used to check the parameters, which are passed to functions.

Note

This macro is defined to empty statement if the `ERS_NO_DEBUG` macro is defined

8.3.2.3 #define ERS_RANGE_CHECK(*min*, *val*, *max*)

Value:

```
ERS_ASSERT_MSG( (min) <= (val) && (val) <= (max), \
    val << " is not in [" << min << ", " << max << "]" range" )
```

the following condition

```
( min <= val ) && ( val <= max )
```

If this expression is not true, then an issue of type `ers::Assertion` is thrown.

Note

This macro is defined to empty statement if the `ERS_NO_DEBUG` macro is defined

8.3.2.4 #define ERS_STRICT_RANGE_CHECK(*min*, *val*, *max*)

Value:

```
ERS_ASSERT_MSG( (min) < (val) && (val) < (max), \
    val << " is not in (" << min << ", " << max << ") range" )
```

the following condition

```
( min < val ) && ( val < max )
```

If this expression is not true, then an issue of type `ers::Assertion` is thrown.

Note

This macro is disabled if the `ERS_NO_DEBUG` macro is defined

8.4 ers/Configuration.h File Reference

ers header and documentation file

```
#include <iostream>
```

Classes

- class [ers::SingletonCreator< class >](#)
- class [ers::Configuration](#)
Manager of ERS streams configuration.

Namespaces

- [ers](#)

Functions

- `std::ostream & ers::operator<< (std::ostream &, const ers::Configuration &)`

8.4.1 Detailed Description

ers header and documentation file This file defines ERS Configuration class.

Author

Serguei Kolos

8.5 ers/Context.h File Reference

```
#include <string>
#include <vector>
#include <ers/Configuration.h>
```

Classes

- class [ers::Context](#)
An abstract interface to access an [Issue](#) context.

Namespaces

- [ers](#)

8.5.1 Detailed Description

This file defines the [ers::Context](#) interface.

8.6 ers/ers.h File Reference

ers main header and documentation file

```
#include <sys/resource.h>
#include <functional>
#include <sstream>
#include <ers/StreamManager.h>
#include <ers/Configuration.h>
#include <ers/Issue.h>
#include <ers/Assertion.h>
#include <ers/Severity.h>
#include <ers/LocalStream.h>
#include <boost/preprocessor/logical/not.hpp>
#include <boost/preprocessor/punctuation/comma_if.hpp>
#include <boost/preprocessor/facilities/is_empty.hpp>
```

Namespaces

- [ers](#)
- [thread](#)

Macros

- #define **ERS_REPORT_IMPL**(stream, issue, message, level)
- #define **ERS_DEBUG**(level, message)
- #define **ERS_INFO**(message)
- #define **ERS_LOG**(message)

Typedefs

- typedef Issue **ers::Exception**

Functions

- IssueCatcherHandler * [ers::set_issue_catcher](#) (const std::function< void(const [ers::Issue](#) &)> &catcher)
- int [ers::debug_level](#) ()
- void [ers::debug](#) (const Issue &issue, int level=debug_level())
- void [ers::error](#) (const Issue &issue)
- void [ers::fatal](#) (const Issue &issue)
- void [ers::info](#) (const Issue &issue)
- void [ers::log](#) (const Issue &issue)
- int [ers::verbosity_level](#) ()
- void [ers::warning](#) (const Issue &issue)
- int [ers::enable_core_dump](#) ()

8.6.1 Detailed Description

ers main header and documentation file This file defines the basic ERS API.

Author

Serguei Kolos

Version

1.1

8.6.2 Macro Definition Documentation**8.6.2.1 #define ERS_DEBUG(level, message)****Value:**

```
do { \
if ( ers::debug_level() >= level ) \
{ \
ERS_REPORT_IMPL( ers::debug, ers::Message, message, level ); \
} } while(0)
```

8.6.2.2 #define ERS_INFO(message)**Value:**

```
do { \
{ \
ERS_REPORT_IMPL( ers::info, ers::Message, message, ERS_EMPTY ); \
} } while(0)
```

8.6.2.3 #define ERS_LOG(message)**Value:**

```
do { \
{ \
ERS_REPORT_IMPL( ers::log, ers::Message, message, ERS_EMPTY ); \
} } while(0)
```

8.6.2.4 #define ERS_REPORT_IMPL(stream, issue, message, level)**Value:**

```
{ \
std::ostringstream ers_report_impl_out_buffer; \
ers_report_impl_out_buffer << message; \
stream( issue( ERS_HERE, ers_report_impl_out_buffer.str() ) \
BOOST_PP_COMMA_IF( BOOST_PP_NOT( ERS_IS_EMPTY( ERS_EMPTY level ) ) ) level ); \
}
```

8.7 ers/InputStream.h File Reference**ers header and documentation file**

```
#include <string>
#include <ers/Issue.h>
#include <ers/IssueReceiver.h>
#include <ers/StreamFactory.h>
```

Classes

- class [ers::InputStream](#)
ERS Issue input stream interface.

Namespaces

- [ers](#)

Macros

- `#define ERS_REGISTER_INPUT_STREAM(class, name, params)`

8.7.1 Detailed Description

ers header and documentation file Defines abstract interface for ERS input streams.

Author

Serguei Kolos

8.7.2 Macro Definition Documentation

8.7.2.1 `#define ERS_REGISTER_INPUT_STREAM(class, name, params)`

Value:

```
namespace { \
    struct InputStreamRegistrar { \
        static ers::InputStream * create( const std::initializer_list<std::string> & params ) \
        { return new class( params ); } \
        InputStreamRegistrar() \
        { ers::StreamFactory::instance().register_in_stream( name, create ); } \
    } registrar_mp; \
}
```

8.8 ers/internal/AbortStream.h File Reference

ers header file

```
#include <ers/OutputStream.h>
```

Classes

- struct [ers::AbortStream](#)
Aborts the current application.

Namespaces

- [ers](#)

8.8.1 Detailed Description

ers header file This file defines AbortStream ERS stream.

Author

Serguei Kolos

8.9 ers/internal/ExitStream.h File Reference

ers header file

```
#include <ers/OutputStream.h>
```

Classes

- struct [ers::ExitStream](#)
Terminates the current application.

Namespaces

- [ers](#)

8.9.1 Detailed Description

ers header file This file defines ExitStream ERS stream.

Author

Serguei Kolos

8.10 ers/internal/FilterStream.h File Reference

ers header file

```
#include <ers/OutputStream.h>
```

Classes

- class [ers::FilterStream](#)
Filtering stream implementation.

Namespaces

- [ers](#)

8.10.1 Detailed Description

ers header file This file defines FilterStream ERS stream.

Author

Serguei Kolos

8.11 ers/internal/FormattedStandardStream.h File Reference

ers header file

```
#include <map>
#include <ers/OutputStream.h>
#include <ers/internal/FormattedStandardStream.inc>
```

Classes

- struct [ers::FormattedStandardStream](#)< Device >

Namespaces

- [ers](#)
- [ers::format](#)

Enumerations

- enum **Token** {
 Severity, Time, Position, Context,
 Host, PID, TID, User,
 CWD, Function, Line, Text,
 Stack, Cause, Parameters, Qualifiers }

8.11.1 Detailed Description

ers header file This file defines FormattedStandardStream ERS stream.

Author

Serguei Kolos

8.12 ers/internal/LockStream.h File Reference

ers header file

```
#include <ers/OutputStream.h>  
#include <mutex>
```

Classes

- struct [ers::LockStream](#)
Lock implementation for an ERS stream.

Namespaces

- [ers](#)

8.12.1 Detailed Description

ers header file This file defines LockStream ERS stream.

Author

Serguei Kolos

8.13 ers/internal/macro.h File Reference

ers header file

```
#include <ers/ers.h>
#include <ers/StreamFactory.h>
#include <ers/StandardStreamOutput.h>
#include <boost/preprocessor/cat.hpp>
```

Macros

- `#define ERS_REGISTER_OUTPUT_STREAM(class, name, param)`
- `#define ERS_INTERNAL_DEBUG(level, message)`
- `#define ERS_INTERNAL_INFO(message)`
- `#define ERS_INTERNAL_WARNING(message)`
- `#define ERS_INTERNAL_ERROR(message)`
- `#define ERS_INTERNAL_FATAL(message)`

8.13.1 Detailed Description

ers header file This file defines some internal ERS macros.

Author

Serguei Kolos

8.13.2 Macro Definition Documentation

8.13.2.1 `#define ERS_INTERNAL_DEBUG(level, message)`

Value:

```
{ \
if ( ers::debug_level() >= level ) \
{ \
std::ostringstream out; \
out << message; \
ers::InternalMessage info( ERS_HERE, out.str() ); \
info.set_severity( ers::Severity( ers::Debug, level ) ); \
ers::StandardStreamOutput::println( std::cout, info, 0 ); \
} }
```

8.13.2.2 `#define ERS_INTERNAL_ERROR(message)`

Value:

```
{ \
std::ostringstream out; \
out << message; \
ers::InternalMessage info( ERS_HERE, out.str() ); \
info.set_severity( ers::Error ); \
ers::StandardStreamOutput::println( std::cerr, info, 0 ); \
}
```

8.13.2.3 `#define ERS_INTERNAL_FATAL(message)`

Value:

```
{ \
  std::ostringstream out; \
  out << message; \
  ers::InternalMessage info( ERS_HERE, out.str() ); \
  info.set_severity( ers::Fatal ); \
  ers::StandardStreamOutput::println( std::cerr, info, 0 ); \
  ::exit( 13 ); \
}
```

8.13.2.4 #define ERS_INTERNAL_INFO(message)

Value:

```
{ \
  std::ostringstream out; \
  out << message; \
  ers::InternalMessage info( ERS_HERE, out.str() ); \
  info.set_severity( ers::Information ); \
  ers::StandardStreamOutput::println( std::cout, info, 0 ); \
}
```

8.13.2.5 #define ERS_INTERNAL_WARNING(message)

Value:

```
{ \
  std::ostringstream out; \
  out << message; \
  ers::InternalMessage info( ERS_HERE, out.str() ); \
  info.set_severity( ers::Warning ); \
  ers::StandardStreamOutput::println( std::cerr, info, 0 ); \
}
```

8.13.2.6 #define ERS_REGISTER_OUTPUT_STREAM(class, name, param)

Value:

```
namespace { \
  struct BOOST_PP_CAT( OutputStreamRegistrar, __LINE__ ) { \
    static ers::OutputStream * create( const std::string & param ) \
    { return new class( param ); } \
    BOOST_PP_CAT( OutputStreamRegistrar, __LINE__ ) () \
    { ers::StreamFactory::instance().register_out_stream( name, create ); } \
  } BOOST_PP_CAT( registrar, __LINE__ ); \
}
```

8.14 ers/internal/NullStream.h File Reference

ers header file

```
#include <ers/OutputStream.h>
#include <ers/InputStream.h>
```

Classes

- struct [ers::NullStream](#)
Null stream.

Namespaces

- [ers](#)

8.14.1 Detailed Description

ers header file This file defines NullStream ERS stream.

Author

Serguei Kolos

8.15 ers/internal/PluginManager.h File Reference

ers header file

```
#include <string>
#include <map>
```

Classes

- class [ers::PluginException](#)
- class [ers::PluginManager](#)

Namespaces

- [ers](#)

8.15.1 Detailed Description

ers header file This file defines PluginManager ERS class.

Author

Serguei Kolos

8.16 ers/internal/RFilterStream.h File Reference

ers header file

```
#include <ers/OutputStream.h>
#include <boost/regex.hpp>
```

Classes

- class [ers::RFilterStream](#)
Filtering stream implementation.

Namespaces

- [ers](#)

8.16.1 Detailed Description

ers header file This file defines RFilterStream ERS stream.

Author

Serguei Kolos

8.17 ers/internal/SingletonCreator.h File Reference

ers header file

```
#include <mutex>
```

Classes

- class [ers::SingletonCreator](#)< class >

Namespaces

- [ers](#)

8.17.1 Detailed Description

ers header file This file defines SingletonCreator ERS class.

Author

Serguei Kolos

8.18 ers/internal/StandardStream.h File Reference

ers header file

```
#include <ers/OutputStream.h>  
#include <ers/StandardStreamOutput.h>
```

Classes

- struct [ers::StandardStream](#)< Device >
Single line, human readable format stream.

Namespaces

- [ers](#)

8.18.1 Detailed Description

ers header file This file defines StandardStream ERS stream.

Author

Serguei Kolos

8.19 ers/internal/ThrowStream.h File Reference

ers header file

```
#include <ers/OutputStream.h>
```

Classes

- struct [ers::ThrowStream](#)
Throws issues as exceptions.

Namespaces

- [ers](#)

8.19.1 Detailed Description

ers header file This file defines ThrowStream ERS stream.

Author

Serguei Kolos

8.20 ers/internal/Util.h File Reference

ers header file

```
#include <string>  
#include <vector>
```

Namespaces

- [ers](#)

Functions

- void **ers::tokenize** (const std::string &text, const std::string &separators, std::vector< std::string > &tokens)
- int **ers::read_from_environment** (const char *name, int default_value)
- const char * **ers::read_from_environment** (const char *name, const char *default_value)

8.20.1 Detailed Description

ers header file This file defines some common utility functions for ERS.

Author

Serguei Kolos

8.21 ers/Issue.h File Reference

ers header and documentation file

```
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <map>
#include <string>
#include <iostream>
#include <sstream>
#include <memory>
#include <chrono>
#include <ers/IssueFactory.h>
#include <ers/LocalContext.h>
#include <ers/Severity.h>
#include <ers/internal/IssueDeclarationMacro.h>
```

Classes

- class [ers::IssueRegistrator< T >](#)
- class [ers::Issue](#)
Base class for any user define issue.

Namespaces

- [ers](#)

Typedefs

- typedef `std::map< std::string, std::string >` **ers::string_map**

Functions

- `std::ostream & ers::operator<< (std::ostream &, const ers::Issue &)`
- `ERS_DECLARE_ISSUE(ers, NoValue,"value for the \\"<< key<< "\" key is not set ",((std::string) key))`
template< typename T > void
ers if (it==m_values.end())
- `std::istream in (it->second)`

Variables

- in **value**

8.21.1 Detailed Description

ers header and documentation file This file defines the [ers::Issue](#) class, which is the base class for any user defined issue.

Author

Serguei Kolos

8.22 ers/IssueCatcherHandler.h File Reference

ers header and documentation file

Classes

- class [ers::IssueCatcherHandler](#)
Implements issue catcher lifetime management.

Namespaces

- [ers](#)

8.22.1 Detailed Description

ers header and documentation file This file declares utility class which is used to unregister [IssueCatcher](#).

Author

Serguei Kolos

8.23 ers/IssueFactory.h File Reference

ers header and documentation file

```
#include <chrono>
#include <string>
#include <vector>
#include <map>
#include <ers/Severity.h>
```

Classes

- class [ers::SingletonCreator< class >](#)
- class [ers::IssueFactory](#)
Implements factory pattern for user defined Issues.

Namespaces

- [ers](#)

Functions

- `std::ostream & ers::operator<< (std::ostream &, const IssueFactory &factory)`
streaming operator

8.23.1 Detailed Description

ers header and documentation file This file defines the IssueFactory class, which is responsible for registration and creation of user defined issues.

Author

Serguei Kolos

8.24 ers/IssueReceiver.h File Reference

ers header and documentation file

```
#include <ers/Issue.h>
```

Classes

- class [ers::IssueReceiver](#)
ERS Issue receiver interface.

Namespaces

- [ers](#)

8.24.1 Detailed Description

ers header and documentation file Defines abstract interface for ERS input streams.

Author

Serguei Kolos

8.25 ers/LocalContext.h File Reference

```
#include <sys/types.h>  
#include <unistd.h>  
#include <ers/Context.h>
```

Classes

- struct [ers::LocalProcessContext](#)
- class [ers::LocalContext](#)

Namespaces

- [ers](#)

Macros

- #define **ERS_PACKAGE** "unknown"
- #define **ERS_HERE_DEBUG** [ers::LocalContext](#)(ERS_PACKAGE, __FILE__, __LINE__, __PRETTY_FUNCTION__, true)
- #define **ERS_HERE** ERS_HERE_DEBUG

8.25.1 Detailed Description

This file defines the `ers::LocalContext` class, which implements the `ers::Context` interface.

8.26 `ers/LocalStream.h` File Reference

ers header and documentation file

```
#include <condition_variable>
#include <functional>
#include <iostream>
#include <queue>
#include <mutex>
#include <thread>
#include <ers/Issue.h>
#include <ers/IssueCatcherHandler.h>
```

Functions

- `ERS_DECLARE_ISSUE` (`ers`, `IssueCatcherAlreadySet`, "Local error catcher has been already set", `ERS_EMPTY`) namespace `ers`

8.26.1 Detailed Description

ers header and documentation file This file defines ERS LocalStream class.

Author

Serguei Kolos

8.26.2 Function Documentation

8.26.2.1 `ERS_DECLARE_ISSUE` (`ers` , `IssueCatcherAlreadySet` , "Local error catcher has been already set" , `ERS_EMPTY`)

The `LocalStream` class can be used for passing issues between threads of the same process.

Author

Serguei Kolos

Version

1.2

returns the singleton

sets local issue catcher

8.27 `ers/OutputStream.h` File Reference

ers header and documentation file

```
#include <string>
#include <memory>
#include <ers/Issue.h>
#include <ers/internal/macro.h>
```

Classes

- class [ers::OutputStream](#)
ERS abstract output stream interface.

Namespaces

- [ers](#)

8.27.1 Detailed Description

ers header and documentation file Defines abstract interface for ERS output streams.

Author

Serguei Kolos

8.28 ers/RemoteContext.h File Reference

```
#include <ers/Context.h>
```

Classes

- struct [ers::RemoteProcessContext](#)
- class [ers::RemoteContext](#)

Namespaces

- [ers](#)

8.28.1 Detailed Description

This file defines the [ers::RemoteContext](#) class, which implements the [ers::Context](#) interface.

8.29 ers/SampleIssues.h File Reference

Sample ERS issues.

```
#include <ers/Issue.h>
```

Variables

- **CantOpenFile**

8.29.1 Detailed Description

Sample ERS issues. Defines examples of user issues.

Author

Serguei Kolos

Version

1.0

8.30 ers/Severity.h File Reference

ers header and documentation file

```
#include <string>
#include <vector>
#include <iostream>
```

Classes

- struct [ers::Severity](#)

Namespaces

- [ers](#)

Enumerations

- enum **severity** {
Debug, Log, Information, Warning,
Error, Fatal }

Functions

- severity [ers::parse](#) (const std::string &s, severity &)
- Severity [ers::parse](#) (const std::string &s, Severity &)
- std::string [ers::to_string](#) (severity s)
- std::string [ers::to_string](#) (Severity s)
Transforms a severity type into the corresponding string.
- std::ostream & [ers::operator<<](#) (std::ostream &out, ers::severity severity)
- std::ostream & [ers::operator<<](#) (std::ostream &out, const [ers::Severity](#) &severity)
- std::istream & [ers::operator>>](#) (std::istream &in, ers::severity &severity)
- std::istream & [ers::operator>>](#) (std::istream &in, [ers::Severity](#) &severity)

8.30.1 Detailed Description

ers header and documentation file This file defines Severity type for ERS.

Author

Serguei Kolos

8.31 ers/StandardStreamOutput.h File Reference

ers header file

```
#include <iostream>
```

Classes

- struct [ers::StandardStreamOutput](#)

Namespaces

- [ers](#)

8.31.1 Detailed Description

ers header file This file defines C++ standard stream output function for the ERS streams.

Author

Serguei Kolos

8.32 ers/StreamFactory.h File Reference

ers header and documentation file

```
#include <ers/Severity.h>
#include <ers/Context.h>
#include <ers/Issue.h>
#include <map>
```

Functions

- [ERS_DECLARE_ISSUE](#) (ers, InvalidFormat, "Creator for the \"<< key<< \" stream is not found\", ((std::string) key)) namespace ers

8.32.1 Detailed Description

ers header and documentation file This file defines the StreamFactory class, which is responsible for registration and creation of ERS streams.

Author

Serguei Kolos

8.32.2 Function Documentation

8.32.2.1 ERS_DECLARE_ISSUE (ers , InvalidFormat , "Creator for the \"<< key<< \" stream is not found" , ((std::string) key))

The `StreamFactory` class is responsible for creating a new instance of a known stream implementation. This class uses singleton pattern. Users should not use this class directly but use the `ERS_REGISTER_OUTPUT_STREAM` macro instead.

Author

Serguei Kolos Factory for ERS stream implementations.

See Also

[ers::StreamManager](#)

<return the singleton

<register a stream creator

<register a stream creator

<create new stream

<create new stream

<create new stream

<collection of factories to build input streams

<collection of factories to build output streams

8.33 ers/StreamManager.h File Reference

ers header and documentation file

```
#include <initializer_list>
#include <memory>
#include <mutex>
#include <ers/Severity.h>
#include <ers/Context.h>
#include <ers/IssueReceiver.h>
#include <ers/StreamFactory.h>
#include <ers/internal/PluginManager.h>
#include <list>
```

Classes

- class [ers::SingletonCreator](#)< class >
- class [ers::StreamManager](#)

This class manages and provides access to ERS streams.

Namespaces

- [ers](#)

Functions

- `std::ostream & ers::operator<< (std::ostream &, const ers::StreamManager &)`

8.33.1 Detailed Description

ers header and documentation file This file defines the StreamManager class, which is responsible for manipulation of ERS streams.

Author

Serguei Kolos