



Eiffel Logging Cluster

Architecture, Design and Implementation

INTRODUCTION	3
ARCHITECTURE	4
ARCHITECTURE GOALS	4
CLUSTER GOALS	4
DESIGN	5
CLUSTER STRUCTURE	5
CLUSTER OVERVIEW	5
MAIN CLUSTER CLASSES	5
LOG_LOGGING_FACILITY	5
LOG_WRITER	6
WRITER DIRECTORY CLASSES	7
LOG_WRITER_STDERR	7
LOG_WRITER_FILE	7
LOG_WRITER_SYSTEM	7
Roll your own	8
IMPLEMENTATION	9
LOG_LOGGING_FACILITY	9
Initialization	9
Output	9
Access	11
LOG_WRITER	16
Initialization	16
Output	16
LOG_WRITER_FILE	17
Initialization	17
Output	17
Access	17
LOG_WRITER_STDERR	18
Initialization	18
Output	18
LOG_WRITER_SYSTEM	19
Inheritance	19
Initialization	19
Disposal	20
Output	20
Access	20

Introduction

For certain projects, it would be nice to be able to write messages to a log file. The problem with abstraction of such functionality is that various platforms have their own method of gathering these messages. Some platforms use log files, some platforms use a so-called syslog daemon, some platforms use a so-called Event Log.

The Eiffel Logging Cluster aims to resolve these differences by providing one interface to access them, while making sure the right platform methods are used under the hood.

One of the main goals of the Eiffel Logging Cluster is to enable users to utilize the same code on multiple platforms, while using various logging mechanisms.

Another main goal is to provide an extendible logging interface that can be extended to log messages to a database, or whatever means seems fit to the developer of the application universe.

Architecture

Architecture goals

The target language for the Cluster implementation is Eiffel, with the platform dependent functions written in ANSI C.

The Cluster must be usable on all platforms on which Eiffel is supported as a programming language.

Cluster goals

- 1) The same Eiffel code should compile and run on many platforms;
- 2) The Logging Cluster must offer a wide range of output forms such as syslog, Event Log, Database, File, stderr. This needs to be extendible by other developers;
- 3) All classes in the system must be capable of utilizing the log functionalities by inheriting from or using a single class that provides the required functionality. Also, within the system, the log functionality should only require configuring once;
- 4) Writing a message to the log must be as simple as the following code snippet, independent from the actual type of log file used.

```
log.write_fatal_error ("A fatal error occurred. Bailing out%N")
```

The logging library has been designed with multi threading in mind. As much concurrency as possible has been allowed by the design, however, in the LOG_WRITERS there might be some cases where two threads actually write to the logs at the same time. To make sure that the two threads are not writing through each other's messages, the actually putstring call is atomic, in that there is only one putstring per write operation. Normally operating systems should be capable of making sure that the file is not being written two at the same time by two threads. The same holds for the system logging log writer that uses syslog (on Unix) and Event Log (on Windows). These system services should be capable of guarding the actual log files from concurrent writes.

When developing a self-made LOG_WRITER it has to be kept in mind that concurrency can occur, and so, if necessary, the developer will have to make sure that this is allowed by the underlying mechanism. If it is not allowed, one can only use MUTEX objects to avoid deadlocks.

The reason why MUTEX objects are preferably not used in the current, default, implementation is that the logging library should also be capable of working in a process that doesn't use multiple threads. Requiring MUTEX objects in such a case, would automatically transform any project into a multi-threaded project.

Design

Cluster Structure

The cluster is provided with one top-level directory in which `LOG_LOGGING_FACILITY` and `LOG_WRITER` are placed. Further to this there is a subdirectory “writers” in which all specific log writers are kept.

Cluster Overview

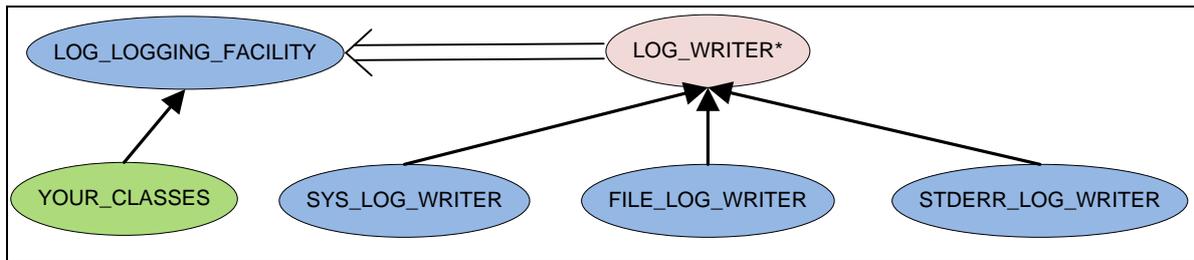


Figure 1: Classes in the Logging Cluster

Main Cluster Classes

LOG_LOGGING_FACILITY

The `LOG_LOGGING_FACILITY` class provides access to the log functionality by exposing a number of `write_*` features. Besides these features, a number of configuration features are exposed.

Writing messages

The following features for writing messages are provided:

```

write_emergency (msg: STRING)
write_alert (msg: STRING)
write_critical (msg: STRING)
write_error (msg: STRING)
write_warning (msg: STRING)
write_notice (msg: STRING)
write_information (msg: STRING)
write_debug (msg: STRING)
  
```

Configuration of the logging functionality

At least one log writer must be registered with the `LOG_LOGGING_FACILITY`. This can be done by using one of several `enable_*` features, or by calling `register_log (a_log: LOG_WRITER)`.

The following functions are provided:

```

disable_all_logs
enable_default_file_log
enable_default_stderr_log
enable_default_system_log
register_log_writer (a_log: LOG_WRITER)
resume_all_logs
resume_default_file_log
resume_default_stderr_log
resume_default_system_log
resume_i_th_log (an_index: INTEGER)
  
```

```
resume_log_writer (a_log_writer: LOG_WRITER)
suspend_all_logs
suspend_default_file_log
suspend_default_stderr_log
suspend_default_system_log
suspend_i_th_log (an_index: INTEGER)
suspend_log_writer (a_log_writer: LOG_WRITER)
```

By default, no log writers are registered.

The `disable_all_logs` feature can be used to completely unregister all logs after a certain event occurred. This can be useful in a situation where a child process is spawned that subsequently should lose connection to the console. It can `disable_all_logs`, and then `enable_default_system_log` in order to close the standard file descriptors 0, 1 and 2.

The `suspend_*` and `resume_*` features can be used to selectively suspend and resume certain or all log writers that are registered in `LOG_LOGGING_FACILITY`.

All `LOG_WRITERS` are kept in a list, and when that list is not empty, the `write_*` features call the `write` feature on each of the `LOG_WRITERS` in the list. If the list is empty, the `write_*` features will not call anything.

Design by Contract

All the `write_*` features require the following:

- There is at least one log writer registered **and**
- The `msg` attribute is not `Void` and not empty

The `write_*` features ensure that the number of log writers has not changed.

The features `enable_system_log`, `enable_file_log`, and `enable_stderr_log` ensure that one more log writer is registered. The `disable_all_logs` feature ensures that no log writer is registered. Lastly, the `register_log` feature requires that the provided log is not `Void`, and ensures that one more log writer is registered with the system.

LOG_WRITER

The `LOG_WRITER` class is an abstract class, which provides the interface used by the `LOG_LOGGING_FACILITY`. All the features are exported to `{LOG_LOGGING_FACILITY}` so that none of the features can be called accidentally by a (user) class in the universe.

The following interface is exposed:

```
initialize
has_errors: BOOLEAN
is_intialized: BOOLEAN
write (priority: INTEGER; msg: STRING)
suspend
resume
suspended: BOOLEAN
```

The `initialize` feature initializes the minimum for the log writer to work. For example, the `LOG_WRITER_FILE` will simply open a text file for appending text.

Design by Contract

The `write` feature requires that the log writer is initialized, and that the message is not `Void` and not empty. The `initialize` feature requires that the log writer is not yet initialized and ensures that it is initialized. The other two features do not have a contract in the abstract `LOG_WRITER` class.

Writer directory classes

The following classes are implemented for the user's usage:

- `LOG_WRITER_FILE`
- `LOG_WRITER_STDERR`
- `LOG_WRITER_SYSTEM`

`LOG_WRITER_STDERR`

This class simply writes everything onto `io.error` by calling `io.error.putstring (msg)`.

`LOG_WRITER_FILE`

This class creates a default log file called `Execution_environment.current_working_directory + Operating_environment.directory_separator + "system.log"`. If this name is not suitable for an application one can create a new object of type `LOG_WRITER_FILE` and call `set_file_name` to provide a more suitable file name.

`LOG_WRITER_SYSTEM`

This class uses the system default event or message logging subsystem to provide integrated message gathering from an application point of view.

On the Windows platform, it uses the Event Log, and on Unix platforms, it uses the `syslogd` or `rsyslogd`.

There are obvious differences between Unix and Windows. These differences are obscured from the class' user in a C library. Nevertheless, one can, by inheriting from the `LOG_WRITER_SYSTEM` set certain default aspects, like the application name and the facility to use, e.g. `LOG_LOCAL3` or System Event Log, instead of the default facilities.

The default facilities are `LOG_LOCAL6` on Unix platforms and the Application Event Log on Windows.

The default syslog options on Unix are simply `LOG_NDELAY`, and `LOG_PID`. Again, by inheritance, these defaults can be changed to suit the needs of the application under development.

Windows Specifics

eif_eventlog_messages.h and eif_eventlog_messages.dll

These two files are generated when `finish_freezing -library` is executed in the `$ISE_EIFFEL\library\logging\Clib`. You need to ship these files with the installation software so that these files can be copied into a known directory, and so that one can already setup appropriate Registry entries.

eif_eventlog_messages.reg

This file contains an example of what to include in the Registry for your own application. The hexadecimal stuff actually reads the full absolute file name of the `eif_message.dll` file.

When an application is using the Event Log for system logging, the Event Viewer expects to find the message definitions in certain places in the Registry. The keywords are `Application`, `Security`, and `System`. So when an application uses the Application Event Log, a key must be generated under `HKLM\System\CurrentControlSet\Services\eventlog\Application`.

The name of that key is the Event Source that the application will use. This is the same value as `{LOG_WRITER_SYSTEM}.application_name`, and is set in `{LOG_WRITER_SYSTEM}.set_application_name`. You can override the default (“EiffelSysLog”) by creating a new object of type `LOG_WRITER_SYSTEM` and calling `set_application_name`. See the multiplatform example, for an example of such usage.

Failure to update the Windows Registry, basically generates a post condition violation of `is_initialized` in the `LOG_WRITER_SYSTEM`’s `initialize` feature, so it is critically important to the well-functioning of an application to adhere by the above guidelines.

Roll your own

It’s fairly to create specific `LOG_WRITERS`. One has to define the `initialize` feature, create a sensible default and put it in `default_create`. The `dispose` feature could be used to close all external resources, e.g. file handles. The only feature left to implement is the `write` feature that actually writes messages to output channel that is being defined in the new `LOG_WRITER`.

Implementation

LOG_LOGGING_FACILITY

Initialization

LOG_LOGGING_FACILITY_make

Requirements

Not applicable

Body

Create the list of log writers.

Guarantees

The list of log writers is created.

Output

write_emergency (msg: STRING)

Requirements

The list of log writers must have at least 1 log writer

Body

For each log writer from the list, call the feature write with the following prefix to the `msg` string: "EMERG - ".

Guarantees

The list of log writers still has the same number of log writers as when execution began

write_alert (msg: STRING)

Requirements

The list of log writers must have at least 1 log writer

Body

For each log writer from the list, call the feature write with the following prefix to the `msg` string: "ALERT - ".

Guarantees

The list of log writers still has the same number of log writers as when execution began

write_critical (msg: STRING)

Requirements

The list of log writers must have at least 1 log writer

Body

For each log writer from the list, call the feature write with the following prefix to the `msg` string: "CRIT - ".

Guarantees

The list of log writers still has the same number of log writers as when execution began

write_error (msg: STRING)

Requirements

The list of log writers must have at least 1 log writer

Body

For each log writer from the list, call the feature write with the following prefix to the `msg` string: "ERROR - ".

Guarantees

The list of log writers still has the same number of log writers as when execution began

write_notice (msg: STRING)

Requirements

The list of log writers must have at least 1 log writer

Body

For each log writer from the list, call the feature write with the following prefix to the `msg` string: "NOTIC - ".

Guarantees

The list of log writers still has the same number of log writers as when execution began

write_warning (msg: STRING)

Requirements

The list of log writers must have at least 1 log writer

Body

For each log writer from the list, call the feature write with the following prefix to the `msg` string: "WARN - ".

Guarantees

The list of log writers still has the same number of log writers as when execution began

write_information (msg: STRING)

Requirements

The list of log writers must have at least 1 log writer

Body

For each log writer from the list, call the feature write with the following prefix to the `msg` string:
"INFO - ".

Guarantees

The list of log writers still has the same number of log writers as when execution began

write_debug (msg: STRING)

Requirements

The list of log writers must have at least 1 log writer

Body

For each log writer from the list, call the feature write with the following prefix to the `msg` string:
"DEBUG - ".

Guarantees

The list of log writers still has the same number of log writers as when execution began

Access

disable_all_logs

Requirements

At least one log writer must have been registered

Body

Empty the log writers list.

Guarantees

No log writer is registered

enable_default_file_log

Requirements

Not applicable

Body

Create a default, but initialized, file log writer and add it to the list of log writers

Guarantees

There is one more log writer registered compared to when execution began

[*enable_default_stderr_log*](#)

Requirements

Not applicable

Body

Create a default, but initialized, stderr log writer and add it to the list of log writers

Guarantees

There is one more log writer registered compared to when execution began

[*enable_default_system_log*](#)

Requirements

Not applicable

Body

Create a default, but initialized, system log writer and add it to the list of log writers

Guarantees

There is one more log writer registered compared to when execution began

[*register_log \(a_log: LOG_WRITER\)*](#)

Requirements

The argument `a_log` may not be Void and `a_log` may not be initialized

Body

Add the the argument `a_log` to the list of log writers and initialize it.

Guarantees

There is one more log writer registered compared to when execution began and `a_log` is now initialized.

[*resume_all_logs*](#)

Requirements

Some log writers must have been registered.

Body

Call resume on all items in the log_writers_list.

Guarantees

Some log writers are still registered.

resume_default_file_log

Requirements

Default file logging must be enabled and the default_log_writer_file object must exist and be initialized.

Body

Call resume on the default_log_writer_file object.

Guarantees

Not applicable.

resume_default_stderr_log

Requirements

Default stderr logging must be enabled and the default_log_writer_stderr object must exist and be initialized.

Body

Call resume on the default_log_writer_stderr object.

Guarantees

Not applicable.

resume_default_system_log

Requirements

Default system logging must be enabled and the default_log_writer_system object must exist and be initialized.

Body

Call resume on the default_log_writer_system object.

Guarantees

Not applicable.

resume_i_th_log_writer (an_index: INTEGER)

Requirements

The argument `an_index` must be greather than zero are less than or equal to the number of registered log writers.

Body

Move the cursor of log_writers_list to the position an_index, and call resume on that item.

Guarantees

Not applicable.

resume_log_writer (a_log: LOG_WRITER)

Requirements

The argument `a_log` must not be Void and it must initialized.

Body

Call resume on `a_log`.

Guarantees

Not applicable.

suspend_all_logs

Requirements

Some log writers must have been registered.

Body

Call suspend on all items in the log_writers_list.

Guarantees

Some log writers are still registered.

suspend_default_file_log

Requirements

Default file logging must be enabled and the default_log_writer_file object must exist and be initialized.

Body

Call suspend on the default_log_writer_file object.

Guarantees

Not applicable.

suspend_default_stderr_log

Requirements

Default stderr logging must be enabled and the default_log_writer_stderr object must exist and be initialized.

Body

Call suspend on the default_log_writer_stderr object.

Guarantees

Not applicable

suspend_default_system_log

Requirements

Default system logging must be enabled and the default_log_writer_system object must exist and be initialized.

Body

Call suspend on the default_log_writer_system object.

Guarantees

Not applicable.

suspend_i_th_log_writer (an_index: INTEGER)

Requirements

The given index `an_index` must be greater than zero and less than or equal to the number of registered log writers.

Body

Move the cursor of log_writers_list to the position an_index, and call suspend on that item.

Guarantees

Not applicable.

suspend_log_writer (a_log: LOG_WRITER)

Requirements

The given log writer `a_log` must exist and it must be initialized

Body

Call suspend on the given log writer.

Guarantees

Not applicable.

LOG_WRITER

Initialization

default_create

Requirements

Not applicable

Body

This feature does not do anything

Guarantees

Not applicable

initialize

Requirements

Not applicable

Body

The body of this feature is deferred

Guarantees

That the log writer is initialized

Output

write (msg: STRING)

Requirements

The log writer must be initialized, it must not be suspended, and `msg` may not be Void or empty

Body

The body of this feature is deferred.

Guarantees

The log writer is still initialized.

LOG_WRITER_FILE

Initialization

default_create

Requirements

Not applicable

Body

This feature sets the appropriate file name to use by default.

Guarantees

Not applicable

initialize

Requirements

Not applicable

Body

Create the `log_file` for appending to that `file_name`. Rescue any raised exceptions, retry the body after setting a local flag, and indicate that an error occurred through `has_errors`.

Guarantees

The log writer is initialized. It also guarantees that the `log_file` can be written to.

Output

write (msg: STRING)

Requirements

The log writer must be initialized, it must not be suspended, and ``msg`` may not be Void or empty

Body

The ``msg`` is written to the `log_file`, with a prefix of the current date and time and a hyphen character.

Guarantees

The log writer is still initialized.

Access

set_file_name (a_file_name: FILE_NAME)

Requirements

The given `a_file_name` must not be Void and the object must not be empty. Also, the log writer may not be initialized yet.

Body

The used file name is set to a twin of the given ``a_file_name'`.

Guarantees

Not applicable.

LOG_WRITER_STDERR

Initialization

default_create

Requirements

Not applicable

Body

This feature does not do anything

Guarantees

Not applicable

initialize

Requirements

Not applicable

Body

This feature does not do anything, except to set the flag indicating that this log writer is initialized.

Guarantees

The log writer is initialized.

Output

write (msg: STRING)

Requirements

The log writer must be initialized, it must not be suspended, and ``msg'` may not be Void or empty

Body

The ``msg'` is written to the `io.error`, with a prefix of the current date and time and a hyphen character.

Guarantees

The log writer is still initialized.

LOG_WRITER_SYSTEM**Inheritance**

In order to provide for some fundamental constant values, the following classes will be introduced as parents of LOG_WRITER_SYSTEM:

- **LOG_FACILITY_CONST**
This class implements the constants for all Unix syslog facilities and the various Windows Event Log constants that let a developer choose which Event Log the message should go into.
- **LOG_OPTIONS_CONST**
This class implements the Unix syslog options, and basically returns a non-descript value on other platforms.
- **LOG_PRIORITY_CONST**
This class implements the Unix syslog options that are mapped to Windows Event Log Categories in the underlying C implementation of the macro's.

Initialization*default_create***Requirements**

Not applicable

Body

This feature sets appropriate options, facility and application name to use by default.

Guarantees

Not applicable

*initialize***Requirements**

Not applicable

Body

Call `eif_logging_open_log` with the default or set information to open the system log.

Indicate through `is_initialized` that this LOG_WRITER_SYSTEM object is ready to write to the system log, and indicate through `has_errors` that an error occurred during initialization.

Guarantees

The log writer is initialized or there were errors.

Disposal

dispose

Requirements

Not applicable

Body

Close the system logger.

Guarantees

Not applicable

Output

write (msg: STRING)

Requirements

The log writer must be initialized, and `msg` may not be Void or empty

Body

Remove the prefix of `msg`, convert that to the appropriate logging priority or category, and write the remainder `msg` is written to the log.

Guarantees

The log writer is still initialized.

Access

set_application_name (an_application_name: STRING)

Requirements

The given `an_application_name` may not be Void or empty, and the log writer may not be initialized.

Body

Set the used application name to a twin of the given `an_application_name`.

Guarantees

Not applicable.

set_facility (a_facility: INTEGER)

Requirements

The log writer may not be initialized.

Body

Set the used facility to the given `a_facility`.

Guarantees

Not applicable.

set_options (some_options: INTEGER)

Requirements

The log writer may not be initialized.

Body

Set the used options to the given `some_options`.

Guarantees

Not applicable.