RFC 234 - Framework Logging

Final

12 Pages

Abstract

Currently there is no API to define how the framework should do logging; code launching a framework needs to know implementation details when it wants to configurate that launching. This RFC proposes a framework implementation independent way to be used by the framework for logging.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

0.2 Trademarks

0.3 Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4 Table of Contents

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 10.1.

```
Source code is shown in this typeface.
```

0.6 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	March 15 2017	Initial version based on #2974 and F2F discussions in Montpellier Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com

1 Introduction

This RFC defines a way to configure the logging of the framework itself independent from the framework implementation.

2 Application Domain

The launch API allows to start a framework in an implementation independent way. While this works, there is at least one thing which needs to be setup depending on the implementation: logging of the framework. As framework logging is an essential part of an application, the current launch API does not provide a full abstraction.

3 Problem Description

The framework launch API defines an implementation independent way to launch any framework implementation. This API covers creating a framework and configuring it. However, one aspect is not covered by this AP: Logging of the framework. This requires launch code like Apache Karaf to use implementation dependent solutions.

For example, by default the Apache Felix framework is logging to system out. But an implementation can provide a org.apache.felix.framework.Logger instance either through the framework properties (tricking the compiler) or by using reflection and checking for a setLogger method on the framework object. Eclipse Equinox is using the embedded LogService to log errors and has a separate tracing mechanism which eventually logs to system out as well.

Usually code that launches a framework also wants to tell the framework in what detail it should log and how to log.

4 Requirements

L0010 - The solution MUST define a way to set the log detail (log level) used by the framework implementation when emitting log messages.

L0020 – The solution MUST provide a way for the framework to detect the current log level to avoid unnecessary calls into the logging mechanism.

L0030 – The solution MUST provide a way to capture the log messages emitted by the framework.

5 Technical Solution

The technical solution describes two parts, a logger instance created by the code launching the framework and how to pass that instance to the framework.

5.1 The Logger

An instance of the Logger is created by the launching code. Initially a logger is created with a log level. The log level can be changed by calling the `setLogLevel` method. Before the framework logs, it can use the

`getLogLevel` method or one of the `isXXXEnabled` methods to detect whether a log messages should be emitted. The framework can use one of the two provided `log` methods to emit a log message:

```
public abstract class Logger {

    public enum LogLevel {

        ERROR,

        WARN,

        INFO,

        DEBUG

    }

    public Logger(LogLevel l) {...}

    public void setLogLevel(LogLevel l) { … }

    public LogLevel getLogLevel() { … }

    public boolean isDebugEnabled() { … }

    public boolean isInfoEnabled() { … }

    public boolean isWarnEnabled() { … }

    public void log(LogLevel level, String msg) { … }

    public final void log(LogLevel level, String msg, Throwable throwable) {...}

    protected abstract void doLog(LogLevel level, String msg, Throwable throwable);

}
```

The Logger class is an abstract class and already implements all the functionality except for processing the log message from the framework. The `log` methods check the log level and call the abstract method `doLog` if the level is enabled. Code launching a framework just needs to create a subclass of `Logger` and implement this single method.

5.2 Providing the Logger to the Framework

The `FrameworkFactory` interface gets a new method to construct a framework which in addition to the configuration map gets a logger instance. The logger instance is optional. Calling the existing method `newFramework(Map)` is equivalent with calling this new method passing in `null` as the second argument:

```
public interface FrameworkFactory {

    Framework newFramework(Map<String, String> configuration, Logger logger);

}
```

Whenever the framework wants to emit a log message it should check the current log level of the logger instance and only emit a message if the level is enabled.

If no logger instance is provided to the framework it is up to the framework implementation how to deal with log messages from the framework implementation.

5.3 Required Framework Logging

If a logger instance is provided to the framework factory, the framework implementation must use this instance to log the following events:

- Whenever an ERROR framework event occurs, this must be logged with log level ERROR.

6 Data Transfer Objects

This RFC does not define any DTOs

7 Java API

8 Considered Alternatives

9 Security Considerations

This RFC does not need any new security considerations.

10 Document Support

10.1 References

[1] Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2] Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3] Data Transfer Objects, Core Release 6

10.2 Author's Address

Name	Carsten Ziegeler
Company	Adobe Systems Incorporated
Address	Barfüsserplatz 6, 4055 Basel, Switzerland
Voice	+41 61 226 55 0
e-mail	cziegele@adobe.com

10.3 Acronyms and Abbreviations

10.4 End of Document