RFP-163 Log Service Update

Draft

15 Pages

Abstract

~~Log Service~~Logging ~~has an important role in the software deployment. It~~is a crucial component ~~can~~to discover ~~the reason for a blocker issues~~software bugs in a software system. The OSGi Log Service was the first compendium service and the Java eco-system gained over time many different log solutions:~~The current deployments are using various log API like:~~ Log4j 2, ~~SLF4J/~~Logback, Java Util Logging, etc. ~~Equinox Log etc.~~ Since the OSGi Log Service was not further developed the API does not take advantages of any of the new features in Java and looks very simplistic in comparison to mainstream Java. ~~The evolution of the log API calls for new Log Service requirements.~~ This document ~~contains proposal for updates to~~seeks proposals to improve the Log Service ~~usage~~ API and add additional roles to upgrade it to Java 8.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is

not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback ("Feedback") on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable, worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future ("Future Specification"), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

OSGi™ is a trademark, registered trademark, or service mark of the OSGi Alliance in the US and other countries. Java is a trademark, registered trademark, or service mark of Oracle Corporation in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

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 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	Sep 17 2014	Initial version introduced on F2F Meeting in Madrid, Spain hosted by Liferay, September 9-11, 2014. Evgeni Grigorov, ProSyst Software, e.grigorov@prosyst.com

Revision	Date	Comments
EnRoute Add.	05.12.14	Moved paragraphs to their correct place and added enRoute experiences and the learned requirements Peter Kriens

1 Introduction

This RFP originates from a general desire in the OSGi community to upgrade the OSGi Log service and provide a more mainstream solution to make OSGi logging look more modern. The RFP also includes the experiences gained in the OSGi enRoute project.

~~It's a great feature that OSGi Log Service is available in the OSGi service registry. Many providers can register their implementations at the same time. The services can be unregistered and registered to provides some updates. That dynamism is good for the providers, but complicates the applications logging.~~

~~When the logging is the next application step, the hard work begins. The application should track the Log Service from the OSGi service registry. If there are no Log Services, the application should buffer or print the log message. If there are more than one Log Services, the application should log the message through all of them.~~

~~Those complications can be cleared and single API can be introduced. The new interface can be used by the framework and all active bundles. There will be single log entry point.~~This RFP was initiated by Prosyst and the augmented by the results of the OSGi enRoute project.

2 Application Domain

2.1 OSGi Log Service

The Log Service API has 4 methods. Each method takes a log level and a *message* string. In the OSGi Log Service this is an int. The ERROR level is 1, the TRACE level is 4, additional levels are accepted and stored. The 4 variations are used to pass a Service Reference and a Throwable.

Since the Log Service is aware of the bundle logging it can automatically provide this bundle in the entries. The OSGi Log Service is a dispatcher, it brokers between a *log client* and zero or more *log appenders*. The clients get the OSGi Log Service and the appenders get the OSGi Log Reader Service and register a listener with the Log Reader Service. The listener is then updated of any logging entries submitted by any client. The OSGi Log Reader has an optional history of recent events. The purpose of the history is to capture the log *entries* before the appender had been able to register itself.

A log entry consists of the bundle, a message, and an optional Service Reference and/or Throwable.

Since the OSGi Log Service uses services there can be multiple implementations and there is no guarantee that there is a log service is present. However, in general, there is only one Log Service and Log Reader service registered. In Declarative Services (DS) terms, the Log service should in general be a static dependency of a log client, which implies that the highest ranking log service is used. Though theoretically possible, few clients log to all registered service. Multiple Log Services is deemed an anomaly because it is a broker model and multiple brokers forfeit the purpose a bit.

Since the Log Service is a service it is possible that there is a need to log before the Log Service is available. Best practices in this case is to record the events until the Log Service becomes available, print to standard out, or ignore events. In Declarative Services, the bind methods that are called before the component is activated (and thus can be called before the Log Service is bound) can throw exceptions that are then logged by the Service Component Runtime.

2.2 Open Source

In the Open Source world a frenzy took place in developing log APIs. The current situation is quite complex because there are so many choices which created their own problems requiring facades that could log to many different logging subsystems. About ten years ago Java introduced java.util.logging but received a lot of flack from the industry because they had not followed best practices. Logging seems to be a quite sensitive product in our industry.

Today it seems that the Simple Logging Facade for Java (SLF4J) is the most mainstream API for clients. The reason of its success is partly its design. It provides an API that is identical to some of the other APIs and it can easily forward the logged entries to other log subsystems. This makes SLF4J attractive from the point of view of the log client.

SLF4J has the concept of a named *logger*. A logger is generally created in a static variable and is obtained from the `LoggerFactory` class. The name is generally the class name (there is an overloaded method on the Logger Factory to give a class object). When the first logger is created, the SLF4J code does some very heavy handed dynamic class loading magic to find a *provider*. The factory classes of the provider are generally implemented in a standard package in the SLF4J namespace. The provider then creates an implementation of the Logger class that is returned to the client. Since this is all static, it happens lazily on the first creation. However, it does require all classes to be visible from the API classes. In OSGi it is therefore necessary to provide the implementation in the same bundle as the API bundle, or use a fragment on the API bundle.

The name of the logger is then used to establish on what level should actually be logged. Since Java class names are hierarchical, wildcards can be used to set the levels for related loggers. In SLF4J, the configuration is set with a properties file/resource that is searched for on well known places. In OSGi, fragments on the API bundle are often used to provide these properties. If a different configuration is needed then the application must be restarted. A logger is set to be *active* for a given level when log messages are passed to the appender.

The SLF4J API is a hodgepodge of log methods that come from different other log APIs and improvements over time. In general, the level is encoded in the method name. i.e. there are error, warn, trace, debug and info.

An important aspect of logging is the performance. Enterprise code is heavily instrumented and logging can take a significant portion of the code and CPU time. It is crucial to minimize the overhead of logging. This is the reason why often the actual log method is not called when the level is not active:

```
if (logger.isDebugEnabled())
  logger.debug("Hello " + name );
```

The reason of this pattern that this way the concatenation of the strings only takes place when the level is active. With the advent of Java 5 we got varargs. Varargs made it easy to defer the cost of computations of the parameters to when it is actually necessary. This made printf like loggers popular:

```
logger.debug("Hello {}", name );
```

This reduced the clutter of log messages significantly. The SLF4J Logger provides printf like methods for all supported levels but does not use the familiar % syntax of the Java String Formatters. It uses the message based format with curly braces.

SLF4J also provides capture of the current threads and *markers*. Markers allow the introduction of variables in the log that can bind different parts in an execution.

~~Most of the applications are using the well-known logging API like Log4j 2 and SLF4J/Logback. They expect to have simple and static way to receive a logger. The applications request different loggers for their internal modules. The log messages are going to be more and more flexible with support of:~~

- ~~Parametrization – the message can be filled with different parameters at runtime.~~
- ~~Context – the message is automatically assigned to a context with additional details like thread info.~~

List of currently used logging frameworks:

- Log4j and the next version Log4j 2
- SLF4J and the update Logback
- Java 2 Logging API

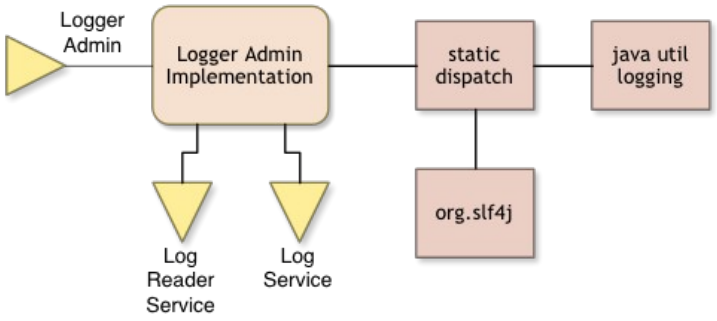
2.3 ~~Equinox Log Service (org.eclipse.equinox.log)~~OSGi enRoute

In the OSGi enRoute [8]. project it was clear from the beginning that SLF4J was so popular in open source that it needed to be supported. However, though SLF4J and its appenders are delivered as bundles, it was not seen as a good idea to bypass the OSGi service model since the factory model is heavily based on dynamic class loading, causing all kinds of visibility problems. Therefore the approach was taken to provide a special enRoute appender that captured the log entries and forwarded them to the Log Service. Since this appender can be created before any OSGi framework is available it cannot rely on any of the OSGi mechanisms. It therefore buffers the log entry in memory as well as any loggers that access it.

Once the enRoute logger bundle becomes available it then accesses the static history and registers itself as the master. This logger bundle is configured with rules for the active levels for each logger. Based on these rules it forwards the entries to the OSGi Log Service. It is assumed to any appenders to other log systems would then use a Log Reader service. Mapping SLF4J entries to OSGi entries was a tad painful since the OSGi entries missed concepts like threads, sequence numbers, and markers.

Since dynamics are so important in OSGi, it also defined a Log Service Admin that could be used to manage the active levels of the loggers as well as provide the names of loggers that have logged. With this service (and corresponding Gogo commands) it is possible to dynamically change the active level of loggers.

The service diagram of the enRoute solution looked as depicted here:



OSGi enRoute registers an SLF4J Logger service and uses a Service Factory to capture the bundle. The name of this logger is then the symbolic name and version of the captured bundle. Though a logging service has the slight disadvantage that it is not available during initialization, it has the huge benefit of the service model. Since Declarative Services takes care of logging for errors in binding methods, the practical disadvantage is actually quite small.

Additionally, the OSGi enRoute also provided a utility to log based on a mechanism pioneered in bnd. The utility took an interface and a Logger and returned a proxy. Each method on the interface was a log message. The proxy handler would take the method name and turn it into a message, interleaving it with the arguments of the method. The level was defined by the return type.

Annotations were added to override the automatic message generation and to allow reorder and format arguments with the Java String Formatter API. By using Java types, the IDE helps finding log messages and refactoring log messages. The overhead is quite minimal since dynamic proxies have become quite fast today.

Last, and maybe least, OSGi enRoute added an additional level: AUDIT. This is a non-maskable level. Many financial institutions use log messages to audit, using a special level can provide more guarantees.

2.4 Apache Sling Log Service

The Apache Sling Log Service follows similar patterns but uses a slightly different route. In addition to consider SLF4J the most used log library, it also acknowledges that logback (http://logback.qos.ch/) is the most common backend for processing log entries.

Instead of funneling all log entries through the OSGi LogService, everything is passed on to SLF4J and from there to logback. Therefore the OSGi LogService logs to SLF4J and all the bridges are in place to pass log entries done through JUL or LOG4J to SLF4J.

Loggers and log levels can be configured through OSGi configurations. Config changes are of course processed dynamically and directly applied. On the other hand instead of using the OSGi LogService as an extension model, logback concepts are used through the whiteboard pattern. Logback TurboFilters, Filters, Appenders etc. can be registered dynamically as well as some other extensions.

This approach uses the most common parts today used for logging: SLF4J for clients and logback for processing.

[1] http://sling.apache.org/documentation/development/logging.html

2.5 Terminology + Abbreviations

- SLF4J - Simple Logging Facade for Java

3 Problem Description

The current OSGi Log Service suffers from the following problems:

- There is no way to set log levels so that not all messages are logged or a certain client.
- The OSGi Log Service API with its separate level is awkward to use in the code, the current practice is to use methods with the level name.
- The log client must construct the full message before the call is executed. This costs time and screen space.
- The current API does not capture threads, nor provides markers, or maintains sequence numbers
- The Log Service is not always available when a entry must be logged, especially during initialization. Then bundles need to buffer the log entries or print them on the console.
- The Log Service specification was developed before the whiteboard model was popular. This means that Log Listeners must first get the Log Reader Service and then register themselves. This is awkward, the whiteboard is much more convenient.
- If more than one Log Service is available, all of them have to be notified in case of new log entry. ## pkr: I disagree, there should be one log service registered.
- ~~The current OSGi approach about the logging facilities is convenient for the Log Service providers, but complicates the consumers. It can be improved to minimize the logging source code, to simplify the usage and to keep the application source code clean and readable.~~
- The applications have an issue if the Log Service is missing. ~~They need to buffer the log entries or print them on the console. If more than one Log Service is available, all of them have to be notified in case of new log entry. Debugging is all about the process to discover new defects and there is a need for as much as possible information. The log messages can be colored with additional details like thread info, log name, context etc.##~~ pkr: this is an advantage since something is wrong and your app should not start if a Log Service is missing. If SLF4J is not on your class path you would get a ClassInitializationError

- Static loggers like SLF4J do not capture the bundle information
- Static loggers like SLF4J require class loading hacks to link them to an appender that are very non-OSGi like.

4 Use Cases

4.1 Static Logging

Al Bundle got fed up with the OSGi Log Service and started to use SLF4J. However, he did wanted to log per bundle so he used the service model:

```
@Component
public class Foo {
  Logger logger
  @Activate
  void activate() {
    doFooInit();
    logger.trace("Initialized");
  }
  @Reference
  void setLogger( Logger logger ) { this.logger = logger; }
}
```

When his bundle runs, Al goes to the Gogo console and checks out the logging configuration:

```
g! loggers
com.example.als.bundle-2.3.4
com.example.als.another-1.2.3
com.example.foo.SomeClass
```

He then changes the level

```
g! level warn com.example.als.bundle-2.3.4
_
```

4.2 Type Safe Logging

After wasting 30 mins in trying to find a log message's logging point Al decides to try out type safe logging. He creates the following logger interface:

```
interface FooLogger extends TypedLoggerBase {
  ERROR noSuchFile(File f);
  TRACE claimProcessed(ClaimId claimId);
}
```

Al then creates the logger:

```
static FooLogger log = TypedLogger.create(FooLogger.class);
```

His code now looks wonderfully simple and it is easy to navigate in Eclipse.

```
void foo(File f) throws Exception {
  try(InputStream in = new FileInputStream(f);) {
    ClaimId id = processClaim(in);
    claimProcessed(id);
  } catch( FileNotFoundException e) {
    log.noSuchFile(f);
  }
}
```

4.3 Single OSGi Log

The OSGi framework integrator would like to have a single Log Service. It'll simplify the log configuration and will place all log messages in a single place. Currently, the bundles are using the Log Service, but the framework (system bundle) is using another own logger.

4.4 Log Service Availability

The OSGi framework is a dynamic environment. The services can be registered and unregistered over and over again. The Log Service is not an exception. The bundles must be prepared with a backup log option in case of missing Log Service. It complicates the implementation and often there are some Log Service wrappers to handle this scenario.

pkr: I really do not buy this.

~~Log Message Parameterization~~

~~The bundle implementation has detailed log. The source code is not readable and hides the real flow because of a lot of:~~

```
if (isDebugEnabled) {
  logger.debug("File:" + fileName + " has been processed.");
}
```

~~The source code will be much clear if this debug is replaced by:~~

```
logger.debug("File:{} has been processed.", fileName);
```

4.5 ~~Named Log Service~~

~~The application bundle is internally organized in modules. There are: configuration, db and UI modules. Each module needs own logger to log the application flow. Currently, all modules are sharing the same Log Service and are using a message prefix to mark that the message is coming from the given module. An example:~~

```
logger.info("[DB] Database has been restored.")
```

~~It'll be convenient to request the logger with a name. The name can be mapped to the bundle internal module name. The prefix from the example will be redundant then.~~

4.6 ~~Log Entry Enhancement~~

~~The system doesn't have an option for remote debug and there is only one option, activated debug. The debug messages must contain as much as possible details like: thread info, logger name, sequence number etc. Currently, those details are missing in OSGi LogEntry and custom solutions are used.~~ It is good to have this dependency since this signals an error

4.7 Multiple Log Services

The application bundle has a critical log message. The OSGi framework maintains a few Log Services. The application bundle should visit all Log Services to submit the message. That scenario can be simplified and automated.

pkr: I really do not buy this. Clients should accept a single log service and generally get the highest ranking. It is imho an error of the deployer if there are 2 Log Services.

5 Requirements

5.1 SLF4J Loggers

- S0010 – It must be possible to use SLF4J API to log
- S0020 – It must be possible to create a static SLF4J logger (the normal way)
- S0030 – It must be possible to use a SFL4J Logger service that is automatically named according to the bundle's symbolic name and version.
- S0040 – Describe how static loggers are cleaned up when a bundle is uninstalled
- S0050 – Define how the the SLF4J go to the OSGi Log Service, taking into account that the bundle implementing the Log Service might be active when the SLF4J log entry is made and no entries may be lost.

5.2 Log Service

- L0010 – The Log Service must provide a new level for AUDIT that cannot be ignored
- L0020 – The Log Service must provide a new level for TRACE to match SLF4J.
- L0030 – The Log Service API should be extended with the methods from the SLF4J Logger API but then only a single variant that takes varargs.

- L0040 – The solution SHOULD provide a mechanism for a notification of multiple Log Services (##pkr disagree, this is not a relevant issue)
- L0050 – Generally ensure that all SLF4J concepts map to a Log Service concept.

5.3 Log Admin

- A0010 – The active levels of the loggers must be dynamically changeable
- A0020 – It must be possible to get a list of active loggers with their active level and last logged entry.
- A0030 – It must be possible to set the active level of a bundle for all logger from that bundle, assuming that static loggers come from the bundle they were loaded from.
- A0040 – It must be possible to get some key statistics of the Logging subsystem like log entries per second, total entries, black listed readers, etc.
- A0050 – Provide a means to throttle the logs when the load reaches a threshold.

5.4 Log Reader

- R0010 – The Log Reader Listener must become whiteboard ~~etc. method, provide location info like classMAYThe solution Application Requirements~~

1. ~~The solution SHOULD provide an option for a single OSGi Log. Log can be used by the bundles and OSGi framework itself (system bundle).~~
2. ~~The solution SHOULD guarantee the availability of the OSGi Log. In this way, the applications can have the assumption that there is always OSGi Log.~~
3. ~~The solution SHOULD provide an option for parameterized log messages.~~
4. ~~The solution SHOULD provide an option for name association with the logger i.e. named logger.~~
5. ~~The solution MAY provide new log levels like: trace.~~
6. ~~The solution MUST use OSGi whiteboard pattern for the Log Listener.~~

~~The solution SHOULD provide a mechanism for a notification of multiple Log Services.~~

5.5 ~~Log Entry Requirements~~

1. ~~The solution SHOULD provide the long entry with thread info.~~

-

5.6 ~~provide the log entry with a logger name.SHOULDprovide the log entry with a sequence number.~~

~~The solution SHOULD The solution~~Log Entry

- E0010 – It must be possible to capture thread information in a Log Entry
- E0020 – It must be possible to capture markers in a log entry (## yes?)
- E0030 – The solution must provide the log entry with a sequence number.
- E0040 – The solution must provide the log entry with a logger name.
- E0050 – The solution should provide an option to include location info like the class and the method and line number if available.

5.7 Type Safe Logging

- T0010 – Provide a utility to log via Java constructs. E.g. a message on an interface.
- T0020 – Provide a mechanism to override the default message that allows reordering of parameters
- T0030 – Provide localization support

6 Document Support

6.1 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]. https://issues.apache.org/jira/browse/FELIX-536

[4]. SLF4J, http://www.slf4j.org

[5]. Apache Log4j 2, http://logging.apache.org/log4j/2.x/

[6]. Equinox Log Service (org.eclipse.equinox.log)

[7].

[8]. ~~*Add references simply by adding new items. You can then cross-refer to them by chosing <Insert><Cross Reference><Numbered Item> and then selecting the paragraph.* ***STATIC REFERENCES (I.E. BODGED) ARE NOT ACCEPTABLE, SOMEONE WILL HAVE TO UPDATE THEM LATER, SO DO IT PROPERLY NOW.***~~http://enroute.osgi.org/services/osgi.enroute.logger.api.html

6.2 Author's Address

Name	Evgeni Grigorov
Company	ProSyst Software
Address	Aachenerstr. 222, 50935 Cologne, Germany
Voice	+49 221 6604 501
e-mail	e.grigorov@prosyst.com

Name	Peter Kriens
Company	aQute
Address	9c, Avenue St. Drézery
Voice	+33 6 33 98 22 60
e-mail	peter.kriens@aqute.biz

6.3 End of Document