# RFP 143 OSGi Connect

Final

7 Pages

## Abstract

Discusses the needs for a specification that would make the service and parts of the life cycle layer available in environments where the expense of modularity encapsulation prevents developers from using OSGi.

# 0 Document Information

## 0.1 Table of Contents

**0 Document Information** **2**
0.1 Table of Contents 2
0.2 Terminology and Document Conventions 2
0.3 Revision History 2

**1 Introduction** **3**

**2 Application Domain** **4**
2.1 Terminology + Abbreviations 4

**3 Problem Description** **5**

**4 Use Cases** **5**
4.1 bnd in Ant and Maven 5
4.2 WARs 5
4.3 Application Frameworks 5
4.4 Non-OSGi Developers 6

**5 Requirements** **6**
5.1 Basic 6

**6 Document Support** **7**
6.1 References 7
6.2 Author's Address 7
6.3 End of Document 7

## 0.2 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 6.1.

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

## 0.3 Revision History

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

| Revision | Date | Comments |
|---|---|---|
| Initial | MARCH 30 2011 | Peter Kriens, Initial draft |
| | 04/19/11 | Peter Kriens, after Austin meeting |
| | 6/17/2011 | Renamed to OSGi Connect. |
| 4th draft | 2012-05-15 | Updated to continue discussion.<br>BJ Hargrave |
| 5th draft | 2012-05-31 | Updated based upon CPEG 24 May call.<br>BJ Hargrave |
| 6th draft | 2012-07-02 | Updated based upon feedback from Mike Keith.<br>BJ Hargrave |
| Final | 2012-10-01 | Mark Final for RFP voting<br>BJ Hargrave |

# 1 Introduction

This RFP discusses the needs for an OSGi Framework that allows people to use the selected capabilities of OSGi without being forced to address all of the modularity related class loading issues before they can enjoy the fruits of the service layer.

This work was first explored by the now defunct OSXA project[1] but was recently revived by Karl Pauls from the Apache Felix project who made a prototype based on Apache Felix called PojoSR.[2]

1 http://blog.osgi.org/2006/09/alternative-osgi-styles.html
2 http://blog.osgi.org/2011/04/osgi-lite.html

# 2 Application Domain

The OSGi framework consists of a number of layers where the module layer is by far the largest and most complex. This layer has shown to be very useful for large and complex applications that require side by side versioning and encapsulation of their classes.

OSGi is currently not widely adopted by many enterprise programmers because class loader hacks/tricks have become the de-facto standards in this world. The class loader hacks require global visibility to the class name space as provided by the WAR format as well as the Java class path. Over the past decade many fixes have been added to Java to provide solutions to the problems caused by these hacks. The Thread Context Class Loader is one of those, as is the Service Loader. The class loading hacks are used because Java has no extension model except for the global ext directory. The  Service Loader is an attempt in this direction but falls short because it became closely tied to class loaders again and is in almost any aspect a poor extension model.

The OSGi class loading model isolates bundles in their own class loader, minimizing the global space. The consequence of these modularity boundaries is that many of the class loading hacks fail to work

The consequence of this lack of extension model is that almost every project above a certain size has implemented a plugin model based on class loaders.

OSGi has an extension model: services. This model provides a very rich collaboration model allowing modules to be highly encapsulated. Only API needs to be shared, all implementation code will never have to be exposed to other modules. The OSGi model can be used to replace the class loading hacks that are used to create extension mechanisms, mostly factories with much less complexity and much more functionality.

One of the more popular class loading hacks is the scanning of the class path. In standard Java, there is no iterator over classes/resources. That is, it is not possible to iterate over classes. This shortcoming is addressed by applications to make assumptions about the class loaders and the sources of where they load their classes from. There are many programs that only work with the URL Class Loader and require the URLs to be either file: or jar:. The OSGi API provides methods to iterator over the resource entries as well as the class path.

One of the key benefits of OSGi is not well understood by non-OSGi developers. An OSGi bundle gets control when the system is started by the mere fact that it is installed. In normal Java applications there is usually a central place that gets control. This can be the main method or the web.xml in WARs. The OSGi is required in collaborative computing.

## 2.1 Terminology + Abbreviations

OSGi Connect – Working name given to this effort.

# 3 Problem Description

The current OSGi framework forces developers to first replace all their class loading hacks before they can run on OSGi. However, once you remove the class loading hacks you need to use services to obtain instances. This makes the adoption of OSGi very painful for the average Java EE developers because they have to perform a lot of work before they can enjoy the fruits of their labor. In almost any situation, it is a bad strategy to require a significant investment before any return can be realized.

There is a need for a solution that allows Java developers to reap the benefits of the OSGi service model and bundle model but that still provides global visibility to the classes. This will allow the quick migration of existing applications to an OSGi model so they can take advantage of the service and bundle model with minimal investment.

# 4 Use Cases

## 4.1 bnd in Ant and Maven

bnd, the OSGi tool to make manifests, is an eclipse plugin, an ant task, a bundle, and a command line utility. Because it has to run inside Ant and Maven it cannot rely on the presence of an OSGi framework. For this reason, it sadly implements a plugin model based on class loaders. If a light weight OSGi was available in around 50K-100K that provided the OSGi service registry then bnd could carry this OSGi Connect in its Ant task and Maven plugin. Though the 400k-1Mb of most frameworks would not be impossible, the class loading issues around using a real framework in Ant or Maven will be hard because these build tools assume global class visibility.

## 4.2 WARs

The Java EE is a highly popular format for deploying web applications. The different servlet bridges have made it possible to use OSGi inside the WARs but especially in Java EE based applications the class loading issues are prevalent. An OSGi Connect could be triggered when a servlet is loaded and thereby activating all “bundles” and let them collaborate through the service registry.

## 4.3 Application Frameworks

Numerous application frameworks exist to provide convenient access to various services ranging from parsing, logging, and security to web access and persistence. The popularity or usefulness of a given framework could lead it to be adopted and even relied upon by an application. Unfortunately, some frameworks have intrinsic dependencies and built-in assumptions about the classloader environment. In some cases these frameworks will not function correctly in OSGi. OSGi Connect is one way to enable applications to reap some of the benefits of OSGi services, but still be able to make use of existing frameworks (that may be inflexibly coded).

## 4.4 Non-OSGi Developers

OSGi has been criticized as being overly complex and hard to use. While many application developers may be interested in the idea of modularity through services, many of them are not willing to make the move to use OSGi because of their perception of it being complex. An architecture that includes a simple and intuitive service layer, without the necessity of having to fully understand the additional OSGi layers, would provide many developers with a convenient portal into a partial OSGi world and allow them to enjoy the benefits of a full-featured service registry.

# 5 Requirements

## 5.1 Basic

- BA0001 – Provide a specification, OSGi Connect, that provides much of the existing OSGi API but that does not provide the module layer functionality. That is, it does not provide any class loaders but loads classes using the class loaders already present in the environment.

- BA0002 – The specification should avoid defining different API for operations where the existing API meets the need. Additional API should only be added, if necessary, for bootstrapping purposes.

- BA0003 – The specification must support the service layer including registering and unregistering of services, service properties, service factories, querying for services using filters, service events, service hooks and security.

- BA0004 – The specification must support a subset of the lifecycle layer functionality, including starting and stopping, bundle events, bundle hooks and environment property access.

- BA0005 – The specification must not require asynchronous thread creation by the framework, or must have an optional mode where such thread creation is guaranteed to not occur. When thread creation is not permitted, asynchronous events must be delivered synchronously.

- BA0010 – It must be possible to author a bundle that can run in both full OSGi and OSGi Connect, with concrete specifiable rules for doing so. For example, no Bundle-Classpath other than "/", no duplicate packages, etc.

- BA0020 – It must be possible to treat JARs, if the environment classloader supports their discovery, as bundles so that applications can use the same lifecycle API in both OSGi Connect and full OSGi .

- BA0050 – The specification must provide sufficient functionality to allow existing service layer extender models, such as Blueprint and Declarative Services, to be supported. However, extender implementations may need minor changes to support operation in OSGi Connect mode and asynchronous thread creation mode may be required.

# 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

## 6.2 Author's Address

| | |
|---|---|
| Name | Peter Kriens |
| Company | aQute |
| Address | 9c, Avenue St. Drézery, Beaulieu, FRANCE |
| Voice | 33467542167 |
| e-mail | Peter.Kriens@aQute.biz |

## 6.3 End of Document