RFP 196 OSGi Connect

Draft

7 Pages

Abstract

Discusses the needs for a specification that would make code and services from outside of an OSGi framework available to OSGi and vice versa.

0 Document Information

0.1 Table of Contents

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	2019-05-29	Carsten Ziegeler, Initial draft based on RFP-143 and F2F discussion in Chicago
	06/05/19	Karl Pauls, update after EEG call
	03/07/19	Karl Pauls, update after EEG call

1 Introduction

This RFP discusses the need for an OSGi Framework that can integrate externally defined modules which are outside of the control of the framework. In other words, to connect the OSGi Framework to code and services from the outside world like jars on the classpath or JPMS modules.

Parts of this work were first explored in RFP-143 OSGiConnect. Although RFP-143 got accepted it never made it into an RFC. This RFP is taking the basic idea and generalizes it to allow for an adjusted set of use cases as well as focusing on a design that could possibly be driven by an SPI.

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 proved to be very useful for large and complex applications that require side by side versioning and encapsulation of their classes.

The OSGi module layer takes care of the class loading for each bundle and isolates bundles in their own class loader, minimizing global space. The consequence of the module layer being responsible for class loading, prevents or at least makes it very difficult to use classes or services that are already present on the class path and whose class space is managed outside of OSGi.

At the same time, functionality running outside of OSGi is not easily able to benefit from the rich service model of OSGi in a standardized way.

Finally, the OSGi framework assumes it is in control of the actual deployment units i.e., the jar files that get installed as Bundles. There is no way to represent outside content as bundles where the framework hasn't been given the deployment unit.

2.1 Terminology + Abbreviations

OSGi Connect – Working name given to this effort.

Connect Bundle – A connect bundle is provided from outside of the OSGi framework and represented within the OSGi framework as a bundle. A connect bundle can function as a proper bundle but might have some limitations with respect to class loading and isolation which need to be defined during the specification process.

3 Problem Description

Code running outside of the OSGi framework is hard to use from the inside. The framework can delegate some of it via the system bundle exports but for "normal" Java applications that is often not enough and somewhat tedious in any case as the delegation is rather static. Furthermore, it isn't possible to represent logical units on the outside (like JPMS Modules, OSGi bundles on the classpath, or other components) inside the framework (as the only delegation method is packages exposed via the system bundle).

That makes it hard and in some cases impossible to create hybrid solutions that can be used inside OSGi as well as in other contexts and that would still work if the two are used within the same application. In other words, there is a need for a solution that allows to bridge part of the outside world into the OSGi framework in such a way that normal OSGi mechanisms apply to it.

~~There is a need for a solution that allows Java developers to connect an OSGi Framework to code running outside of the OSGi framework. This opens up OSGi to "normal" Java applications and allows to use OSGi functionality for dedicated use cases. At the same time, OSGi gets access to the outer world and is not losing any of its core functionality and benefits. Furthermore, if it would be possible to connect outside code to an OSGi Framework in such a way that it doesn't have to be in control of the actual deployment unit in order to represent it as a bundle, it could serve as a light weight service registry more easily. Finally, it would allow developers to create hybrid modules that can be used inside OSGi as well as in other contexts and would still work if the two are used within the same application.~~

4 Use Cases

4.1 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 without the need to actually copy them into the framework cache.

4.2 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 to 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 use some of the benefits of OSGi services and modules, but still be able to make use of existing frameworks.

4.3 Build a hybrid OSGi-JPMS application

A Java application may have some of it’s functionality implemented as bundles and another part implemented as JPMS modules. It should be possible to use JPMS modules within an OSGi framework. This could make it possible to accurately reflect available system packages in a JPMS environment and would allow the framework to itself be a JPMS module. Ultimately, it could be possible to create JPMS modules that carry enough meta-data to work inside OSGi as well and vice versa where one could still use them inside OSGi even in case they are provided as JPMS modules in a given application.

4.4 ~~Runnable~~Bundles available via an outside classloader ~~fat jar~~

In some scenarios, an application has the bundles already available in a classloader (e.g., they are loaded as normal jars from the classpath). ~~its common to package a Java application as a single jar with all the jars embedded. Usually a custom classloader is used to add all embedded jars to the classpath and then the application is launched.~~ In these scenarios ~~all jars/modules are already on the classpath and~~ a contained OSGi framework does not need to provide module isolation and can directly use the jars as bundles, still providing most of the OSGi functionality.

4.5 Native compilation

In some scenarios, it is becoming common place that Java applications are compiled to native code AOT. In AOT compilation, custom classloaders can be a problem. This approach would potentially allow to make OSGi bundles work from the outside where they (together with the framework) are AOT compiled.

5 Requirements

5.1 Basic

- BA0010 – It must be possible to install Connect Bundles.
- BA0020 – It must be possible for Connect bundles to participate in the Service Layer as usual for Bundles.
- BA0030 – It must be possible to use OSGi services implementing an API provided by a Connect Bundle outside of the OSGi framework.

5.2 Connect Bundle

- CB0010 – It must be possible to provide the class loader for a connect bundle.
- CB0020 – It must be possible to influence the wiring of a connect bundle similar to existing resolver hooks.
- CB0030 – It must be possible to provide the manifest headers of a connect bundle.
- CB0040 – It must be possible to provide the resources/entries of a connect bundle.
- CB0050 – Connect bundles must be treated and behave like any other bundle except for a limited set of functionalities that are explicitly not supported.
- CB0060 – Connect Bundles must be subject to the standard OSGi resolution rules.
- CB0070 – All existing lifecycle methods must be well-defined wrt. Connect Bundles.
- ~~CB0080 – Installed Connect Bundles must be able to be persisted across restarts.~~
- CB0080 – Installed Connect Bundles must be able to be persisted across restarts.

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	Carsten Ziegeler
Company	Adobe
Address	
Voice	
e-mail	cziegele@adobe.com

Name	Karl Pauls
Company	Adobe
Address	
Voice	
e-mail	kpauls@adobe.com

6.3 End of Document