RFC 6 - A Technique for Defining a Minimum Execution Environment for OSGi Bundles

Members Only, Final
cpeg-rfc_6_bundle_env-1_00

8 Pages

Abstract

This RFC specifies how an Execution Environment for OSGi Bundles will be specified and how it may be used. It does not define an Execution Environment. This will be defined by other RFCs.

0 Document Information

0.1 Table of Contents

0.2 Status

This document specifies a technique for defining Execution Environments for OSGi Bundles, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within OSGi.

0.3 Acknowledgement

0.4 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.
```

All Page Within This Box

0.5 Revision History

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

Revision	Date	Comments
First draft.	26 October 2000	BJ Hargrave, IBM, hargrave@us.ibm.com
Second draft	7 November 2000	BJ Hargrave, IBM, hargrave@us.ibm.com, and Peter Kriens, Peter.Kriens@aQute.se
Third draft	4 April 2001	David Bowen, Sun
Fourth draft	9 April 2001	BJ Hargrave, IBM, hargrave@us.ibm.com, and Peter Kriens, Peter.Kriens@aQute.se
Review Draft	2 May 2001	BJ Hargrave, IBM, hargrave@us.ibm.com, and Peter Kriens, Peter.Kriens@aQute.se
Final	16 May 2001	BJ Hargrave, IBM, hargrave@us.ibm.com. Added sentence to 3.3 for package private constructors.

1 Introduction

The purpose of this RFC is to define the technique by which an Execution Environment for OSGi Bundles will be specified. An Execution Environment is defined to be the collection of public and protected methods and fields in public classes and interfaces provided by the runtime environment (RE) that an OSGi Bundle can assume exists and can be accessed by the bundle at runtime.

Note 1: An Execution Environment is not defined by simply enumerating a collection of classes and interfaces. In some cases, a class or interface in an Execution Environment may not contain all the methods and fields with which the class or interface is defined in the appropriate documentation for that class.

Note 2: An Execution Environment is not constrained to Java classes only but can also contain OSGi classes and interfaces, javax packages and any other classes that must be present in the runtime environment before the Bundle can execute.

The goals satisfied by this technique will be a machine-readable specification that can be used to:

1. Validate that a vendor's RE implementation supports the required methods and fields.
2. Validate that bundles do not use classes, interfaces, methods or fields not in the specification.
3. Support the upcoming OSGi validation process for REs, frameworks and bundles.
4. Minimize tool development work to support this specification.
   - Use existing parsers.
5. No new specification language syntax.

All Page Within This Box

Other RFCs will define Execution Environments using the technique defined by this RFC.

2 Motivation and Rationale

It is critical for OSGi's success to be able to provide a validation process for REs, framework implementations, and bundles. Having such a validation process significantly improves interoperability among OSGi components, reduces confusion for bundle programmers and framework implementers and allows REs to provide support for OSGi Execution Environments.

In order to implement such a validation process, it is necessary to have a formal specification technique that describes an execution environment "contract". The formal specification technique described in this RFC has the following attributes:

- It is an open specification, defined by OSGi. It does not give any company an unfair advantage over others as intended by the OSGi Membership Agreement and By-Laws.
- It clearly and formally specifies which classes and interfaces, and within them, which methods and fields, with their full prototype information, will be available for use by a bundle.
- It is machine readable and verifiable so anyone can claim compliance and verify this compliance.

3 Technical Discussion

3.1 Overview

Execution Environment specifications for OSGi Bundles will only specify signature information regarding classes, interfaces, methods and fields. That is, only the Java syntax information is captured: member name, parameter types, throws clause, modifiers and return type. No attempt is made to capture semantic details. This is beyond the scope of this effort. It is assumed that the semantic specification is captured in the appropriate documentation.

Using signature information, tools can assess whether a specific method or field is required or present. For the purposes of analyzing REs to determine if they implement an Execution Environment specification, it is assumed that if the method or field is present, that is has the correct semantic behavior as described in the relevant documentation for that method or field.

3.2 Binary Format

The binary format of an Execution Environment for OSGi Bundles specification will be a collection of Java class files contained in a JAR file. More specifically, the JAR file will contain a collection of `.class` files in the Java

class file format. Each `.class` file will contain a valid Java class file. The class file will describe a public Java class or interface and its public and protected methods and fields. Package private and private classes, methods and fields are not included since they are not part of a public API. The bytecodes comprising the bodies of the methods for the class will be empty to the extent permitted for the class file to be a valid class file. *i.e.* a method that returns a value must have a method body that returns a value of the correct type in order for the method body to be valid.

3.3 Source Format

The source format of an Execution Environment specification will be a collection of Java source files. Each Java source file represents a public Java class or interface and contains valid, compilable Java source code defining the public and protected methods and fields of the Java class or interface.

- The fields will be declared with no initializer value, thus taking the default value.
- The constructors will be declared with an empty method body except when the superclass does not have a default constructor. In that case, the method body will consist of a single call to one of the superclass constructors with dummy argument values.

Parameter type	**Argument value**
Numerical	`0`
`boolean`	`False`
Object references	`Null`

  To avoid the compiler automatically inserting a public (or protected) constructor, a package private constructor will be declared for classes with only private or package private constructors.
- The methods will be declared with minimal bodies depending upon the return type.

Return Type	**Method Body**
`void`	`{}`
Numerical	`{ return 0; }`
`boolean`	`{ return false; }`
Object references	`{ return null; }`

The collection of these Java source files can be compiled without error by a valid Java compiler (*e.g.* `javac`) and the collection of output Java class files will be package into a JAR file thus producing the binary format of an Execution Environment specification.

3.4 Use Of The Specification

The purpose of the source format of the specification is to provide a conveniently editable specification format for humans that can be easily converted into the binary format. The binary format of the specification can be use by many different possible tools. The Java class file is the central data format to be processed by these tools. The binary format of the specification is a collection of Java class files, just like a bundle is a collection of Java class files, and also the base classes of a RE can be seen as a collection of Java class files. Many different Java packages that process and analyze Java class files are available to be used as a basis for development of tools.

Note: The analysis performed by such tools would only be a static analysis. Classes, interfaces, methods and fields referenced by `Class.forName` and reflection would not be visible and thus missed from analysis.

3.4.1 Example Tools

3.4.1.1 RE Compatibility

A tool can be written to verify that a RE implements the classes, interfaces, methods and fields specified in an Execution Environment specification. The tool would read the binary format of the specification and analyze the class files in the RE to verify that the RE implements all the class, interface, method and field signatures specified by an Execution Environment specification.

3.4.1.2 Bundle Validation

A tool can be written to validate that a bundle does not use any classes, interfaces, methods and fields that are not contained in an Execution Environment specification. The tool would read the binary format of the specification and analyze the class files in the bundle. The tool would ignore references to classes, interfaces, methods and fields in imported packages (defined by `Import-Package` headers in the bundle's manifest), as well as those defined within the bundle. Reference to all other classes, interfaces, methods and fields would be compared against an Execution Environment specification to validate that the bundle did not use any classes, interfaces, methods or fields not in an Execution Environment specification.

3.4.1.3 Bundle Set Closure

Finally, a sophisticated tool can be written to validate that a set of bundles, an Execution Environment specification, and the OSGi framework specification together provide a complete closure of referenced methods and fields. Such a tool would ensure that all `Export-Package` and `Import-Package` headers resolve, and that all referenced classes, interfaces, methods and fields are defined within the set of bundles, an Execution Environment and the OSGi framework specification. Thus, this tool will provide a complete static analysis and validate that the set of bundles will resolve on any OSGi framework.

4 Security Considerations

This RFC has no security considerations. This RFC specifies a formal technique for specifying an Execution Environment for OSGi Bundles. An Execution Environment specification will be used by tools in development and deployment environments and generally will not be used in the OSGi execution environment.

All Page Within This Box

5 Document Support

5.1 References

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

5.2 Author's Addresses

Name	BJ Hargrave
Company	IBM Pervasive Computing
Address	11400 Burnet Road Austin, TX 78758 USA
Voice	+1 512 838 9938
e-mail	hargrave@us.ibm.com

Name	Peter Kriens
Company	Ericsson
Address	Finnasandsvägen 22 Onsala 43933 Sweden
Voice	+46 300 39800
e-mail	Peter.Kriens@aQute.se

Name	Tommy Bohlin
Company	Gatespace
Address	Stora Badhusgatan 18-20 Göteborg 41121 Sweden
Voice	+46 31 743 9800
e-mail	tommy@gatespace.com

Name	James Smith
Company	Verizon

All Page Within This Box

Address	700 Hidden Ridge HQW02J62 Irving, TX 75038
Voice	+1 972 718 6564
e-mail	james.smith@verizon.com

5.3 Acronyms and Abbreviations

OSGi	Open Services Gateway Initiative
JRE	Java runtime environment
RE	runtime environment (general)

5.4 End of Document

All Page Within This Box