

**OSGi Working Group
OSGi Compendium**

**Release 8.1
December 2022**



Licensed under the Eclipse Foundation Specification

Copyright © 2000, 2024 Eclipse Foundation

LICENSE

Eclipse Foundation Specification License – v1.0

By using and/or copying this document, or the Eclipse Foundation document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, and distribute the contents of this document, or the Eclipse Foundation document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

- link or URL to the original Eclipse Foundation document.
- All existing copyright notices, or if one does not exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright © [date-of-document] Eclipse Foundation, Inc. <<url to this license>>".

Inclusion of the full text of this NOTICE must be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of Eclipse Foundation documents is granted pursuant to this license, except anyone may prepare and distribute derivative works and portions of this document in software that implements the specification, in supporting materials accompanying such software, and in documentation of such software, PROVIDED that all such works include the notice below. HOWEVER, the publication of derivative works of this document for use as a technical specification is expressly prohibited.

The notice is:

"Copyright © [date-of-document] Eclipse Foundation. This software or document includes material copied from or derived from [title and URI of the Eclipse Foundation specification document]."

Disclaimers

THIS DOCUMENT IS PROVIDED "AS IS," AND THE COPYRIGHT HOLDERS AND THE ECLIPSE FOUNDATION MAKE 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 DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

THE COPYRIGHT HOLDERS AND THE ECLIPSE FOUNDATION WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of the copyright holders or the Eclipse Foundation may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

Preface

Implementation Requirements

An implementation of a Specification: (i) must fully implement the Specification including all its required interfaces and functionality; (ii) must 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 Specification. An implementation that does not satisfy limitations (i) (ii) is not considered an implementation of the Specification and must not be described as an implementation of the Specification. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. An implementation of a Specification must not claim to be a compatible implementation of the Specification unless it passes the Technology Compatibility Kit ("TCK") for the Specification.

Feedback

This specification can be downloaded from the OSGi Documentation web site:

<https://docs.osgi.org/specification/>

Comments about this specification can be raised at:

<https://github.com/osgi/osgi/issues>

Table of Contents

159	Feature Service Specification	5
159.1	Introduction.....	5
159.2	Feature.....	6
159.3	Comments.....	8
159.4	Bundles.....	8
159.5	Configurations.....	10
159.6	Variables.....	10
159.7	Extensions.....	11
159.8	Framework Launching Properties.....	13
159.9	Resource Versioning.....	14
159.10	Capabilities.....	14
159.11	org.osgi.service.feature.....	14
159.12	org.osgi.service.feature.annotation.....	26
159.13	References.....	26
160	Feature Launcher Service Specification	27
160.1	Introduction.....	27
160.2	Features and Artifact Repositories.....	28
160.3	The Feature Launcher.....	30
160.4	org.osgi.service.featurelauncher.....	32

Licensed under the Eclipse Foundation Specification License – V1.0. Copyright © Contributors to the Eclipse Foundation.

DRAFT

159 Feature Service Specification

Version 1.0

159.1 Introduction

OSGi has become a platform capable of running large applications for a variety of purposes, including rich client applications, server-side systems and cloud and container based architectures. As these applications are generally based on many bundles, describing each bundle individually in an application definition becomes unwieldy once the number of bundles reaches a certain level.

When developing large scale applications it is often the case that few people know the role of every single bundle or configuration item in the application. To keep the architecture understandable a grouping mechanism is needed that allows for the representation of parts of the application into larger entities that keep reasoning about the system manageable. In such a domain members of teams spread across an organization will need to be able to both develop new parts for the application as well as make tweaks or enhancements to parts developed by others such as adding configuration and resources or changing one or more bundles relevant to their part of the application.

The higher level constructs that define the application should be reusable in different contexts, for example if one team has developed a component to handle job processing, different applications should be able to use it, and if needed tune its configuration or other aspects so that it works in each setting without having to know each and every detail that the job processing component is built up from.

Applications are often associated with additional resources or metadata, for example database scripts or custom artifacts. By including these with the application definition, all the related entities are encapsulated in a single artifact.

By combining various applications or subsystems together, systems are composed of existing, reusable building blocks, where all these blocks can work together. Architects of these systems need to think about components without having to dive into the individual implementation details of each subcomponent. The Features defined in this specification can be used to model such applications. Features contain the definition of an application or component and may be composed into larger systems.

159.1.1 Essentials

- *Declarative* - Features are declarative and can be mapped to different implementations.
- *Extensible* - Features are extensible with custom content to facilitate all information related to a Feature to be co-located.
- *Human Readable* - No special software is needed to read or author Features.
- *Machine Readable* - Features are easily be processed by tools.

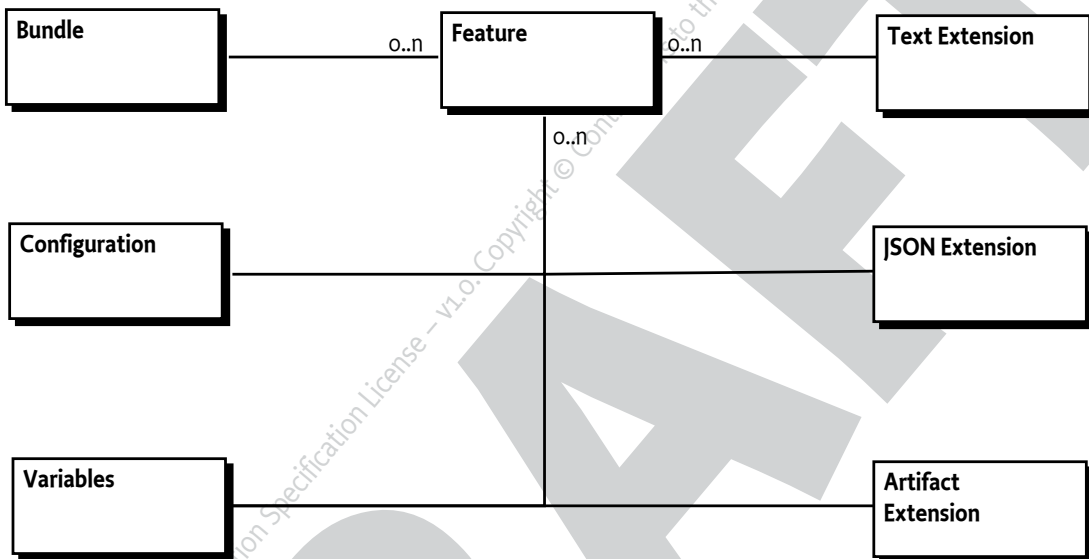
159.1.2 Entities

The following entities are used in this specification:

- *Feature* - A Feature contains a number of entities that, when provided to a launcher can be turned into an executable system. Features are building blocks which may be assembled into larger systems.

- *Bundles* - A Feature can contain one or more bundles.
- *Configuration* - A Feature can contain configurations for the Configuration Admin service.
- *Extension* - A Feature can contain a number of extensions with custom content.
- *Launcher* - A launcher turns one or more Features into an executable system.
- *Processor* - A Feature processor reads Features and perform a processing operation on them, such as validation, transformation or generation of new entities based on the Features.
- *Properties* - Framework launching properties can be specified in a Feature.

Figure 159.1 Features Entity overview



159.2 Feature

Features are defined by declaring JSON documents or by using the [Feature API](#). Each Feature has a unique ID which includes a version. It holds a number of entities, including a list of bundles, configurations and others. Features are extensible, that is a Feature can also contain any number of custom entities which are related to the Feature.

Features may have dependencies on other Features. Features inherit the capabilities and requirements from all bundles listed in the Feature.

Once created, a Feature is immutable. Its definition cannot be modified. However it is possible to record caching related information in a Feature through transient extensions. This cached content is not significant for the definition of the Feature or part of its identity.

159.2.1 Identifiers

Identifiers used throughout this specification are defined using the Maven Identifier model. They are composed of the following parts:

- Group ID
- Artifact ID
- Version
- Type (optional)

- Classifier (optional)

Note that if Version has the -SNAPSHOT suffix, the identifier points at an unreleased artifact that is under development and may still change.

For more information see [3] *Apache Maven Pom Reference*. The format used to specify identifiers is as follows:

```
groupId ':' artifactId ( ':' type ( ':' classifier )? )? ':' version
```

159.2.2 Feature Identifier

Each Feature has a unique identifier. Apart from providing a persistent handle to the Feature, it also provides enough information to find the Feature in an artifact repository. This identifier is defined using the format described in *Identifiers* on page 6.

159.2.2.1 Identifier type

Features use as identifier type the value `osgifeature`.

159.2.3 Attributes

A Feature can have the following attributes:

Table 159.1

Feature Attributes

Attribute name	Data Type	Kind	Description
categories	Array of String	Optional, defaults to an empty array	The categories this Feature belongs to. The values are user-defined.
complete	boolean	Optional, defaults to false	Completeness of the Feature. A Feature is complete when it has no external dependencies.
description	String	Optional	A longer description of the Feature.
docURL	String	Optional	A location where documentation can be found for the Feature.
license	String	Optional	The license of the Feature. The license only relates to the Feature itself and not to any artifacts that might be referenced by the Feature. The license follows the Bundle-License format as specified in the Core specification.
SCM	String	Optional	SCM information relating to the feature. The syntax of the value follows the Bundle-SCM format. See the 'Bundle Manifest Headers' section in the OSGi Core specification.
vendor	String	Optional	The vendor of the Feature.

An initial Feature without content can be declared as follows:

```
{
  "feature-resource-version": "1.0",
  "id": "org.acme:acmeapp:1.0.0",
  "name": "The ACME app",
```

```

    "description":
      "This is the main ACME app, from where all functionality is reached."

    /*
     * Additional Feature entities here
     * ...
     */
  }

```

159.2.4 Using the Feature API

Features can also be created, read and written using the Feature API. The main entry point for this API is the [FeatureService](#). The Feature API uses the builder pattern to create entities used in Features.

A builder instance is used to create a single entity and cannot be re-used to create a second one. Builders are created from the [BuilderFactory](#), which is available from the FeatureService through [getBuilderFactory\(\)](#).

```

FeatureService fs = ... // from Service Registry
BuilderFactory factory = fs.getBuilderFactory();

FeatureBuilder builder = factory.newFeatureBuilder(
    fs.getID("org.acme", "acmeapp", "1.0.0"));
builder.setName("The ACME app");
builder.setDescription("This is the main ACME app, "
    + "from where all functionality is reached.");

Feature f = builder.build();

```

The Feature API can also be useful in environments outside of an OSGi Framework where no service registry is available, for example in a build-system environment. In such environments the FeatureService can be obtained by using the `java.util.ServiceLoader` mechanism.

159.3 Comments

Comments in the form of [2] *JSMIn (The JavaScript Minifier)* comments are supported, that is, any text on the same line after `//` is ignored and any text between `/* */` is ignored.

159.4 Bundles

Features list zero or more bundles that implement the functionality provided by the Feature. Bundles are listed by referencing them in the bundles array so that they can be resolved from a repository. Bundles can have metadata associated with them, such as the relative start order of the bundle in the Feature. Custom metadata may also be provided. A single Feature can provide multiple versions of the same bundle, if desired.

Bundles are referenced using the identifier format described in *Identifiers* on page 6. This means that Bundles are referenced using their Maven coordinates. The bundles array contains JSON objects which can contain the bundle IDs and specify optional additional metadata.

159.4.1 Bundle Metadata

Arbitrary key-value pairs can be associated with bundle entries to store custom metadata alongside the bundle references. Reverse DNS naming should be used with the keys to avoid name clashes

when metadata is provided by multiple entities. Keys not using the reverse DNS naming scheme are reserved for OSGi use.

Bundle metadata supports string keys and string, number or boolean values.

The following example shows a simple Feature describing a small application with its dependencies:

```
{
  "feature-resource-version": "1.0",
  "id": "org.acme:acmeapp:1.0.1",

  "name": "The Acme Application",
  "license": "https://opensource.org/licenses/Apache-2.0",
  "complete": true,

  "bundles": [
    { "id": "org.osgi.org.osgi.util.function:1.1.0" },
    { "id": "org.osgi.org.osgi.util.promise:1.1.1" },
    {
      "id": "org.apache.commons:commons-email:1.5",

      // This attribute is used by custom tooling to
      // find the associated javadoc
      "org.acme.javadoc.link":
        "https://commons.apache.org/proper/commons-email/javadocs/api-1.5"
    },
    { "id": "com.acme:acmelib:1.7.2" }
  ]
  /*
   Additional Feature entities here
   ...
  */
}
```

159.4.2 Using the Feature API

A Feature with Bundles can be created using the Feature API as follows:

```
FeatureService fs = ... // from Service Registry
BuilderFactory factory = fs.getBuilderFactory();

FeatureBuilder builder = factory.newFeatureBuilder(
  fs.getID("org.acme", "acmeapp", "1.0.1"));
builder.setName("The Acme Application");
builder.setLicense("https://opensource.org/licenses/Apache-2.0");
builder.setComplete(true);

FeatureBundle b1 = factory
  .newBundleBuilder(fs.getIDfromMavenCoordinates(
    "org.osgi.org.osgi.util.function:1.1.0"))
  .build();
FeatureBundle b2 = factory
  .newBundleBuilder(fs.getIDfromMavenCoordinates(
    "org.osgi.org.osgi.util.promise:1.1.1"))
  .build();
```

```

FeatureBundle b3 = factory
    .newBundleBuilder(fs.getIDfromMavenCoordinates(
        "org.apache.commons:commons-email:1.1.5"))
    .addMetadata("org.acme.javadoc.link",
        "https://commons.apache.org/proper/commons-email/javadocs/api-1.5")
    .build();
FeatureBundle b4 = factory
    .newBundleBuilder(fs.getIDfromMavenCoordinates(
        "com.acme:acmelib:1.7.2"))
    .build();

builder.addBundles(b1, b2, b3, b4);
Feature f = builder.build();

```

159.5 Configurations

Features support configuration using the OSGi Configurator syntax, see ???. This is specified with the configurations key in the Feature. A Launcher can apply these configurations to the Configuration Admin service when starting the system.

It is an error to define the same PID twice in a single Feature. An entity processing the feature must fail in this case.

Example:

```

{
  "feature-resource-version": "1.0",
  "id": "org.acme:acmeapp:osgifeature:configs:1.0.0",
  "configurations": {
    "org.apache.felix.http": {
      "org.osgi.service.http.port": 8080,
      "org.osgi.service.http.port.secure": 8443
    }
  }
}

```

159.6 Variables

Configurations and Framework Launching Properties support late binding of values. This enables setting these items through a Launcher, for example to specify a database user name, server port number or other information that may be variable between runtimes.

Variables are declared in the variables section of the Feature and they can have a default value specified. The default must be of type string, number or boolean. Variables can also be declared to *not* have a default, which means that they must be provided with a value through the Launcher. This is done by specifying null as the default in the variable declaration.

Example:

```

{
  "feature-resource-version": "1.0",
  "id": "org.acme:acmeapp:osgifeature:configs:1.1.0",
  "variables": {
    "http.port": 8080,
    "db.username": "scott",

```

```

        "db.password": null
    },
    "configurations": {
        "org.acme.server.http": {
            "org.osgi.service.http.port:Integer": "${http.port}"
        },
        "org.acme.db": {
            "username": "${db.username}-user",
            "password": "${db.password}"
        }
    }
}

```

Variables are referenced with the curly brace placeholder syntax: `${variable-name}` in the configuration value or framework launching property value section. To support conversion of variables to non-string types the configurator syntax specifying the datatype with the configuration key is used, as in the above example.

Multiple variables can be referenced for a single configuration or framework launching property value and variables may be combined with text. If no variable exist with the given name, then the `${variable-name}` must be retained in the value.

159.7 Extensions

Features can include custom content. This makes it possible to keep custom entities and information relating to the Feature together with the rest of the Feature.

Custom content is provided through Feature extensions, which are in one of the following formats:

- *Text* - A text extension contains an array of text.
- *JSON* - A JSON extension contains embedded custom JSON content.
- *Artifacts* - A list of custom artifacts associated with the Feature.

Extensions can have a variety of consumers. For example they may be handled by a Feature Launcher or by an external tool which can process the extension at any point of the Feature life cycle.

Extensions are of one of the following three kinds:

- *Mandatory* - The entity processing this Feature *must* know how to handle this extension. If it cannot handle the extension it must fail.
- *Optional* - This extension is optional. If the entity processing the Feature cannot handle it, the extension can be skipped or ignored. This is the default.
- *Transient* - This extension contains transient information which may be used to optimize the processing of the Feature. It is not part of the Feature definition.

Extensions are specified as JSON objects under the extensions key in the Feature. A Feature can contain any number of extensions, as long as the extension keys are unique. Extension keys should use reverse domain naming to avoid name clashing of multiple extensions in a single Feature. Extensions names without a reverse domain naming prefix are reserved for OSGi use.

159.7.1 Text Extensions

Text extensions support the addition of custom text content to the Feature. The text is provided as a JSON array of strings.

Example:

```

{
  "feature-resource-version": "1.0",
  "id": "org.acme:acmeapp:2.0.0",

  "name": "The Acme Application",
  "license": "https://opensource.org/licenses/Apache-2.0",

  "extensions": {
    "org.acme.mydoc": {
      "type": "text",
      "text": [
        "This application provides the main acme ",
        "functionality."
      ]
    }
  }
}

```

159.7.2**JSON Extensions**

Custom JSON content is added to Features by using a JSON extension. The content can either be a JSON object or a JSON array.

The following example extension declares under which execution environment the Feature is complete, using a custom JSON object.

```

{
  "feature-resource-version": "1.0",
  "id": "org.acme:acmeapp:2.1.0",

  "name": "The Acme Application",
  "license": "https://opensource.org/licenses/Apache-2.0",

  "extensions": {
    "org.acme.execution-environment": {
      "type": "json",
      "json": {
        "environment-capabilities":
          ["osgi.ee; filter:=\("&(osgi.ee=JavaSE)(version=11))\""],
        "framework": "org.osgi:core:6.0.0",
        "provided-features": ["org.acme:platform:1.1"]
      }
    }
  }
}

```

159.7.3**Artifact list Extensions**

Custom extensions can be used to associate artifacts that are not listed as bundles with the Feature.

For example, database definition resources may be listed as artifacts in a Feature. In the following example, the extension `org.acme.ddlfiles` lists Database Definition Resources which *must* be handled by the launcher agent, that is, the database must be configured when the application is run:

```

{
  "feature-resource-version": "1.0",
  "id": "org.acme:acmeapp:2.2.0",

```

```

    "name": "The Acme Application",
    "license": "https://opensource.org/licenses/Apache-2.0",
    "complete": true,

    "bundles": [
      "org.osgi.org.osgi.util.function:1.1.0",
      "org.osgi.org.osgi.util.promise:1.1.1",
      "com.acme:acmelib:2.0.0"
    ],

    "extensions": {
      "org.acme.ddlfiles": {
        "kind": "mandatory",
        "type": "artifacts",
        "artifacts": [
          { "id": "org.acme:appddl:1.2.1" },
          {
            "id": "org.acme:appddl-custom:1.0.3",
            "org.acme.target": "custom-db"
          }
        ]
      }
    }
  }
}

```

As with bundle identifiers, custom artifacts are specified in an object in the artifacts list with an explicit id and optional additional metadata. The keys of the metadata should use a reverse domain naming pattern to avoid clashes. Keys that do not use reverse domain name as a prefix are reserved for OSGi use. Supported metadata values must be of type string, number or boolean.

159.8 Framework Launching Properties

When a Feature is launched in an OSGi framework it may be necessary to specify Framework Properties. These are provided in the Framework Launching Properties extension section of the Feature. The Launcher must be able to satisfy the specified properties. If it cannot ensure that these are present in the running Framework the launcher must fail.

Framework Launching Properties can reference Variables as defined in *Variables* on page 10. These variables are substituted before the properties are set.

Example:

```

{
  "feature-resource-version": "1.0",
  "id": "org.acme:acmeapp:osgifeature:fw-props:2.0.0",

  "variables": {
    "fw.storage.dir": "/tmp" // Can be overridden through the launcher
  },

  "extensions": {
    "framework-launching-properties": {
      "type": "json",
      "json": {

```

```

    "org.osgi.framework.system.packages.extra":
      "javax.activation;version=\\"1.1.1\\" ",
    "org.osgi.framework.bootdelegation": "javax.activation",
    "org.osgi.framework.storage": "${fw.storage.dir}"
  }
}
}
}

```

159.9 Resource Versioning

Feature JSON resources are versioned to support updates to the JSON structure in the future. To declare the document version of the Feature use the `feature-resource-version` key in the JSON document.

```

{
  "feature-resource-version": "1.0",
  "id": "org.acme:acmeapp:1.0.0"

  /*
   * Additional Feature entities here
   * ...
   */
}

```

The currently supported version of the Feature JSON documents is 1.0. If no Feature Resource Version is specified 1.0 is used as the default.

159.10 Capabilities

159.10.1 `org.osgi.service` Capability

The bundle providing the Feature Service must provide a capability in the `org.osgi.service` namespace representing the services it is registering. This capability must also declare uses constraints for the relevant service packages:

```

Provide-Capability: org.osgi.service;
  objectClass:List<String>="org.osgi.service.feature.FeatureService";
  uses:="org.osgi.service.feature"

```

This capability must follow the rules defined for the ???.

159.11 `org.osgi.service.feature`

Feature Package Version 1.0.

Bundles wishing to use this package must list the package in the `Import-Package` header of the bundle's manifest. This package has two types of users: the consumers that use the API in this package and the providers that implement the API in this package.

Example import for consumers using the API in this package:

```

Import-Package: org.osgi.service.feature; version="[1.0,2.0)"

```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.feature; version="[1.0,1.1]"
```

159.11.1 Summary

- **BuilderFactory** - The Builder Factory can be used to obtain builders for the various entities.
- **Feature** - The Feature Model Feature.
- **FeatureArtifact** - An Artifact is an entity with an ID, for use in extensions.
- **FeatureArtifactBuilder** - A builder for FeatureArtifact objects.
- **FeatureBuilder** - A builder for Feature Models.
- **FeatureBundle** - A Bundle which is part of a feature.
- **FeatureBundleBuilder** - A builder for Feature Model FeatureBundle objects.
- **FeatureConfiguration** - Represents an OSGi Configuration in the Feature Model.
- **FeatureConfigurationBuilder** - A builder for Feature Model FeatureConfiguration objects.
- **FeatureConstants** - Defines standard constants for the Feature specification.
- **FeatureExtension** - A Feature Model Extension.
- **FeatureExtension.Kind** - The kind of extension: optional, mandatory or transient.
- **FeatureExtension.Type** - The type of extension
- **FeatureExtensionBuilder** - A builder for Feature Model FeatureExtension objects.
- **FeatureService** - The Feature service is the primary entry point for interacting with the feature model.
- **ID** - ID used to denote an artifact.

159.11.2 public interface BuilderFactory

The Builder Factory can be used to obtain builders for the various entities.

Provider Type Consumers of this API must not implement this type

159.11.2.1 public FeatureArtifactBuilder newArtifactBuilder(ID id)

id The artifact ID for the artifact object being built.

- Obtain a new builder for Artifact objects.

Returns The builder.

159.11.2.2 public FeatureBundleBuilder newBundleBuilder(ID id)

id The ID for the bundle object being built. If the ID has no type specified, a default type of `@{code jar}` is assumed.

- Obtain a new builder for Bundle objects.

Returns The builder.

159.11.2.3 public FeatureConfigurationBuilder newConfigurationBuilder(String pid)

pid The persistent ID for the Configuration being built.

- Obtain a new builder for Configuration objects.

Returns The builder.

159.11.2.4 public FeatureConfigurationBuilder newConfigurationBuilder(String factoryPid, String name)

factoryPid The factory persistent ID for the Configuration being built.

name The name of the configuration being built. The PID for the configuration will be the factoryPid + '+' + name

- Obtain a new builder for Factory Configuration objects.

Returns The builder.

159.11.2.5 **public FeatureExtensionBuilder newExtensionBuilder(String name, FeatureExtension.Type type, FeatureExtension.Kind kind)**

name The extension name.

type The type of extension: JSON, Text or Artifacts.

kind The kind of extension: Mandatory, Optional or Transient.

- Obtain a new builder for Feature objects.

Returns The builder.

159.11.2.6 **public FeatureBuilder newFeatureBuilder(ID id)**

id The ID for the feature object being built. If the ID has no type specified, a default type of osgifeature is assumed.

- Obtain a new builder for Feature objects.

Returns The builder.

159.11.3 **public interface Feature**

The Feature Model Feature.

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

159.11.3.1 **public List<FeatureBundle> getBundles()**

- Get the bundles.

Returns The bundles. The returned list is unmodifiable.

159.11.3.2 **public List<String> getCategories()**

- Get the categories.

Returns The categories. The returned list is unmodifiable.

159.11.3.3 **public Map<String, FeatureConfiguration> getConfigurations()**

- Get the configurations. The iteration order of the returned map should follow the definition order of the configurations in the feature.

Returns The configurations. The returned map is unmodifiable.

159.11.3.4 **public Optional<String> getDescription()**

- Get the description.

Returns The description.

159.11.3.5 **public Optional<String> getDocURL()**

- Get the documentation URL.

Returns The documentation URL.

159.11.3.6 **public Map<String, FeatureExtension> getExtensions()**

- Get the extensions. The iteration order of the returned map should follow the definition order of the extensions in the feature.

Returns The extensions. The returned map is unmodifiable.

159.11.3.7 **public ID getID()**

- Get the Feature's ID.

Returns The ID of this Feature.

159.11.3.8 **public Optional<String> getLicense()**

- Get the license of this Feature. The syntax of the value follows the Bundle-License header syntax. See the 'Bundle Manifest Headers' section in the OSGi Core specification.

Returns The license.

159.11.3.9 **public Optional<String> getName()**

- Get the name.

Returns The name.

159.11.3.10 **public Optional<String> getSCM()**

- Get the SCM information relating to the feature. The syntax of the value follows the Bundle-SCM format. See the 'Bundle Manifest Headers' section in the OSGi Core specification.

Returns The SCM information.

159.11.3.11 **public Map<String, Object> getVariables()**

- Get the variables. The iteration order of the returned map should follow the definition order of the variables in the feature. Values are of type: String, Boolean or BigDecimal for numbers. The null JSON value is represented by a null value in the map.

Returns The variables. The returned map is unmodifiable.

159.11.3.12 **public Optional<String> getVendor()**

- Get the vendor.

Returns The vendor.

159.11.3.13 **public boolean isComplete()**

- Get whether the feature is complete or not.

Returns Completeness value.

159.11.4 **public interface FeatureArtifact**

An Artifact is an entity with an ID, for use in extensions.

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

159.11.4.1 **public ID getID()**

- Get the artifact's ID.

Returns The ID of this artifact.

159.11.4.2 **public Map<String, Object> getMetadata()**

- Get the metadata for this artifact.

Returns The metadata. The returned map is unmodifiable.

159.11.5 public interface FeatureArtifactBuilder

A builder for FeatureArtifact objects.

Concurrency Not Thread-safe

Provider Type Consumers of this API must not implement this type

159.11.5.1 public FeatureArtifactBuilder addMetadata(String key, Object value)

key Metadata key.

value Metadata value.

- Add metadata for this Artifact.

Returns This builder.

159.11.5.2 public FeatureArtifactBuilder addMetadata(Map<String, Object> metadata)

metadata The map with metadata.

- Add metadata for this Artifact by providing a map. All metadata in the map is added to any previously provided metadata.

Returns This builder.

159.11.5.3 public FeatureArtifact build()

- Build the Artifact object. Can only be called once on a builder. After calling this method the current builder instance cannot be used any more.

Returns The Feature Artifact.

159.11.6 public interface FeatureBuilder

A builder for Feature Models.

Concurrency Not Thread-safe

Provider Type Consumers of this API must not implement this type

159.11.6.1 public FeatureBuilder addBundles(FeatureBundle... bundles)

bundles The Bundles to add.

- Add Bundles to the Feature.

Returns This builder.

159.11.6.2 public FeatureBuilder addCategories(String... categories)

categories The Categories.

- Adds one or more categories to the Feature.

Returns This builder.

159.11.6.3 public FeatureBuilder addConfigurations(FeatureConfiguration... configs)

configs The Configurations to add.

- Add Configurations to the Feature.

Returns This builder.

159.11.6.4 public FeatureBuilder addExtensions(FeatureExtension... extensions)

extensions The Extensions to add.

- Add Extensions to the Feature

Returns This builder.

159.11.6.5 public FeatureBuilder addVariable(String key, Object defaultValue)

key The key.

defaultValue The default value.

- Add a variable to the Feature. If a variable with the specified key already exists it is replaced with this one. Variable values are of type: String, Boolean or BigDecimal for numbers.

Returns This builder.

Throws IllegalArgumentException– if the value is of an invalid type.

159.11.6.6 public FeatureBuilder addVariables(Map<String, Object> variables)

variables to be added.

- Add a map of variables to the Feature. Pre-existing variables with the same key in are overwritten if these keys exist in the map. Variable values are of type: String, Boolean or BigDecimal for numbers.

Returns This builder.

Throws IllegalArgumentException– if a value is of an invalid type.

159.11.6.7 public Feature build()

- Build the Feature. Can only be called once on a builder. After calling this method the current builder instance cannot be used any more.

Returns The Feature.

159.11.6.8 public FeatureBuilder setComplete(boolean complete)

complete If the feature is complete.

- Set the Feature Complete flag. If this method is not called the complete flag defaults to false.

Returns This builder.

159.11.6.9 public FeatureBuilder setDescription(String description)

description The description.

- Set the Feature Description.

Returns This builder.

159.11.6.10 public FeatureBuilder setDocURL(String docURL)

docURL The Documentation URL.

- Set the documentation URL.

Returns This builder.

159.11.6.11 public FeatureBuilder setLicense(String license)

license The License.

- Set the License.

Returns This builder.

159.11.6.12 public FeatureBuilder setName(String name)

name The Name.

- Set the Feature Name.

Returns This builder.

159.11.6.13 **public FeatureBuilder setSCM(String scm)**

scm The SCM information.

- Set the SCM information.

Returns This builder.

159.11.6.14 **public FeatureBuilder setVendor(String vendor)**

vendor The Vendor.

- Set the Vendor.

Returns This builder.

159.11.7 **public interface FeatureBundle**

A Bundle which is part of a feature.

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

159.11.7.1 **public ID getID()**

- Get the bundle's ID.

Returns The ID of this bundle.

159.11.7.2 **public Map<String, Object> getMetadata()**

- Get the metadata for this bundle.

Returns The metadata. The returned map is unmodifiable.

159.11.8 **public interface FeatureBundleBuilder**

A builder for Feature Model FeatureBundle objects.

Concurrency Not Thread-safe

Provider Type Consumers of this API must not implement this type

159.11.8.1 **public FeatureBundleBuilder addMetadata(String key, Object value)**

key Metadata key.

value Metadata value.

- Add metadata for this Bundle.

Returns This builder.

159.11.8.2 **public FeatureBundleBuilder addMetadata(Map<String, Object> metadata)**

metadata The map with metadata.

- Add metadata for this Bundle by providing a map. All metadata in the map is added to any previously provided metadata.

Returns This builder.

159.11.8.3 **public FeatureBundle build()**

- Build the Bundle object. Can only be called once on a builder. After calling this method the current builder instance cannot be used any more.

Returns The Bundle.

159.11.9 public interface FeatureConfiguration

Represents an OSGi Configuration in the Feature Model.

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

159.11.9.1 public Optional<String> getFactoryPid()

- Get the Factory PID from the configuration, if any.

Returns The Factory PID, or null if there is none.

159.11.9.2 public String getPid()

- Get the PID from the configuration.

Returns The PID.

159.11.9.3 public Map<String, Object> getValues()

- Get the configuration key-value map.

Returns The key-value map. The returned map is unmodifiable.

159.11.10 public interface FeatureConfigurationBuilder

A builder for Feature Model FeatureConfiguration objects.

Concurrency Not Thread-safe

Provider Type Consumers of this API must not implement this type

159.11.10.1 public FeatureConfigurationBuilder addValue(String key, Object value)

key The configuration key.

value The configuration value. Acceptable data types are the data type supported by the Configuration Admin service, which are the Primary Property Types as defined for the Filter Syntax in the OSGi Core specification.

- Add a configuration value for this Configuration object. If a value with the same key was previously provided (regardless of case) the previous value is overwritten.

Returns This builder.

Throws IllegalArgumentException– if the value is of an invalid type.

159.11.10.2 public FeatureConfigurationBuilder addValues(Map<String, Object> configValues)

configValues The map of configuration values to add. Acceptable value types are the data type supported by the Configuration Admin service, which are the Primary Property Types as defined for the Filter Syntax in the OSGi Core specification.

- Add a map of configuration values for this Configuration object. Values will be added to any previously provided configuration values. If a value with the same key was previously provided (regardless of case) the previous value is overwritten.

Returns This builder.

Throws IllegalArgumentException– if a value is of an invalid type or if the same key is provided in different capitalizations (regardless of case).

159.11.10.3 public FeatureConfiguration build()

- Build the Configuration object. Can only be called once on a builder. After calling this method the current builder instance cannot be used any more.

Returns The Configuration.

159.11.11 **public final class FeatureConstants**

Defines standard constants for the Feature specification.

159.11.11.1 **public static final String FEATURE_IMPLEMENTATION = "osgi.feature"**

The name of the implementation capability for the Feature specification.

159.11.11.2 **public static final String FEATURE_SPECIFICATION_VERSION = "1.0"**

The version of the implementation capability for the Feature specification.

159.11.12 **public interface FeatureExtension**

A Feature Model Extension. Extensions can contain either Text, JSON or a list of Artifacts.

Extensions are of one of the following kinds:

- **Mandatory:** this extension must be processed by the runtime
- **Optional:** this extension does not have to be processed by the runtime
- **Transient:** this extension contains transient information such as caching data that is for optimization purposes. It may be changed or removed and is not part of the feature's identity.

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

159.11.12.1 **public List<FeatureArtifact> getArtifacts()**

- Get the Artifacts from this extension.

Returns The Artifacts. The returned list is unmodifiable.

Throws `IllegalStateException`— If called on an extension which is not of type `ARTIFACTS`.

159.11.12.2 **public String getJSON()**

- Get the JSON from this extension.

Returns The JSON.

Throws `IllegalStateException`— If called on an extension which is not of type `JSON`.

159.11.12.3 **public FeatureExtension.Kind getKind()**

- Get the extension kind.

Returns The kind.

159.11.12.4 **public String getName()**

- Get the extension name.

Returns The name.

159.11.12.5 **public List<String> getText()**

- Get the Text from this extension.

Returns The lines of text. The returned list is unmodifiable.

Throws `IllegalStateException`— If called on an extension which is not of type `TEXT`.

159.11.12.6 **public FeatureExtension.Type getType()**

- Get the extension type.

Returns The type.

159.11.13 **enum FeatureExtension.Kind**

The kind of extension: optional, mandatory or transient.

159.11.13.1 **MANDATORY**

A mandatory extension must be processed.

159.11.13.2 **OPTIONAL**

An optional extension can be ignored if no processor is found.

159.11.13.3 **TRANSIENT**

A transient extension contains computed information which can be used as a cache to speed up operation.

159.11.13.4 **public static FeatureExtension.Kind valueOf(String name)**

159.11.13.5 **public static FeatureExtension.Kind[] values()**

159.11.14 **enum FeatureExtension.Type**

The type of extension.

159.11.14.1 **JSON**

A JSON extension.

159.11.14.2 **TEXT**

A plain text extension.

159.11.14.3 **ARTIFACTS**

An extension that is a list of artifact identifiers.

159.11.14.4 **public static FeatureExtension.Type valueOf(String name)**

159.11.14.5 **public static FeatureExtension.Type[] values()**

159.11.15 **public interface FeatureExtensionBuilder**

A builder for Feature Model FeatureExtension objects.

Concurrency Not Thread-safe

Provider Type Consumers of this API must not implement this type

159.11.15.1 **public FeatureExtensionBuilder addArtifact(FeatureArtifact artifact)**

artifact The artifact to add.

- Add an Artifact to the extension. Can only be called for extensions of type FeatureExtension.Type.ARTIFACTS.

Returns This builder.

159.11.15.2 **public FeatureExtensionBuilder addText(String text)**

text The text to be added.

- Add a line of text to the extension. Can only be called for extensions of type `FeatureExtension.Type.TEXT`.

Returns This builder.

159.11.15.3 **public FeatureExtension build()**

- Build the Extension. Can only be called once on a builder. After calling this method the current builder instance cannot be used any more.

Returns The Extension.

159.11.15.4 **public FeatureExtensionBuilder setJSON(String json)**

json The JSON to be added.

- Add JSON in String form to the extension. Can only be called for extensions of type `FeatureExtension.Type.JSON`.

Returns This builder.

159.11.16 **public interface FeatureService**

The Feature service is the primary entry point for interacting with the feature model.

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

159.11.16.1 **public BuilderFactory getBuilderFactory()**

- Get a factory which can be used to build feature model entities.

Returns A builder factory.

159.11.16.2 **public ID getID(String groupId, String artifactId, String version)**

groupId The group ID (not null, not empty).

artifactId The artifact ID (not null, not empty).

version The version (not null, not empty).

- Obtain an ID.

Returns The ID.

159.11.16.3 **public ID getID(String groupId, String artifactId, String version, String type)**

groupId The group ID (not null, not empty).

artifactId The artifact ID (not null, not empty).

version The version (not null, not empty).

type The type (not null, not empty).

- Obtain an ID.

Returns The ID.

159.11.16.4 **public ID getID(String groupId, String artifactId, String version, String type, String classifier)**

groupId The group ID (not null, not empty).

artifactId The artifact ID (not null, not empty).

version The version (not null, not empty).

type The type (not null, not empty).

classifier The classifier (not null, not empty).

- Obtain an ID.

Returns The ID.

159.11.16.5 **public ID getIdFromMavenCoordinates(String coordinates)**

coordinates The Maven Coordinates.

- Obtain an ID from a Maven Coordinates formatted string. The supported syntax is as follows:
groupId ':' artifactId (':' type (':' classifier)?)? ':' version

Returns the ID.

159.11.16.6 **public Feature readFeature(Reader jsonReader) throws IOException**

jsonReader A Reader to the JSON input

- Read a Feature from JSON

Returns The Feature represented by the JSON

Throws IOException— When reading fails

159.11.16.7 **public void writeFeature(Feature feature, Writer jsonWriter) throws IOException**

feature the Feature to write.

jsonWriter A Writer to which the Feature should be written.

- Write a Feature Model to JSON

Throws IOException— When writing fails.

159.11.17 **public interface ID**

ID used to denote an artifact. This could be a feature model, a bundle which is part of the feature model or some other artifact.

Artifact IDs follow the Maven convention of having:

- A group ID
- An artifact ID
- A version
- A type identifier (optional)
- A classifier (optional)

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

159.11.17.1 **public static final String FEATURE_ID_TYPE = "osgifeature"**

ID type for use with Features.

159.11.17.2 **public String getArtifactId()**

- Get the artifact ID.

Returns The artifact ID.

159.11.17.3 **public Optional<String> getClassifier()**

- Get the classifier.

Returns The classifier.

159.11.17.4 public String getGroupId()

- Get the group ID.

Returns The group ID.

159.11.17.5 public Optional<String> getType()

- Get the type identifier.

Returns The type identifier.

159.11.17.6 public String getVersion()

- Get the version.

Returns The version.

159.11.17.7 public String toString()

- This method returns the ID using the following syntax:
groupId ':' artifactId (':' type (':' classifier)?)? ':' version

Returns The string representation.

159.12 org.osgi.service.feature.annotation

Feature Annotations Package Version 1.0.

This package contains annotations that can be used to require the Feature Service implementation.

Bundles should not normally need to import this package as the annotations are only used at build-time.

159.12.1 Summary

- `RequireFeatureService` - This annotation can be used to require the Feature implementation.

159.12.2 @RequireFeatureService

This annotation can be used to require the Feature implementation. It can be used directly, or as a meta-annotation.

Retention CLASS

Target TYPE, PACKAGE

159.13 References

- [1] *JSON (JavaScript Object Notation)*
<https://www.json.org>
- [2] *JSMIn (The JavaScript Minifier)*
<https://www.crockford.com/javascript/jsmin.html>
- [3] *Apache Maven Pom Reference*
<https://maven.apache.org/pom.html>

160 Feature Launcher Service Specification

Version 1.0

160.1 Introduction

The *Feature Service Specification* on page 5 defines a model to design and declare Complex Applications and reusable Sub-Components that are composed of multiple bundles, configurations and other metadata. These models are, however, only descriptive and have no standard mechanism for installing them into an OSGi framework.

This specification focuses on turning these Features into a running system, by introducing the Feature Launcher and Feature Runtime. The Feature Launcher takes a Feature definition, obtains a framework instance for it and then starts the Feature in that environment. The Feature Runtime extends this capability to a running system, enabling one or more Features to be installed, updated, and later removed from a running OSGi framework.

The Launcher and Runtime also interact with the Configuration Admin Service, that is, they provide configuration to the system if it is present in the Feature being launched or installed.

160.1.1 Essentials

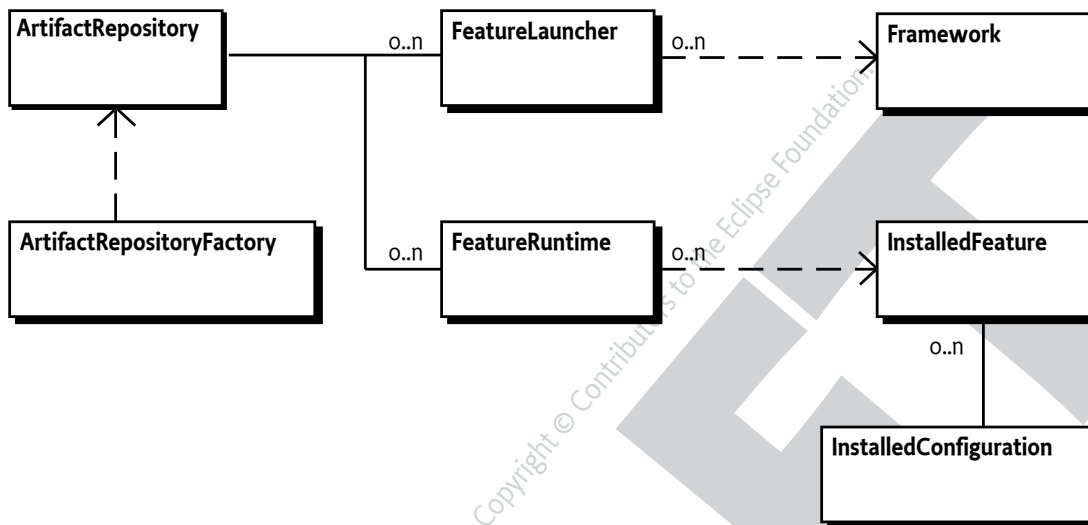
- *Dynamic* - The Feature Runtime dynamically adds, updates and removes Features in a running system.
- *Parameterizable* - Feature installation may be customised using local parameters if the Feature supports it.
- *Zero code* - The Feature Launcher can launch a framework containing an installed Feature in an implementation independent way without a user writing any code .

160.1.2 Entities

The following entities are used in this specification:

- *Feature* - A Feature as defined by the *Feature Service Specification* on page 5
- *Artifact Repository* - A means of accessing the installable bytes for bundles in a Feature
- *Feature Launcher* - A Feature Launcher obtains an OSGi Framework instance and installs a Feature into it.
- *Framework* - A running implementation of the OSGi core specification.
- *Launch Properties* - Framework launching properties defined in a Feature.
- *Feature Parameters* - Key value pairs that can be used to customise the installation of a Feature.
- *Configuration* - A configuration for the Configuration Admin service.
- *Feature Runtime* - A Feature Runtime is an OSGi service capable of installing Features into the running OSGi framework, removing installed Features from the OSGi framework, and updating an installed Feature with a new Feature definition.
- *Installed Feature* - A representation of a Feature installed by the Feature Runtime.
- *Installed Configuration* - A representation of a Configuration installed by the Feature Runtime.

Figure 160.1 Features Entity overview



160.2 Features and Artifact Repositories

OSGi Features exist either as JSON documents, or as runtime objects created by the Feature Service API. The primary purpose of a Feature is to define a list of bundles and configurations that should be installed, however the Feature provides no information about the location of the bundle artifacts. A key challenge with installing a Feature is therefore finding the appropriate artifacts to install.

The [ArtifactRepository](#) interface is designed to be implemented by users of the Feature Launcher Service to provide a way for the Feature Launcher Service to find an installable `InputStream` of bytes for a given bundle artifact using the `getArtifact(ID)` method. Artifact Repository implementations are free to use any mechanism for locating the bundle artifact data. If no artifact can be found for the supplied ID then the implementation of the Artifact Repository should return null. If the Artifact Repository throws an exception then this must be logged by the Feature Launcher Service and then treated in the same manner as a null return value.

160.2.1 The Artifact Repository Factory

In order to support the *Zero Code* objective of this specification, and to simplify usage for most users, the [ArtifactRepositoryFactory](#) provides a factory for commonly used repository types.

160.2.1.1 Obtaining an Artifact Repository Factory

The Artifact Repository Factory is useful both for the Feature Launcher and the Feature Runtime, and as such it must be easy to access both inside and outside an OSGi framework. The Feature Launcher Service implementation must provide an implementation of the Artifact Repository Factory interface. A user of the Artifact Repository Factory service may use the following ways to find an instance.

When outside OSGi:

- Using the Java ServiceLoader API to find instances of `org.osgi.service.featurelauncher.ArtifactRepositoryFactory`
- From configuration, and then using `Class.forName()`, `getConstructor()` and `newInstance()`

- By hard coding the implementation and using the new operator.

When inside an OSGi framework:

- Using the OSGi service registry to find instances of `org.osgi.service.featurelauncher.ArtifactRepositoryFactory`
- Using the Java ServiceLoader API and the OSGi Service Loader Mediator to find instances of `org.osgi.service.featurelauncher.ArtifactRepositoryFactory`
- By hard coding the implementation type and using the new operator.

160.2.1.2

Local Repositories

A Local Repository is one that exists on a locally accessible file system. Note that this does not require that the file system is local, and technologies such as NFS or other network file systems would still be considered as Local Repositories. The key aspects of a Local Repository are that:

- The root of the repository can be accessed and resolved as a `java.nio.file.Path` or `file: URI`.
- The repository uses the Maven2 Repository Layout
Add bibliography link to <https://maven.apache.org/repository/layout.html#maven2-repository-layout>

An Artifact Repository representing a Local Repository can be created using the `createRepository(Path)` method, passing in the path to the root of the repository. A `NullPointerException` must be thrown if the path is null and an `IllegalArgumentException` must be thrown if the path does not exist, or represents a file which is not a directory.

An Artifact Repository representing a Local Repository can also be created using the `createRepository(URI,Map)` method, passing a URI using the file scheme which points to the root of the repository. A `NullPointerException` must be thrown if the URI is null and an `IllegalArgumentException` must be thrown if the path does not exist, or represents a file which is not a directory.

Once created this Artifact Repository will search the supplied repository for any requested artifact data. Implementations are free to optimise checks using repository metadata.

160.2.1.3

Remote Repositories

A Remote Repository is one that exists with an accessible `http` or `https` endpoint for retrieving artifact data. Note that this does not require that the repository is on a remote machine, only that the means of accessing data is via HTTP requests. The key aspects of a Remote Repository are that:

- The root of the repository can be accessed and resolved as a `http` or `https` URI
- The repository uses the Maven2 Repository Layout
Add bibliography link to <https://maven.apache.org/repository/layout.html#maven2-repository-layout>

An Artifact Repository representing a Remote Repository can be created using the `createRepository(URI,Map)` method, passing in the uri to the root of the repository. A `NullPointerException` must be thrown if the uri is null and an `IllegalArgumentException` must be thrown if the uri does not use the `http` or `https` scheme.

In addition to the repository URI the user may pass configuration properties in a `Map`. Implementations may support custom configuration properties, but those properties should use Reverse Domain Name keys. Keys not using the reverse DNS naming scheme are reserved for OSGi use. Implementations must ignore any configuration property keys that they do not recognise. All implementations must support the following properties:

- `REMOTE_ARTIFACT_REPOSITORY_NAME` - The name for this repository
- `REMOTE_ARTIFACT_REPOSITORY_USER` - The user name to use for authenticating with this repository

- `REMOTE_ARTIFACT_REPOSITORY_PASSWORD` - The password to use for authenticating with this repository
- `REMOTE_ARTIFACT_REPOSITORY_BEARER_TOKEN` - A bearer token to use when authenticating with this repository
- `REMOTE_ARTIFACT_REPOSITORY_SNAPSHOTS_ENABLED` - A Boolean indicating that SNAPSHOT versions are supported. Defaults to true
- `REMOTE_ARTIFACT_REPOSITORY_RELEASES_ENABLED` - A Boolean indicating that release versions are supported. Defaults to true
- `REMOTE_ARTIFACT_REPOSITORY_TRUST_STORE` - A trust store to use when validating a server certificate. May be a file system path or a data URI.
Add bibliography link to https://en.wikipedia.org/wiki/Data_URI_scheme
- `REMOTE_ARTIFACT_REPOSITORY_TRUST_STORE_FORMAT` - The format of the trust store to use when validating a server certificate.
- `REMOTE_ARTIFACT_REPOSITORY_TRUST_STORE_PASSWORD` - The password to use when validating the trust store integrity.

Once created this Artifact Repository will search the supplied repository for any requested artifact data. Implementations are free to optimise checks using repository metadata.

160.3 The Feature Launcher

The `FeatureLauncher` is the main entry point for creating a running OSGi framework containing the bundles and configurations defined in a Feature. As such the Feature Launcher is primarily designed for use outside of an OSGi framework.

To support usage in a non-OSGi environment implementations of the Feature Launcher Service must register the following implementation classes with the Java ServiceLoader API, and any necessary module metadata.

- `org.osgi.service.featurelauncher.FeatureLauncher`
- `org.osgi.service.featurelauncher.ArtifactRepositoryFactory`

160.3.1 Obtaining and configuring a Feature Launcher

A Feature Launcher Service implementation must provide an implementation of the Feature Launcher interface. A user of the Feature Launcher service may use the following ways to find this class and create an instance:

- Using the Java ServiceLoader API to find instances of `org.osgi.service.featurelauncher.FeatureLauncher`
- From configuration, and then using `Class.forName()`, `getConstructor()` and `newInstance()`
- By hard coding the implementation type and using the new operator.

Once instantiated the `FeatureLauncher` may be configured in a fluent manner using the `configure(Map)` and `addRepository(ArtifactRepository)` methods. Configuration properties for the Feature Launcher are implementation specific, and any unrecognised property names should be ignored. Artifact Repository instances may be created by the user using as described in [The Artifact Repository Factory on page 28](#), or using custom implementations.

160.3.2 Using a Feature Launcher

Once a configured Feature Launcher instance has been created, one of the launch methods can be used to launch an OSGi framework containing the supplied Feature. Features can either be supplied as a Reader providing access to the JSON text of a Feature document or a parsed `Feature`. The Feature

Launcher will then return a running Framework instance representing the launched OSGi framework and the Feature that it contains. If an error occurs creating the framework, or locating and installing any of the feature bundles, then a [LaunchException](#) must be thrown.

160.3.2.1 Overriding Feature variables

Some Feature definitions include variables which can be used to customise their deployment. These variables are intended to be set at the point where a Feature is installed, and may contain default values. To enable these variables to be overridden there are launch methods which permit a Map of variables to be provided. The values in this map must be one of the types permitted by the *Feature Service Specification* on page 5

If a Feature declares a variable with no default value then this variable *must* be provided to the Feature Launcher. If no value is provided then the Feature Launcher must fail to launch by throwing a [LaunchException](#)

160.3.2.2 Providing Framework Launch Properties

Framework launch properties are key value pairs which are passed to the OSGi framework as it is created. They can control many behaviours, including operations which happen before the framework starts, meaning that is not always possible to set them *after* startup.

Feature definitions that require particular framework launch properties can define them using a Feature Extension named [FRAMEWORK_LAUNCHING_PROPERTIES](#). The Type of this Feature Extension must be TEXT, where each entry is in the form `key=value`. All implementations of the Feature Launcher must support this extension, and use it to populate the Framework Launch Properties.

In addition to Framework Launch properties defined inside the Feature, users of the Feature Launcher can add and override Framework Launch Properties using one of the launch methods which permit a Map of framework properties to be provided. Any key value pairs defined in this map must take precedence over those defined in the Feature. A key with a null value must result in the removal of a key value pair if it is defined in the Feature.

160.3.2.3 Selecting a framework implementation

When defining a feature it is not always possible to be framework independent. Sometimes specific framework APIs, or licensing restrictions, will require that a particular implementation is used. In this case a Feature Extension named [LAUNCH_FRAMEWORK](#) with Type ARTIFACTS can be used to list one or more artifacts representing OSGi framework implementations.

The list of artifacts is treated as a preference order, with the first listed artifact being used if available, and so on, until a framework is found. If a listed artifact is not an OSGi framework implementation then the Feature Launcher must log a warning and continue on to the next artifact in the list. If the Kind of the feature is MANDATORY and none of the listed artifacts are available then launching must fail with a [LaunchException](#).

160.3.3 The Feature Launching Process

The following section defines the process through which the Feature Launcher must locate, initialize and populate an OSGi framework when launching a feature. Unless explicitly stated implementations may perform one or more parts of this process in a different order to that described in the specification.

160.3.3.1 Locating a framework implementation

Before a framework instance can be created the Feature Launcher must identify a suitable implementation using the following search order:

1. If any provider specific configuration has been given to the Feature Launcher implementation then this should be used to identify the framework.

2. If the Feature declares an Extension `LAUNCH_FRAMEWORK` then the Feature Launcher implementation must use the first listed artifact that can be found in any configured Artifact Repositories, as described in [Selecting a framework implementation on page 31](#).
Currently this only fails if the extension is mandatory
3. If no framework implementation is found in the previous steps then the Feature Launcher implementation must search the classpath using the Thread Context Class Loader, or, if the Thread Context Class Loader is not set, the Class Loader which loaded the caller of the Feature Launcher's launch method. The first suitable framework instance located is the instance that will be used.
4. In the event that no suitable OSGi framework can be found by any of the previous steps then the Feature Launcher implementation may provide a default framework implementation to be used.

If no suitable OSGi framework implementation can be found then the Feature Launcher implementation must throw a `LaunchException`.

160.3.3.2

Creating a Framework instance

Once a suitable framework implementation has been located the Feature Launcher implementation must create and initialize a framework instance. Implementations are free to use implementation specific mechanisms for framework implementations that they recognise. The result of this initialization must be the same as if the Feature Launcher used the `org.osgi.framework.launch.FrameworkFactory` registered by the framework implementation to create the framework instance.

When creating the framework any framework launch properties defined in the Feature must be used. These are defined as described in [Providing Framework Launch Properties on page 31](#) and must include any necessary variable replacement as defined by [Overriding Feature variables on page 31](#).

Once instantiated the framework must be initialised appropriately so that it has a valid `BundleContext`. Once initialised the framework is ready for the Feature Launcher implementation to begin populating the framework.

160.3.3.3

Installing bundles and configurations

TODO

160.4 org.osgi.service.featurelauncher

Feature Launcher Package Version 1.0.

Bundles wishing to use this package must list the package in the `Import-Package` header of the bundle's manifest. This package has two types of users: the consumers that use the API in this package and the providers that implement the API in this package.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.service.feature; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.feature; version="[1.0,1.1)"
```

160.4.1

Summary

- `ArtifactRepository` - An `ArtifactRepository` is used to get hold of the bytes used to install an artifact.

- **ArtifactRepositoryFactory** - A **ArtifactRepositoryFactory** is used to create implementations of **ArtifactRepository** for one of the built in repository types:
 - Local File System
 - HTTP repository
- **FeatureLauncher** - The Feature launcher is the primary entry point for launching an OSGi framework and set of bundles.
- **FeatureLauncherConstants** - Defines standard constants for the Feature Launcher specification.
- **LaunchException** - A **LaunchException** is thrown by the **FeatureLauncher** if it is unable to:
 - Locate or start an OSGi Framework instance
 - Locate the installable bytes of any bundle in a **Feature**
 - Install a bundle in the **Feature**
 - Determine a value for a **Feature** variable that has no default value defined

160.4.2 **public interface ArtifactRepository**

An **ArtifactRepository** is used to get hold of the bytes used to install an artifact. Users of this specification may provide their own implementations for use when installing feature artifacts. Instances must be Thread Safe.

Concurrency Thread-safe

160.4.2.1 **public InputStream getArtifact(ID id)**

id the id of the artifact

- Get a stream to the bytes of an artifact

Returns an **InputStream** containing the bytes of the artifact or null if this repository does not have access to the bytes

160.4.3 **public interface ArtifactRepositoryFactory**

A **ArtifactRepositoryFactory** is used to create implementations of **ArtifactRepository** for one of the built in repository types:

- Local File System
- HTTP repository

Provider Type Consumers of this API must not implement this type

160.4.3.1 **public ArtifactRepository createRepository(Path path)**

path a path to the root of a Maven Repository Layout containing installable artifacts

- Create an **ArtifactRepository** using the local file system

Returns an **ArtifactRepository** using the local file system

Throws **IllegalArgumentException**– if the path does not exist, or exists and is not a directory

NullPointerException– if the path is null

160.4.3.2 **public ArtifactRepository createRepository(URI uri, Map<String, Object> props)**

uri the URI for the repository. The http, https and file schemes must be supported by all implementations.

props the configuration properties for the remote repository. See **FeatureLauncherConstants** for standard property names

- Create an **ArtifactRepository** using a remote Maven repository.

Returns an ArtifactRepository using the local file system

Throws IllegalArgumentException– if the uri scheme is not supported by this implementation
 NullPointerException– if the path is null

160.4.4 **public interface FeatureLauncher extends ArtifactRepositoryFactory**

The Feature launcher is the primary entry point for launching an OSGi framework and set of bundles. As it is a means for launching a framework it is designed to be used from outside OSGi and therefore should be obtained using the ServiceLoader.

Provider Type Consumers of this API must not implement this type

160.4.4.1 **public FeatureLauncher addRepository(ArtifactRepository repository)**

repository the repository to add

- Add a repository to this FeatureLauncher that will be used to locate installable artifact data.

Returns this

160.4.4.2 **public FeatureLauncher configure(Map<String, Object> configuration)**

configuration the repository to add

- Configure this FeatureLauncher with the supplied properties.

Returns this

160.4.4.3 **public Framework launch(Feature feature)**

feature the feature to launch

- Launch a framework instance based on the supplied feature

Returns A running framework instance.

Throws LaunchException–

160.4.4.4 **public Framework launch(Reader jsonReader)**

jsonReader a Reader for the input Feature JSON

- Launch a framework instance based on the supplied feature JSON

Returns A running framework instance.

Throws LaunchException–

160.4.4.5 **public Framework launch(Feature feature, Map<String, Object> variables)**

feature the feature to launch

variables key/value pairs to set variables in the feature

- Launch a framework instance based on the supplied feature and variables

Returns A running framework instance.

Throws LaunchException–

IllegalArgumentException– if any of the variable values is not one of the values defined for Feature.getVariables():

- java.lang.String
- java.lang.Boolean

- java.math.BigDecimal

160.4.4.6 **public Framework launch(Reader jsonReader, Map<String, Object> variables)**

jsonReader a Reader for the input Feature JSON

variables key/value pairs to set variables in the feature

- Launch a framework instance based on the supplied feature JSON and variables

Returns A running framework instance.

Throws LaunchException– if an error occurs in launching

IllegalArgumentException– if any of the variable values is not one of the values defined for Feature.getVariables():

- java.lang.String
- java.lang.Boolean
- java.math.BigDecimal

160.4.4.7 **public Framework launch(Framework feature, Map<String, Object> variables, Map<String, String> frameworkProperties)**

feature the feature to launch

variables key/value pairs to set variables in the feature

frameworkProperties set or override framework properties when launching the framework

- Launch a framework instance based on the supplied feature, variables and framework properties

Returns A running framework instance.

Throws LaunchException–

IllegalArgumentException– if any of the variable values is not one of the values defined for Feature.getVariables():

- java.lang.String
- java.lang.Boolean
- java.math.BigDecimal

160.4.4.8 **public Framework launch(Reader jsonReader, Map<String, Object> variables, Map<String, String> frameworkProperties)**

jsonReader a Reader for the input Feature JSON

variables key/value pairs to set variables in the feature

frameworkProperties set or override framework properties when launching the framework

- Launch a framework instance based on the supplied feature JSON

Returns A running framework instance.

Throws LaunchException–

IllegalArgumentException– if any of the variable values is not one of the values defined for Feature.getVariables():

- java.lang.String
- java.lang.Boolean
- java.math.BigDecimal

160.4.5 public final class FeatureLauncherConstants

Defines standard constants for the Feature Launcher specification.

160.4.5.1 public static final String FEATURE_LAUNCHER_IMPLEMENTATION = "osgi.featurelauncher"

The name of the implementation capability for the Feature specification.

160.4.5.2 public static final String FEATURE_LAUNCHER_SPECIFICATION_VERSION = "1.0"

The version of the implementation capability for the Feature specification.

160.4.5.3 public static final String FRAMEWORK_LAUNCHING_PROPERTIES = "framework-launching-properties"

The name for the FeatureExtension which defines the framework properties that should be used when launching the feature.

160.4.5.4 public static final String LAUNCH_FRAMEWORK = "launch-framework"

The name for the FeatureExtension which defines the framework that should be used to launch the feature. The extension must be of Type.ARTIFACTS and contain one or more ID entries corresponding to OSGi framework implementations. This extension must be processed even if it is Kind.OPTIONAL or Kind.TRANSIENT.

If more than one framework entry is provided then the list will be used as a priority order when determining the framework implementation to use. If none of the frameworks are present then an error is raised and launching will be aborted.

160.4.5.5 public static final String REMOTE_ARTIFACT_REPOSITORY_BEARER_TOKEN = "token"

The configuration property key used to set the bearer token when creating an ArtifactRepository using FeatureLauncher.createRepository(URI, Map)

160.4.5.6 public static final String REMOTE_ARTIFACT_REPOSITORY_NAME = "name"

The configuration property key used to set the repository name when creating an ArtifactRepository using FeatureLauncher.createRepository(URI, Map)

160.4.5.7 public static final String REMOTE_ARTIFACT_REPOSITORY_PASSWORD = "password"

The configuration property key used to set the repository password when creating an ArtifactRepository using FeatureLauncher.createRepository(URI, Map)

160.4.5.8 public static final String REMOTE_ARTIFACT_REPOSITORY_RELEASES_ENABLED = "release"

The configuration property key used to set that release versions are enabled for an ArtifactRepository using FeatureLauncher.createRepository(URI, Map)

160.4.5.9 public static final String REMOTE_ARTIFACT_REPOSITORY_SNAPSHOTS_ENABLED = "snapshot"

The configuration property key used to set that SNAPSHOT release versions are enabled for an ArtifactRepository using FeatureLauncher.createRepository(URI, Map)

160.4.5.10 public static final String REMOTE_ARTIFACT_REPOSITORY_TRUST_STORE = "truststore"

The configuration property key used to set the trust store to be used when accessing a remote ArtifactRepository using FeatureLauncher.createRepository(URI, Map)

160.4.5.11 public static final String REMOTE_ARTIFACT_REPOSITORY_TRUST_STORE_FORMAT = "truststoreFormat"

The configuration property key used to set the trust store format to be used when accessing a remote ArtifactRepository using FeatureLauncher.createRepository(URI, Map)

160.4.5.12 **public static final String REMOTE_ARTIFACT_REPOSITORY_TRUST_STORE_PASSWORD = "truststorePassword"**

The configuration property key used to set the trust store password to be used when accessing a remote ArtifactRepository using FeatureLauncher.createRepository(URI, Map)

160.4.5.13 **public static final String REMOTE_ARTIFACT_REPOSITORY_USER = "user"**

The configuration property key used to set the repository user when creating an ArtifactRepository using FeatureLauncher.createRepository(URI, Map)

160.4.6 **public class LaunchException** **extends RuntimeException**

A LaunchException is thrown by the FeatureLauncher if it is unable to:

- Locate or start an OSGi Framework instance
- Locate the installable bytes of any bundle in a Feature
- Install a bundle in the Feature
- Determine a value for a Feature variable that has no default value defined

160.4.6.1 **public LaunchException(String message)**

message

- Create a LaunchException with the supplied error message

160.4.6.2 **public LaunchException(String message, Throwable cause)**

message

cause

- Create a LaunchException with the supplied error message and cause

Licensed under the Eclipse Foundation Specification License – V1.0. Copyright © Contributors to the Eclipse Foundation.

DRAFT

Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

DRAFT

Licensed under the Eclipse Foundation Specification License – V1.0. Copyright © Contributors to the Eclipse Foundation.

DRAFT

End Of Document