# 160 Feature Launcher Service Specification

*Version 1.0*

## 160.1 Introduction

The *Feature Service Specification* on page 1375 defines a model to design and declare Complex Applications and reuable Sub-Components that are composed of multiple bundles, configurations and other metadata.

This specification focuses on turning these Features into a running system, by introducing the Feature launcher. The launcher takes a Feature definition, obtains a runtime environment for it and then starts the Feature in that environment.

The launcher also interacts with the Configuration Admin Service, that is, it provides configuration to the system if present in the Feature.

## 160.2 Launching a Feature

To launch a Feature, the launcher must find or create a target environment for the Feature first. For example it can launch an OSGi framework that the Feature should run in.

The launcher should deploy all the bundles referenced by the Feature in this Framework. It must first install all bundles, then resolve them and finally start all the bundles. The order in which this happens between the bundles is not defined. ### Introduce start order in metadata

Once all bundles are started and all bundle fragments resolved and attached the launcher should provide the specified configurations to the Configuration Admin Service.

A Feature launcher can be obained using the `LauncherFactory` service. This service can be obtained from the Service registry if running in an OSGi Framework or using the ServiceLoader mechanism otherwise.

```
ServiceLoader<LauncherFactory> sl =
  ServiceLoader.load(LauncherFactory.class);

LauncherFactory factory = sl.iterator().next();
Launcher launcher = factory.newLauncher(
  new URL("file:///home/david/myfeature.json"),
  Collections.emptyMap());
launcher.start();

launcher.waitForStop(0); // Start is asynchronous
```

If a Feature can't be launched `waitForStop()` will throw a `LauncherException`.

## 160.3 Handling Bundles

All bundles listed in the Feature will first be installed, then resolved and finally started in the Framework chosen by the launcher.

Bundle fragments are installed and resolved and attached to their host(s).

If a Bundle cannot resolve or start a `LauncherException` must be thrown.

## 160.4 Handling Configuration

If configuration is found in the Feature then it is passed to the Configuration Admin service. If a Feature contains a `configuration` section but the Configuration Admin service is not found in the running system, the launcher will abort with an `LauncherException`.

## 160.5 Specifying Framework Properties

Framework Launching Properties can be provided in the Feature through the `framework-launching-properties` extension. The launcher must ensure that the Framework it provides for the feature has these properties set. If it cannot provide a Framework with the requested Framework properties set it must fail with a `LauncherException`.

For example, to ensure the `org.osgi.framework.bsnversion` Framework property is set for the Feature, specify the following in the Feature:

```
"extensions": {
  "framework-launching-properties": {
    "type": "json",
    "kind": "mandatory",
    "json": {
       "org.osgi.framework.bsnversion": "multiple"
    }
  }
}
```

## 160.6 Specifying Runtime Preconditions

A Feature can specify the preconditions it places on its runtime environment. That is, the Framework used to run the Feature in, must satisfy these constraints. If the Launcher cannot provide a Framework with the specified conditions, it must fail.

Preconditions are specified as requirements in the Feature. If no preconditions are specified, the Launcher is free to choose a Java and OSGi implementation of its choice.

For example:

```
"requirements": [
  {
    "namespace": "osgi.ee",
    "filter":    "(&(osgi.ee=JavaSE)(version=11))"
  }, {
    "namespace": "osgi.wiring.package",
```

```
      "filter":     "(&(osgi.wiring.package=org.osgi.framework)(version=1.10))"
    }
  ]
```

## 160.7 Specifying Variables

Variables allow for late binding of configuration values and Framework properties. Variables are provided through the LauncherFactory:

```
Map<String,Object> variables = new HashMap<>();
variables.put("user.name", "scott");
variables.put("db.driver", "postgresql");

LauncherFactory factory = ... // From Service Registry or Service Loader
Launcher launcher = factory.newLauncher(
  new URL("https://repo.maven.apache.org/maven2/org/foo/Bar/1.0.0/Bar-1.0.0.osgifeature"),
  variables);

launcher.start();
```

## 160.8 Specifying Extension Handlers

### TODO

## 160.9 Specifying Post-processors

### TODO

## 160.10 org.osgi.service.feature.launcher

Feature Launcher Package 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.launcher; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

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

### 160.10.1 Summary

- Launcher - A launcher can launch a Feature model into a running system.
- LauncherConstants - Defines standard constants for the Feature Launcher specification.
- LauncherException - Exception thrown when the launcher isn't able to launch the Feature.
- LauncherFactory - Create a Feature Launcher.

## 160.10.2 public interface Launcher

A launcher can launch a Feature model into a running system.

### 160.10.2.1 public Framework start()

- Start the launcher. This method is asynchronous and will return as soon as the launching has been initiated.

*Returns* The Framework the Feature is launched into.

### 160.10.2.2 public void waitForStop(long timeout) throws InterruptedException, LauncherException

*timeout* Maximum number of milliseconds to wait. A value of zero will wait indefinitely.

- Wait until the system has stopped.

*Throws* InterruptedException– If another thread interrupted the current thread before or while the current thread was waiting for the system to stop. The *interrupted status* of the current thread is cleared when this exception is thrown.

LauncherException– When the launch is not successful.

## 160.10.3 public final class LauncherConstants

Defines standard constants for the Feature Launcher specification.

### 160.10.3.1 public static final String LAUNCHER_SPECIFICATION_VERSION = "1.0"

The version of the Feature specification.

## 160.10.4 public class LauncherException extends Exception

Exception thrown when the launcher isn't able to launch the Feature.

### 160.10.4.1 public LauncherException.Reason getReason()

- Get the reason for the exception;

*Returns* The reason

## 160.10.5 public interface LauncherFactory

Create a Feature Launcher.

### 160.10.5.1 public Launcher newLauncher(URL feature, Map<String, Object> variables)

*feature* URL to the Feature file.

*variables* The feature variables to use.

- Create a new launcher based on the provided URLs.

*Returns* the new launcher;

### 160.10.5.2 public Launcher newLauncher(Feature feature, Map<String, Object> variables)

*feature* The feature the launcher should use.

*variables* The feature variables to use.

- Create a new launcher based on the provided Feature instances;

*Returns* the new launcher.