OSGI stands for Open Services Gateway Initiative. OSGI provides an architecture for developing and deploying modular applications and libraries. Sometimes OSGI is also referred as Dynamic Module System for Java. In this article we will explore the basics of OSGI.

Advantages of using OSGI:

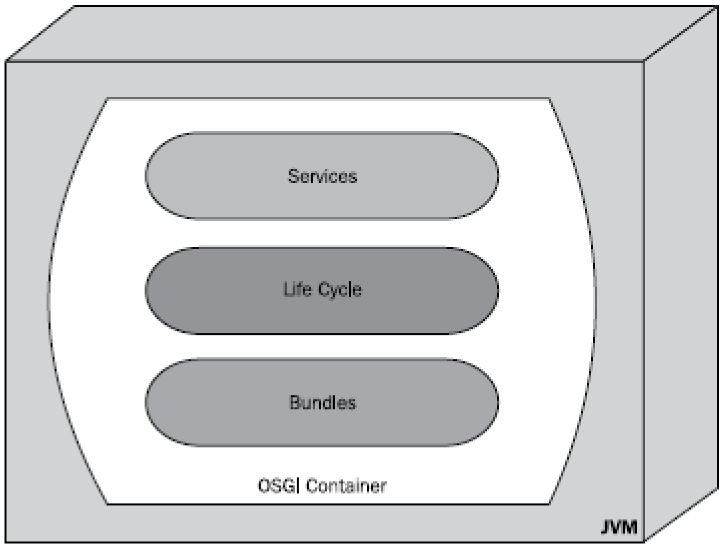
1. **OSGI** allows us to create dynamic, live architectures and applications.
2. **OSGI** provides a framework to develop modular application. We can decompose a complex application into multiple modules. Each modules focuses on specific task.It also motivates us as a developer to build distinctly identifiable JAR files, small sets of code that do a few things in a clearly defined manner.
3. We can install, uninstall, start, and stop various modules of our application dynamically without restarting the container.
4. Our application can have more than one version of a particular module running at the same time.
5. **OSGI** is lightweight and customizable.
6. **OSGI** provides extensibility without eroding the system.
7. **OSGI** manages the complexity of large systems.
8. Starting with bundles, developers are able to encapsulate functional portions of code into single deployable units, with explicitly stated imported and exported packages.
9. In contrast to a regular classloading environment, we as a developer can control explicitly what you expose, share,and how it is versioned.

Existence of OSGI:

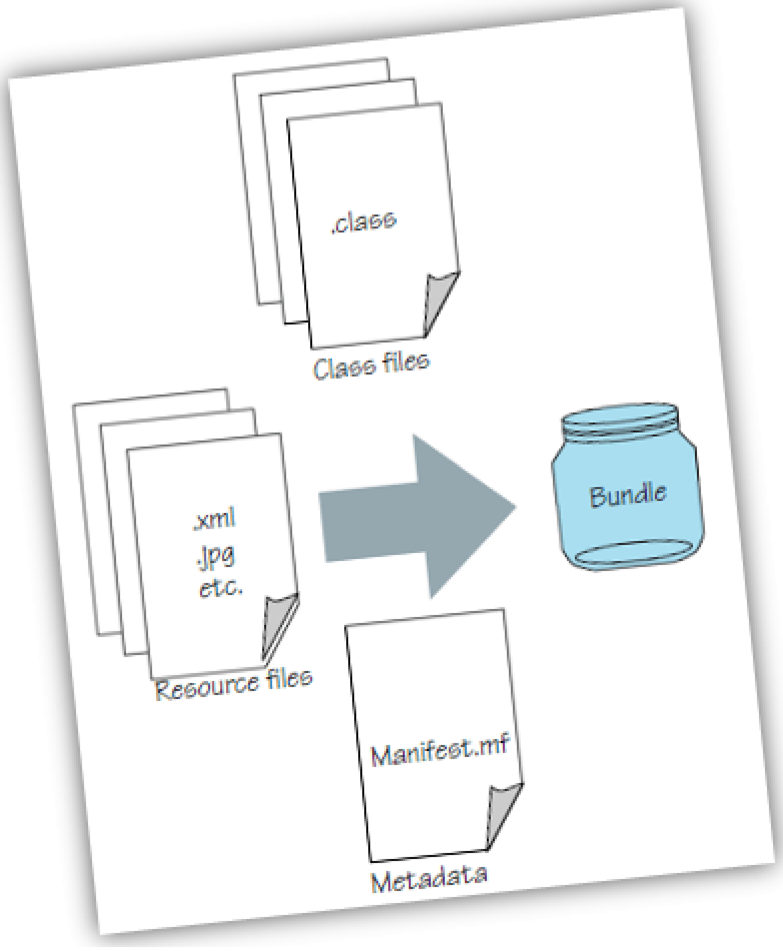
OSGI is a very mature standard. The original OSGI Service Gateway Specification **Release 1.0** was released in **May 2000**.
For more information
https://www.osgi.org/developer/specifications/

Architecture Of OSGI:

The key architectural aspects of **OSGI** that we will be focusing on is shown in the following diagram

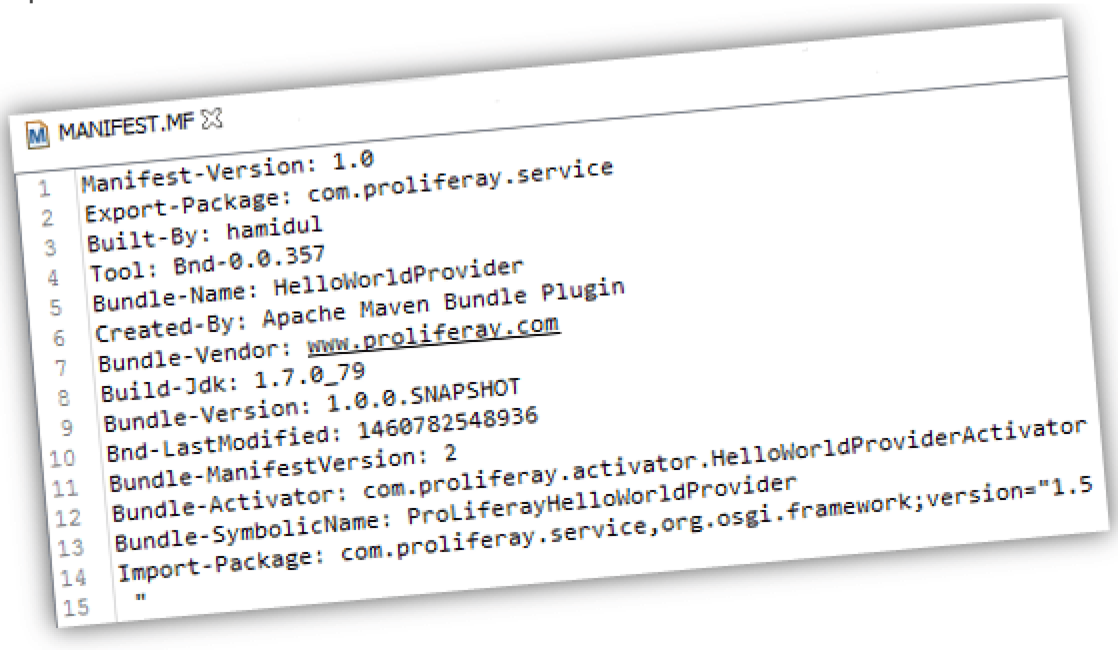


OSGI Bundle:
An **OSGI** bundle is nothing but a **JAR** file with extra meta data.A bundle contains our class files and their related resources, as depicted in the below



figure

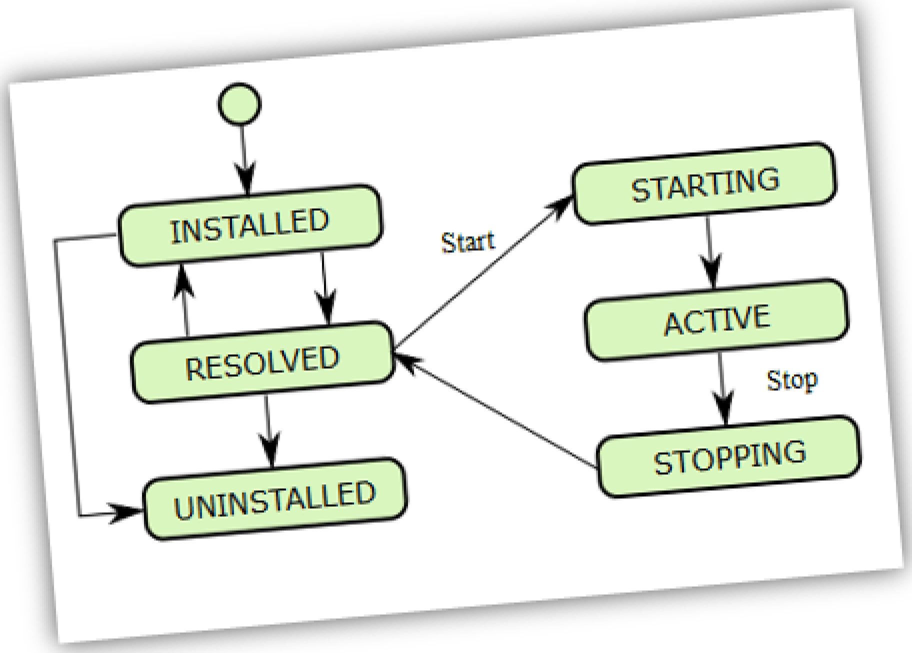
So we can say that every **OSGI** bundle is JAR file. But reverse is not true. That means every JAR can't be a OSGI bundle. To convert a JAR file into bundle we must add extra meta data into **MANIFEST.MF** file. A sample **MANIFEST.MF** file of a bundle is shown below



```
Manifest-Version: 1.0
Export-Package: com.proliferay.service
Built-By: hamidul
Tool: Bnd-0.0.357
Bundle-Name: HelloWorldProvider
Created-By: Apache Maven Bundle Plugin
Bundle-Vendor: www.proliferay.com
Build-Jdk: 1.7.0_79
Bundle-Version: 1.0.0.SNAPSHOT
Bnd-LastModified: 1460782548936
Bundle-ManifestVersion: 2
Bundle-Activator: com.proliferay.activator.HelloWorldProviderActivator
Bundle-SymbolicName: ProLiferayHelloWorldProvider
Import-Package: com.proliferay.service,org.osgi.framework;version="1.5
 "
```

OSGI Headers:
As shown in the above **MANIFEST.MF** file there are few meta data which are only specific to **OSGI**specifications. For example **Bundle-Activator, Bundle-SymbolicName** etc. These are called **OSGI Headers**. So **OSGI Headers** are the key elements which turns a normal JAR file to a Bundle. There are many headers. Among them only **Bundle-SymbolicName** is mandatory.

OSGI Bundle Life Cycle:
The life cycle of a bundle is depicted in the below figure



Below are the explanations of various states of a bundle

INSTALLED : The bundle has been successfully installed.
RESOLVED : All Java classes that the bundle needs are available. This state indicates that the bundle is either ready to be started or has stopped.
STARTING: The bundle is being started, the **BundleActivator.start** method has been called but the start method has not yet returned. When the bundle has an activation policy, the bundle will remain in the**STARTING** state until the bundle is activated according to its activation policy.
ACTIVE : The bundle has been successfully activated and is running. Its Bundle Activator start method has been called and returned.
STOPPING : The bundle is being stopped. The **BundleActivator.stop** method has been called but the stop method has not yet returned.
UNINSTALLED : The bundle has been uninstalled. It cannot move into another state.

Implementation of OSGI Specifications :

Following are the well known implementation **OSGI Specifications**
1. Apache Felix
2. Eclipse Equinox
3. Knopflerfish
4. ProSyst