Tutorial 2 on OSGI (Open Service Gateway Initiative)

Abstract: This tutorial is meant to teach the students, how to develop the service bundles. We outline the steps to develop a bundle that provides a service, to supply multiple implementations for the same service interface, and to retrieve resources packaged in the same bundle.

Bundles: these are the software components plugged into the framework. A bundle may provide zero, one, or multiple services. A bundle is a form of packaging for the services. It is also a functional unit with life cycle operations and class loading capability.

Steps for Developing Bundles:

1. Design the service interface
2. Implement the service
3. Create the bundle activator that usually registers in its 'start' method and unregisters the service in its stop method.
4. Edit the manifest file
5. Compile and generate a JAR file

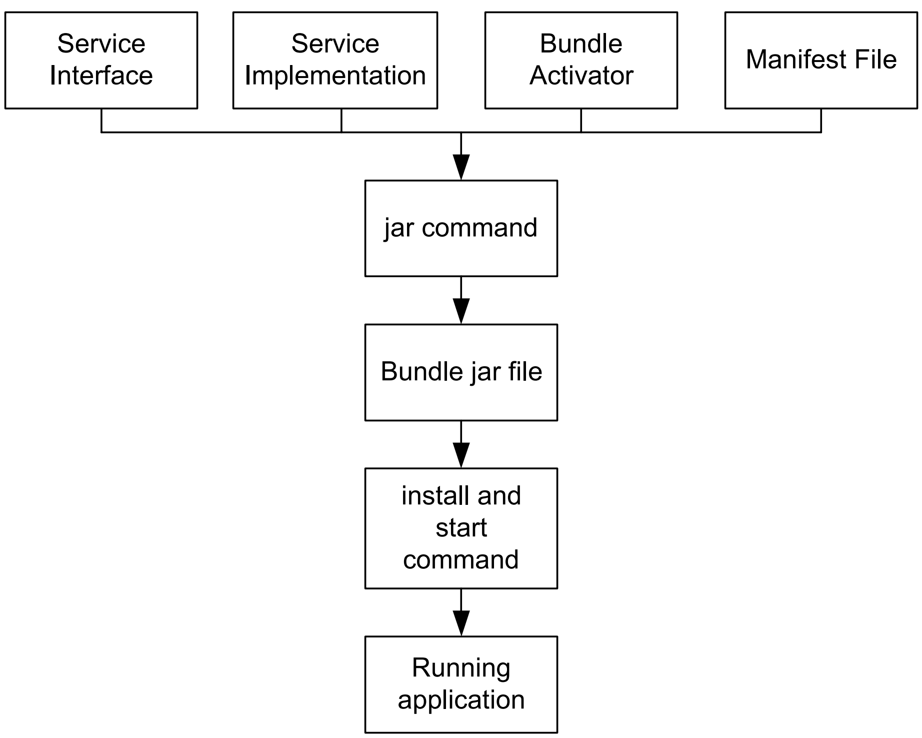

Fig 1: Developing service bundle

Step 1: Service Interface

- There must be an interface for each service that will be included in the bundle.
- This interface represents the service to the other bundles in the server.
- The service interface should be placed in a dedicated package, as it is to be exported to and accessible by other bundles.
- Bundle developers register and access a service in a bundle through the service interface. The service interface defines the methods that the consumers of the service can call.

Step 2: Implement the Service

- Each service has its own implementation.
- There must be at least one class that implements the interface.

Step 3: Create the bundle Activator

- The Bundle Activator object implements the *org.osgi.framework.BundleActivator* interface.
- In the *start* / *stop* method of the Bundle Activator, the programmer encloses the operations that the bundle executes when started/stopped. Such operations are:
- On bundle *start*. When started, the Framework calls 'start' method.
    - Registering the services in a bundle.
    - Referring to other services to retrieve data for the proper operation of the bundle.
    - Starting threads
- On bundle *stop*. When stopped, the Framework invokes 'stop' method.
    - Unregistering the services in the bundle.
    - Releasing all engaged services when resolving the bundle.
    - Terminating threads
- The Activator class must be specified as "public" because it is to be accessed by the framework.
- The activator must have a public default no-argument constructor.

Step 4: Manifest file

- This is a text file that contains miscellaneous properties and values of bundles.

- If your bundle has a Bundle Activator and the Bundle-Activator header is missing in the manifest, the bundle may not start and no error may be indicated.
- You may experience problems with dynamic class loading if you have not correctly specified the Import-Package and Export-Package headers.
- If in the Bundle-Activator header you specify a class that does not implement org.osgi.framework.BundleActivator, the framework will throw a 'ClassCastException' at bundle startup.
- The same header cannot be repeated in the manifest.
- There cannot be an extra empty line before any header.
- A line must not be longer than 72 bytes.
- Use only a colon followed by a white space between the header and it's value. Don't use TAB (\t). Otherwise, the framework will not read the header information and load the necessary resources.
- Use only a white space (not TAB) at the beginning of a line to indicate continuation of the previous line.
- There must be an extra new line at the end of the Manifest.

Step 5: Generate JAR File

- Compile your source code with –d option
- Run the command

  jar cmf Manifest yourJar.jar *.class from the appropriate directory
- If the bundle JAR does not include the Bundle Activator specified in the manifest, the framework will throw a 'ClassNotFoundException' at bundle startup.
- When in doubt, you can unpack the bundle JAR file in a temporary folder and examine.

Summary:

This tutorial will help you to develop a service bundle.

Today's work:

1. Using explorer find the path where the jes2.0 is located on your machine.
2. jes_path = location of installation of jes2.0
3. At the command prompt set CLASSPATH=jes_path\lib\framework.jar

4. Go to Start → Run → explorer
    - Select ‘C’ drive
    - Create a folder ‘com’
    - Under ‘com’, create a folder ‘acme’
    - Under ‘com’, create three folders ‘service’, ‘impl’ and ‘caller’.
    - Under ‘service’, create a folder ‘dictionary’
    - Under ‘impl’, create a folder ‘dictionary’
    - Under ‘caller’, create a folder ‘dictionary’
5. Developing Dictionary service bundle:
    - **Step 1:** <u>Design the service interface:</u>

Suppose we want to develop a dictionary service that return’s word definitions. The service interface may be defined as:

```
/*	This service produces definitions of words.		*/
package com.acme.service.dictionary;
public interface DictionaryService {
/**
* Gets the definitions of a word.
* @param word the word whose definition is sought
* @return the definition or null if the word is not found
*/
	public String getDefinition (String word);
/**
* Gets the definitions of the words beginning with the specified letter
* @param alpha the first letter of the words
* @return  an array of definitions, or null if either no definition is found or the
letter is invalid
*/
public String[] getDefinitions (char alpha);
}
```

Save above file under com\acme\service\dictionary as 'DictionaryService.java'

-- ------------------------------ ----------- ---------

- **Step 2:** Implement the Service

```
package com.acme.impl.dictionary;

import java.util.*;
import com.acme.service.dictionary.DictionaryService;

class Devil implements DictionaryService {
    private Properties defs;
    Devil( ) {
            defs= new Properties();
            defs.put("accuse", "To affirm  another's guilt or unworth;"+
                        " most commonly as a justification of ourselves for" + "having
                        wronged him.");
            defs.put("admire","Our politee recognition of "+
                        "another's resemblance to ourseleves");

            defs.put("calamity","Misfortune to ourselves, "+
                        "and good fortunes to others.");
            defs.put("kill","To create a vacancy without nominating "+
                        "a successor.");
    }
    public String getDefinition(String word)
    {
            return (String) defs.get(word);
    }
    public String[] getDefinitions(char alpha)
    {
            Vector v = new Vector();
            for (Enumeration e = defs.keys();e.hasMoreElements(); )
            {
                    String w = (String)e.nextElement();
                    if(w.length()>0 && w.charAt(0)== alpha)
                    {
```

```
                v.addElement(w+": "+defs.get(w));
            }
        }

        String[] result = null;
        if(v.size()>0)
        {
            result = new String[v.size()];
            v.copyInto(result);
        }

        return result;
    }
}
```

Save above file under com\acme\impl\dictionary as 'Devil.java'

-- ------------------------------ ----------- ---------

- **Step 3:** <u>Activator file</u>

We register the dictionary service in the activator's 'start' method, so that our service becomes available when the bundle is started. We unregister it in the bundle activator's 'stop' method, so that it is withdrawn when the bundle is stopped.

The bundle activator is defined as:

```
package com.acme.impl.dictionary;

import java.util.Properties;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;
import com.acme.service.dictionary.DictionaryService;

public class Activator implements BundleActivator
{
    private ServiceRegistration devilReg;
```

```
        public void start (BundleContext ctxt)
        {
                DictionaryService devil = new Devil();
                Properties devilProps = new Properties();
                devilProps.put("author", "Ambrose Bierce");
                devilProps.put("description", " The devil's Dictionary");

                devilReg= ctxt.registerService(
                        "com.acme.service.dictionary.DictionaryService",
                        devil, devilProps);
        }
        public void stop( BundleContext ctxt)
        {
                if(devilReg != null)
                {
                        devilReg.unregister();
                }
        }
}
```

Save above file under com\acme\impl\dictionary as 'Activator.java'

-- ------------------------------ ---------- ---------

- **Step 4: <u>Manifest file</u>** (keep an extra blank line at the end of this file)

Save this file under com\acme\impl\dictionary as "Manifest" (with quotes)

```
Bundle-Activator: com.acme.impl.dictionary.Activator
Export-package: com.acme.service.dictionary
```

-- ------------------------------ ---------- ---------

- **Step 5: Crate a Service Calling bundle**

```
package com.acme.caller.dictionary;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;
```

```
import com.acme.service.dictionary.DictionaryService;

public class Activator implements BundleActivator
{
    public void start (BundleContext ctxt)throws Exception
    {
            ServiceReference[] ref = ctxt.getServiceReferences(
                    "com.acme.service.dictionary.DictionaryService",
                    "(description= The devil's Dictionary)");

            if(ref==null)
            {
    System.out.println("The dictionary service has not been registered");
    return;
            }
            DictionaryService ds =
            (DictionaryService) ctxt.getService(ref[0]);

            String[] aDefs = ds.getDefinitions('a');

            if(aDefs!=null)
            {
                    for(int i=0; i<aDefs.length; i++)
                    {
                            System.out.println(aDefs[i]);
                    }
            }

            ctxt.ungetService(ref[0]);
    }

    public void stop( BundleContext ctxt)
    {
            // nothing
    }
}
```

Save above file under com\acme\caller\dictionary as 'Activator.java'

-- ------------------------------ ----------- ---------

- **Step 6:** **<u>Manifest file</u>** (keep an extra blank line at the end of this file)

Save this file under com\acme\caller\dictionary as "Manifest1" (with quotes)

Bundle-Activator: com.acme.caller.dictionary.Activator

Import-package: com.acme.service.dictionary

-- ------------------------------ ----------- ---------

6. Go to Start → Run → Cmd

   - Change your prompt to c:>
   - Compile your java files under com\acme\service\dictionary

   C:\>javac –classpath .\;jes_path\lib\framework.jar com\acme\service\dictionary*.java

   - Compile your java files under com\acme\impl\dictionary

   C:\>javac –classpath .\;jes_path\lib\framework.jar com\acme\impl\dictionary*.java

   - Create a dictionary.jar with service interface

   C:\>jar -cmf com\acme\impl\dictionary\Manifest dictionary.jar com\acme\service\dictionary*.class

   - Add implementation and activator class files and manifest file to the dictionary.jar from com\acme\impl\dictionary

   C:\>jar -uf dictionary.jar com\acme\impl\dictionary*.class

   - Compile your java files under com\acme\caller\dictionary

   C:\>javac –classpath .\;c:\jes2.0\lib\framework.jar com\acme\caller\dictionary*.java

   - Create a dictcaller.jar

   C:\>jar -cmf com\acme\caller\dictionary\Manifest1 dictcaller.jar com\acme\caller\dictionary*.class

-- ------------------------------ ----------- ---------

7. The dictionary.jar and dictcaller.jar will get created under c:, copy those two bundles under jes_path\bundles\

-- ------------------------------ ----------- ---------

8. Go to Start → Programs → Java Embedded server 2.0 → Java Embedded server 2.0

- install dictionary.jar
- install dictcaller.jar
- bundles
- start 1
- bundles
- services
- start 2
- bundles
- stop 1
- stop 2
- start 2
- stop 2
- un 1
- bundles
- un 2
- bundles
- shutdown

-- ------------------------------ ----------- ---------