**Playing with Apache Karaf Console**

In my [previous post](http://www.javaindeed.com/getting-to-know-apache-karaf/), I’ve tried to explain overall architecture of Apache Karaf and its important components and services. Now in order to start using it, we have to get more familiar with the interaction mechanism provided by Apache Karaf, yes I mean the console, one of the most important features of Apache Karaf.

**What you’ll learn in this post?**

In this post I’ll try to cover following topics:

* Requirements to install and run Apache Karaf
* Review Apache Karaf directory structure
* How to start and stop Apache Karaf
* Some common shell commands and the concept of command groups
* How to manage bundles (install, start, stop, etc)
* Introducing bundles auto-deployment mechanism

**What is Apache Karaf Console?**

Every OSGi container has its own console to manage bundles and interact with the container. In OSGi specification there is nothing about OSGi console implementation, so containers have implemented their own console. Even though they have some common commands, generally you have to learn each container’s shell commands in order to master it. This is an area where Apache Karaf console can be useful. If you use Apache Karaf on top of containers, then you just have to learn a single command set. But the most important part is that Apache Karaf console is actually based on a subproject of Apache [Felix](http://felix.apache.org/) named [Gogo](http://felix.apache.org/documentation/subprojects/apache-felix-gogo.html) shell with an added layer that makes it simple to extend by adding custom commands to interact with bundles or container services.

In this post, I’ll just try to get you familiar with the console and how to use it to manage your bundles. Extending the console by adding custom commands would be a title for my future posts.  
OK, that’s enough. I’ve never liked lengthy introductions. Lets get our hands dirty by downloading and installing Apache Karaf and see console features in practice.

**Prerequisites**

In order to get on with this post, following requirements have to be met in your system:

* Windows >= XP or Linux distribution (Ubuntu, Debian, and almost any)
* JDK 1.6 or greater. Remember that Apache Karaf is not currently compatible with JRockit JDK. (I’ve used [HotSpot 1.7.0\_45](http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html))
* JAVA\_HOME environment variable must be set to your JDK installation folder
* [Maven 3](http://maven.apache.org/download.cgi) (I’ve used 3.0.5)

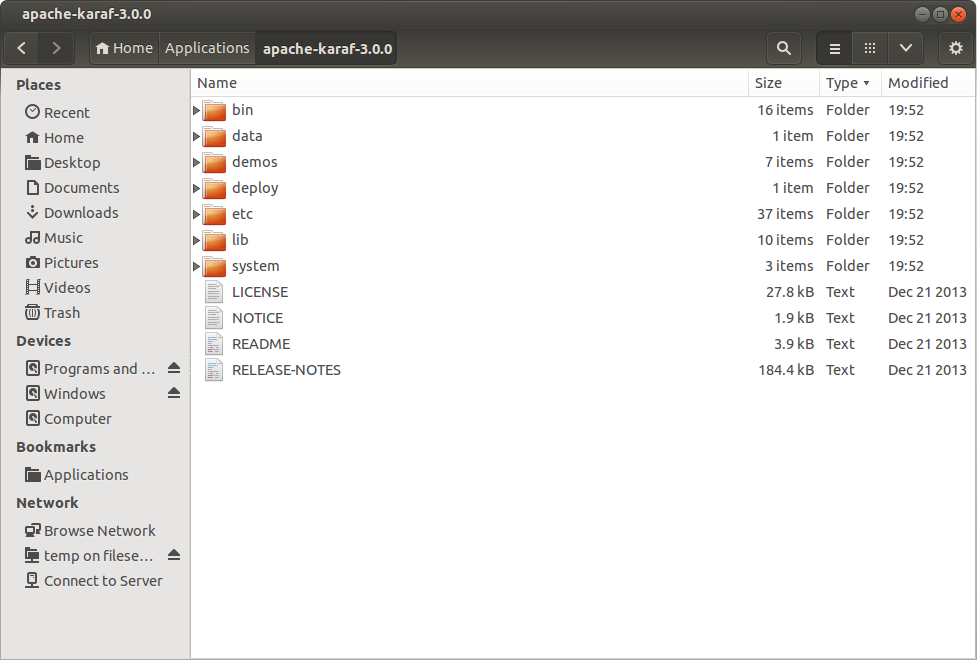
**Downloading and Installing Apache Karaf**

As I write this post, the latest stable version of Apache Karaf is [3.0.1](http://karaf.apache.org/index/community/download.html#Karaf3.0.1) which has been released recently. As some features that I’ll try to cover in this post are part of the new 3.0.0 branch, you have to download 3.0.0 or 3.0.1.

After you’ve downloaded the binary zip file, extract it to a folder which from now on, I’ll call it {KARAF\_HOME}. That’s it. Installation completed.

**Directory Structure**

The following screenshot displays directory structure of Apache Karaf.

Apache Karaf Directory Structure

Here is a brief description about each folder:

* **bin**: contains scripts to start/stop the container or connecting to a running container
* **data**: this folder is mostly for internal use of Apache Karaf and at initial installation is almost empty (there is just an empty tmp folder inside it). As this folder contains all working data of Apache Karaf, by simply deleting its contents you’ll have a fresh Apache Karaf installation. When you run the Apache Karaf for the first time, following subfolder will be created automatically inside it:
  + **cache**: this folder would be used as OSGi bundle cache and there is a subfolder for each bundle named after its id (like bundle0, bundle1, etc)
  + **generated-bundles**: the deployer service uses this folder as temp
  + **log**: Karaf by default redirects all logs to this folder, in a file named karaf.log
  + **tmp**: temporary folder for internal use only
* **demos**:  contains some sample bundles demonstrating main capabilities of Apache Karaf
* **deploy**: hot deployment folder. Every bundle you place in this folder would be automatically deployed in container (we will come to it later in this post)
* **etc**: all configuration files (like logging configuration, user definition file, etc) are stored in this folder
* **instances**: when you create a new child instance of Apache Karaf, a new subfolder would be created in this folder named after instance name and all data and configurations related to that instance would be stored in it. This folder would be created when Apache Karaf starts for the first time
* **lib**: contains bootstrapping libraries which would be used when Apache Karaf is starting up. It contains a subfolder for JRE extensions, named *ext* and another one for endorsed libraries named *endorsed*. All jar files in this folders would be used by bootstrap class loader and will not be available for bundles by default
* **system**: it’s an OSGi bundle repository organized in maven repository structure

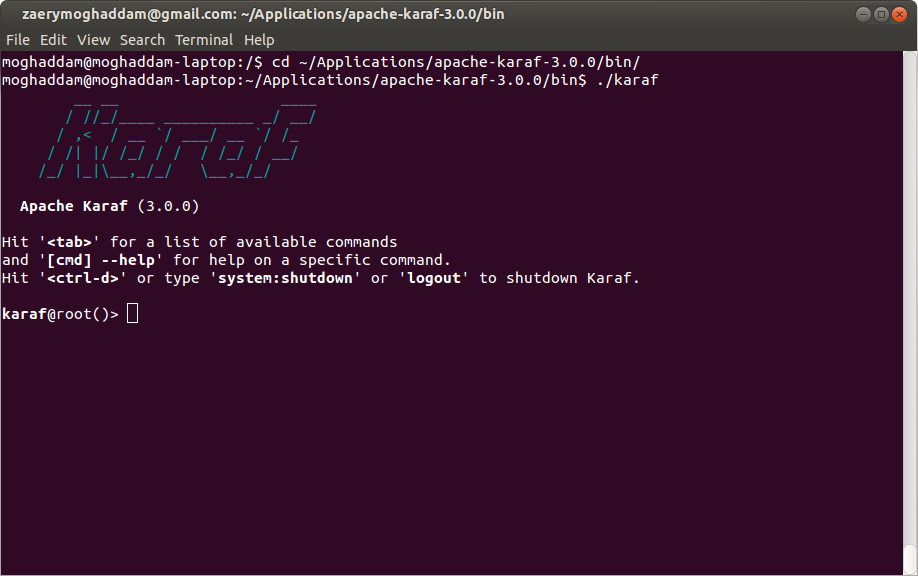
**Starting Karaf**

In order to start Apache Karaf, in a terminal window, navigate to {KARAF\_HOME}/bin and type the following command and hit enter:

Windows: *karaf.bat*

Linux: *./karaf*

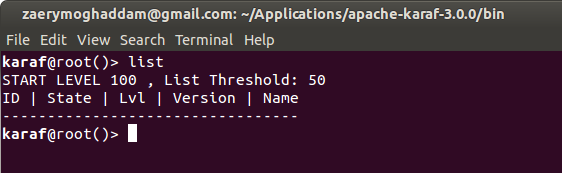
When the karaf bootstrap process get finished, as can be seen in following screenshot, an ascii Karaf logo will appear in your terminal meaning that Apache Karaf console is waiting for you to command him (or maybe her).

Apache Karaf Console Startup

As can be seen in the picture, in order to shutdown Apache Karaf, you can use *logout* or *system:shutdown* command or hit *Ctrl+D* on keyboard.

**Basic Shell Commands**

The first and one of the mostly used command in Apache Karaf is the ‘*list*‘ command which displays list of all bundles installed. When you issue this command for the first time, the output should be something like this:

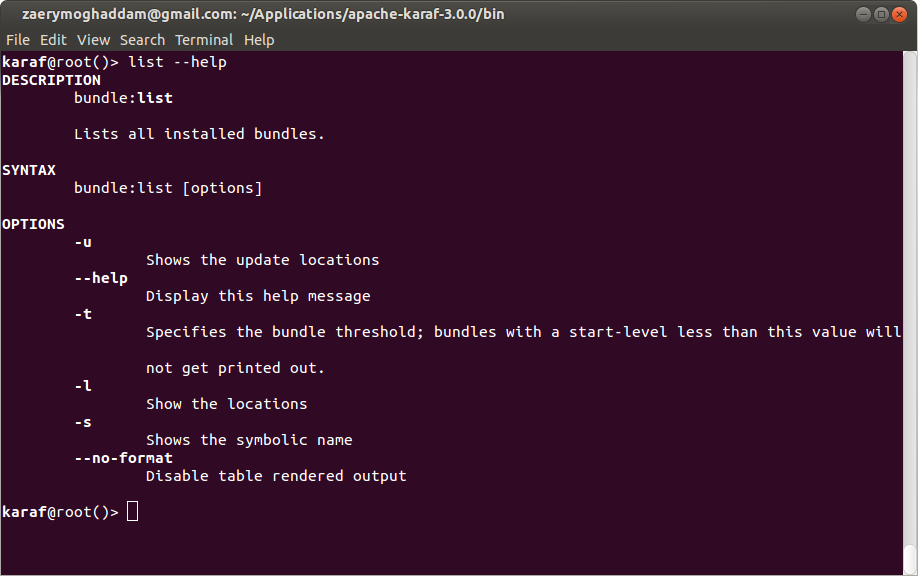
Issuing list command for the first time

The ‘*list*‘ command displays ID, State, Level, Version and Name of bundles in a table. As can be seen, there is no bundle in the table by default. This is because of the ‘List Threshold’ value mentioned on top of the table. The ‘*list’* command displays only those bundles that their level value is less than this threshold. It has a default value of 50 and as by default, there is no bundle with the level value greater than 50, nothing would be displayed.

In order to change the threshold value, we have to specify some parameters when calling the *‘list*‘ command. To get familiar with its parameters, we can use the *–help* switch after its name; so type the following command in console:

list --help

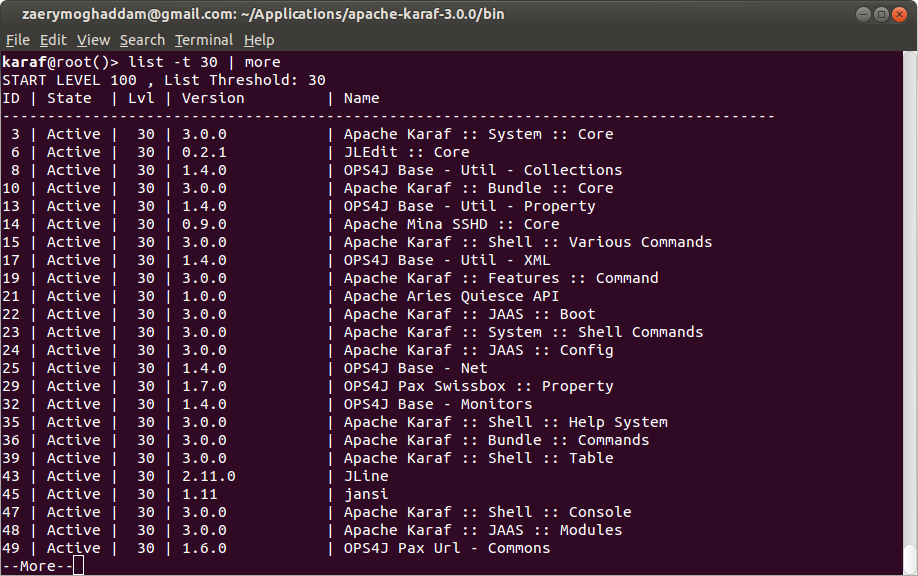
This will cause the following output:

Issuing list –help command to get list command parameters

The very first part of the output is the full name of the command including its command group. (I’ll talk about commands group in next section). As can be seen in screenshot, we can use the *-t* switch to specify a threshold value for bundles. The *–help* switch can be used with any console command to get familiar with its parameters and their usage. In order to list all bundles having level value greater than or equal to 30, we have to specify the following command:

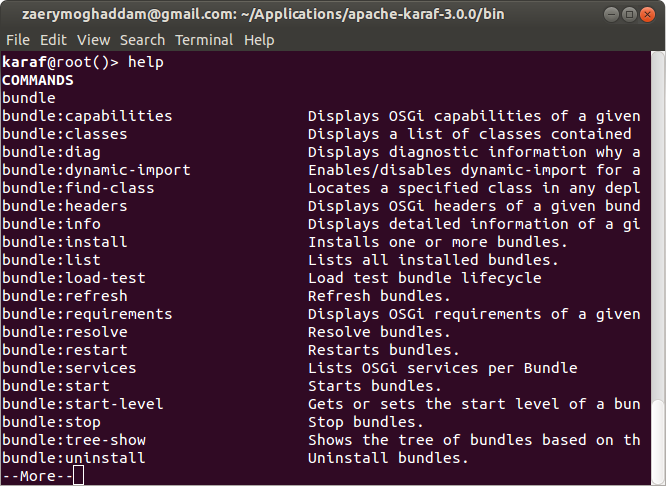
list -t 30

Due to the large number of bundles with the level value specified in threshold parameters, the result will be too long and may scrolls out of the terminal window. In order to access the result in page by page basis, another command named *‘more*‘ (yes more, just like *‘more’* command in Linux and Windows terminal) is provided in Apache Karaf console. You can pipe output of any command to ‘*more*‘ in order to display them page by page. The result of issuing the *‘list*‘ command and piping its output to the *‘more’*  is pictured in following screenshot:

Result of issuing list -t 30 – more command

There is some other utility commands like *‘tail’, ‘grep’, ‘sort’, ‘head’, ‘history’* and . . . which act like their corresponding commands in a Linux terminal. You can use them to interact with Karaf just as you use them in a Linux terminal to interact with your OS. In order to access the list of commands, you can hit the *‘Tab’* key on your keyboard. Hitting the *‘Tab’* key triggers the autocomplete feature. If you’ve not typed any character yet, it displays all available console commands as a list.

As I mentioned above, any command in console has an option named *‘–help’* that prints all of its options and arguments. You can also use a command named *‘help’* individually to see the list of all available commands with a short description about each one (like following screenshot).

Issuing help command

The *man* command also gives the name of a command as input and has the same output as *‘help*‘. It is actually an alias for *‘help | more’*. You can use it like this:

man {command\_name}

**Introducing Commands Group**

As displayed in picture above, most commands have a prefix separated with a colon. This prefix is the name of the group which that commands belongs to and contains some related commands. It acts like a namespace and give us the option to have similar command names in different groups. Actually, every command you run in console belongs to a command group. If the command name is unique among all groups (like *‘help’*, *‘man’* or ‘shutdown’), you can omit the group name prefix. Apache Karaf console can find such commands and run it, but it’s recommended to always use the prefixed version to prevent any confusion.

**Note:**

For commands that exist in more than one group (for example *feature:list, bundle:list, service:list, instance:list*, etc.), if you omit the group prefix, the console will try to find it in *‘shell*‘ group, then *‘bundle’* group and if it couldn’t find it, would try to find the first command in its available commands list that its name ends with given command name. So run your commands without prefix ONLY WHEN you are sure the command name is unique among available groups in Apache Karaf console.

Some well known commands group which are available by default in Apache Karaf are:

* **shell**:contains basic console commands like those listed in previous section (for example *clear, cat, more, sort, echo, etc.*)
* **bundle**: contains commands related to manipulating bundles through console (like *list, install, uninstall, resolve, start, stop, refresh, etc.)*
* **instance**: collection of commands related to managing a Apache Karaf instance (like *start, stop, status, list, create, connect, destroy, etc.)*
* **feature**: A Feature is a provisioning concept introduced by Apache Karaf. This group contains a collection of commands related to management of Karaf features (like install, uninstall, list, info, etc.)
* **log**: all commands related to Karaf logs are gathered in this group (e.g list, display, tail, clear and so on)
* **config**: contains commands related to manipulating configurations (for example edit, update, list, delete, cancel, etc.)
* **jaas**: collection of commands for managing users, roles and groups (like group-add, role-add, user-add, user-list, group-delete, user-delete, etc.)

You can use *‘help {group\_name}’* to see the list of commands in a group (like *‘help bundle’* or *‘help shell’)*.

**Managing Bundles**

Bundles are first-class citizens in OSGi world and are basic building blocks of Apache Karaf itself. By managing bundles, I mean installing, uninstalling, starting, stopping, getting information about its services, etc. In order to demonstrate the operations on a bundle, I’ve developed a simple bundle (helloworld-1.0.0.jar) that displays a “Hello World!” on Apache Karaf console on start and a “Goodbye World!” when you try to stop it. You can download its source or binary jar file from my [GitHub](https://github.com/moghaddam/developmentor) account.

**Installing a bundle**

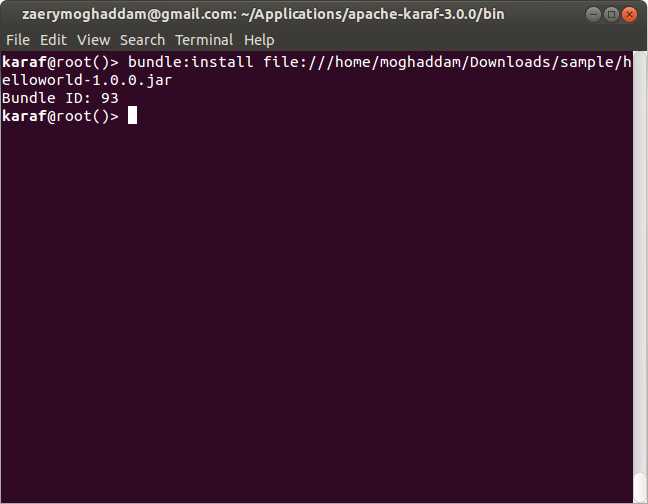
In order to install the sample bundle in Apache Karaf, you can use *‘bundle:install’* command as follow:

bundle:install file://{bundle\_folder}/helloworld-1.0.0.jar

where *{bundle\_folder}* is the folder that you’ve downloaded the jar file to. In my case, it’s something like this:

bundle:install file:///home/moghaddam/Downloads/sample/helloworld-1.0.0.jar

As displayed in picture below, by issuing the ‘*bundle:install*‘ command, Apache Karaf tries to install the bundle in its underlying OSGi container and displays its bundle ID in output (your value probably is different from mine). This ID would be used as a parameter for most *bundle:\** commands (those commands in bundle group).

Installing a bundle in Apache Karaf

As I’ve stated in my [previous post](http://www.javaindeed.com/getting-to-know-apache-karaf/), Apache Karaf deployer architecture lets you deploy a bundle from file system, web address or maven repository. In the example above, we have used the ‘file://’ prefix to inform the deployer that we want to deploy a bundle from file system. You can deploy a bundle directly from web using ‘http://’ or ‘https://’ protocols. As an example, instead of downloading the helloworld-1.0.0.jar, you can try to install it directly from GitHub by issuing the following command:

bundle:install https://github.com/moghaddam/developmentor/blob/master/helloworld/target/helloworld-1.0.0.jar?raw=true

Another option is to install the bundle in your local Maven repository and ask Apache Karaf to install it from there. If you’ve a copy of the source repository of *helloworld* bundle, you can install it in your local Maven repository by issuing following command in your terminal (the *{bundle\_source\_folder}* is the folder where the project’s *pom.xml* file is located):

cd {bundle\_source\_folder}

mvn install

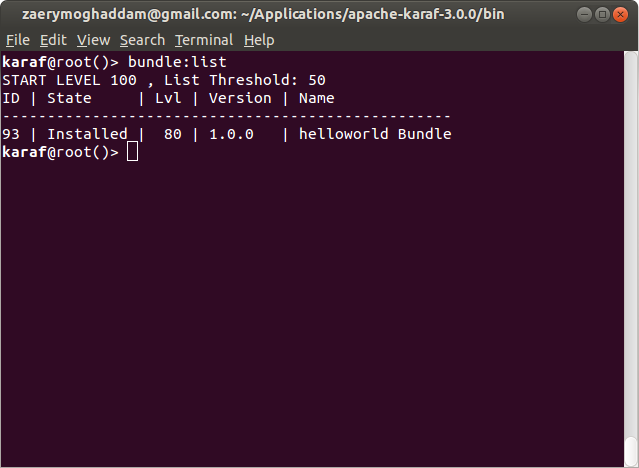
then you can try to install the bundle in your Apache Karaf console using *‘mvn:’* as protocol:

bundle:install mvn:com.blogspot.developmentor/helloworld/1.0.0

the syntax for specifying a bundle in Maven repository is:

mvn:group-id/artifact-id/version

As stated in OSGi specification, a bundle should be in installed state after initial installation. You can check this using  bundle:list command. The output of this command is captured in following screenshot:

Issuing bundle-list command

As can be seen, the newly installed bundle is in *Installed* state and Apache Karaf set its start level to 80.

**Note:**

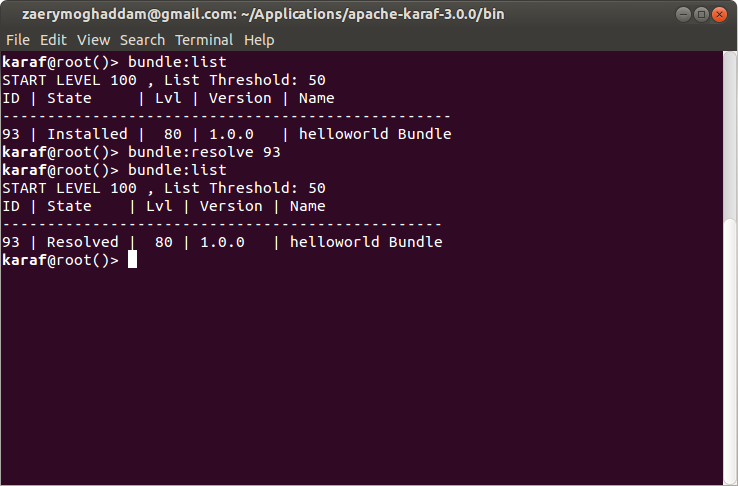
The default start level for bundles is configured in Apache Karaf configuration property file in *{KARAF\_HOME}/etc/config.properties*. If you check this file, you can see a number of different configurations available to be customized if you like. The default start level for bundles is specified as *karaf.startlevel.bundle*  
I’ll take a look in different configuration options in my later posts.

**Resolving, Starting and Stopping a bundle**

As specified in OSGi specification, in order to start a bundle, it should be first transitioned to the *Resolved* state. When resolving a bundle, OSGi container have to ensure that all the dependencies specified in bundle meta-data are met.  
You can resolve a bundle by issuing following command in Apache Karaf console:

bundle:resolve 93

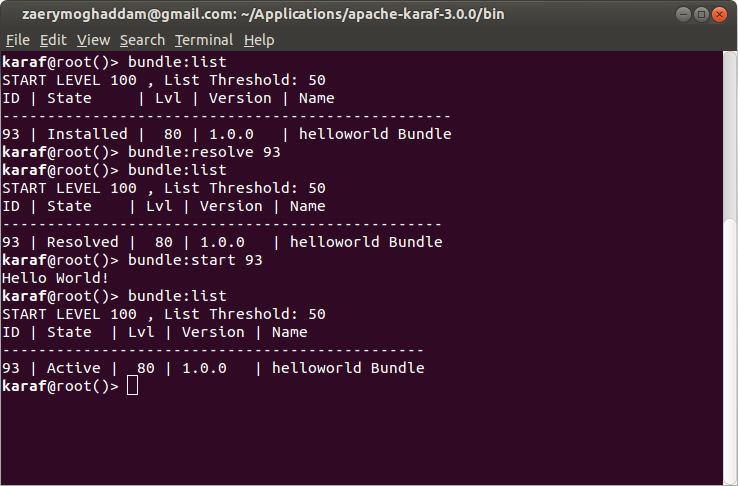
where the number 93 is the identifier assigned to your bundle when installing it in previous step. As *bundle:resolve* command has no output, I’ve tried to use *bundle:list* command to check bundle status just before and after resolving my bundle. The output of resolving a bundle is pictured below:

Issuing bundle-resolve command

As can be seen, the bundle has been successfully transitioned to *Resolved* state.  
Now it’s time for action. We can start the bundle and see the actual output. In order to start a bundle (as you may guessed) we have to issue the *bundle:start* command, passing the identifier of our bundle:

bundle:start 93

When you issue the start command, the container temporary changes the bundle state to Starting and just after its start-up process finishes, it changes the bundle state to Active. By starting the bundle, you can see its well known ‘Hello World!’ message appears on Apache Karaf console. Again, I’ve tried to list bundles, just after starting it, to make sure it’s in Active state.

issuing bundle-start and checking its state

The helloworld bundle has an Activator class (as defined in OSGi specification) implementing *BundleActivator* class as follow:

package com.blogspot.developmentor;

import org.osgi.framework.BundleActivator;

import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

public void start(BundleContext context) {

System.out.println("Hello World!");

}

public void stop(BundleContext context) {

System.out.println("Goodbye World!");

}

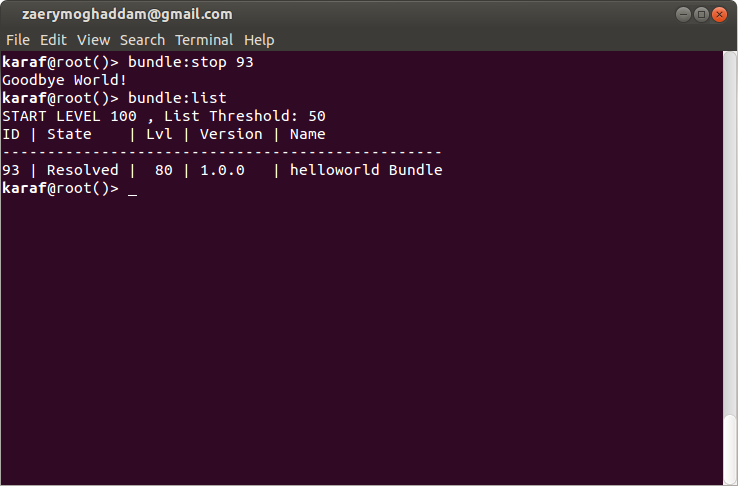
}

When the container tries to start the bundle, it actually calls the *start* method of *Activator* class (if any exists) which causes the *‘Hello World’* string to appear in Apache Karaf console. The same is true for *stop* method when container wants to stop a bundle. Having an activator class is not necessary for all bundles, but in our case, it helps to illustrate the life-cycle of the bundle.

In order to stop a bundle, you have to issue the *bundle:stop* command with your bundle identifier:

bundle:stop 93

just like starting process, OSGi container changes the state of the bundle to *Stopping* temporarily, and after stop process finishes, it changes back to *Resolved* again.

Issuing bundle-stop and checking its state

**Uninstalling a bundle**

To uninstall a bundle you have to just issue the *bundle:uninstall* command with your bundle identifier:

bundle:uninstall 93

That’s it. The bundles has vanished from Apache Karaf and you’ll get an empty list if you issue *bundle:list* command.

**Using Auto-Deployment for bundles**

When you’re developing a new bundle, it would be tedious to manage your bundles in the way we explained in above paragraphs. For development purpose, you’ll mostly use the auto-deployment feature of Apache Karaf by just dropping the jar file of your bundle in *‘deploy’* folder in *{KARAF\_HOME}*. It’ll be updated as you copy a new version of your jar file to this folder. It tries to install and start your bundles immediately after you copy them.  
For example, if you copy the *helloworld-1.0.0.jar* bundle in this folder, the *‘Hello World!’* message immediately appears on Apache Karaf console. If your delete it, the *‘Goodby World!’* message would be displayed, because Apache Karaf stops and then uninstalls a bundle which has been removed from this folder.

If you copy a new version, the sequence would be to stop and uninstall the old version and then install and start the new version immediately.  
By copying a new version, I mean a new build of your jar file and it’s not necessary to change the version number of the bundle.

**Conclusion**

The topics we’ve review in this post were enough to start using Apache Karaf console and deploy your OSGi bundles in it to check whether there is any incompatibility. I personally encourage anyone who wants to start OSGi development, to get used to Apache Karaf. The console features you’ve seen in this post were just part of services added by Apache Karaf on top of well-known OSGi containers like Felix and Equinox.  
In my next posts, I’ll show you how to enrich the console by developing your custom commands to interact with your application bundles at runtime.

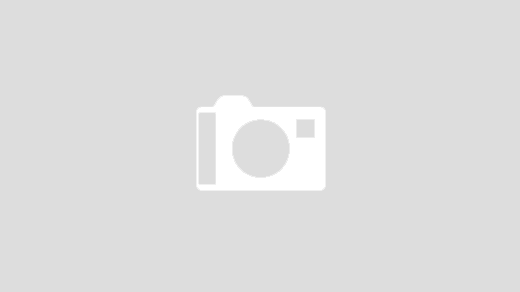
Feel free to share you thought about this post with me (through your valuable comments) or suggest it to others (by sharing this post :))

Please follow and like us:

Tags: [Apache Karaf](https://www.javaindeed.com/tag/apache-karaf/)[OSGi](https://www.javaindeed.com/tag/osgi/)

* [**Next story** Getting started with JSON-B](https://www.javaindeed.com/getting-started-with-json-b/)
* [**Previous story** Getting to know Apache Karaf](https://www.javaindeed.com/getting-to-know-apache-karaf/)

**You may also like...**

* [](https://www.javaindeed.com/getting-to-know-apache-karaf/)

[**Getting to know Apache Karaf**](https://www.javaindeed.com/getting-to-know-apache-karaf/)

**2 Responses**

* [Comments2](https://www.javaindeed.com/playing-with-apache-karaf-console/#commentlist-container)
* [Pingbacks0](https://www.javaindeed.com/playing-with-apache-karaf-console/#pinglist-container)

1. *Sharron Abraham A/L Thangasamy*

[8th December 2019 at 3:25 pm](https://www.javaindeed.com/playing-with-apache-karaf-console/#comment-1670)

Hi Sir ,

I would like to get an help on how can i create a script to restart the servicemix and i wanted to to run this weekly using crontab , as for your info servicemix should be restarted in proper way where we have to  
Make sure the servicemix logs stop fully in order to start back the servicemix again from bin , we also have amq running on the same server with under different user which we will also stop and start before we start back the servicemix. Hope you can help me on how to automate this process for weekly activity.

[Reply](https://www.javaindeed.com/playing-with-apache-karaf-console/#comment-1670)

* + [*Ehsan Zaery Moghaddam*](https://www.javaindeed.com)

[25th December 2019 at 8:06 am](https://www.javaindeed.com/playing-with-apache-karaf-console/#comment-1677)

Hi Sharron

Sorry for the late answer. I think the best place to ask such question is [ServiceMix User Mailing List](https://servicemix.apache.org/community/mailing-lists.html). I don’t have much practical knowledge about ServiceMix to help you in this regard.

[Reply](https://www.javaindeed.com/playing-with-apache-karaf-console/#comment-1677)