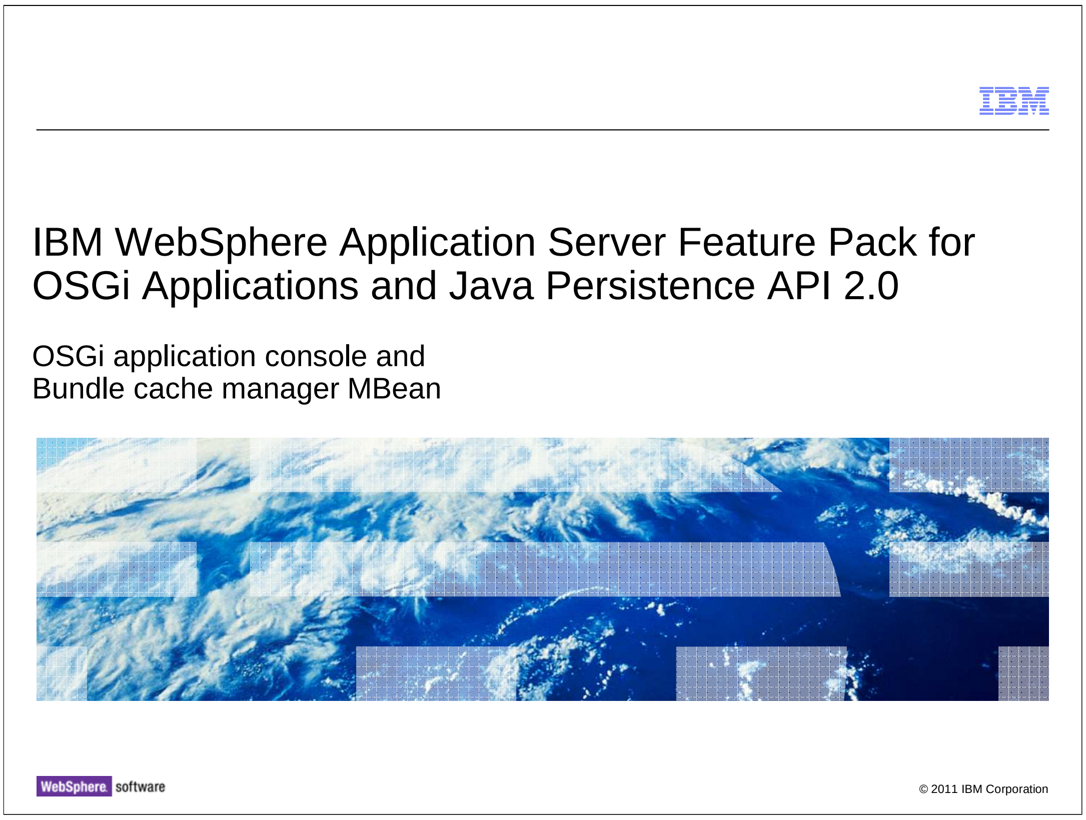
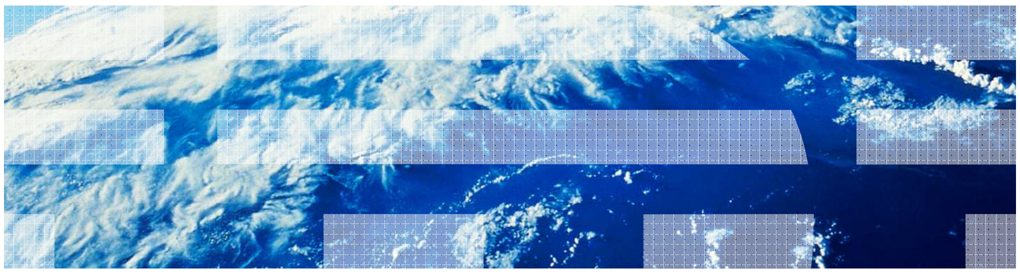


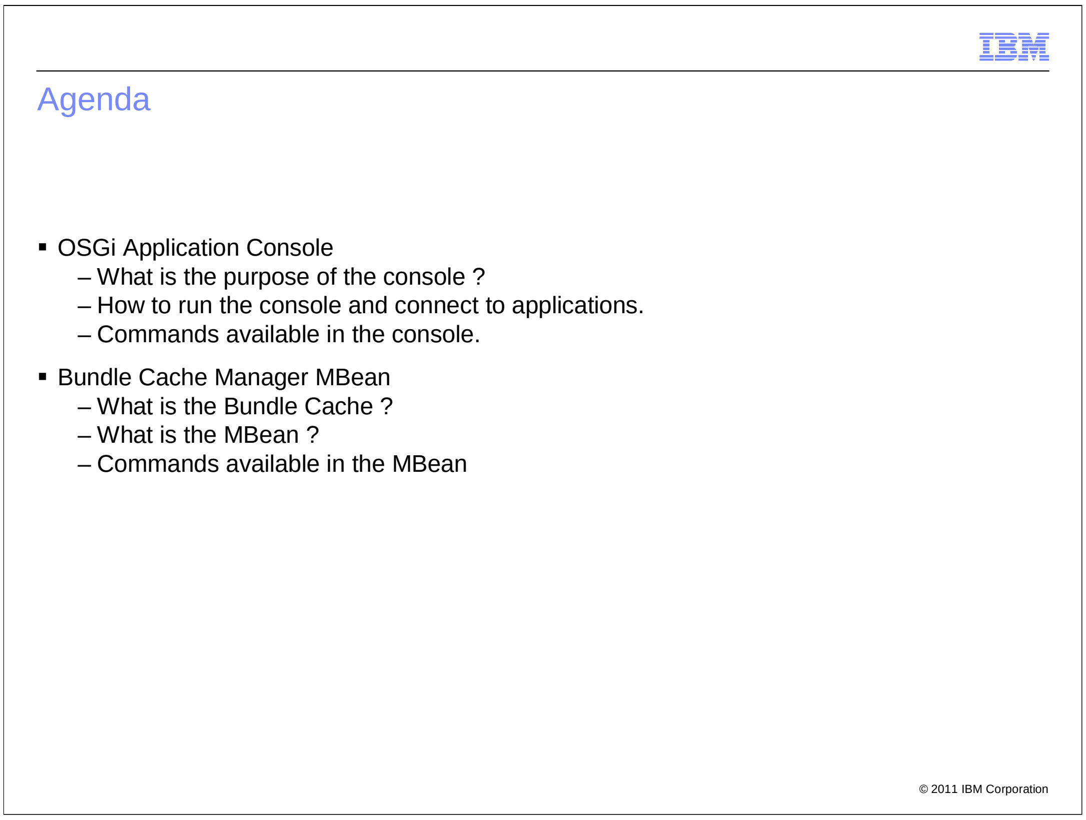
IBM WebSphere Application Server Feature Pack for OSGi Applications and Java Persistence API 2.0

OSGi application console and
Bundle cache manager MBean



WebSphere software

This module provides education on the IBM WebSphere® Application Server Feature Pack for OSGi Applications and Java™ Persistence API 2.0 OSGi Application Console and Bundle Cache Manager MBean.



Agenda

- OSGi Application Console
  - What is the purpose of the console ?
  - How to run the console and connect to applications.
  - Commands available in the console.
- Bundle Cache Manager MBean
  - What is the Bundle Cache ?
  - What is the MBean ?
  - Commands available in the MBean

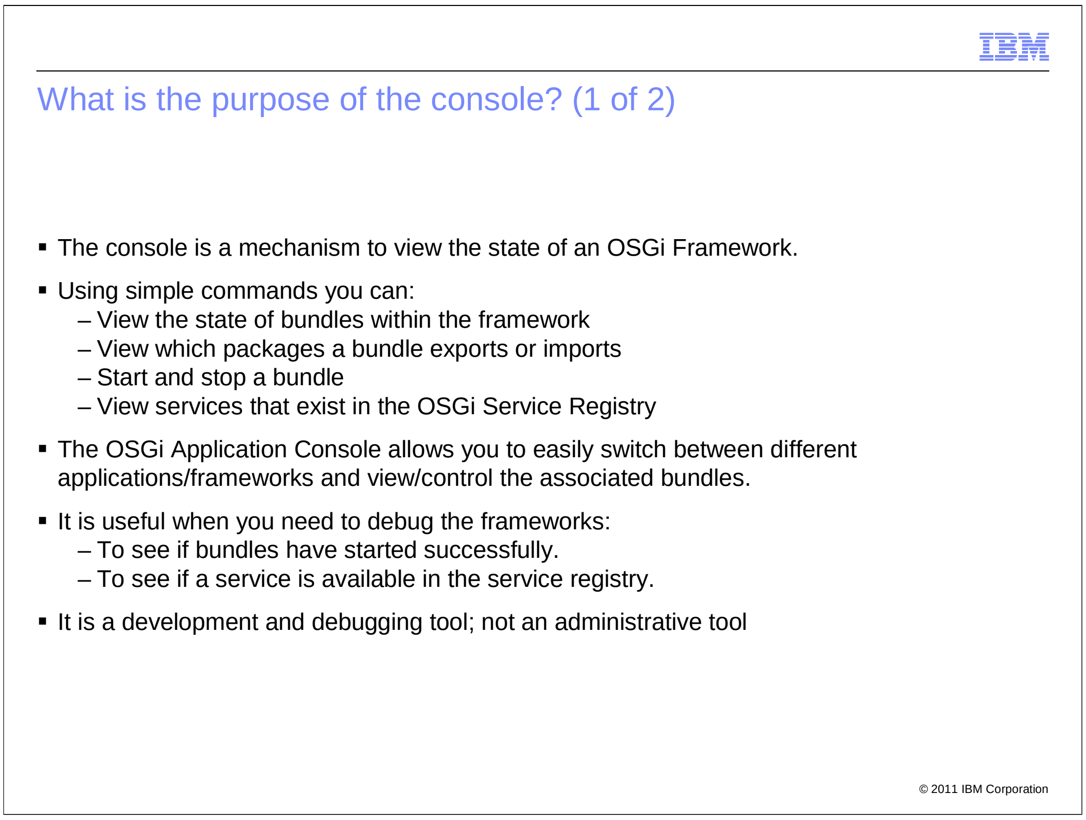
This module gives an overview of two troubleshooting tools supplied with the OSGi applications feature pack: the OSGi application console and the bundle cache manager MBean.

The OSGi Applications Console is more useful to developers and the Bundle Cache Manager MBean is more useful for administrators.

Section

OSGi application console

This section will cover the OSGi application console.



What is the purpose of the console? (1 of 2)

- The console is a mechanism to view the state of an OSGi Framework.
- Using simple commands you can:
  - View the state of bundles within the framework
  - View which packages a bundle exports or imports
  - Start and stop a bundle
  - View services that exist in the OSGi Service Registry
- The OSGi Application Console allows you to easily switch between different applications/frameworks and view/control the associated bundles.
- It is useful when you need to debug the frameworks:
  - To see if bundles have started successfully.
  - To see if a service is available in the service registry.
- It is a development and debugging tool; not an administrative tool

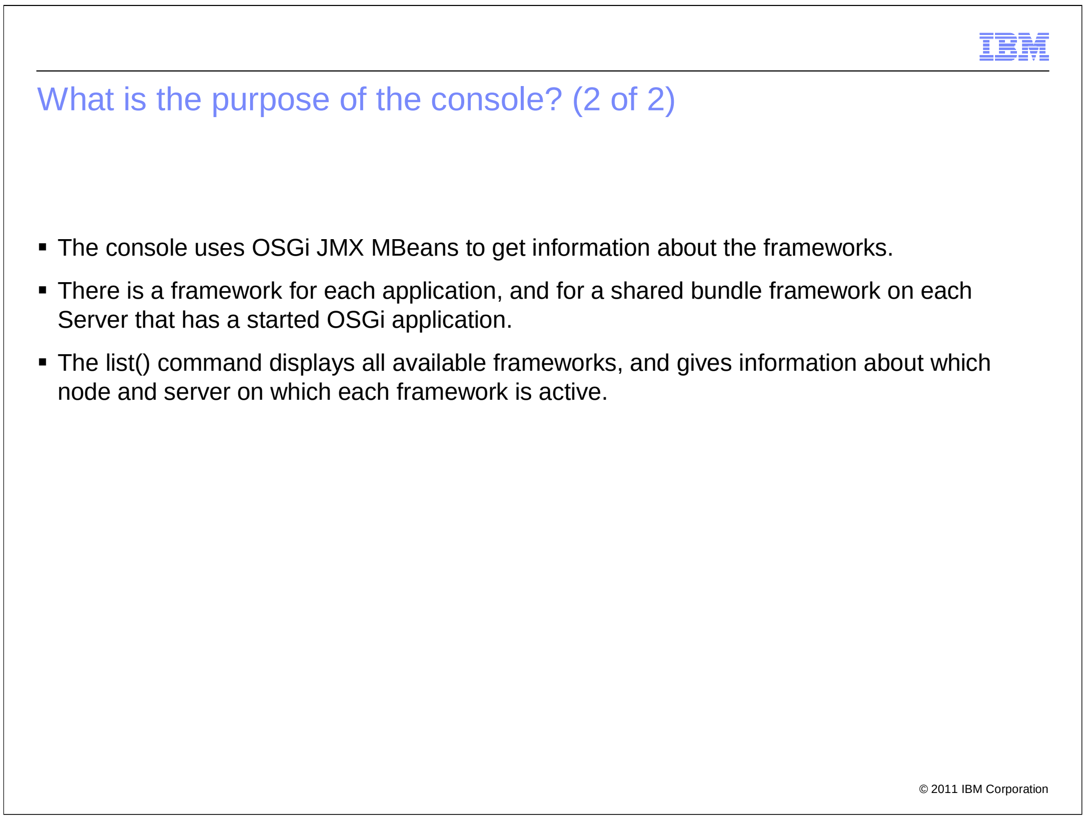
All the major OSGi framework implementations provide some way of introspecting the state of the framework. Typically this is in the form of a command line console to inspect bundles, services and try changes.

Similarly, for the OSGi applications feature there is a console to view the state of bundles, view packages exported or imported by bundles, trigger life cycle events like starting or stopping bundles, or view services in the registry or registered by a bundle.

This console provides introspection on a per application basis, for every OSGi application running. This console is invaluable for troubleshooting integration issues occurring only during deployment, for example bundles that appear not to have started, bundles that get incorrectly wired, blueprint bundles not started, missing service registrations and so on.

Note though that this tool is probably more for use by developers rather than administrators. The console should be used for tracking down hard to diagnose application bugs during development or integration testing.

Starting and stopping a bundle capability is provided for historical prescient and can be useful if you want to trigger events during a debugging scenario but this should not be used in a production environment.



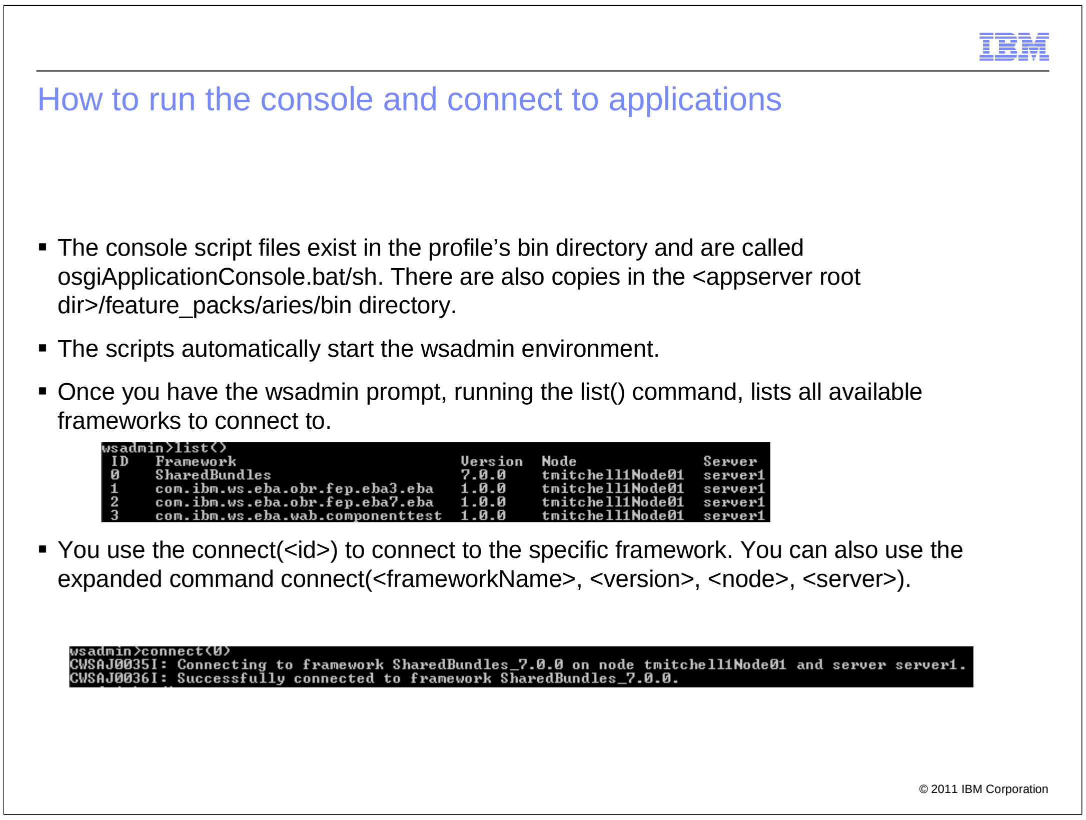
What is the purpose of the console? (2 of 2)

- The console uses OSGi JMX MBeans to get information about the frameworks.
- There is a framework for each application, and for a shared bundle framework on each Server that has a started OSGi application.
- The list() command displays all available frameworks, and gives information about which node and server on which each framework is active.

Under the covers the OSGi application console uses the JMX MBeans from the Apache Aries MBeans component.

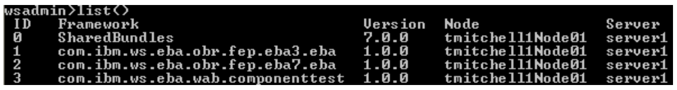
Every application runs in its own separate framework for each server. In addition there is a framework for all shared bundles (usually the bundles that get provisioned to satisfy unsatisfied package requirements).

To get an overview of all those frameworks use the list() command. This shows the active frameworks and the nodes and servers on which they run.



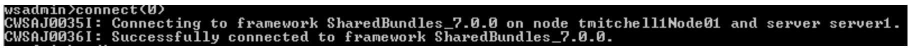
How to run the console and connect to applications

- The console script files exist in the profile’s bin directory and are called osgiApplicationConsole.bat/sh. There are also copies in the <appserver root dir>/feature_packs/aries/bin directory.
- The scripts automatically start the wsadmin environment.
- Once you have the wsadmin prompt, running the list() command, lists all available frameworks to connect to.



```
wsadmin>list()
 ID   Framework                          Version  Node              Server
 0    SharedBundles                      7.0.0    tmitchell1Node01  server1
 1    com.ibm.ws.eba.obr.fep.eba3.eba    1.0.0    tmitchell1Node01  server1
 2    com.ibm.ws.eba.obr.fep.eba7.eba    1.0.0    tmitchell1Node01  server1
 3    com.ibm.ws.eba.wab.componenttest   1.0.0    tmitchell1Node01  server1
```

- You use the connect(<id>) to connect to the specific framework. You can also use the expanded command connect(<frameworkName>, <version>, <node>, <server>).

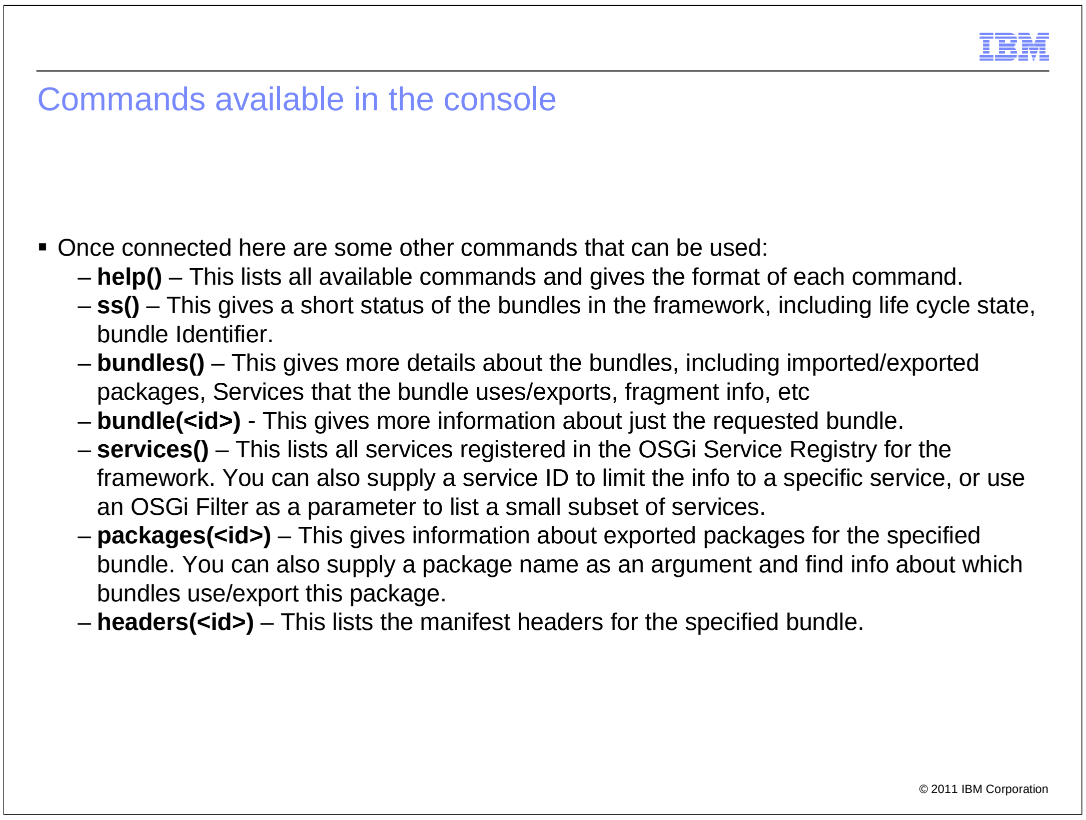


```
wsadmin>connect(0)
CWSAJ0035I: Connecting to framework SharedBundles_7.0.0 on node tmitchell1Node01 and server server1.
CWSAJ0036I: Successfully connected to framework SharedBundles_7.0.0.
```

The OSGi applications console resides in a WebSphere Application Server profile's bin directory or also in the feature packs installation directory shown on this slide. This script starts a wsadmin session (Jython syntax) with the additional commands for introspecting application OSGi frameworks.

The first screen capture shows the results of a “list()” command. In this particular example there are three applications installed in addition to the framework for shared bundles. To actually inspect one of these frameworks, one has to first select it. To do this, use the “connect(<id>)” command. This command also supports an extended syntax for specifying among other things the node and server name.

The connect command provides some diagnostic information, an example is shown in the second screen capture.



Commands available in the console

- Once connected here are some other commands that can be used:
  - **help()** – This lists all available commands and gives the format of each command.
  - **ss()** – This gives a short status of the bundles in the framework, including life cycle state, bundle Identifier.
  - **bundles()** – This gives more details about the bundles, including imported/exported packages, Services that the bundle uses/exports, fragment info, etc
  - **bundle(<id>)** - This gives more information about just the requested bundle.
  - **services()** – This lists all services registered in the OSGi Service Registry for the framework. You can also supply a service ID to limit the info to a specific service, or use an OSGi Filter as a parameter to list a small subset of services.
  - **packages(<id>)** – This gives information about exported packages for the specified bundle. You can also supply a package name as an argument and find info about which bundles use/export this package.
  - **headers(<id>)** – This lists the manifest headers for the specified bundle.

The power of the console is the commands to inspect an application's framework. These are modeled after the commands supported by the OSGi console included in Eclipse Equinox with very similar semantics.

First of all there is the built in help command. Then there are “bundles” and “ss” to inspect bundles: you can look at the state of bundles, the bundles themselves, and headers to find out more about a particular bundle. Whereas “packages” and “services” supply information again about the global state of the framework.

A typical session for checking persistence related problems might for example go like this.

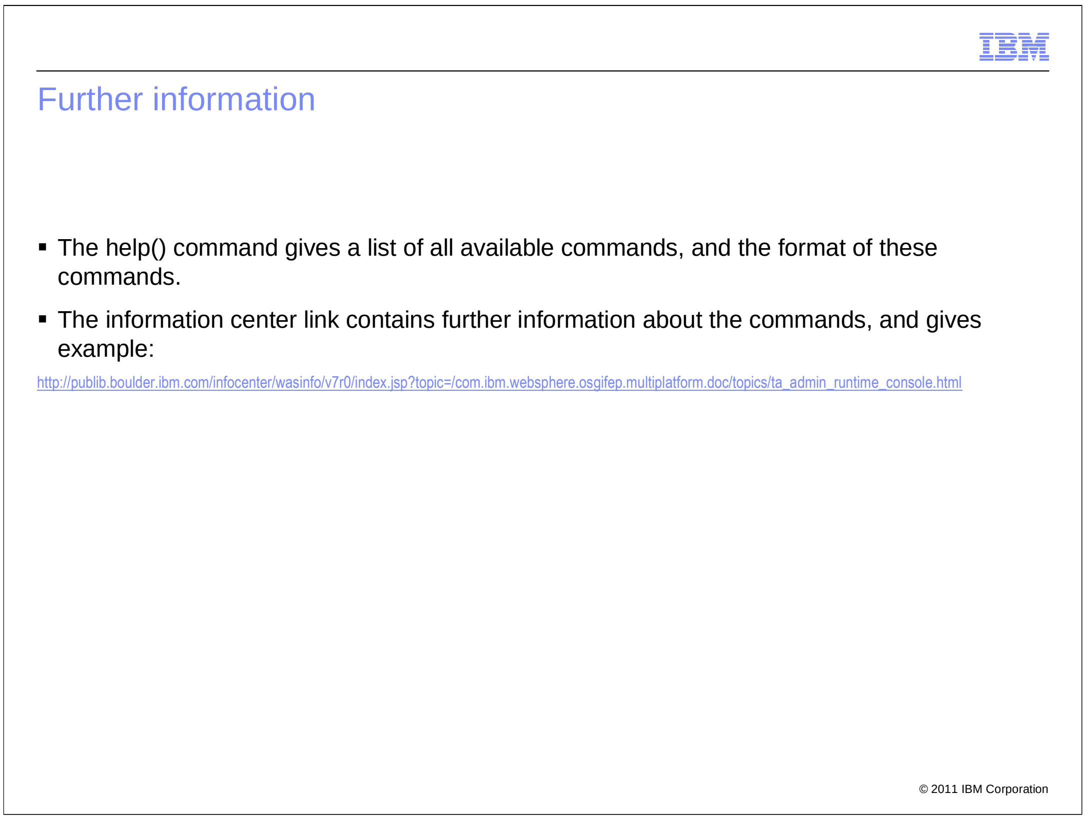
Do a list() to see what framework corresponds to the application.

Do a connect(<id>) to it.

Do an ss() to see what ID the persistence bundle is assigned.

Do a bundle(<persistence bundle id>) to inspect the persistence bundle, in particular this will also list the registered services, which for a persistence bundle should be various persistence unit services.

Do a services() to inspect services in the registry for example DataSource services that could explain why the JPA services are not there, and so on.

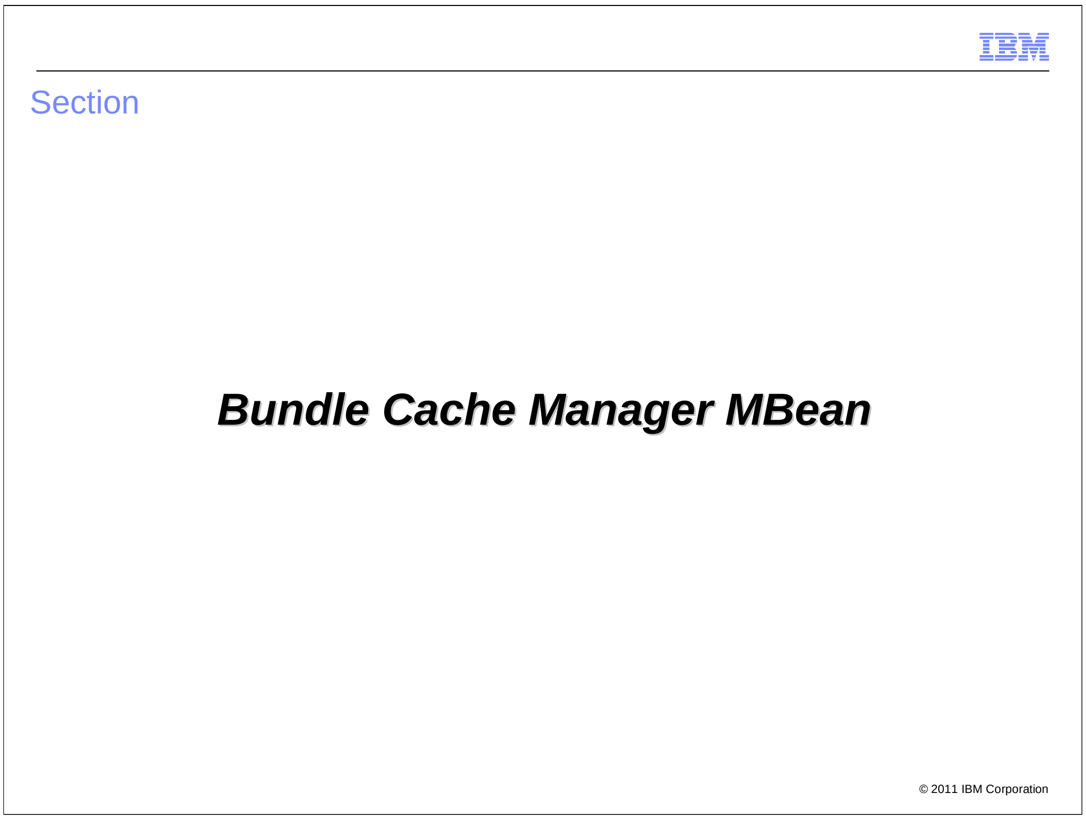


Further information

- The help() command gives a list of all available commands, and the format of these commands.
- The information center link contains further information about the commands, and gives example:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.osgifep.multiplatform.doc/topics/ta_admin_runtime_console.html

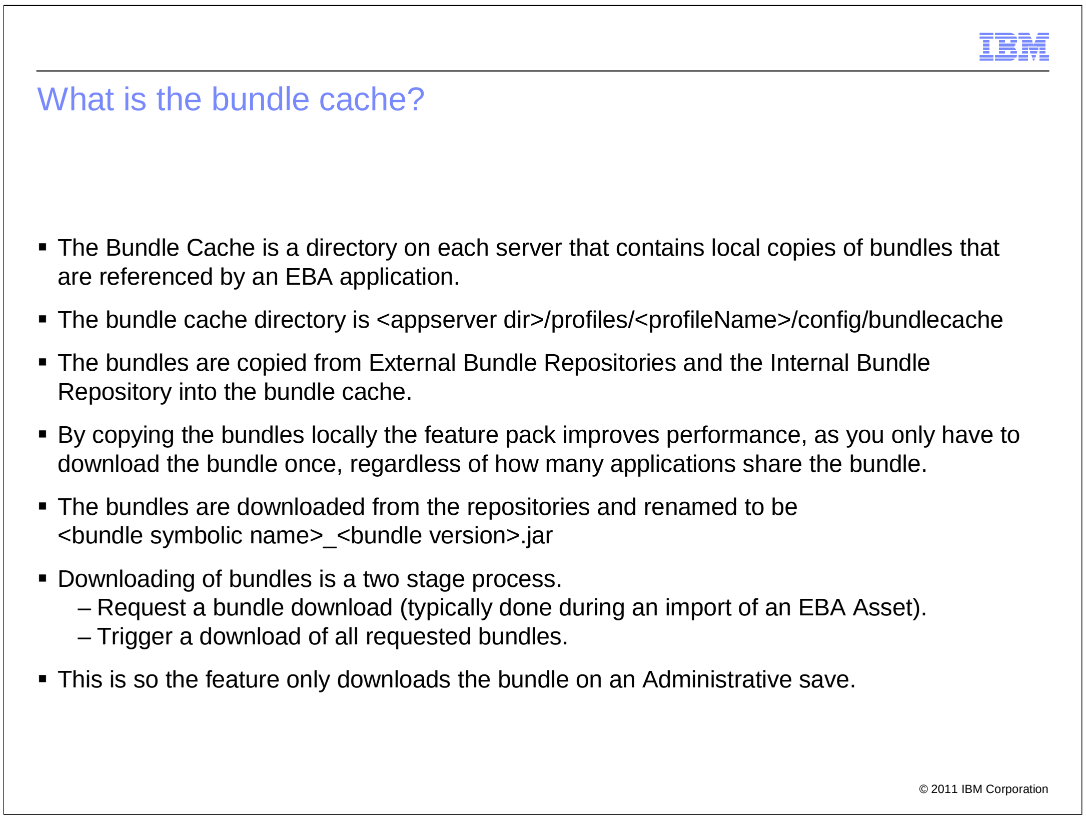
For additional information the best two sources are the “help()” command mentioned previously and the information center.



Section

Bundle Cache Manager MBean

This section will discuss the Bundle Cache Manager MBean



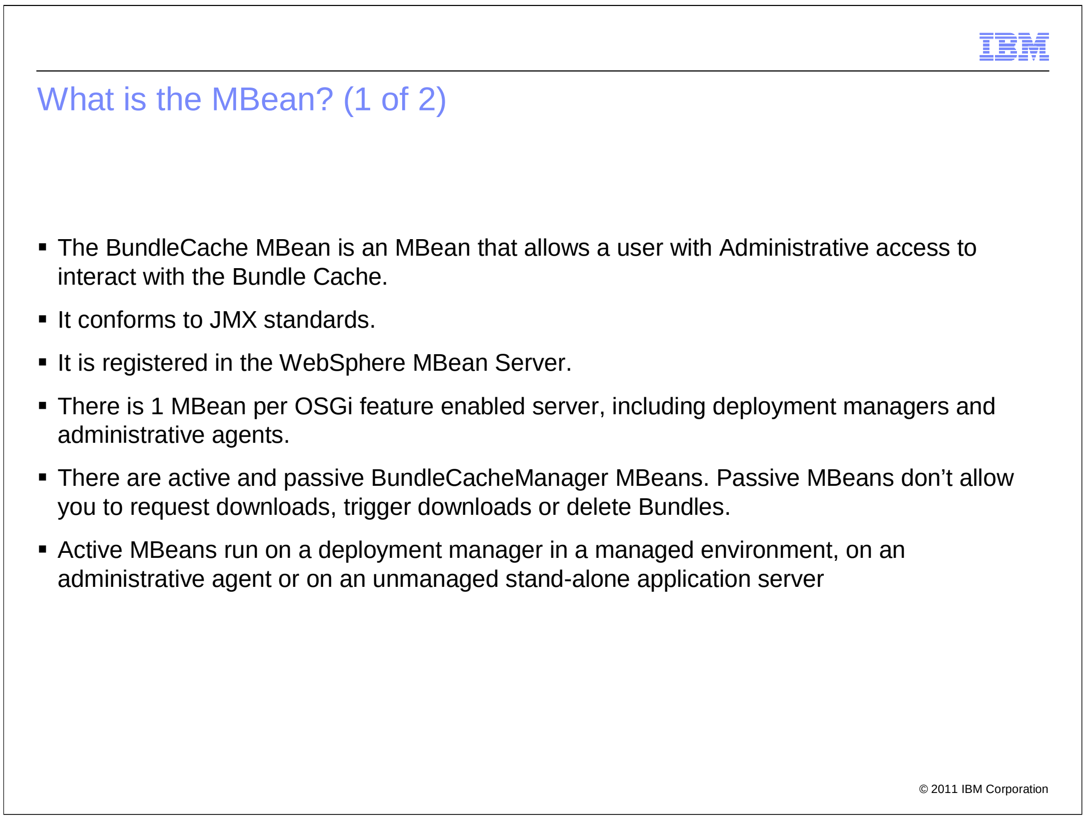
What is the bundle cache?

- The Bundle Cache is a directory on each server that contains local copies of bundles that are referenced by an EBA application.
- The bundle cache directory is <appserver dir>/profiles/<profileName>/config/bundlecache
- The bundles are copied from External Bundle Repositories and the Internal Bundle Repository into the bundle cache.
- By copying the bundles locally the feature pack improves performance, as you only have to download the bundle once, regardless of how many applications share the bundle.
- The bundles are downloaded from the repositories and renamed to be <bundle symbolic name>_<bundle version>.jar
- Downloading of bundles is a two stage process.
  - Request a bundle download (typically done during an import of an EBA Asset).
  - Trigger a download of all requested bundles.
- This is so the feature only downloads the bundle on an Administrative save.

The bundle cache, as the name suggests, is a local cache for bundles that come from an internal or external repository. To use a repository bundle for configuring an application bundle or loading it into the runtime it needs to reside locally on the server. For this purpose it needs to be downloaded, to the cache. Note that all applications share bundles in the cache and furthermore unlike other caches, the bundle cache does not by default get invalidated.

The bundle cache lives in a server's profile directory under config/bundlecache. It contains several bundles, canonically named as <symbolic name>_<version>.jar.

As discussed in the administration module, bundle downloads are requested during the import asset operation but they are not kicked off until the workspace changes are saved.



What is the MBean? (1 of 2)

- The BundleCache MBean is an MBean that allows a user with Administrative access to interact with the Bundle Cache.
- It conforms to JMX standards.
- It is registered in the WebSphere MBean Server.
- There is 1 MBean per OSGi feature enabled server, including deployment managers and administrative agents.
- There are active and passive BundleCacheManager MBeans. Passive MBeans don't allow you to request downloads, trigger downloads or delete Bundles.
- Active MBeans run on a deployment manager in a managed environment, on an administrative agent or on an unmanaged stand-alone application server

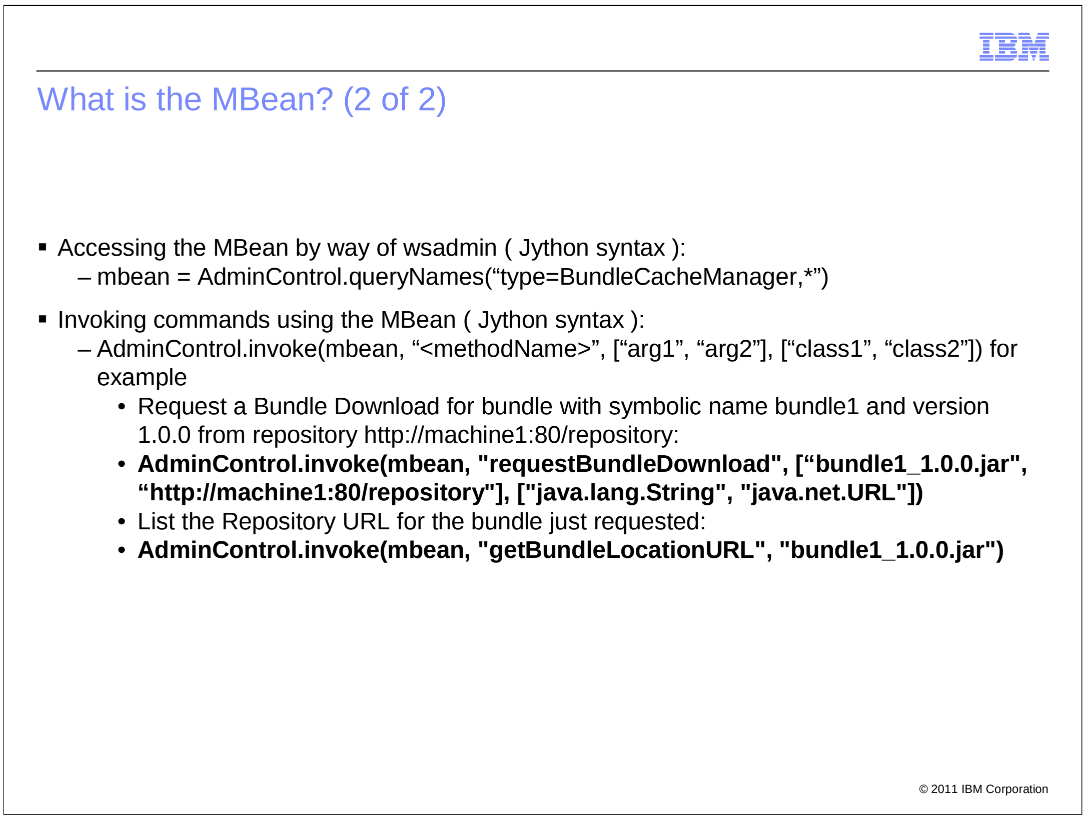
The OSGi application feature provides an MBean for inspecting the bundle cache manager's state and triggering operations on it.

Almost always this should not be necessary, however if and when a bundle download actually goes wrong this functionality can prove invaluable.

The MBean conforms to the typical JMX conventions and is registered with the WebSphere MBean server like usual. There is a single MBean on each server augmented with OSGi feature pack, regardless of whether the server is a stand-alone server, a deployment manager, or an administrative agent.

However, there are two different types, active and passive. Passive MBeans have certain restrictions like not allowing new download requests and so on.

Active MBeans run on administrative systems and have full capabilities, that means there is an active cache manager on a deployment manager or a stand-alone server, also there is one active MBean per managed node on administrative agent.



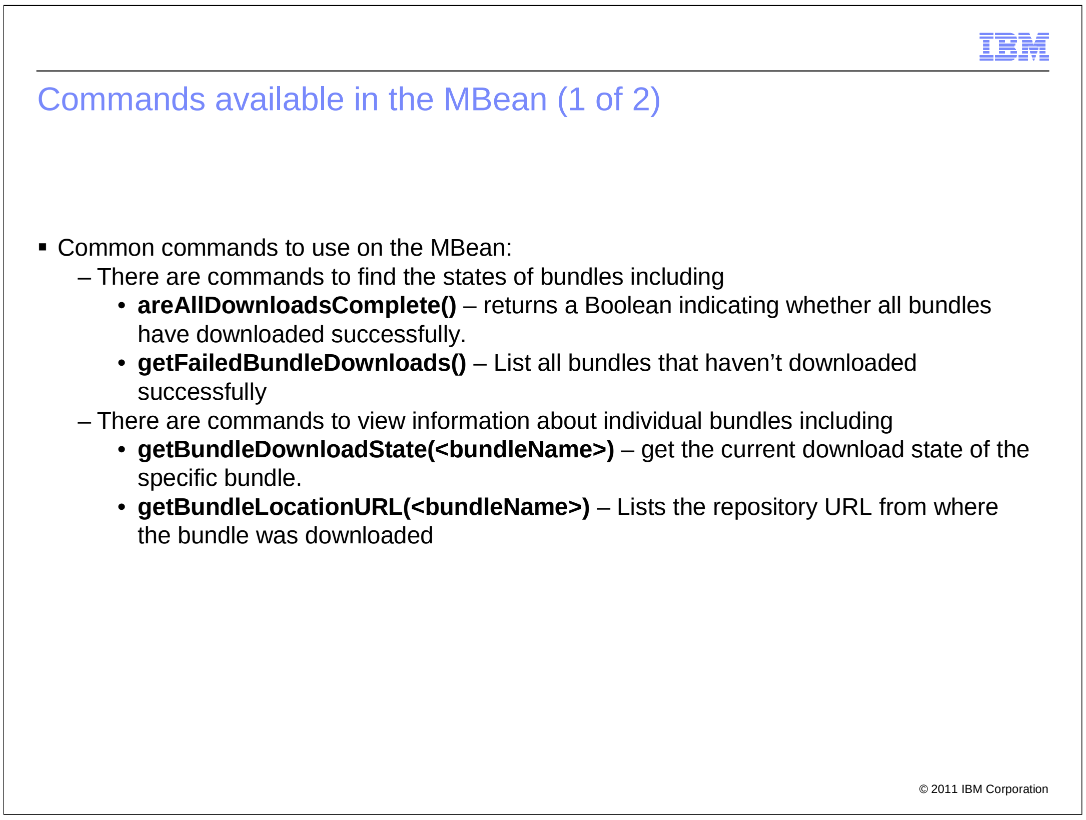
What is the MBean? (2 of 2)

- Accessing the MBean by way of wsadmin (Jython syntax):
  – mbean = AdminControl.queryNames(“type=BundleCacheManager,*”)
- Invoking commands using the MBean (Jython syntax):
  – AdminControl.invoke(mbean, “<methodName>”, [“arg1”, “arg2”], [“class1”, “class2”]) for example
    - Request a Bundle Download for bundle with symbolic name bundle1 and version 1.0.0 from repository http://machine1:80/repository:
    - **AdminControl.invoke(mbean, "requestBundleDownload", [“bundle1_1.0.0.jar", “http://machine1:80/repository"], ["java.lang.String", "java.net.URL"])**
    - List the Repository URL for the bundle just requested:
    - **AdminControl.invoke(mbean, "getBundleLocationURL", "bundle1_1.0.0.jar")**

In a typical wsadmin session with Jython syntax the single MBean is obtained as shown in the first snippet.

On all but administration agent this will give the single MBean. On an administrative agent the individual MBeans carry a profile root attribute that allows you to distinguish MBeans for different managed nodes.

To invoke commands, use the AdminControl.invoke operation. The two snippets at the bottom of the slide show example commands to request a download and check the download URL of the new request.

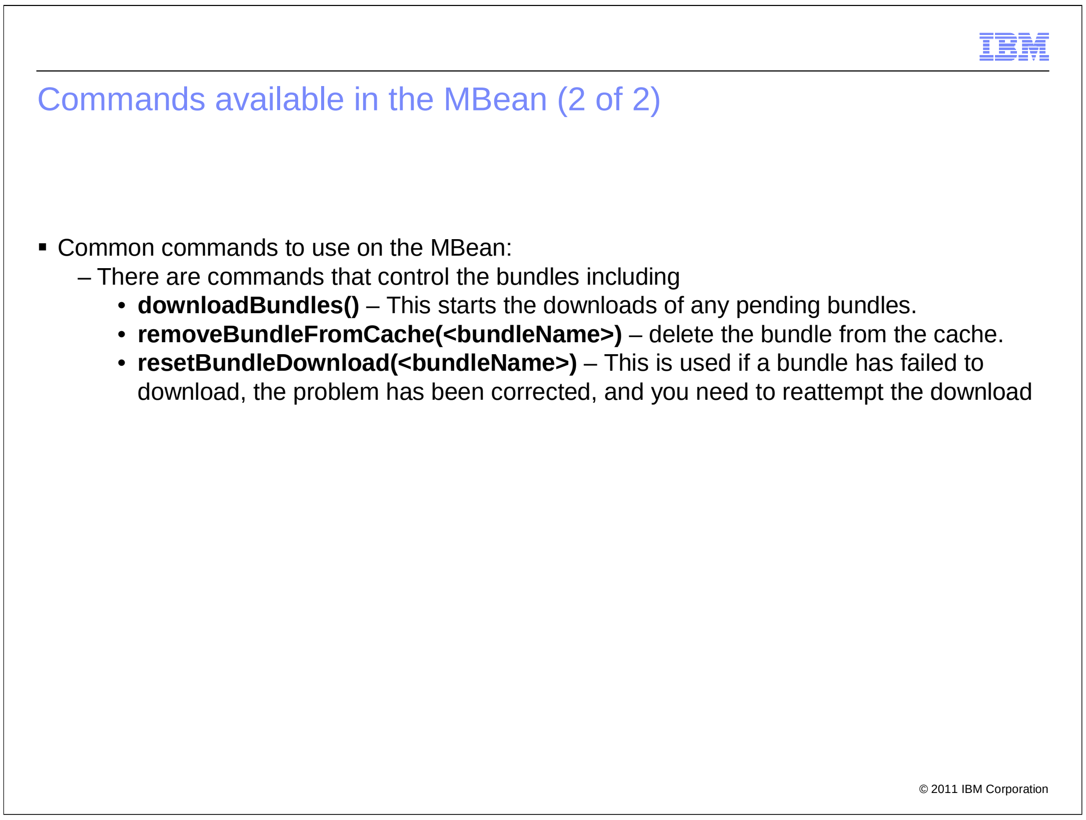


Commands available in the MBean (1 of 2)

- Common commands to use on the MBean:
  - There are commands to find the states of bundles including
    - **areAllDownloadsComplete()** – returns a Boolean indicating whether all bundles have downloaded successfully.
    - **getFailedBundleDownloads()** – List all bundles that haven't downloaded successfully
  - There are commands to view information about individual bundles including
    - **getBundleDownloadState(<bundleName>)** – get the current download state of the specific bundle.
    - **getBundleLocationURL(<bundleName>)** – Lists the repository URL from where the bundle was downloaded

The list of all commands on the bundle cacher manage MBean is well documented in the information center (link on the last slide).

A number of information commands are shown on this slide. For example, the getFailedBundleDownloads() command can be used to determine the cause why an .eba asset cannot be added to BLA and the asset detail panel reads downloads failed. After that the getBundleLocationURL command can be used to show the, potentially incorrect, location of the bundle binary.



Commands available in the MBean (2 of 2)

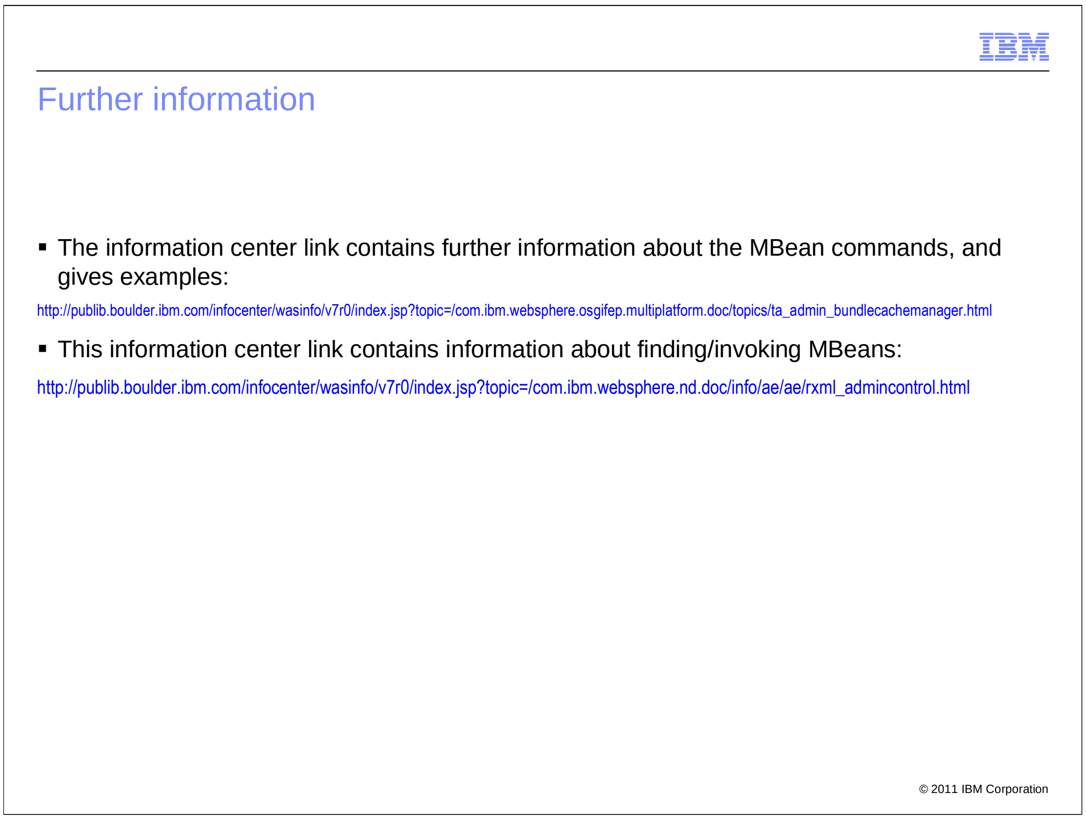
- Common commands to use on the MBean:
  - There are commands that control the bundles including
    - **downloadBundles()** – This starts the downloads of any pending bundles.
    - **removeBundleFromCache(<bundleName>)** – delete the bundle from the cache.
    - **resetBundleDownload(<bundleName>)** – This is used if a bundle has failed to download, the problem has been corrected, and you need to reattempt the download

In addition to troubleshooting and information gathering commands, there are a few commands to clean up potential messes.

Most importantly for temporary network failures the resetBundleDownload and downloadBundles commands are useful. For failed downloads these should always be used in conjunction.

If a download URL is outright wrong, then removeBundleFromCache should be used to remove any reference to the incorrect URL and then the bundle can be re-downloaded either through the MBean or by re-importing the asset that triggered the download initially.

Note that removing bundles from the cache is very dangerous for bundles that have correctly been downloaded. In such a case the administrator must make sure that no application uses the bundle to be removed.



Further information

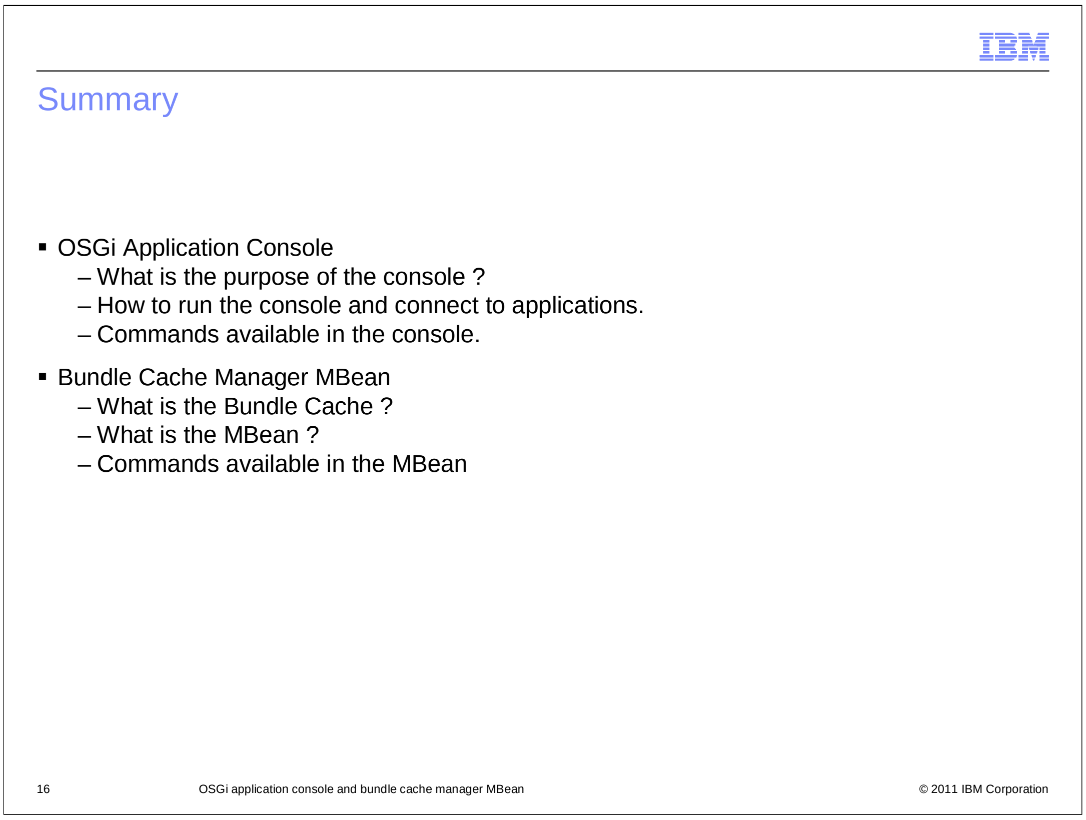
- The information center link contains further information about the MBean commands, and gives examples:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.osgifep.multiplatform.doc/topics/ta_admin_bundlecachemanager.html

- This information center link contains information about finding/invoking MBeans:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/rxml_admincontrol.html

Some further information about the bundle cache manager in particular and MBeans in general can be found in the information center.

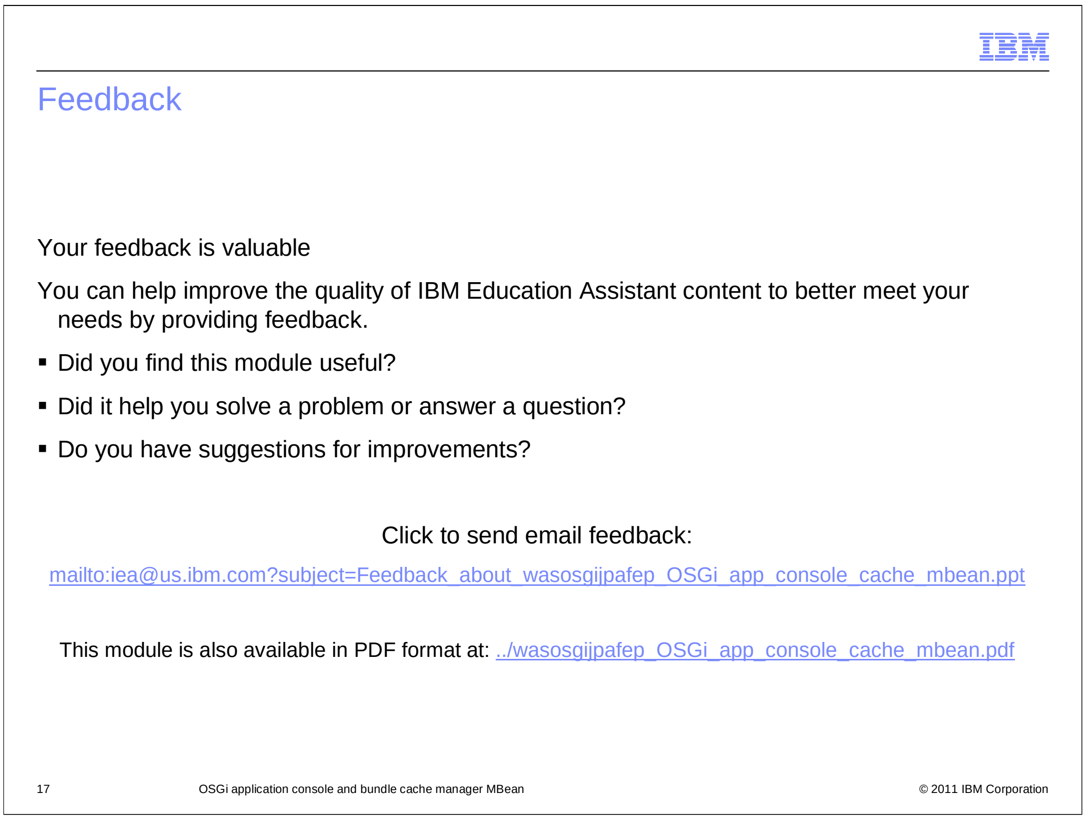


Summary

- OSGi Application Console
  - What is the purpose of the console ?
  - How to run the console and connect to applications.
  - Commands available in the console.
- Bundle Cache Manager MBean
  - What is the Bundle Cache ?
  - What is the MBean ?
  - Commands available in the MBean

16 OSGi application console and bundle cache manager MBean © 2011 IBM Corporation

In this module two tools were discussed. First the OSGi Application console which is helpful to developers when debugging applications and secondly the Bundle Cache Manager MBean which is of particular use to administrators to view information about the bundle cache and is currently the only way to clean a bundle out of the bundle cache.



Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

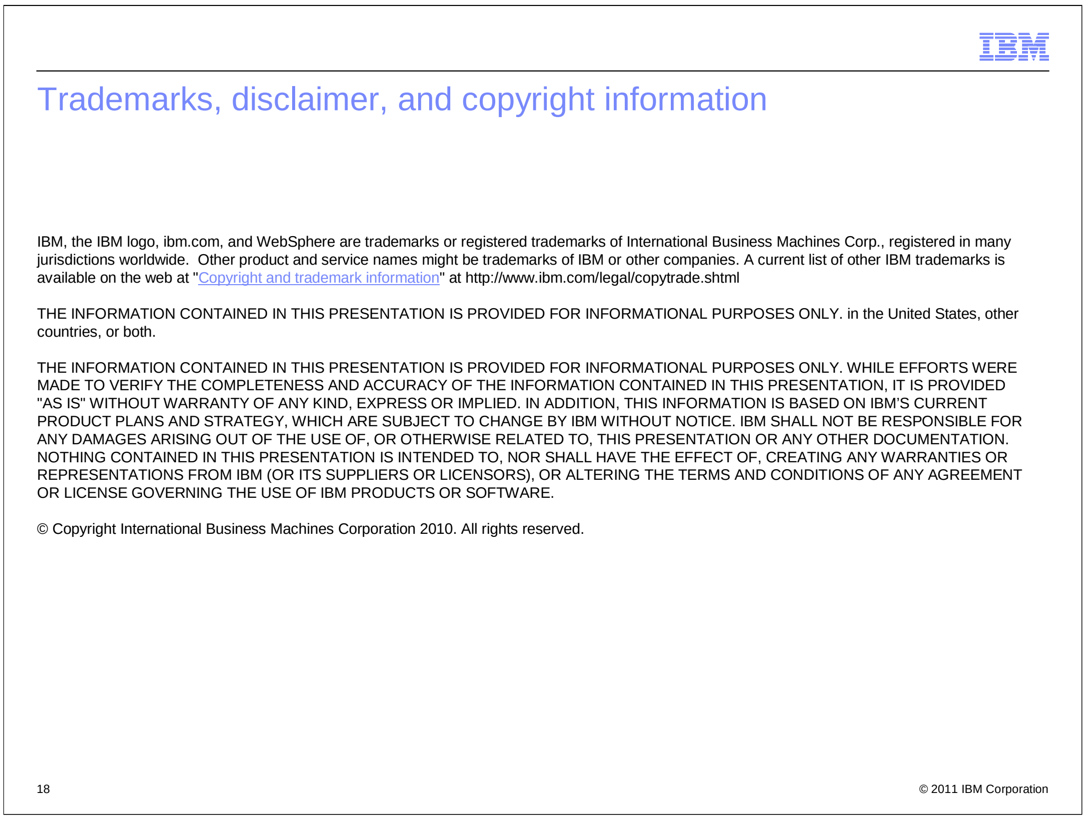
Click to send email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_wasosgijpafep_OSGi_app_console_cache_mbean.ppt

This module is also available in PDF format at: ../wasosgijpafep_OSGi_app_console_cache_mbean.pdf

17 OSGi application console and bundle cache manager MBean © 2011 IBM Corporation

You can help improve the quality of IBM Education Assistant content by providing feedback.



Trademarks, disclaimer, and copyright information