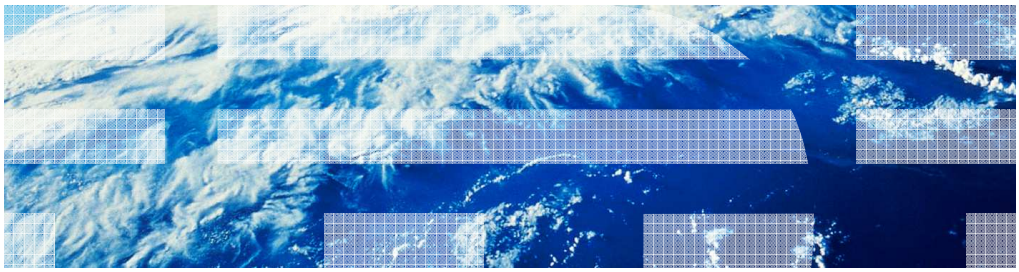
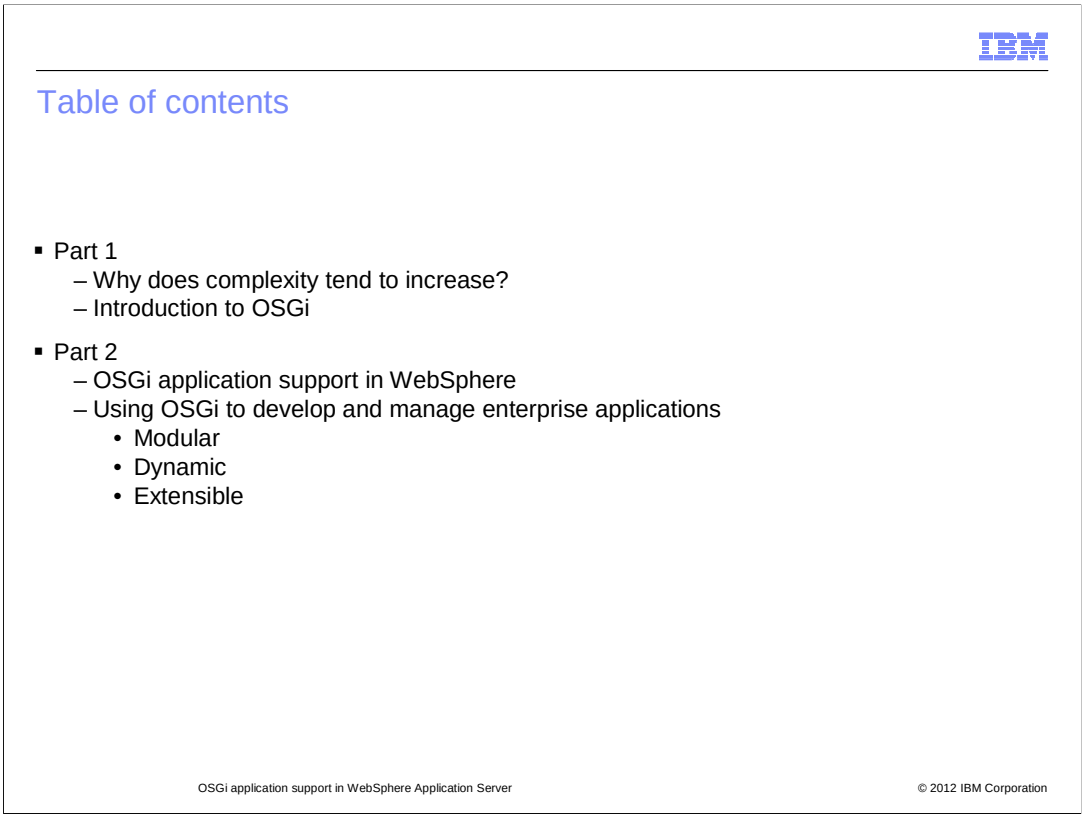
WebSphere Application Server V8.5

Modular and dynamic OSGi applications
Part 2: OSGi application support in WebSphere Application Server



This presentation covers modular and dynamic OSGi applications in WebSphere Application Server. This is Part two - OSGi applications support in WebSphere Application Server.



This is the second part of a two part session that looks at the WebSphere Application Server OSGi Applications feature, introduced in version seven and extended in version 8.

Part one described some common problems experienced with application deployments using Java Enterprise Edition (or Java EE for short), and how OSGi and Java Enterprise Edition have come together over the last two years in standards and in open source.

This part looks at the concept of an OSGi application and how this is supported in WebSphere Application Server to produce modular, dynamic, and extensible applications based on OSGi and Java Enterprise Edition technology.

OSGi applications in WebSphere Application Server

- First surfaced to applications in a WebSphere Application Server V7 Feature Pack
  - http://www-01.ibm.com/software/webservers/appserv/was/featurepacks/
  - Modular development, deployment and management
  - Blueprint (Standardized Spring Component Model)
  - Web applications (Java EE 5)
  - Remote Services and Heterogeneous Assembly (SCA)
- Included in the WebSphere Application Server Base in V8 and added:
  - Java EE 6 web technologies
  - Post-deployment configuration
  - Performance metrics
  - In-place Update
  - Application extension
- Added in WebSphere Application Server V8.5
  - Modular Enterprise JavaBeans
  - Blueprint Role-based Security
  - OSGi Applications Web Console

OSGi application support in WebSphere Application Server

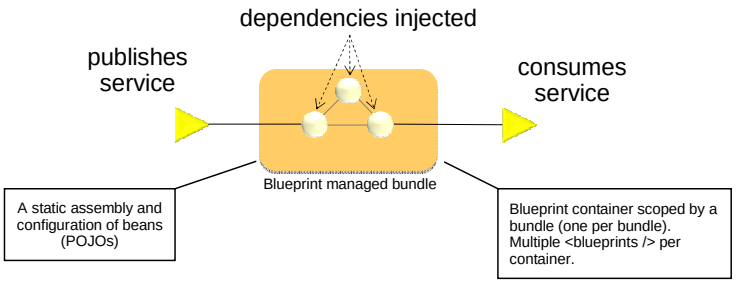
The WebSphere Application Server V7 Feature Pack for OSGi Applications and JPA 2.0 integrates the Apache Aries project with the WebSphere Application Server to provide support for OSGi based enterprise applications.

It introduced the Blueprint standardization of the Spring XML bean definition format, the ability to form web applications from OSGi bundles, and the ability to integrate with the Service Component Architecture for assembling JEE applications and OSGi applications together, and enabling OSGi remote services. It also includes basic support for application update and extension. The feature pack extends the Application Server to provide an end-to-end development, deployment, and administrative integration for OSGi Applications.

In WebSphere Application Server V8 the OSGi applications capability is included in the base product. It is enhanced to provide support for updating and extending applications while they continue to run. Configuration can be changed after application deployment. Performance metrics were added for OSGi applications, and there is support for using JEE 6 web technologies within OSGi applications.

In WebSphere Application Server V8.5 OSGi applications can take advantage of the Enterprise JavaBeans programming model. Blueprint has been extended to add the ability to specify for role-based security. As a debugging tool, through the administrative console, it now possible to view and navigate the details of running OSGi applications.

Blueprint components and service



- Dependency Injection container
  - Standardizing established Spring conventions
- Configuration and dependencies declared in Blueprint XML
  - Standardization of Spring “application context” XML
  - Extended for OSGi: publish/consume components as OSGi services
- Simplifies unit test outside either Java EE or OSGi runtime
- **Integrated into server runtime to simplify deployment & support**

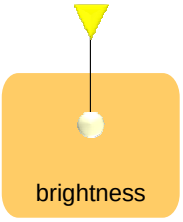
By standardizing the Spring XML configuration format in the OSGi Alliance and delivering the container as an OSGi bundle, it has become possible to pull the dependency injection container out of the application and into the middleware. The standards-based evolution of the dependency injection container is called the Blueprint container.

The Blueprint XML configuration file has the same structure as the Spring XML configuration file but in an OSGi namespace. The Blueprint XML is a bean definition file for all the beans provided by a single bundle. In addition to the bean definitions that is familiar to Spring developers, the Blueprint model adds new service and reference elements as part of the integration with the OSGi environment. Service elements direct the Blueprint container to expose a service interface for a component outside the bundle and a reference element directs the Blueprint container to locate a service that can be consumed from outside the bundle. The yellow arrows in the figure indicate OSGi services that are published to and consumed from the OSGi Service Registry by the blueprint container. The OSGi service registry is a standard part of OSGi and provides a mechanism akin to JNDI for the publication of OSGi services. The use of Blueprint abstracts application developers provides a simpler, declarative use of the OSGi services.

Ultimately, the Blueprint container manages the lifecycle and dependencies of the POJO beans that contain the application logic and the services and references each bundle provides, and ensures references are wired to available services.

Blueprint extensions

- Namespaces extend core component model
  - Transactions
  - Resource References
  - JPA integration
  - Security (new in V8.5)




```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:sec="http://www.ibm.com/appserver/schemas/blueprint/security/v1.0.0"
  xmlns:tx="http://aries.apache.org/xmlns/transactions/v1.0.0"
  xmlns:jpa="http://aries.apache.org/xmlns/jpa/v1.0.0">
  <service id="adjustmentService" ref="BrightnessDeltaBean"
    interface="colors.adjustment.api.BrightnessService">
  </service>
  <bean id="BrightnessDeltaBean"
    class="colors.brightness.BrightnessDelta">
    <sec:access-constraint method="setBrightnessDelta" role="adjuster" />
    <tx:transaction method="*" value="Required" />
    <jpa:context property="em" unitname="brightnessUnit"/>
  <property name="delta" value="100"/>
  </bean>
</blueprint>
```


The Blueprint specification provides an extension mechanism. This is used to provide declarative mechanisms for specifying additional properties to managed beans. The transaction extension enables methods to be included within a transaction and a way of specifying the scope of that transaction. The Java Persistence API extension provides a short-hand form of injecting an entity manager into a managed bean. The security extension provides a mechanism for securing access at the method level to a specific role which you can map to a role defined in the application server at the time you install the application.

EJB bundle (New in 8.5)

- EJB Bundle = EJB JAR + OSGi Metadata
- **Export-EJB:** Opt-in header for EJB Bundles
  - Existence: process bundle for EJBs
  - Absence: do not process bundle for EJBs, even if it contains them
- Header value governs registration of EJBs as OSGi services
  - Excludes message-driven and stateful beans
  - Best practice: only export EJBs to be shared outside bundle

Example	Meaning
Export-EJB:	Process all EJBs and register them as OSGi services
Export-EJB: BlogBiz, BlogPersistence	Process all EJBs, register BlogBiz and BlogPersistence as services if they exist
Export-EJB: NONE	Process all EJBs but don't register them as OSGi services

- EJBs run in the same WebSphere Application Server EJB container
- EJB 3 style supported (EJB 2 style not supported)
- Uses OSGi for classloading and lifecycle

From WebSphere Application Server V8.5 Enterprise JavaBeans can be deployed in OSGi bundles. Just as a Bundle is a JAR with extra OSGi metadata, an “EJB Bundle” is an EJB JAR with extra OSGi metadata.

A bundle that contains Enterprise JavaBeans is not processed as an EJB Bundle by default. An opt-in header indicates that Enterprise JavaBeans are present in the bundle and should be processed. If the value of the header is empty, all the EJBs in the bundle are processed and additionally registered as OSGi services in the OSGi service registry. Alternatively a list of comma separated EJB names can be specified that should be processed and registered as OSGi services. The special value ‘NONE’ means EJBs are processed, but not registered as OSGi services.

Example EJB bundle

- Local and remote EJBs supported
  - EJB 3 style
- Best practices:
  - Put interfaces and EJB implementations in separate bundles and separate packages
  - Annotate @Local or @Remote on the EJB implementation classes*
- Blue explicitly named in Export-EJB header:
  - Blue processed as an EJB and registered as a service
  - Other EJBs ignored
- Imports the packages it needs:
  - Colors API
  - EJB API

```
…
@Stateless
@Local(ColorService.class)
@LocalBean
public class Blue implements ColorService {

  private Color color = new Color(0, 0, 150);

  public Color getColor() {
    return color;
  }

}
```

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Acme order processing service
Bundle-SymbolicName: com.acme.order.service
Bundle-Version: 1.0.0
Export-EJB: Blue
Import-Package:
 com.provider.api;version="[1.0.0,1.1.0)",
 javax.ejb;version="3.1"
```

*annotations not processed outside the EJB bundle

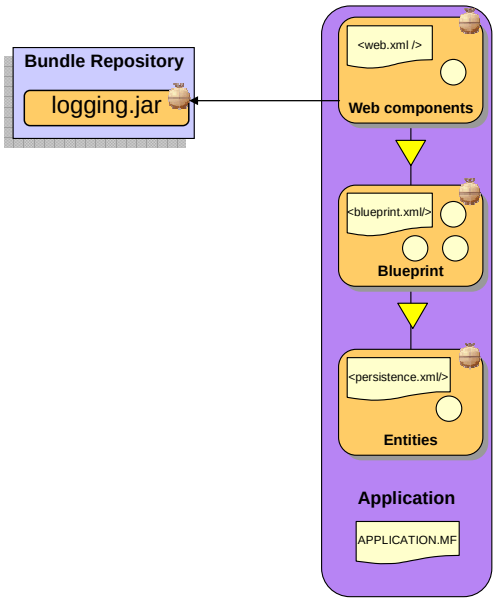
As an example, a class annotated as a LocalBean EJB, called “Blue,” implements a business interface called “ColorService.”

The OSGi best practice of separating interfaces from their implementations into separate bundles allows those implementations to be dynamically replaced at runtime within affecting the availability of the consumers of those interfaces. This best practice should be applied to the use of Enterprise JavaBeans in OSGi: business interfaces should be separated from their EJB implementations, into separate bundles.

In this example, the Export-EJB specifies “Blue” to be processed and registered in the OSGi service registry against the ColorService interface.

Application-level metadata and archive

- An isolated, cohesive application consisting of a collection of bundles, is deployed as a logical unit in a ".eba" archive
  - An "OSGi Application".
- Constituent bundles are listed in the **APPLICATION.MF** and can be contained ("by-value") in the archive
  - if referred to, then they are provisioned from a bundle repository at deployment time
- Services provided by the bundles in the application are isolated to the application unless explicitly exposed through EBA-level application manifest
- Configuration by exception – optional application-level metadata in APPLICATION.MF

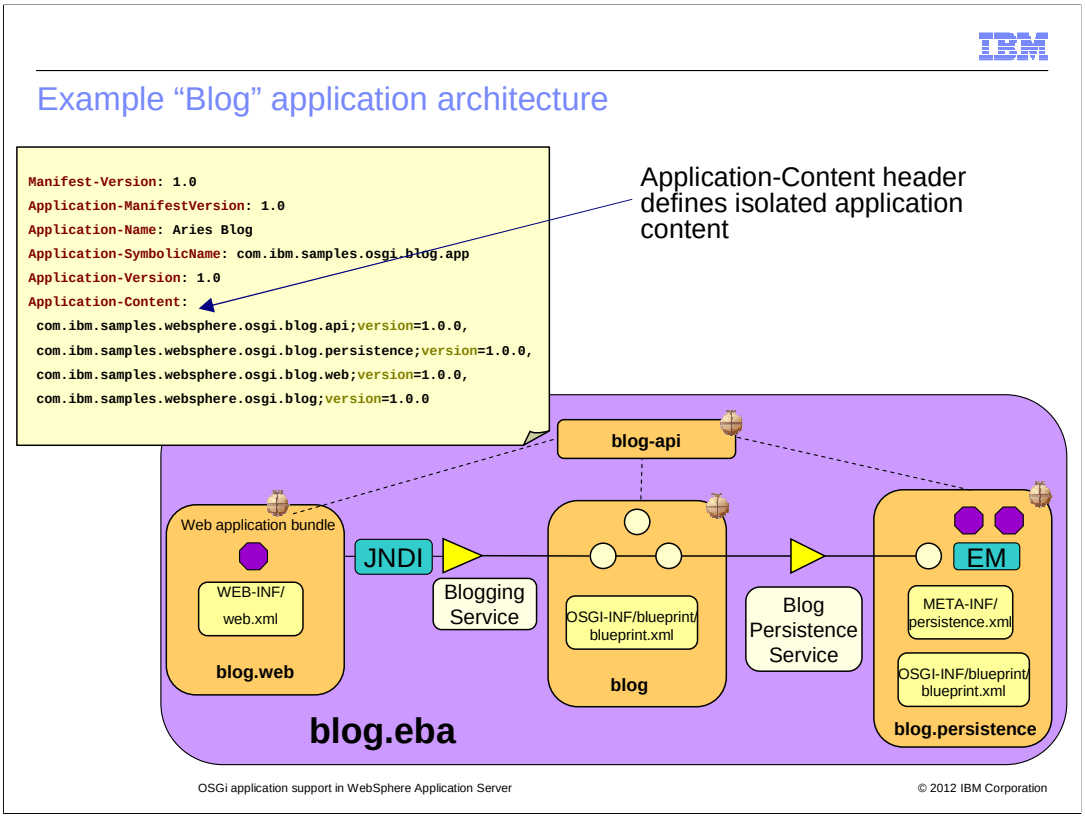


OSGi applications are deployed to WebSphere Application Server through wsadmin or with the administrative console just like any other application but are packaged in a new type of archive called an "enterprise bundle archive" or "EBA" archive. This is similar to a JEE Enterprise Application Archive (or 'EAR') except that its modules are deployed as bundles to the required target servers. An EBA archive represents a single isolated OSGi application consisting of one of more bundles and is the unit of deployment for an enterprise OSGi application. Like an EAR file, an EBA archive can contain all the constituent modules that make up the application but it may just contain the metadata required to locate those bundles from a configured bundle repository.

The metadata is in the form of an EBA-level "APPLICATION MANIFEST" file that describes the content of the application and whether the application exposes any external services and references. Just like a bundle manifest describes the modularity characteristics of a bundle, the application manifest describes the modularity characteristics of the application and the deployable content of the application.

The example here shows an OSGi Application consisting of a web application bundle (or WAB) providing the UI content, a blueprint bundle providing the business logic and a persistence bundle encapsulating entities that are persisted through the Java Persistence API to a relational database. The WAB depends on a common 'logging' library that is not packaged as part of the archive but provisioned from a common bundle repository.

Configuration is by exception – the absence of an APPLICATION MANIFEST indicates that all application content is contained within the archive and the application exposes no services or references externally.



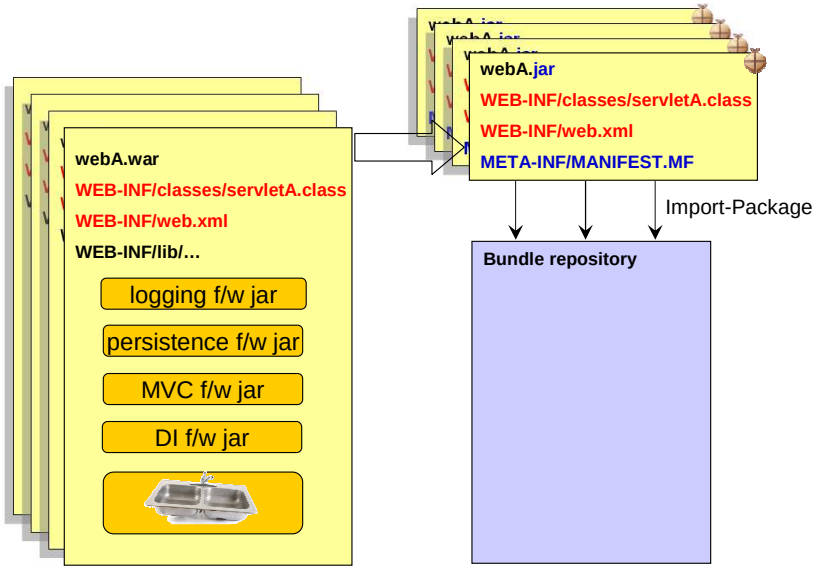
The APPLICATION MANIFEST here shows how isolated content is defined by the Application-Content header.

The figure describes one of the sample applications shipped with the OSGi Application feature pack. It is a web application that provides a Blog. It consists of: A web bundle to provide the user interface through standard servlets and Dojo. A blueprint bundle containing three beans that encapsulate the business logic. The entry point is a Blogging Service that is accessed through JNDI by the web application. A persistence bundle containing a standard persistence.xml and entities representing the persistent data. And a database where blog entries and author information are read from and written to through JPA.

Enhanced modular deployment process (1 of 2)

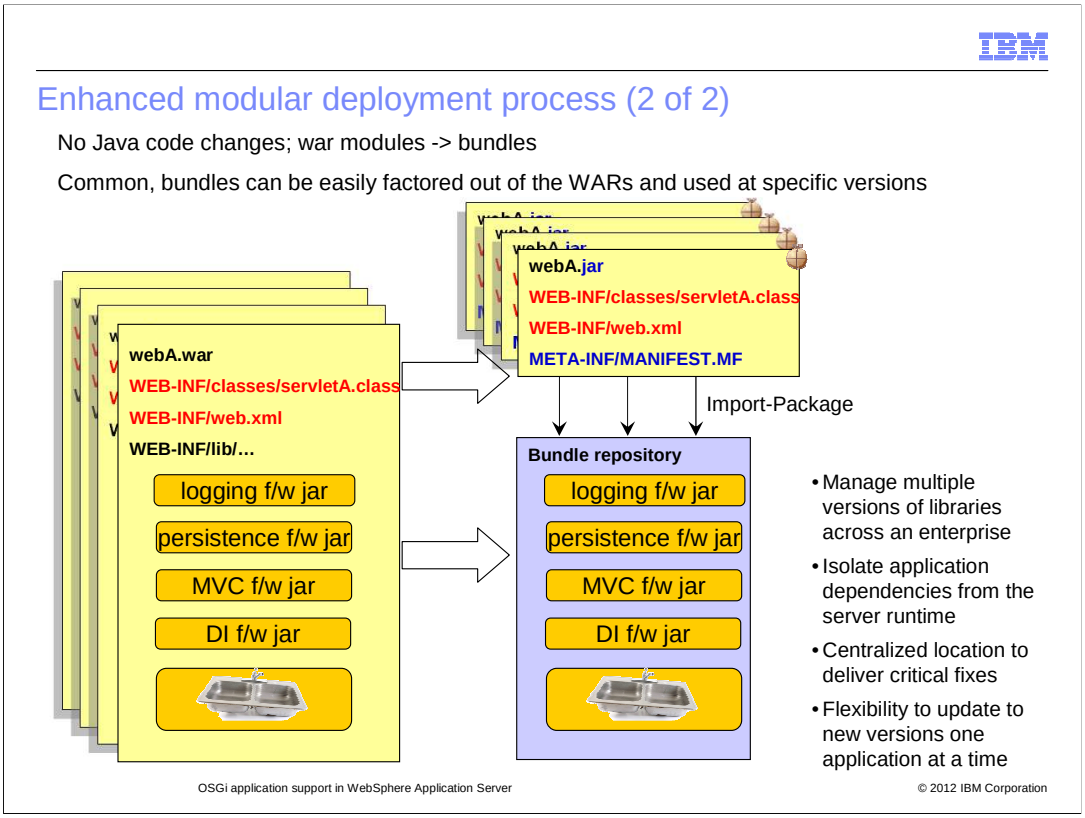
No Java code changes; war modules -> bundles

Common, bundles can be easily factored out of the WARs and used at specific versions



Getting started with OSGi application support in WebSphere Application Server is made very simple with no need to make any changes to a web application implementation.

Web Application Archives, also known as W.A.R modules, can be deployed to WebSphere Application Server as web application bundles with no change to their runtime behavior. When you have multiple applications that use common libraries they are often placed within each WAR file that uses them. With a small amount of refactoring, you can place versioned, common libraries in an OSGi bundle repository so that each application using these libraries delivers only their unique modules and are wired to the common libraries at deployment time.

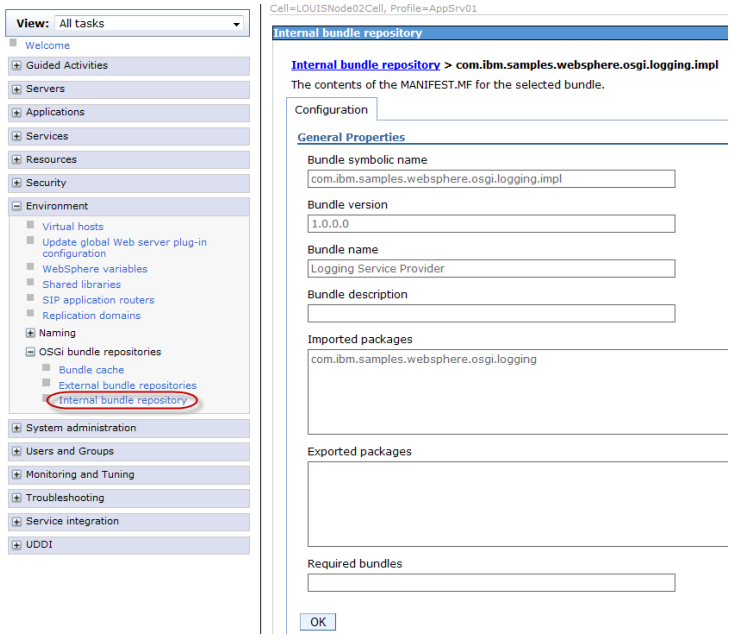


Remember- a bundle is just a JAR with additional OSGi metadata and a class loader that respects that metadata. Throughout this presentation the bundle symbol is used to indicate a JAR that is actually a bundle.

In this illustration four web application archives, which each include several common libraries, are refactored as four web application bundles containing only the unique content. The common libraries are installed once into an OSGi bundle repository.

The use of a bundle repository enables you to manage multiple versions of libraries across your enterprise, providing a central location to deliver critical fixes and the flexibility to update bundles used by the applications, one application at a time.

Bundle repository for modular deployment

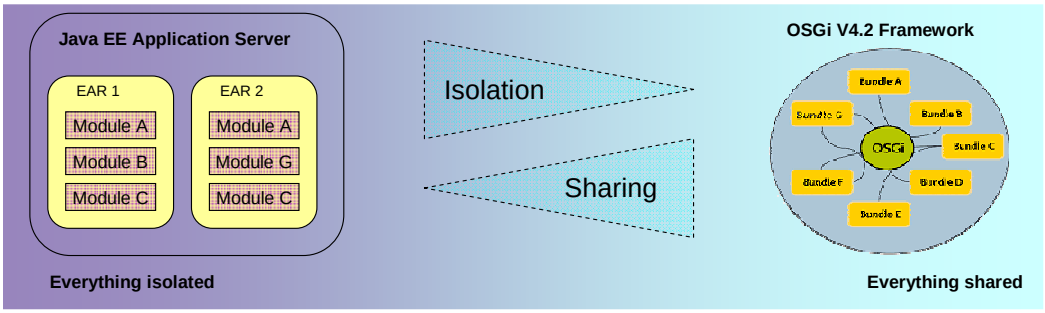


WebSphere Application Server provides administrative support for using OSGi bundle repositories to simplify the deployment of applications that use common libraries. WebSphere Application Server can be configured either with the locations of external bundle repositories or can use an internal bundle repository provided in the product. External bundle repositories provide their own tools for populating them and maintaining their content; the WebSphere Application Server internal OSGi bundle repository is managed through WebSphere administration with the administrative console or wsadmin scripting.

Common bundles can be installed once into the configured OSGi Bundle repository and used by many applications, reducing both disk usage and memory footprint.

Isolated and shared bundles (1 of 2)

- In Java EE, modules are isolated within an application and applications are isolated from one another.
  - Makes sharing modules difficult
- In OSGi all bundles have shared visibility to the externals of all others bundles within an OSGi framework (JVM)
  - Makes isolating applications difficult



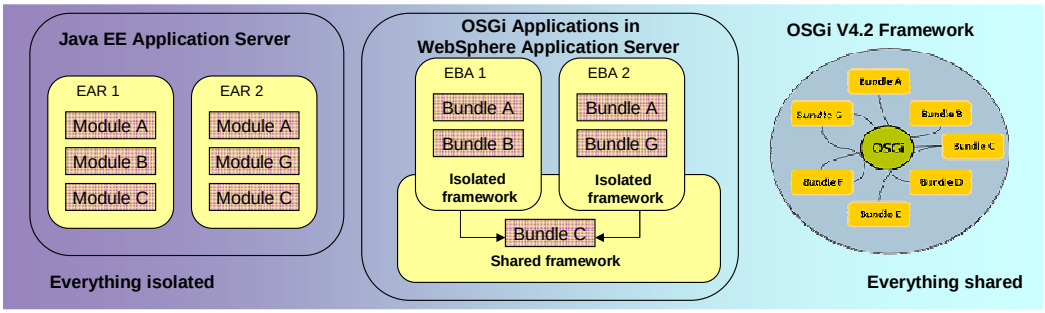
Application Isolation is an important consideration. At one extreme, Java EE provides no portable notion of module sharing between enterprise applications – everything is isolated and sharing libraries is difficult.

At the other extreme, OSGi bundles have shared visibility to the externals of all other bundles within an OSGi framework, which typically means within a JVM. This makes isolating applications difficult.

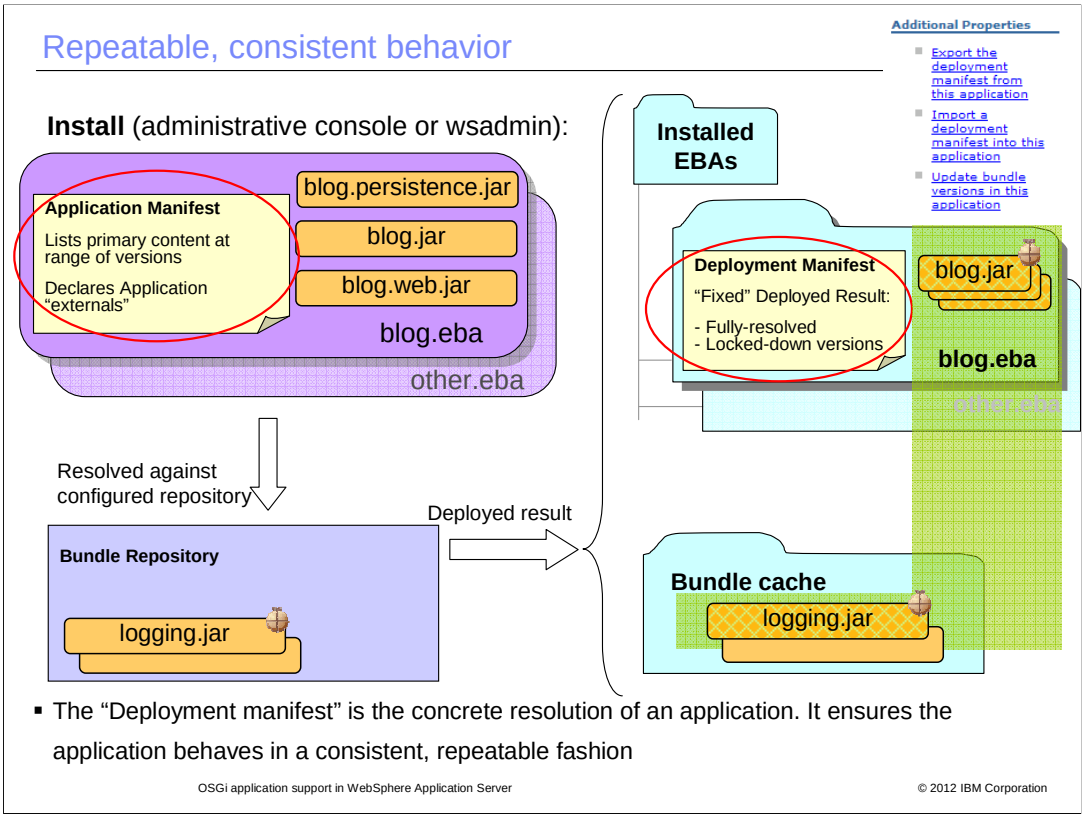
Something in between is needed.

Isolated and shared bundles (2 of 2)

- The Equinox OSGi framework used by WebSphere Application Server enables “composite bundles” to run in isolated child frameworks
  - WebSphere installs each OSGi application into an isolated child framework
  - Shared bundles are installed into the (single) parent framework



WebSphere Application Server achieves a mixture of isolation and sharing within applications by effectively providing each OSGi application with its own OSGi framework. Bundles that are not to be shared across applications run in an application’s “isolated framework.” Bundles that are to be shared run in a single “shared framework” whose contents are visible to all running applications.

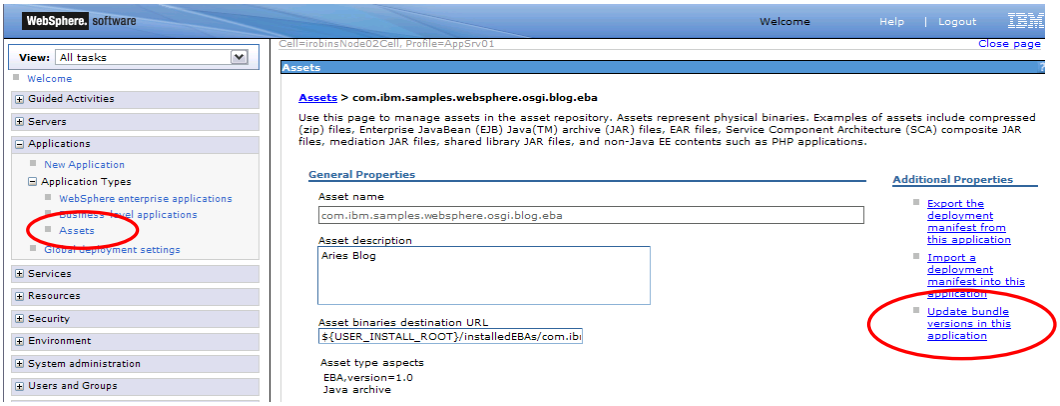


The optional application manifest is authored by the assembler (for example using Rational Application Developer). It must list the bundles required to be isolated at runtime in the Application-Content header. Each entry in the header includes a "version range," which indicates which versions of the bundle are acceptable for use within the application. The dependencies, which will ultimately run in the Shared Framework, are calculated at deployment time using a process called 'resolution'. This calculation is made using the set of packages imported by the isolated bundles and the services required by Blueprint managed beans. Second order dependencies are also pulled in as are their dependencies and so on. Any missing package dependencies are reported as errors and are reported at deployment time, so if the application deploys successfully then its package dependencies will always be present at runtime.

The result of the resolution process is the full list of bundles, each at a specific version number, to be used by the deployed application. This is stored in the deployment manifest. Unlike the authored application manifest, the generated deployment manifest contains the transitively closed content – the result of the deploy-time resolution. The deployment manifest can be exported from one deployment and used in a subsequent deployment of the same application, to ensure the application is deployed in the same way. This is useful when moving an application from a Test to a Production system where you need to ensure the application deploys in the same way.

Once an OSGi application has been successfully deployed then all its constituent bundles are pushed out to the appropriate target servers and the application can be administratively started. Starting the application causes its constituent bundles to be started.

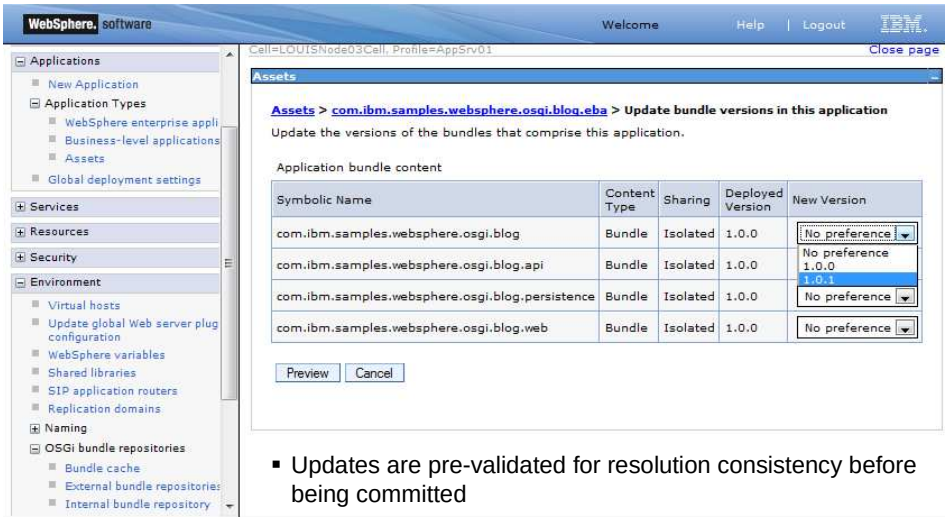
Application-centric bundle management (1 of 2)



- EBA archives are installed as application assets in WebSphere Application Server systems management

EBA Archives are installed as application assets in WebSphere Application Server systems management. Once installed, the application can be updated using the “update bundle versions in the application” link.

Application-centric bundle management (2 of 2)



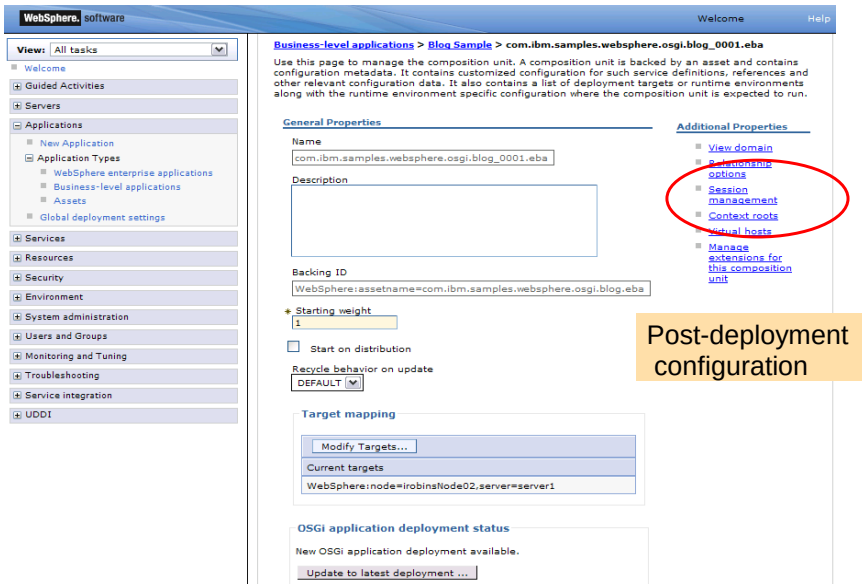
- Updates are pre-validated for resolution consistency before being committed
- Only the modified bundles are installed
- In V7, the update takes effect after the application is restarted

After clicking on the “update bundle versions in this application” link the application content bundles are listed. A drop-down list of new versions is available for each bundle. Initially there are no new versions.

In this example, all the bundles are deployed at version 1.0.0. The drop-down list indicates version 1.0.1 of the first bundle is available in a bundle repository. If the administrator looks at the available bundles now he sees the blog bundle available at versions 1.0.0 and 1.0.1. If the administrator wants to move to version 1.0.1, he can preview the changes to make sure the new version can still enable the application to be fully resolved, that is, there are no new dependencies that cannot be found. If successful, this procedure creates a new deployment manifest.

The administrator can then go ahead and commit the change. In V7 the update takes effect after the application is restarted. In V8 the change takes effect through an additional step described later.

Post deployment configuration - from V8.0

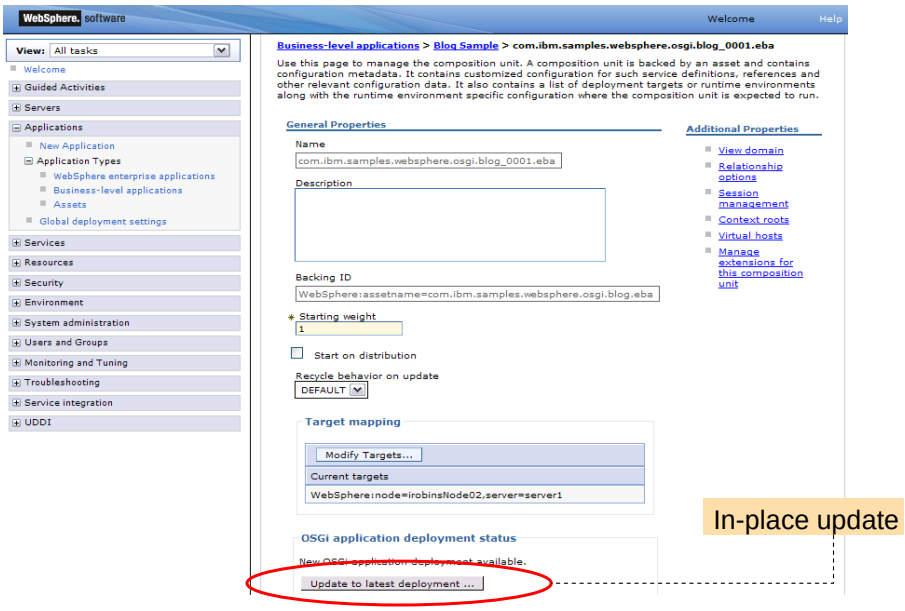


When using the V7 feature pack, once an OSGi application has been installed as a compositional unit (which includes configuring the application), reconfiguring the application can only be done by removing and re-adding the composition unit.

In version 8 and 8.5 all options that can be configured during installation can also be changed without having to remove and re-add the composition unit. This is done from the composition unit detail panel. Note that post deployment configuration is also available for V7 feature pack nodes when administered by a V8 or V8.5 deployment manager.

In addition, session management and run-as role mappings for OSGi web bundles can be configured in the same place.

In-place update – from V8.0 (1 of 2)



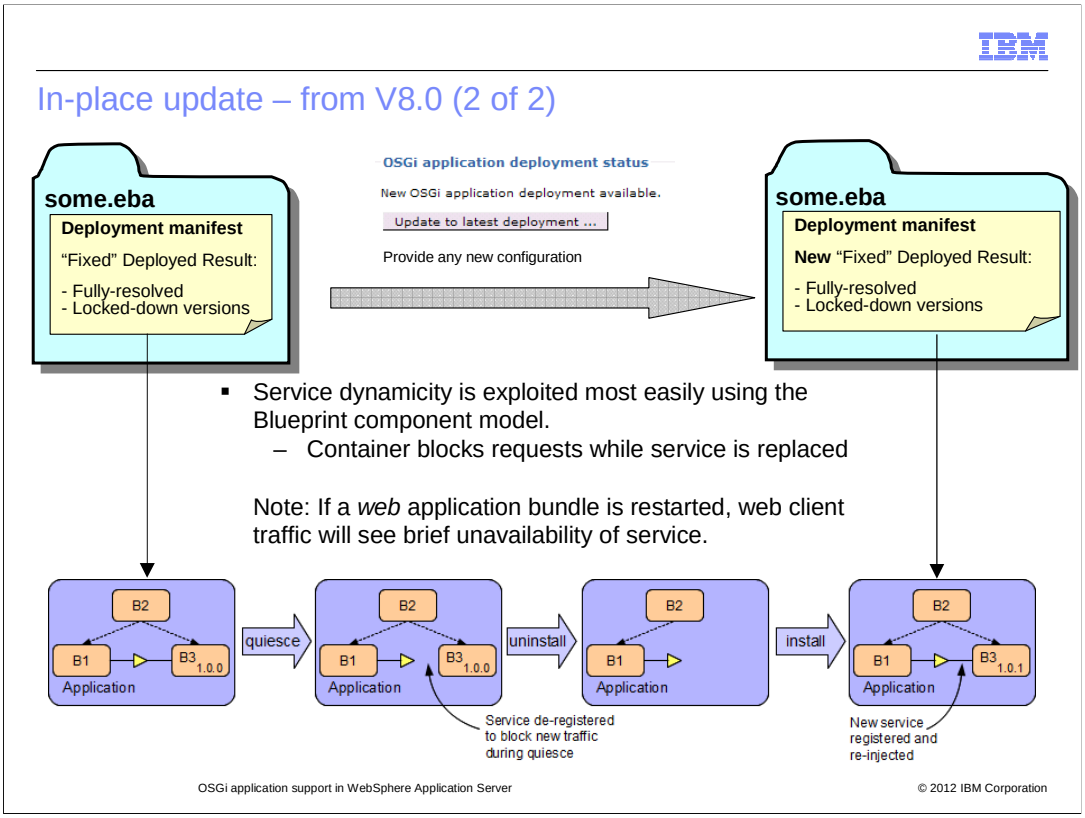
Although the V7 feature pack has the capability to upgrade individual bundles within an application, the capability has undergone a major overhaul to support both in-place update and updates that require new configuration. The additional button at the end of the composition unit detail panel is labeled “Update to latest deployment” and is enabled when a new deployment has been created, but not yet enabled. It is used to initiate an in-place update to an OSGi application - an update to a running application that replaces a subset of the running bundles within it.

Note however that major changes that include package and service imports and exports might well trigger an implicit restart of the entire application. Configuration wizard is displayed at this point if the new replacement bundles require additional configuration information such as virtual hosts or security bindings.

Updating an OSGi application in V8 and V8.5 occurs using a two-step process:

First, change bundle versions at the asset level, exactly as previously described in the V7 feature pack. This creates a new deployment manifest.

Second, instead of restarting the application, the administrator updates the application to use the new deployment manifest by clicking the ‘Update to latest deployment’ button. At this stage a wizard will pop up to allow any new required configuration to be set before the update. Existing configuration will be migrated to the new bundle versions so only new configuration needs to be configured. After this step, saving the configuration changes will trigger the update. The bundles that need to be updated within the running application will be stopped and the new ones started in their place.



This slide will show how in-place update works in detail.

As an example, an OSGi application contains three bundles: B1 is a user of a service interface provided by bundle B2 and implemented by B3. The current deployment manifest for the application lists bundle B3 at version 1.0.0. Then a new deployment for the application is created that lists bundle B3 at version 1.0.1. After clicking "Update to latest deployment," reviewing the changes, and then clicking OK, these four steps are taken to update the application.

In step 1, The existing B3 bundle is quiesced, meaning it receives no new traffic and is allowed to finish processing existing requests. This is done by de-registering its service from the service registry.

Step 2, the B3 bundle is stopped and uninstalled from the OSGi framework.

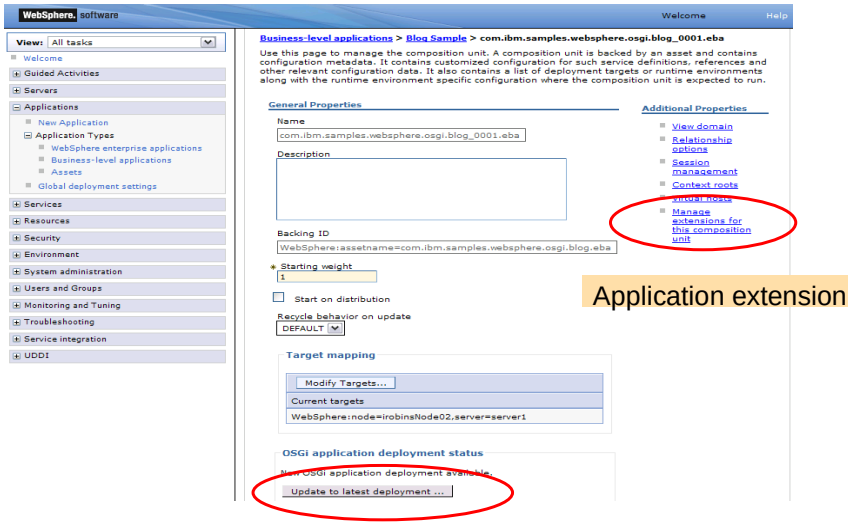
In step 3, the new B3 bundle at version 1.0.1 is installed into the OSGi framework and started.

And finally, its implementation of the service is then registered in the service registry.

If at any of these steps an error occurs, the update is rolled back if possible.

The best way to exploit in-place update is to use OSGi service dynamics such as in the depicted scenario. In the application, B3 supplies no packages to any other bundle - just a service implementation. Therefore updating B3 does not cause any other bundle of the application to be restarted. However, B1 needs to be written to cope with the short disappearance of the service that B3 provides. Using Blueprint in the implementation of bundle B1 will provide 'service damping', meaning when the new service is available, it will be injected into bean in bundle B1. That is, bundle B1 does not need to concern itself with the dynamic aspects.

Application extensions – from V8.0 (1 of 2)

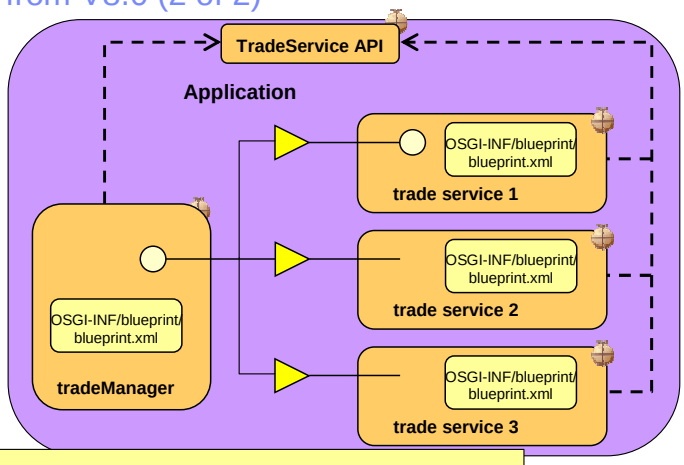


A completely new feature added in version 8 is application extensions. Application extensions allow the deployer to add and remove capabilities, in the form of composite bundles, dynamically while the application is running.

Any composite bundle available from an external bundle repository or the internal bundle repository can be installed as an extension. Installing or removing an extension creates a new deployment, in a similar way to creating a new deployment by updating bundles in an application. An application with a new deployment manifest containing an application extension can be applied to the running application in the same way as a new deployment manifest containing just updates: by using the “Update to latest deployment” button as shown.

Application extensions – from V8.0 (2 of 2)

For example using
Blueprint reference-listener

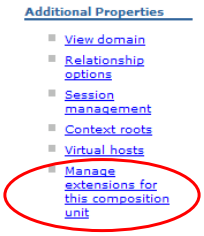


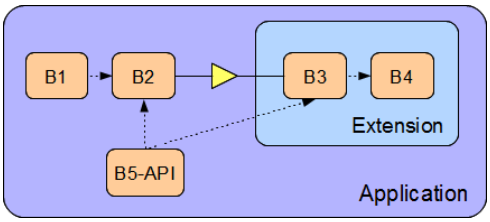

```
<blueprint>
  <bean id="manager" class="org.acme.TradeManagerImpl">
    <property name="tradeServices" ref="tradeServices" />
  </bean>
  <reference-list id="tradeServices" interface="org.example.TradeService">
        <reference-listener
              bind-method="bind" unbind-method="unbind">
           <bean class="org.acme.ReferenceListener"/>
        </reference-listener>
  </reference-list>
</blueprint>
```


A convenient way to make use of application extensions is with Blueprint reference-listeners. In the depicted application the tradeManager bundle can interact with several different trade services. The trade services are located in composite bundles that are installed as extensions and are transparently made available in the list of trade services available to the tradeManager. In addition the ReferenceListener receives callbacks for new trade services or trade services going away.

Administering application extensions – from V8.0

- An extension is a composite bundle archive (CBA) containing one or more bundles. WebSphere Application Server administrator steps:
  - Install extensions (.cba) in Internal Bundle Repository
  - Add extensions to Application through “Manage extensions”
  - When ready, update to latest deployment
    - Providing any extension configuration
  - Well designed extensions cause zero down-time
  - Golden design for extension:
    - Import packages from application
    - Export services to application
  - An OSGi equivalent to an Eclipse extension

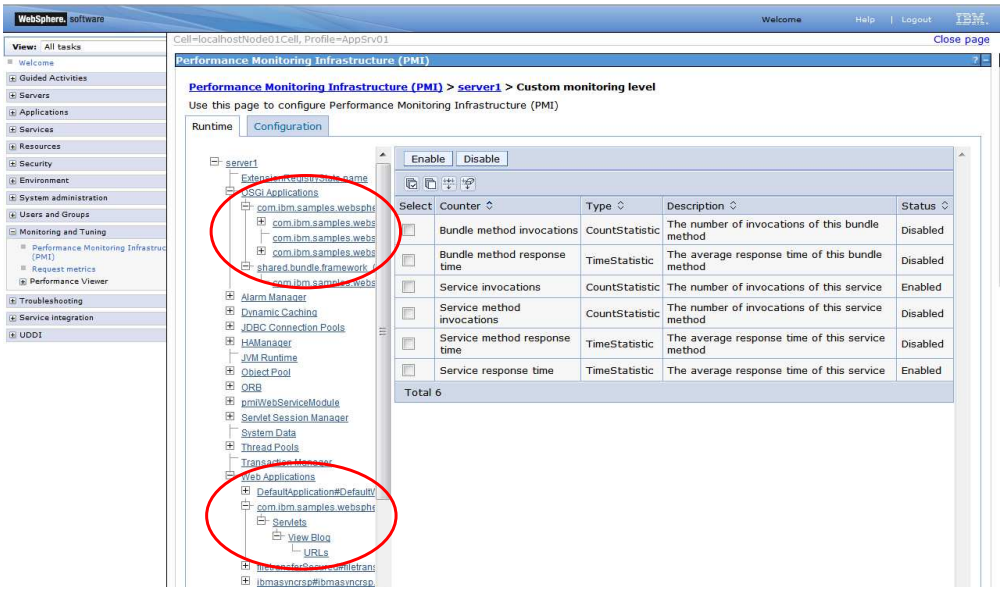




First the composite bundle containing the extension must be installed to either the internal bundle repository or made available in a configured external bundle repository. The next step is to navigate to the business level application in which the OSGi application is deployed and then the deployed asset for the OSGi application. The link named “Manage extensions for this composition unit” will allow you to start the process for creating a deployment manifest. Then the “Update to Latest Deployment” button will activate it.

With application extensions, OSGi applications can be extended with additional capability with zero down-time to the running application.

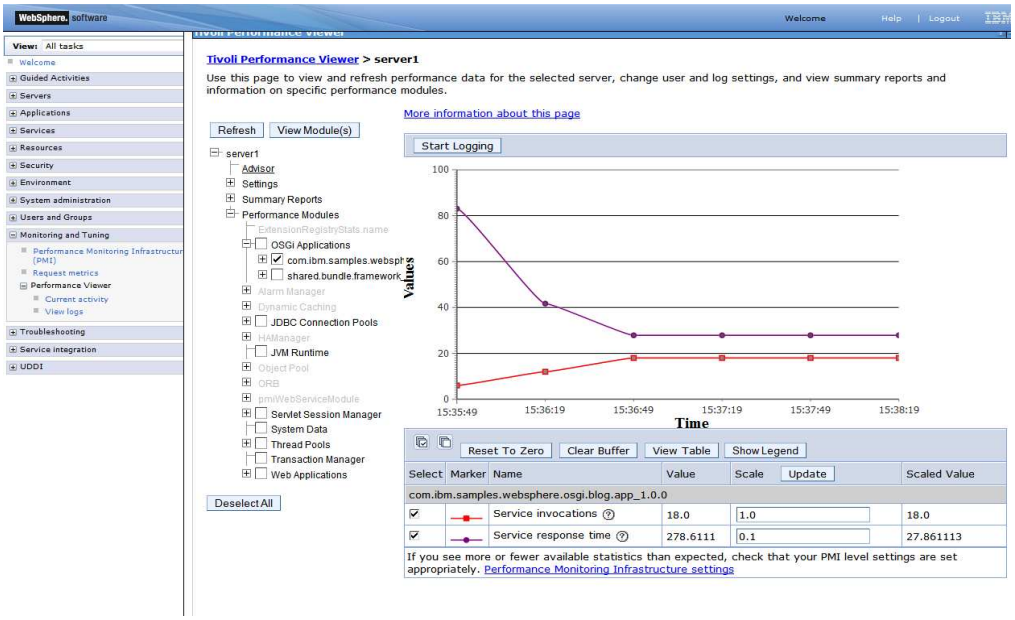
Performance monitoring infrastructure (PMI) – from V8.0 (1 of 2)



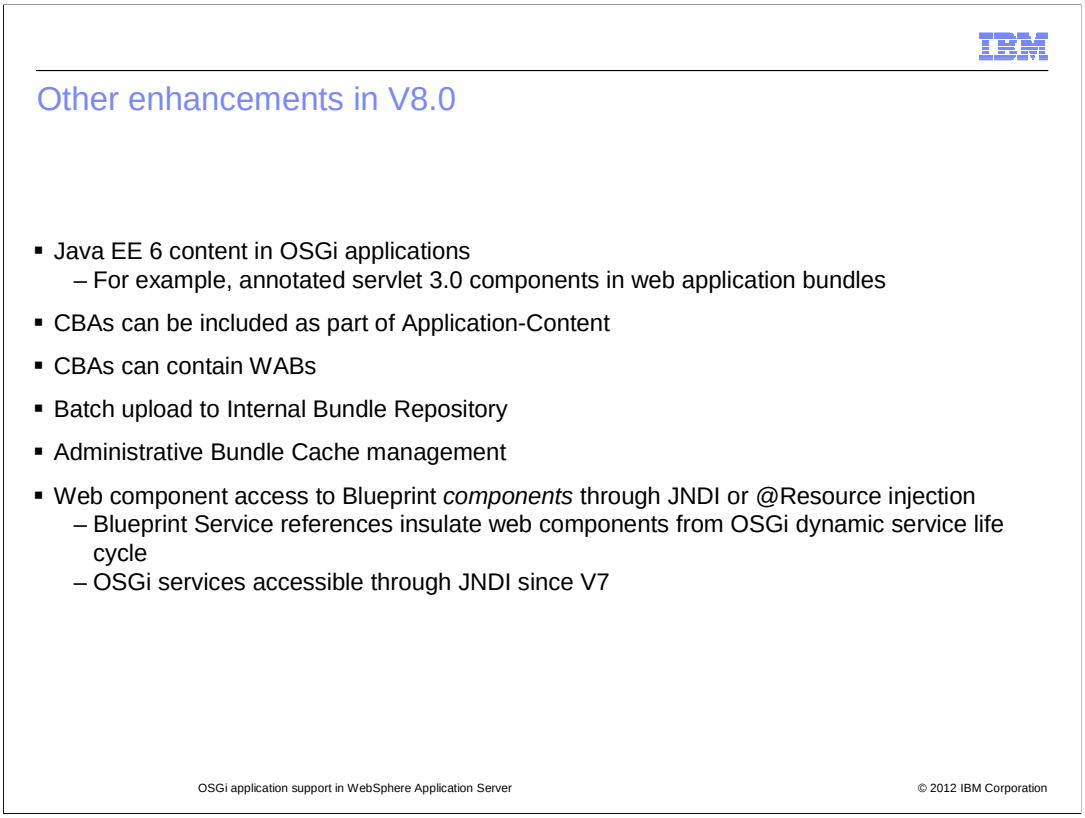
The Performance Monitoring Infrastructure has been enhanced to track the number of invocations and the response times of OSGi applications. There are three levels of granularity: basic, which monitors at the OSGi service level; extended, which monitors at the method level of OSGi services; and all, which monitors at the blueprint bean method level.

Web Application Bundles are monitored by the regular web container monitoring.

Performance monitoring infrastructure (PMI) – from V8.0 (2 of 2)



The performance viewer can be used to visualize the performance data for OSGi applications.



Other enhancements have been made in WebSphere Application Server V8.0 to the OSGi applications capability. Web Application bundles now contain annotated servlet 3.0 components.

With V8, Composite Bundle Archives can be included as isolated content in an application. That is, they can be present in the Application-Content header of the application's application manifest file. Additionally these composite bundles can contain Web Application Bundles

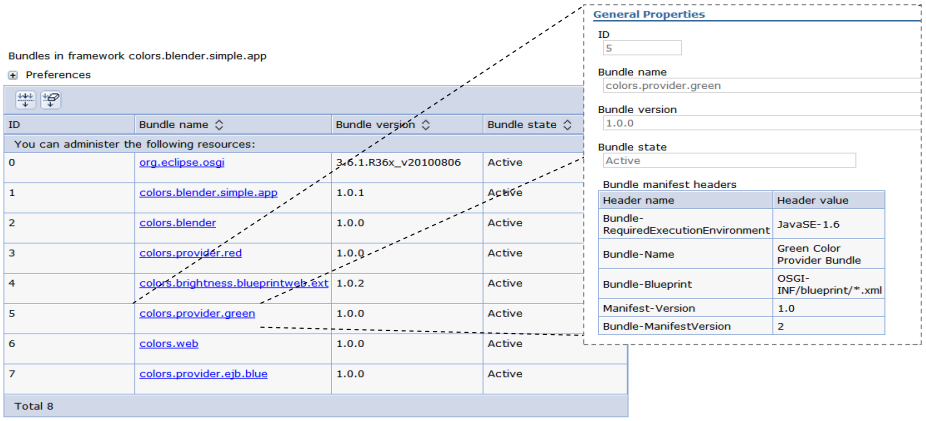
Multiple bundles can be uploaded to the Internal Bundle Repository in one action by providing them in an archive file.

The Bundle Cache can now be managed administratively.

Web components may now access Blueprint components through JNDI or @Resource injection. In addition Blueprint Service references will insulate web components from the OSGi dynamic service life cycle.

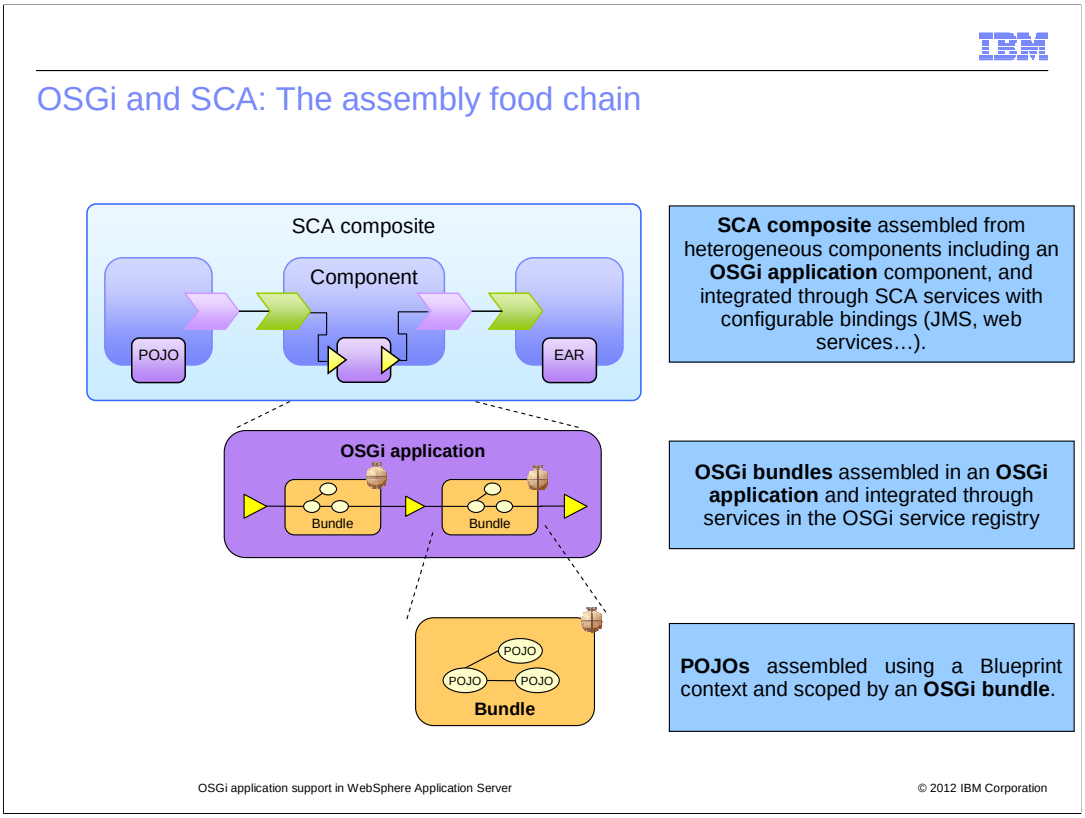
OSGi applications console (new in V8.5)

- Command-line console from V7 now complemented by integrated administrative console version
- View applications
- Drill down on content bundles
- View packages, services and dependencies
- View wiring into the shared bundle space



The OSGi Applications Console is an addition to WebSphere Application Server V8.5 administrative console. You can explore or debug bundles by interrogating the contents of OSGi frameworks used by a running application. You can view the package and service dependencies of bundles running in the application and the way individual bundles are wired to each other to ensure their package dependencies are resolved.

This facility is provided as a valuable debugging aid and to generally view how the application's modules are connected at runtime.



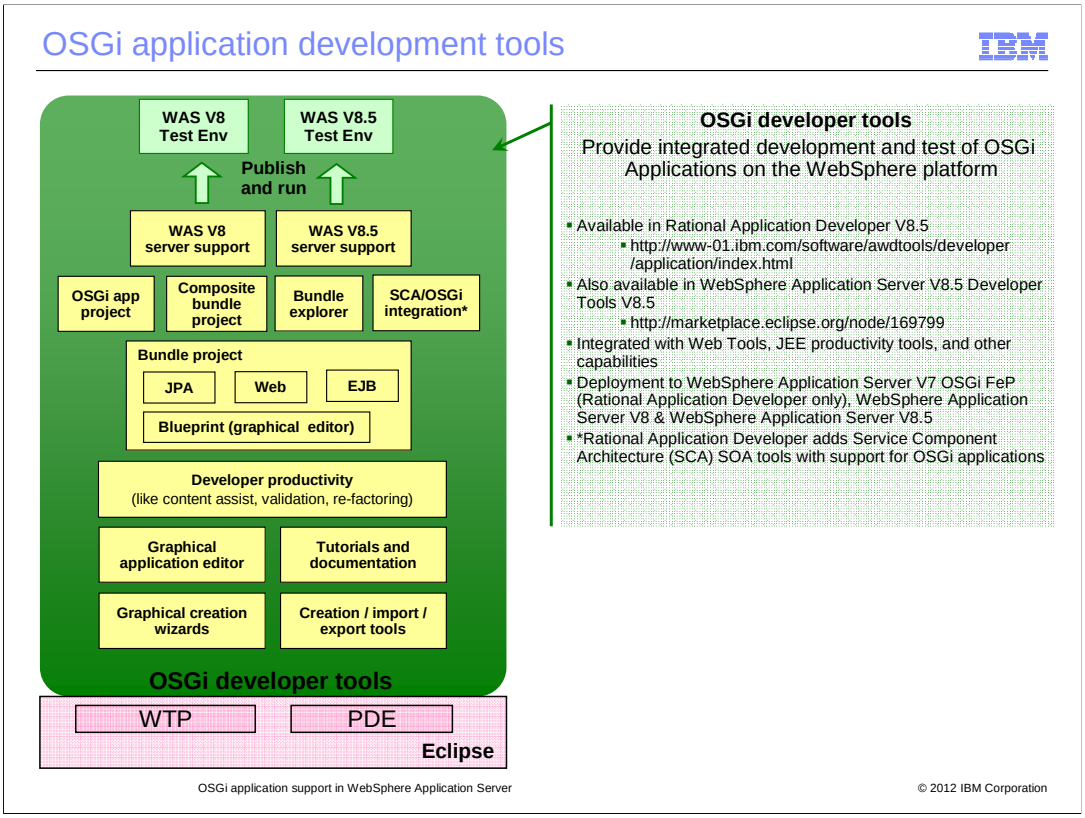
You have seen how Plain-Old-Java-Objects (or POJOs for short) can be assembled and configured in a Blueprint bundle and how multiple bundles including web and persistence bundles can be assembled into an isolated OSGi Application.

There is a further level of assembly available to OSGi applications into a Service Component Architecture composite to provide a Service Oriented Architecture abstraction. Within an SCA composite the OSGi Application is a component that can be wired to other components with different implementation types. For example, an SCA composite can contain an OSGi-application component, a JEE component containing Enterprise Java Beans and so on. Each component within an SCA composite declares abstract services and references to which concrete bindings can be applied and it is through these services and references that the components of an SCA composites are wired together. The OSGi Application architecture was designed with this form of assembly in mind so that the services and references declared in a Blueprint XML configuration can be exposed through the Application manifest to be visible outside the application. Such exposed services and references can then be mapped to SCA services and references with the full range of available SCA bindings applied to them.

This enables OSGi applications to participate in two new scenarios:

First, they can be assembled into heterogeneous composites of OSGi and non-OSGi components.

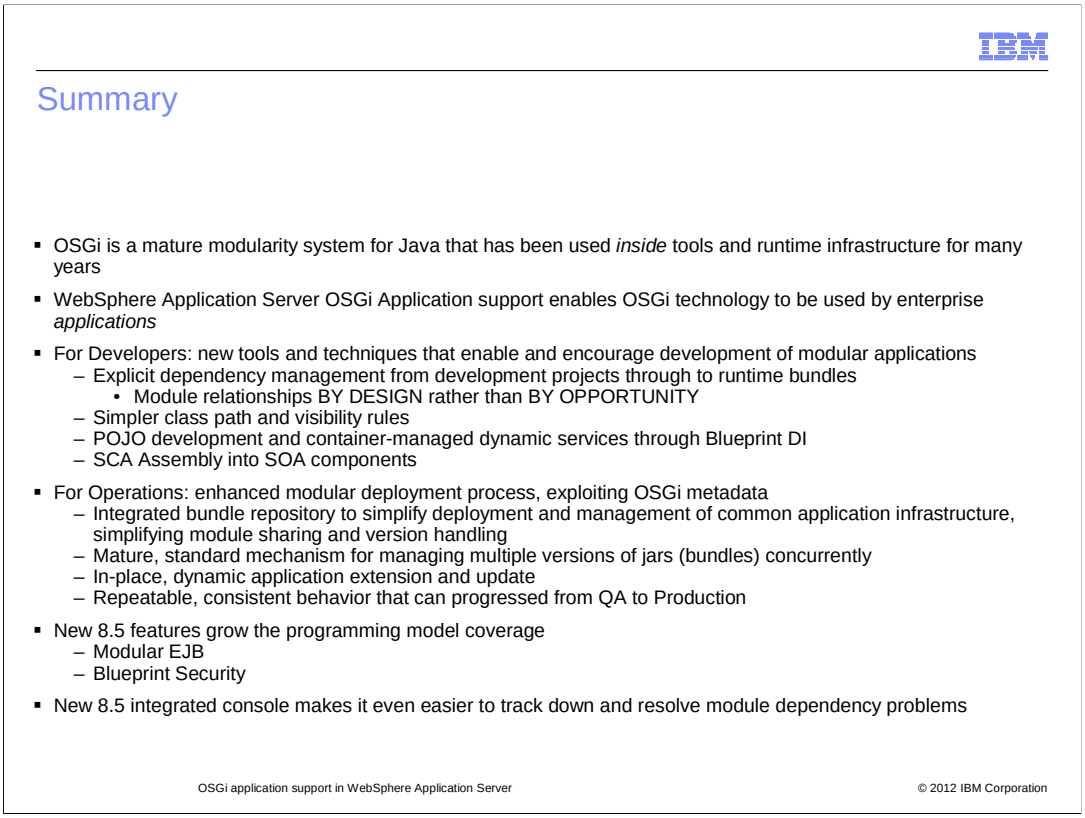
Second, OSGi services they provide can be provided remotely through SCA services with a variety of bindings including JMS, SOAP over HTTP, IIOP, and JSON-RPC.



The tools support is structured so that server-independent development and assembly tools can be installed as a plug-in into any Eclipse Web Tools Platform (or WTP for short) environment. While this is pre-integrated in Rational Application Developer V8.5, the availability of the new tools in Eclipse WTP configurations other than Rational Application Developer better enables these common tools to be used to develop OSGi Applications for deployment to Geronimo and, in the future, application servers other than WebSphere that integrate the Apache Aries runtime components.

The development tools include the OSGi Application project type, the OSGi Composite Bundle project type and the OSGi Bundle project type. Enterprise Bundle Archive files, or .eba files can be imported and exported. Form-based editors for bundle manifests, application manifests and Blueprint configuration files, tutorials, and documentation all assist with the creation of OSGi Applications.

Additionally integrated into Rational Application Developer V8.5 are WebSphere deployment tools, a WebSphere Application Server test environment, enhanced validation tools, and integration with Web and JEE productivity tools.



OSGi is a mature modularity system for Java that has been used inside tools and runtime infrastructure for many years. WebSphere Application Server OSGi Application support enables OSGi technology to be used by enterprise applications. For Developers there are new tools and techniques that enable and encourage development of modular applications. There is explicit dependency management from development projects through to runtime bundles. The module relationships at runtime are BY DESIGN rather than BY OPPORTUNITY, which has the effect of providing simpler class path and class visibility rules.

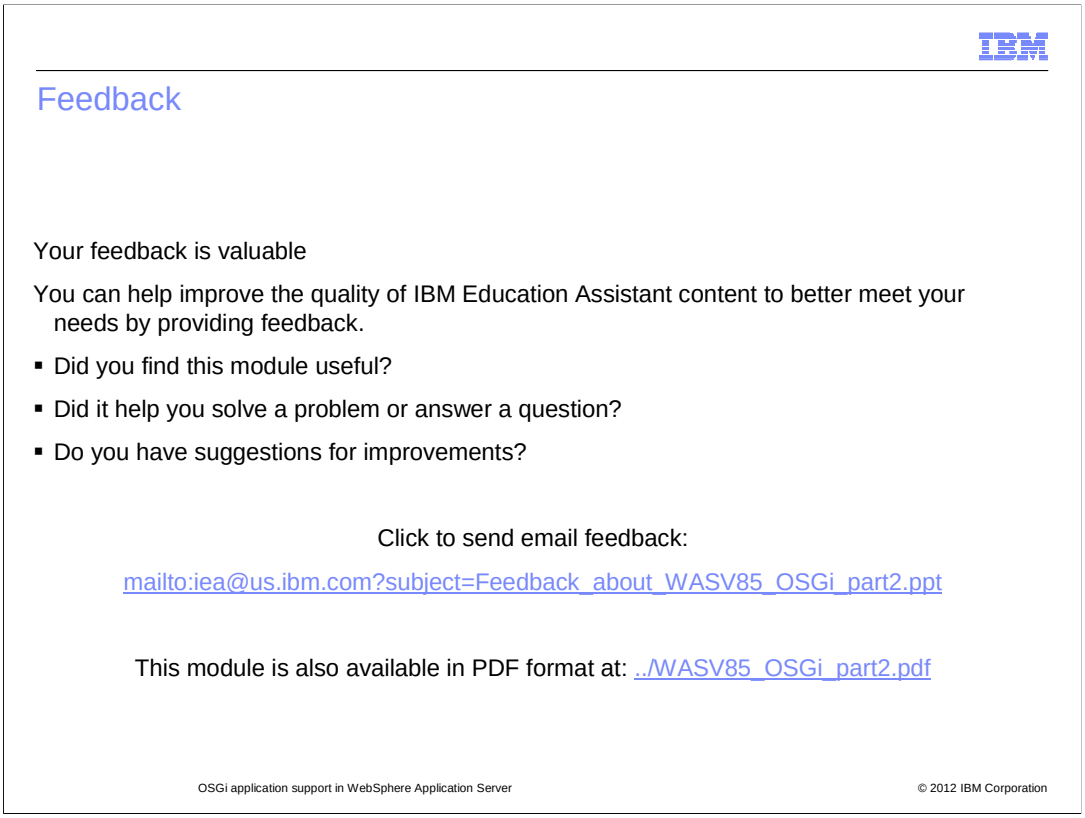
The tools promote POJO development and container-managed dynamic services through standard Blueprint Dependency Injection. And OSGi applications can be assembled into heterogeneous applications using the Service Component Architecture. For server operators there is an enhanced modular deployment process, exploiting OSGi metadata. The integrated bundle repository enables deployment and management of common application infrastructure, simplifying module sharing, and version handling. A mature, standard mechanism is provided for managing multiple versions of bundle jars concurrently.

The in-place update and dynamic application extension available from WebSphere Application Server V8 provides additional options for dynamic application management. And the deployment mechanism provides a repeatable, consistent behavior that can progress from Quality Assurance to Production. From version 8.5 OSGi applications now include Enterprise JavaBeans and secure Blueprint access to bean methods by security role. The new integrated consoles makes it even easier to track down and resolve module dependency problems.

Additional references

- Intro paper on WebSphere OSGi application feature
  http://www.ibm.com/developerworks/websphere/techjournal/1007_robinson/1007_robinson.html
- OSGi best practices
  http://www.ibm.com/developerworks/websphere/techjournal/1007_charters/1007_charters.html
- Enterprise OSGi YouTube Channel:
  http://www.youtube.com/user/EnterpriseOSGi
- Redbook: OSGi Applications and JPA 2.0:
  http://www.redbooks.ibm.com/redpieces/abstracts/sg247911.html?Open
- WebSphere Discussion forum for OSGi applications:
  http://www.ibm.com/developerworks/forums/forum.jspa?forumID=1928
- WebSphere Application Server V8
  http://www.ibm.com/software/webservers/appserv/was/
- WebSphere Application Server V7 Feature Pack for OSGi Applications
  http://www.ibm.com/websphere/was/osgi
- Rational Application Developer V8.0
  http://www.ibm.com/software/awdtools/developer/application/index.html

Here are several links where you can learn more about OSGi.



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

Trademarks, disclaimer, and copyright information

IBM, the IBM logo, ibm.com, Rational, and WebSphere are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Java, and all Java-based trademarks and logos are trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.