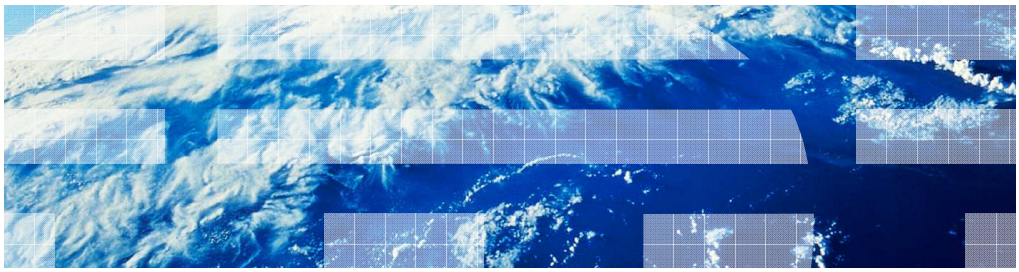
WebSphere Application Server Version 8

Bean validation for RAR modules



WebSphere. software

This presentation describes the Bean Validation support for Resource Adapter (RAR) modules introduced in IBM WebSphere® Application Server V8.

Table of contents

Contents of this presentation include an overview of the support for RAR bean validation, descriptions of validation scenarios involving RAR JavaBeans types, and an examination of a bean validation configuration specific to a RAR module.

This section contains an overview of bean validation and how it is applied by WebSphere Application Server to validate JavaBeans supplied in a RAR module.

Resource providers can now use bean validation metadata to specify the operational constraints of the JavaBeans composing a resource adapter.

The application server uses the metadata to verify whether a resource adapter has valid state to ensure that applications do not encounter unexpected behaviors when interacting with that resource.

The effects of RAR bean validation on the interaction between applications and resources is elaborated shortly, but first: Bean validation is discussed.

What is bean validation?

- A **metadata model** for declaring the validation constraints of a JavaBeans type
- An **application programming interface** (API) for determining whether a JavaBeans instance violates any validation constraints declared for that JavaBeans type
- Bean Validation requirements specified in JSR 303

Bean validation is a **metadata model** for declaring the validation constraints of a JavaBeans, and an **API** for determining whether a JavaBeans instance violates any validation constraints declared for that JavaBeans type.

The bean validation metadata model and API are specified in JSR 303.

An introspective Java EE bean validation example

```
public class Stuart implements SmalleyBean {

    @Resource ValidatorFactory validatorFactory;
    @Resource Validator validator;

    @NotNull public String state = “acceptance”;

    public Boolean validateMe(Stuart me)  {
        Set<ConstraintViolation<Stuart>> cvSet = null;
        try {
            cvSet = validator.validate(me);
        }
        catch (Throwable t) {
            t.printStackTrace();  // The validator failed unexpectedly!
        }
        return cvSet != null && cvSet.size() == 0;
    }
. . .
```


This slide displays an enterprise Java™ bean to exemplify the basic components of bean validation.

A **validation constraint** is a declarative condition that a Java bean must satisfy to be considered valid.

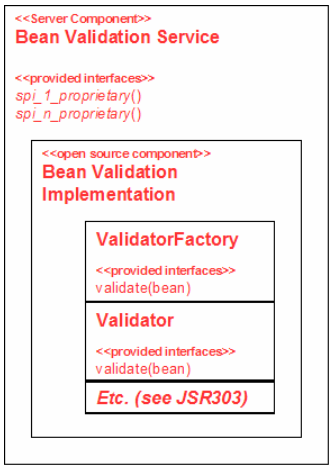
The example uses the NotNull built-in constraint. By default, validation constraints are defined by **metadata annotations**, and every bean validation implementation supplies several *built-in* validation constraint annotations. Validation constraints can also be defined, extended or overridden using XML descriptors.

The application acquires an instance of a **ValidatorFactory** and uses the factory to obtain an instance of a **validator**. In JEE 6, a ValidatorFactory and validator might be injected within an enterprise Java bean by using the resource annotation.

The application then invokes the **validate()** method to validate a JavaBeans instance. In this case, it validates an instance of “stuart”. Validation fails whenever the set of ViolationConstraints returned by validate() contains at least one element. So, stuart will fail validation if its state is not set.

Note that enterprise JavaBeans are not RAR JavaBeans.

WebSphere bean validation



- Adds a **Bean Validation service** to WebSphere Application Server
- Provides a default **Bean Validation implementation** and exports the JSR 303 **Bean Validation API** for use by **application** modules deployed on a server
- Exports a proprietary Bean Validation SPI to server components that support Java EE Bean Validation requirements (e.g. The Connection service)
- Allows the use of third-party Bean Validation implementations

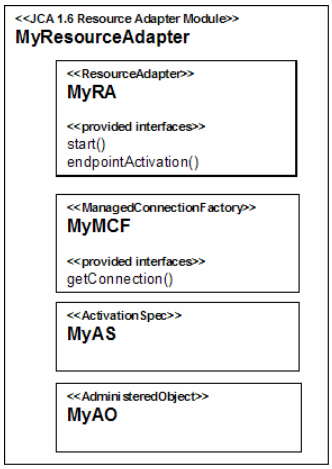
Version 8 of WebSphere Application Server adds a **bean validation service** to the server runtime.

The service exports the JSR 303 **Bean Validation API** and a default **bean validation implementation** to all application modules deployed on a server.

The bean validation service enables a deployed JEE module to use the default bean validation implementation, and if required, enables a module to use a third-party bean validation implementation that is either specific to that module or shared (used) across several modules.

The service also provides a **proprietary bean validation SPI** to server components that support JEE bean validation requirements – Like the WebSphere connection service.

Bean validation for RAR modules (1 of 2)



- Resource adapter providers apply constraints, using annotations or an XML descriptor, to the fields and properties of JCA JavaBeans types:
  - **`ResourceAdapter`**
  - **`ManagedConnectionFactory`**
  - **`ActivationSpec`**
  - **`AdministeredObject`**
- The constraints declare range limits and mandatory attributes of resource adapter configuration properties that are configurable by the RAR deployer

The connection service validates RAR JavaBeans to ensure that resource adapters provide the expected connectivity behaviors between applications and resources.

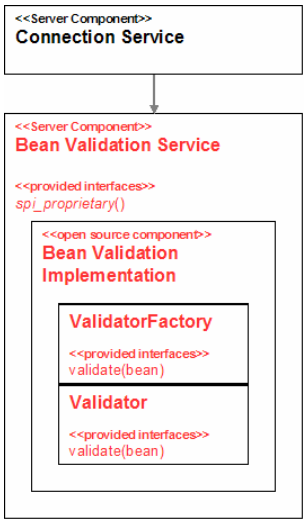
There are **four types of RAR JavaBeans**: ResourceAdapter, ManagedConnectionFactory, ActivationSpec, and AdministeredObject.

To use RAR bean validation, resource developers must declare the bean validation constraints of a RAR JavaBeans' configuration properties either by decorating the JavaBeans classes with bean validation constraint annotations, or by supplying a bean validation XML configuration within the RAR.

This is known as the constraint metadata, which ultimately specifies the range and mandatory attributes of the RAR JavaBeans configuration properties

The Java connector architecture encourages resource developers to use built-in constraint annotations.

Bean validation for RAR modules (2 of 2)



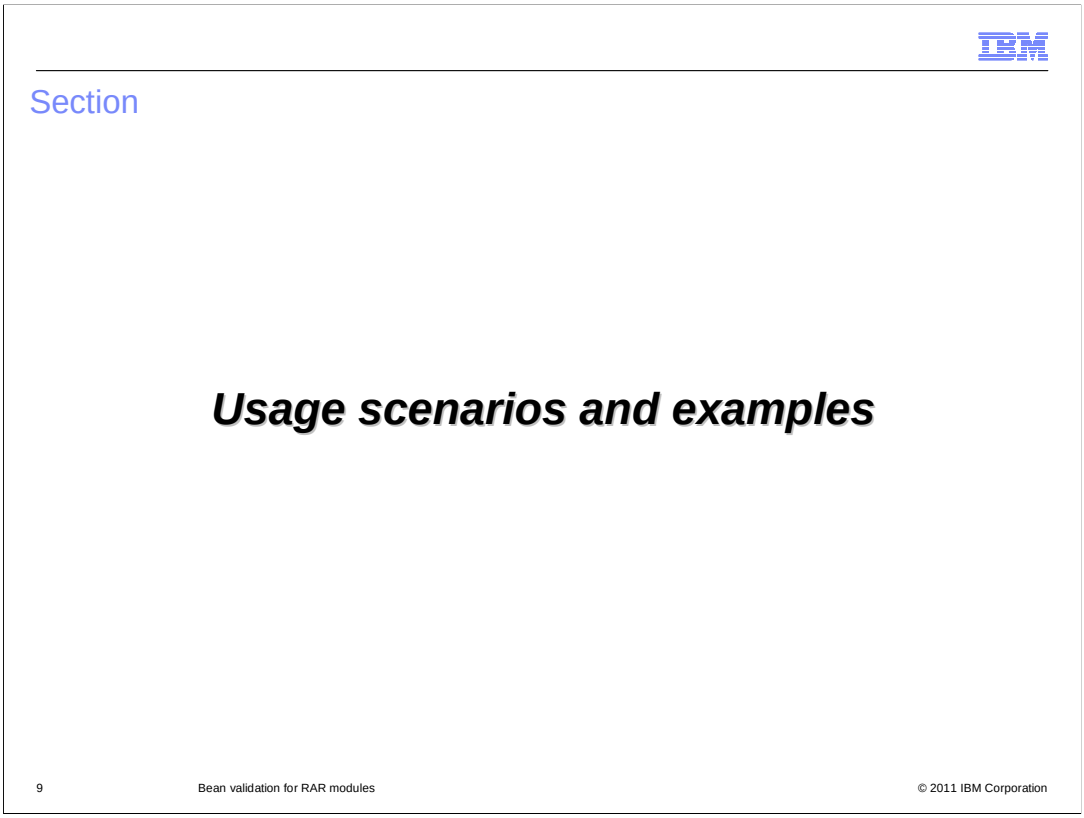
- The server creates a **ValidatorFactory** configured specifically for each JCA 1.6 RAR module
- The server uses the RAR-specific ValidatorFactory to create a **Validator** and then invokes **validate()** on each new RAR JavaBeans instance immediately after initializing its custom properties
- The server creates a **ConstraintValidationException** whenever validate() returns at least one ConstraintViolation and prevents the not valid RAR JavaBeans instance from providing service to applications

To perform RAR Bean Validation, the application server configures a ValidatorFactory for each RAR module.

Before placing a RAR JavaBeans into service, the Connection service must first instantiate the bean and set the state of the configuration properties. To help ensure the bean has valid state, the Connection service will now also validate the bean instance by obtaining a **Validator** from the RAR-specific **ValidatorFactory**, and then invoking the Validator to validate the bean.

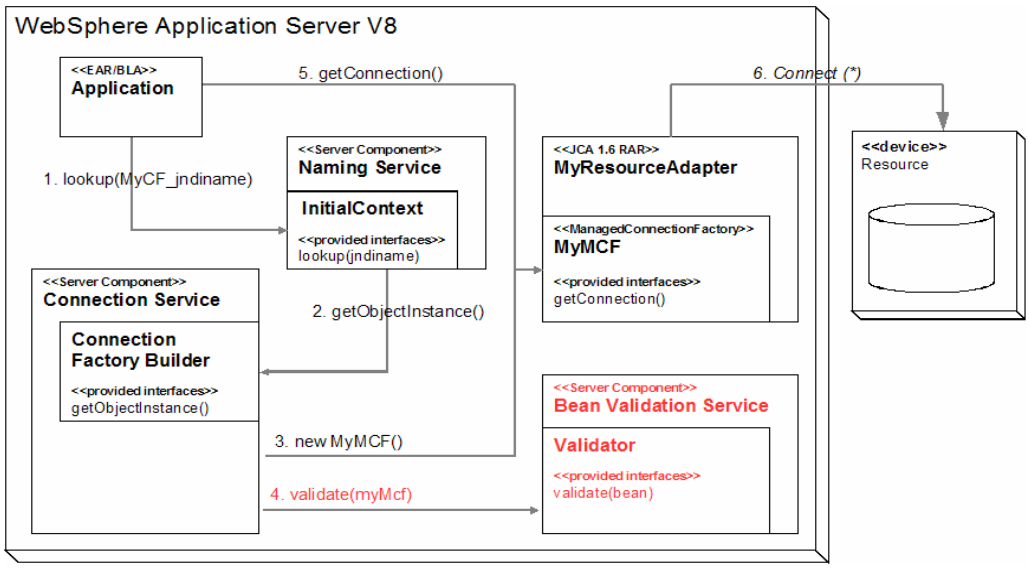
If a validation constraint is violated, the Connection service creates a **ConstraintValidationException** and prevents the not valid bean from providing service. The effect of a constraint violation differs according to the RAR JavaBeans type and how the RAR is deployed.

Administrators will determine any constraint violations and, if possible, modify the indicated property values to satisfy their constraints.



This section describes how to use RAR bean validation and exemplifies the interactions between the application, the resource adapter, and the server components described in the overview when performing bean validation on RAR JavaBeans.

Connecting to a resource



This slide shows the interaction of components that occur when an application looks-up a connection factory and attempts to establish a connection to a resource.

The application server creates and configures a **ManagedConnectionFactory** (MCF) JavaBeans instance, provided in a RAR, when an application looks-up a ConnectionFactory.

Before returning the connection factory to the application, the server now validates the MCF instance as depicted in red in step 4. If validation fails, the MCF instance is not placed into service, and the server returns a ConstraintViolationException to the application rather than a connection factory.

For all this to occur, the RAR provider must have developed the MCF class with properties and fields constrained with bean validation constraint metadata. Next an administrator must have deployed the RAR, and of course, the application. And finally, the administrator must have created a connection factory for the resource adapter and configured the custom properties of the connection factory for any required connection behaviors.

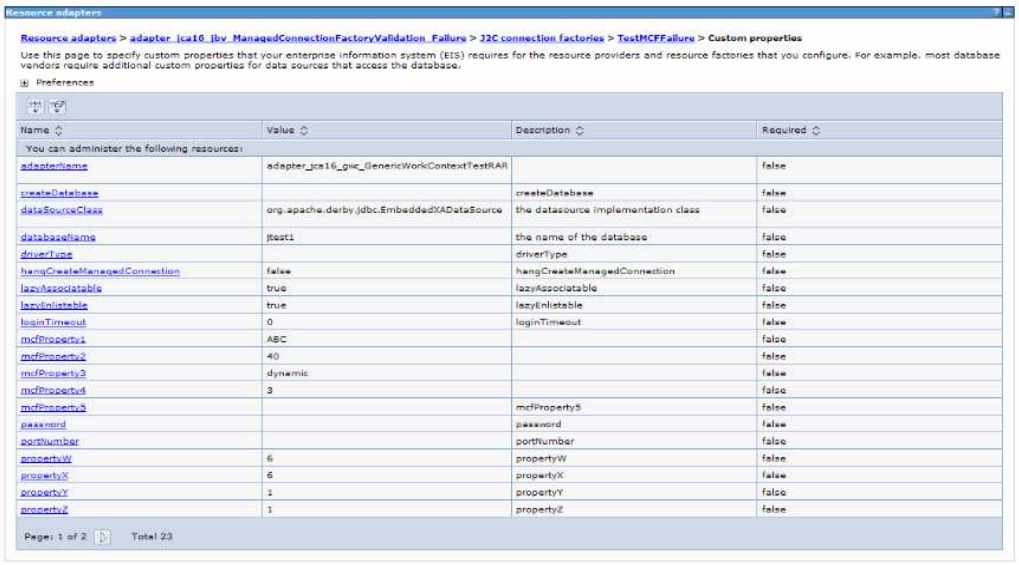
ManagedConnectionFactory JavaBeans property constraint

```
package com.ibm.adapter.spi.jbv;

import javax.resource.spi.ManagedConnectionFactory;
import javax.resource.spi.ResourceAdapterAssociation;
import javax.validation.constraints.Min;
import java.io.Serializable;

public class JBVFATMCFFailureImpl
      extends ManagedConnectionFactory, Serializable, ResourceAdapterAssociation
{
   @Min(value=10, message="The value should be greater than 10")
   Float mcfProperty4;

   public Float getMcfProperty4() {return mcfProperty4;}

   public void setMcfProperty4(Float mcfProperty4) {this.mcfProperty4 = mcfProperty4;}
   . . .
```


To use RAR Bean Validation, the resource provider must develop a ManagedConnectionFactory (MCF) JavaBeans class with properties and fields constrained with Bean Validation constraint metadata.

This slide shows an example of an MCF Java bean with a configuration property named “mcfProperty4”. The property is constrained using the built-in “Min” annotation specifying a minimal value of 10.

Connection factory custom properties



After deploying a RAR, an administrator must create a connection factory in the resource adapter configuration that applications might use to connect to the resource.

The administrator configures the connection factory and its custom properties for required connectivity behaviors.

Custom properties in the CF are the configuration properties of the MCF JavaBeans class supporting the connection factory.

This slide shows the custom properties of the ConnectionFactory named “TestMCFFailure”. Notice that the “mcfProperty4” property is configured with a value that is not valid – that is, its value is less than 10.

Managed connection factory constraint violation (1 of 2)

If validation fails, the server rejects the MCF instance during the JNDI lookup and creates a ContraintViolationException nested within a NamingException

```
[9/30/10 7:58:58:734 CDT] 00000023 BeanValidatio E   J2CA0238E:
  JavaBeanscom.ibm.adapter.spi.jbv.JBVFATMCFFailureImpl@7dd07dd0 failed Bean Validation
  due to one or more not valid configuration settings indicated in the following list of
  constraint violations:
  …
  ConstraintViolationImpl{interpolatedMessage='The value should be greater than 10',
  propertyPath=mcfProperty4, rootBeanClass=class
  com.ibm.adapter.spi.jbv.JBVFATMCFFailureImpl, messageTemplate='The value should be
  greater than 10'}
  …
[9/30/10 7:58:58:765 CDT] 00000023 ConnectionFac E   J2CA0009E: An exception occurred
  while trying to instantiate the ManagedConnectionFactory class
  com.ibm.adapter.spi.jbv.JBVFATMCFFailureImpl used by resource jms/TestMCFFailure :
  com.ibm.ejs.j2c.metadata.ConstraintViolationException
  at
  com.ibm.ejs.j2c.metadata.BeanValidationHelper.validate(BeanValidationHelper.java:477)
  at com.ibm.ejs.j2c.ServerFunction.validate(ServerFunction.java:1323)
  at com.ibm.ejs.j2c.J2CUtilityClass.createMCFEntry(J2CUtilityClass.java:810)
  …
  at javax.naming.InitialContext.lookup(InitialContext.java:455)
  at suite.r80.base.jca16.jbv.JavaBeanValidatorTest.testJavaBeanValidationMCF(…
```

The application cannot get a connection to the resource

When an application performs a JNDI lookup on a ConnectionFactory, the Connection Service creates and validates an MCF instance and then uses the instance to provide managed connections when the application uses the factory to get a connection

If a constraint violation occurs during validation, the service will create a ContraintViolationException back to the application, rather then returning a connection factory, which prevents the application from getting a connection to the resource.

This slide shows a constraint violation that occurred while validating an instance of the MCF containing property “mcfProperty4”, which was configured with a value that is not valid. The exception lists the names of all property constraints that were violated along with their values.

Managed connection factory constraint violation (2 of 2)

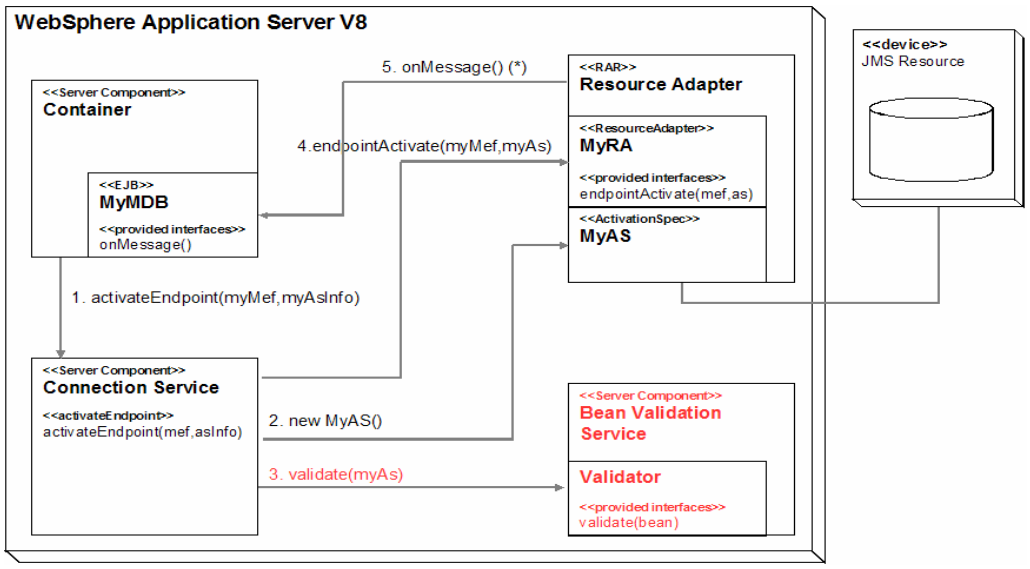
- The administrator should determine the problem, stop application service, reconfigure any MCF properties to the appropriate settings, and restart application service
- Recovery of in-doubt transactions involving the resource through connections created by the ConnectionFactory cannot occur while its MCF configuration violates its validation constraints
- The effects are the same whether the ConnectionFactory is defined within a stand-alone or an embedded RAR

When a violation occurs, an Administrator should examine the violation and determine the correct value for the property.

Because the MCF Java bean is not placed into service, and the MCF is also the XAResource factory, the server cannot recover in-doubt transactions over resource connections previously established with the Connection Factory until the violation is alleviated.

Finally, the connectivity failures described to this point are the same whether the Connection Factory is defined within a stand-alone or an embedded RAR.

Activating a message endpoint



This slide shows the interaction of components that activate a message endpoint within an application that accepts messages delivered by a messaging provider.

The application server creates and configures an **ActivationSpec** (AS) Java bean to activate the delivery of messages from a message provider to the message endpoints of an application.

Before activating an endpoint for the first time, the server now **validates the AS** instance as depicted in red in step 3. If validation fails, the AS instance is not placed into service, the endpoint is not activated, and the resource adapter does not deliver messages to the endpoint.

For all this to occur, the RAR provider must have developed the AS class with properties and fields constrained with Bean Validation constraint metadata. Next an administrator must have deployed the RAR, and of course, the application. And finally, the Administrator must have created a activation specification for the resource adapter and configured the custom properties of the activation specification for any required connection behaviors.

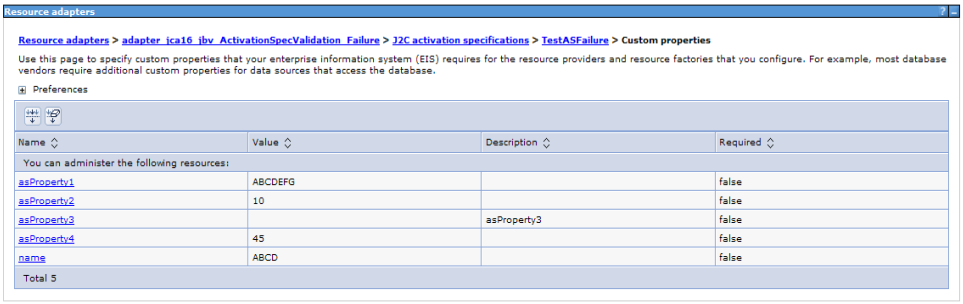
ActivationSpec JavaBeansconstraint

```
package com.ibm.adapter.activationspec.jbv;

import javax.resource.spi.ActivationSpec;
import javax.validation.constraints.Size;
import java.io.Serializable;

public class JBVFATActSpecFailureImpl implements ActivationSpec, Serializable
{
   public String asProperty1;
   public Integer asProperty2;

   @Size(min=2,max=4,message="Size should be between 2 and 4")
   public Integer getAsProperty1() {return asProperty1;}

   @Max(value=30,message="Should be less < 30")
   public Integer getAsProperty2() {return asProperty2;}
   ...
```


To use bean validation, the resource provider must develop an ActivationSpec (AS) JavaBeans class with properties and fields constrained with bean validation constraint metadata.

This slide shows an example of an AS Java bean with configuration properties named “asProperty1” and “asProperty2”. The getter methods for these properties are constrained using the built-in “@size” and “@max” annotations.

Activation specification custom properties



After deploying a resource adapter module, an administrator can create an activation specification in the resource adapter configuration. The administrator configures the activation specification and its custom properties to specify how the messaging resource will deliver messages to a message endpoint. Custom properties in the activation specification are the configuration properties of the ActivationSpec JavaBean class supporting the activation specification.

This slide shows the custom properties of an activation specification named “TestASFailure”. Notice that the “asProperty1” property is configured with a value that is not valid – that is, its size is not between 2 and 4, inclusive.

ActivationSpec constraint violation (1 of 2)

If validation fails, the server rejects the AS instance during initial endpoint activation by throwing a ContraintViolationException, nested in a ResourceException, to the endpoint container

```
[9/29/10 10:52:05:125 CDT] 00000009 BeanValidatio E   J2CA0238E: JavaBean
  com.ibm.adapter.activationspec.jbv.JBVFATActSpecFailureImpl@51625162 failed Bean
  Validation due to one or more not valid configuration settings indicated in the
  following list of constraint violations:
  ConstraintViolationImpl{interpolatedMessage='Size should be between 2 and 4',
  propertyPath=asProperty1, rootBeanClass=class
  com.ibm.adapter.activationspec.jbv.JBVFATActSpecFailureImpl, messageTemplate='Size
  should be between 2 and 4'}
…
[9/29/10 10:52:05:171 CDT] 00000009 RAWrapperImpl E   J2CA0089E: The method
  activateEndpoint on ResourceAdapter JavaBean cells/IBM-
  46DF84D297BNode01Cell/nodes/IBM-
  46DF84D297BNode01/resources.xml#J2CResourceAdapter_1285109389828 failed with the
  following exception: javax.resource.ResourceException:
  com.ibm.ejs.j2c.metadata.ConstraintViolationException
  at com.ibm.ejs.j2c.ActivationSpecWrapperImpl.validateActivationSpecInstance(Acti…
  at com.ibm.ejs.j2c.ActivationSpecWrapperImpl.createAndInitializeActivationSpecIns…
  at com.ibm.ejs.j2c.ActivationSpecWrapperImpl.activateEndpoint(ActivationSpecWrapperI…
…
[9/29/10 10:52:05:750 CDT] 00000009 ApplicationMg A   WSVR0217I: Stopping application:
  sampleapp_jca16_jbv_standaloneasfailureApp…
```

The endpoint is not activated and messages cannot be delivered to the endpoint.

At runtime, the server creates and validates an AS instance when activating the endpoint during application startup. The message provider starts message delivery to the endpoint using the AS configuration.

If validation fails, the AS is not placed into service and message endpoint is not activated, that is, the messaging provider cannot deliver messages to the application hosting the endpoint.

This slide shows a constraint violation that occurred while validating an instance of the AS containing property “asProperty1”, which was configured with a non-valid value. The exception lists the names of all property constraints that were violated along with their values.

ActivationSpec constraint violation (2 of 2)

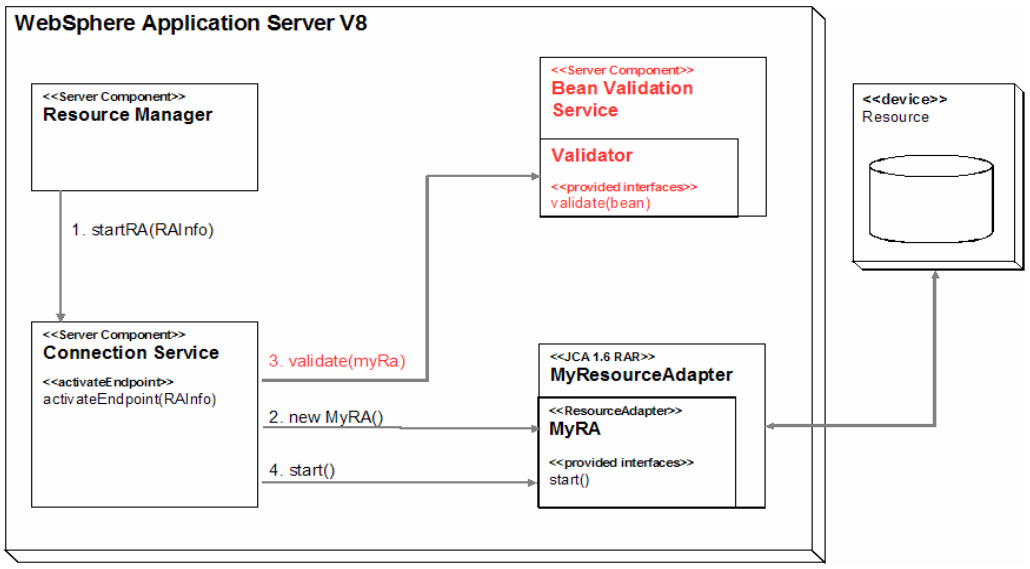
- The administrator should determine the problem, reconfigure any AS properties to the appropriate settings, and restart application service to activate message endpoints (that is, start message delivery)
- Applications bound to the activation specification cannot receive messages while the AS configuration violates any constraints
- Failed transactional messages delivered to endpoints established using the activation specification cannot be recovered while the AS configuration violates its validation constraints

When a violation occurs, an Administrator should examine the violation and determine the correct value for the property.

Because the ActivationSpec is not placed into service, applications bound to the activation specification cannot receive messages until the violation is alleviated.

Furthermore, the server cannot redeliver – that is, recover -- failed transactional messages to endpoints that do not activate due to constraint violations.

Starting a resource adapter



This slide shows the interaction of server components that start a resource adapter.

When the application server starts a resource adapter, it creates and configures a **ResourceAdapter** (RA) Java bean to activate a resource over the life cycle of the server for a stand-alone RAR, or for an embedded RAR, the life cycle of an application.

Before starting a resource adapter, the server now **validates the RA** instance as depicted in red in step 3. If validation fails, the RA instance is not placed into service, the adapter is not started, and the server will provide no connectivity to the resource.

For all this to occur, the RAR provider must have developed the RA class with properties and fields constrained with Bean Validation constraint metadata, and an administrator must have deployed the RAR onto a node. The administrator also might have configured the custom properties of the resource adapter for any required connection behaviors.

ResourceAdapter JavaBeans constraint

```
package com.ibm.adapter.jbv;

import javax.resource.spi.ResourceAdapter;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import java.io.Serializable;
...

public class JBVFATAdapterFailureImpl implements ResourceAdapter, Serializable {

    private int idleTimeout;

    @Max(value = 100) @Min(value=10)
    public int getIdleTimeout() {return idleTimeout;}

    public void setIdleTimeout(int idleTimeout) {this.idleTimeout = idleTimeout;}

    ...
```


This slide shows an example of an RA Java bean with a configuration property named “idleTimeout“. The getter methods for the property is constrained using the built-in “@Max” and “@Min” annotations defining a valid range for the value between 10 and 100, inclusive.

ResourceAdapter constraint violation (1 of 2)

If validation fails, the server rejects the RA instance during resource adapter startup by throwing a ContraintViolationException

```
[9/29/10 10:51:24:125 CDT] 00000000 BeanValidatio E   J2CA0238E: JavaBean
  com.ibm.adapter.jbv.JBVFATAdapterFailureImpl@7efa7efa failed Bean Validation due to
  one or more invalid configuration settings indicated in this list of constraint
  violations:
  ConstraintViolationImpl{interpolatedMessage='The minimum size is 2',
  propertyPath=dataBaseName, rootBeanClass=class
  com.ibm.adapter.jbv.JBVFATAdapterFailureImpl, messageTemplate='The minimum size is 2'}
  ConstraintViolationImpl{interpolatedMessage='must be greater than or equal to 10',
  propertyPath=idleTimeout, rootBeanClass=class
  com.ibm.adapter.jbv.JBVFATAdapterFailureImpl,
  messageTemplate='{javax.validation.constraints.Min.message}'}
…
[9/29/10 10:51:24:468 CDT] 00000000 RALifeCycleMa E   J2CA0128E: An Exception occurred
  while trying to start ResourceAdapter cells/IBM-46DF84D297BNode01Cell/nodes/IBM-
  46DF84D297BNode01/resources.xml#J2CResourceAdapter_1285109360562. The exception is:
  com.ibm.ejs.j2c.metadata.ConstraintViolationException
  at com.ibm.ejs.j2c.metadata.BeanValidationHelper.validate(
  at com.ibm.ejs.j2c.metadata.BeanValidationHelper.validate(
  at com.ibm.ejs.j2c.RAWrapperImpl.createAndConfigureRA(
  at com.ibm.ejs.j2c.RAWrapperImpl.startRA(
  at com.ibm.ejs.j2c.RALifeCycleManagerImpl.startRA(…
```

The resource adapter fails to start and applications can neither establish outbound connections to the resource nor receive messages from the resource

This slide shows a constraint violation that occurred while a starting resource adapter. The adapter is identified by its archive path shown on the slide.

The violation occurred because the instance of RA class “JBVFATAdapterFailureImpl” contains property “idleTimeout”, which was configured with an incorrect value.

If a ResourceAdapter Java bean fails validation, the resource adapter fails to start and the server cannot establish any connectivity to the resource.

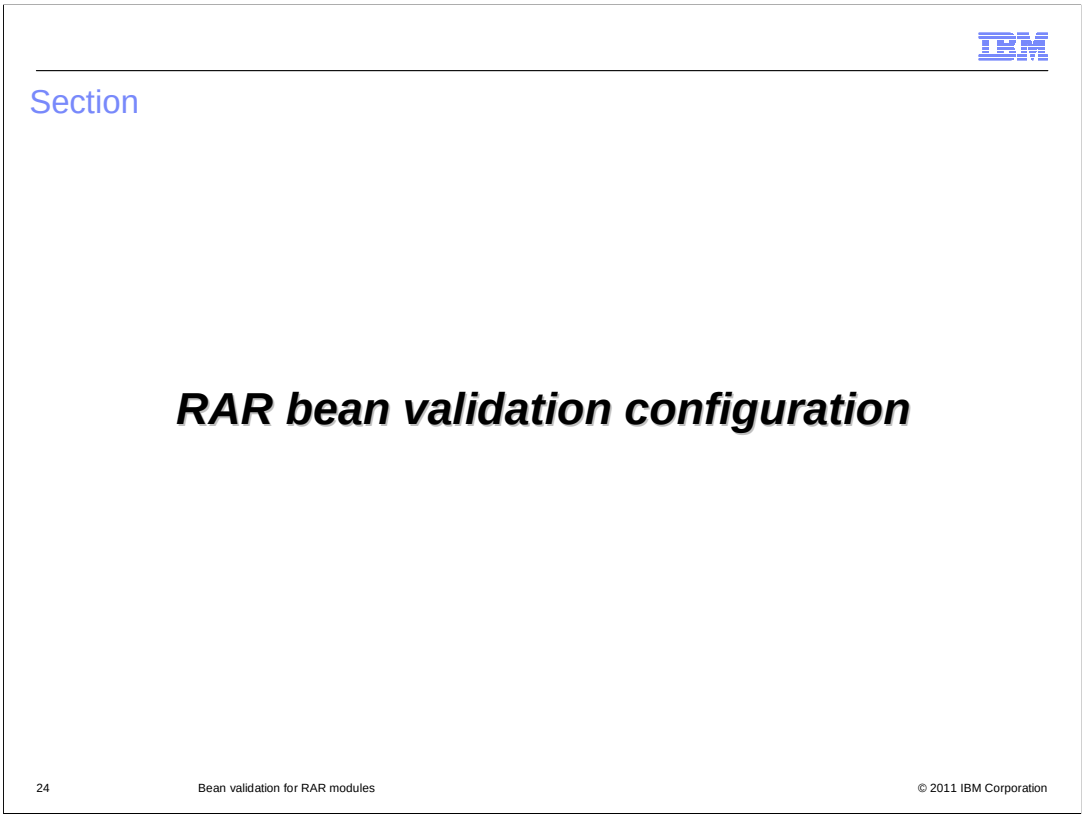
ResourceAdapter constraint violation (2 of 2)

- The administrator determines the problem, reconfigures any RA properties to the appropriate settings, and restarts the resource adapter.
  - For stand-alone resource adapters, restart the server
  - For embedded resource adapters, restart the application embedding the RAR
- Applications that require the resource cannot establish connections or receive messages while the RA configuration violates any constraints
- In-doubt transactions, including transactional messages, cannot be recovered while the RA configuration violates any constraints

When a violation occurs, an Administrator should examine the violation and determine the correct value for the property.

After setting the property to a valid value, the administrator might restart the resource adapter. If the resource adapter is embedded, the adapter will restart when the containing application is restarted; if the adapter is stand-alone, the adapter will restart when the server is restarted.

Because the ResourceAdapter is not placed into service, applications using the resource can neither get connections to the resource nor receive messages from the resource until the violation is alleviated.



This section describes the declaration of bean validation constraints within a validation XML configuration and how to package the validation configuration in a RAR module.

RAR bean validation descriptor - Validation.Xml

A Bean Validation descriptor that references constraint metadata

```
<?xml version="1.0" encoding="UTF-8"?>
<validation-config
   xmlns="http://jboss.org/xml/ns/javax/validation/configuration“
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=
      “http://jboss.org/xml/ns/javax/validation/configuration validation-configuration-1.0.xsd">
   <constraint-mapping>META-INF/constraints.xml</constraint-mapping>
</validation-config>
```


In the simplest case, a RAR validation XML configuration consists of a validation configuration descriptor, validation.xml, and at least one constraint resource XML file that is specified in a constraint-mapping element of the validation.xml file.

This slide displays a bean validation descriptor that references constraint metadata for the RAR JavaBeans.

RAR bean validation descriptor – Constraints.xml

Constraint-mappings of a Bean Validation descriptor

```
<constraint-mappings
   xmlns="http://jboss.org/xml/ns/javax/validation/mapping“
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance“
   xsi:schemaLocation=
      "http://jboss.org/xml/ns/javax/validation/mapping validation-mapping-1.0.xsd">

   <default-package>com.ibm.adapter.activationspec.jbv</default-package>
   <bean class="JBVFATActSpecEmbeddedImpl" ignore-annotations="false">
      <field name="asProperty5">
         <valid />
         <!-- @Max(10) -->
         <constraint annotation="javax.validation.constraints.Max">
            <message>Maximum possible value is 10</message>
            <element name="value">10</element>
         </constraint>
      </field>
   </bean>
</constraint-mappings>
```


This slide displays the constraint metadata referenced by the validation.xml file.

The metadata declares the value of the built-in annotation type “@Max” applied to a field of the JavaBeans class.

RAR bean validation descriptor – Packaging

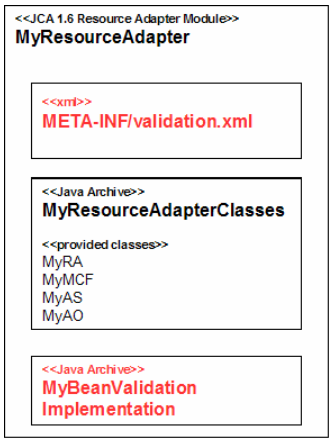
The RAR Bean Validation descriptor must be packaged in the META-INF directory

```
MyResourceAdapter.rar
   META-INF/
      ra.xml
      validation.xml
      constraints.xml
   com/
      my_company/
         MyResourceAdapter.class
         …
```


The validation configuration must be packaged in the META-INF directory of a RAR module. Any user-defined constraint annotation classes that are declared in the XML configuration must also be packaged in the RAR module.

This slide shows the organization of a RAR with classes contained package “com/my_company”, and a simple validation configuration in the META-INF directory.

Third-party RAR bean validation



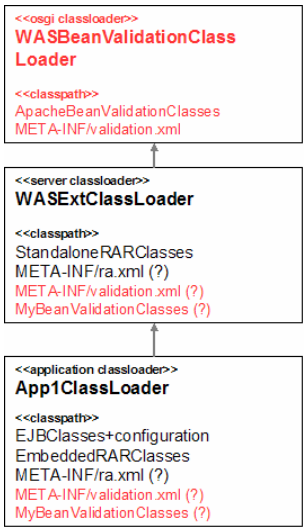
- Package the JAR file containing the bean validation implementation in the root directory of the RAR
- Package a single validation configuration descriptor (validation.xml) in the META-INF directory of the RAR module or the BeanValidation jar, but not both
- For RAR modules embedded within EAR modules, set the delegation mode of the application class loader to *Parent-Last* (*Child-First*)

WebSphere Application Server supports using different bean validation implementations.

If a resource adapter requires a bean validation implementation different from the implementation provided by the product, and the RAR provides the bean validation implementation, then package the JAR file that contains the bean validation implementation in the RAR module root directory.

The RAR module must also contain a single validation configuration descriptor (validation.xml), which can be packaged in the META-INF directory of the RAR module, or in the META-INF/services directory of the bean validation JAR file, but not both.

Validation configuration discovery



- The server attempts to discover and load the bean validation configuration specific to the RAR using the validation descriptor deployed in the RAR's META-INF directory
- If the descriptor does not exist, the server attempts to bootstrap the configuration using the first validation descriptor discovered in the RAR class loading context
- Finally, the server uses the default validation configuration provided by the product
- The server then creates a validator factory for the RAR specific to the discovered bean validation configuration
- Only one validation configuration might exist in the RAR class loading context!

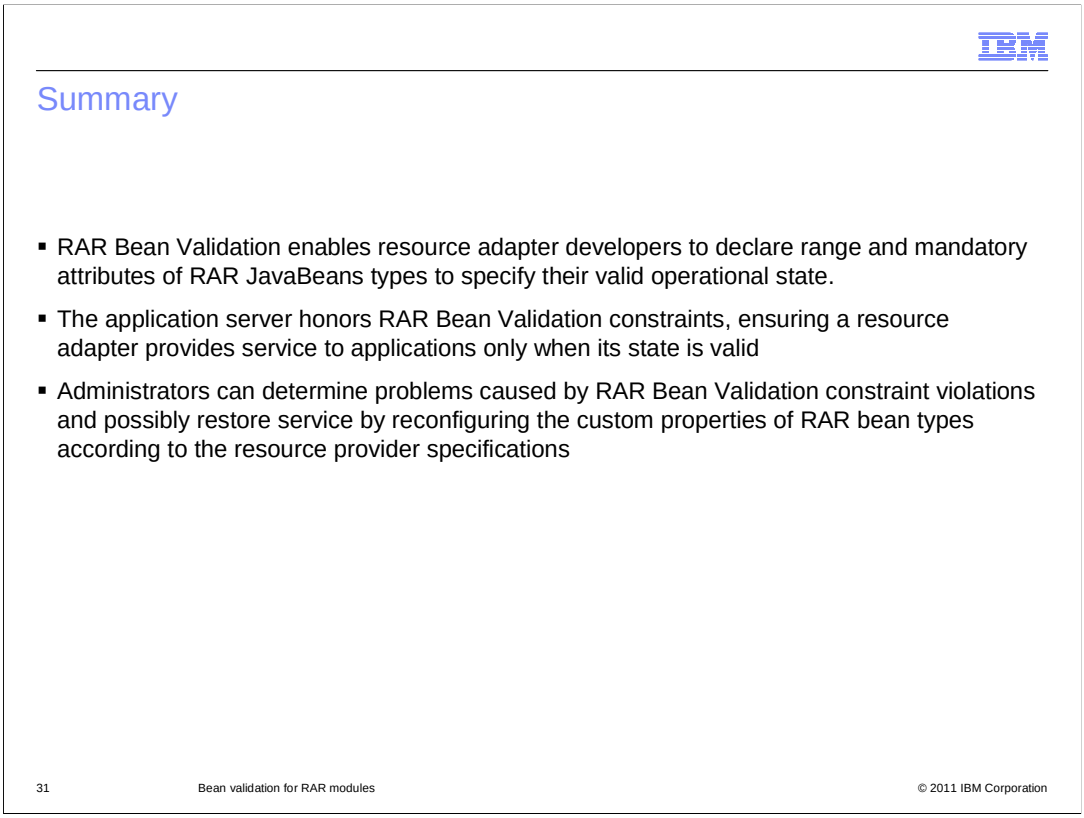
When validating RAR beans, the application server bootstraps the bean validation configuration, specific to the RAR, according to the bean validation descriptor supplied in the RAR META-INF directory.

If the descriptor does not exist, the server bootstraps the configuration using the first validation descriptor discovered in the RAR class loading context, such as that supplied in a third-party bean validation that is packaged in the RAR.

Finally, the server uses the default validation configuration provided by the product.

The server then creates a validator factory specific to the discovered bean validation configuration and uses this factory to create validator instances for validating the RAR bean instances.

This section provides a summary of the support for RAR bean validation and a list of end-user references.



RAR bean validation enables resource providers to declare range and mandatory attributes of RAR JavaBeans types in order to specify their valid operational state. The application server honors RAR bean validation constraints, better ensuring a resource adapter provides service to applications only when its state is valid. Administrators can determine problems caused by RAR bean validation constraint violations and possibly restore service by reconfiguring the custom properties of RAR bean types according to the resource provider specifications

References

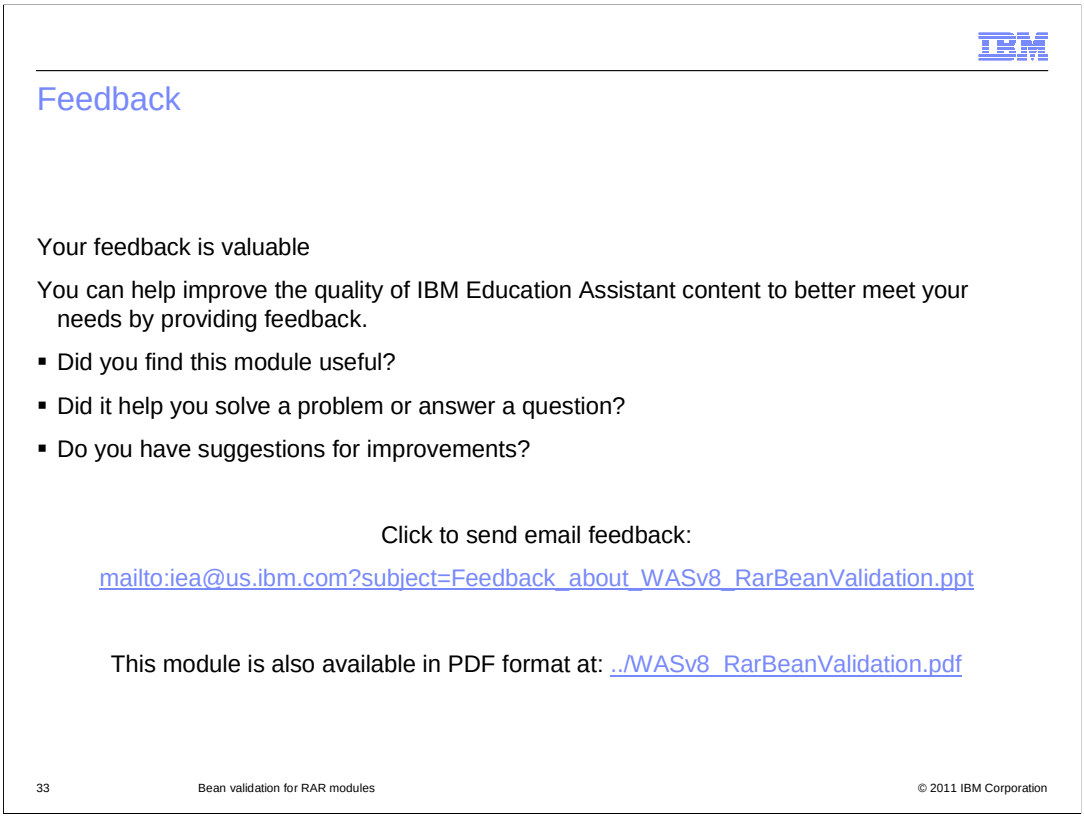
- WebSphere Application Server V8 Information Center topic *"Bean Validation in RAR modules"*
  http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/cdat_rarbeabval.html
- JSR 322 *"Java EE Connector Architecture 1.6"*
  http://jcp.org/en/jsr/detail?id=322
- JSR 303 *"Bean Validation"*
  http://jcp.org/en/jsr/detail?id=303

This slide lists references available to end-users.

The WebSphere Application Server V8 Information Center topic *"Bean Validation in RAR modules"* discusses RAR bean validation in detail.

JSR 322 "*Java EE Connector Architecture 1.6"* specifies the requirements of RAR bean validation for JCA1.6-compliant resource adapters.

And finally, JSR 303 *"Bean Validation"* specifies the requirements for bean validation for the Java platform.



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, 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.