



# BEA WebLogic Workshop™ Help

Version 8.1 SP4  
December 2004

# Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

## Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software–Restricted Rights Clause at FAR 52.227–19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227–7013, subparagraph (d) of the Commercial Computer Software—Licensing clause at NASA FAR supplement 16–52.227–86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

## Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E–Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

# Table of Contents

<b>Annotations Reference.....</b>	<b>1</b>
<b>Common Annotations.....</b>	<b>2</b>
<b>@common:context Annotation.....</b>	<b>3</b>
<b>@common:control Annotation.....</b>	<b>4</b>
<b>@common:define Annotation.....</b>	<b>5</b>
<b>@common:message-buffer Annotation.....</b>	<b>7</b>
<b>@common:operation Annotation.....</b>	<b>9</b>
<b>@common:schema Annotation.....</b>	<b>10</b>
<b>@common:security Annotation.....</b>	<b>11</b>
<b>@common:target-namespace Annotation.....</b>	<b>14</b>
<b>@common:xmlns Annotation.....</b>	<b>16</b>
<b>Java Control Annotations.....</b>	<b>17</b>
<b>@jc:connection Annotation.....</b>	<b>19</b>
<b>@jc:conversation Annotation.....</b>	<b>21</b>
<b>@jc:ejb Annotation.....</b>	<b>22</b>
<b>@jc:handler Annotation.....</b>	<b>24</b>
<b>@jc:jms Annotation.....</b>	<b>26</b>
<b>@jc:jms-headers Annotation.....</b>	<b>29</b>
<b>@jc:jms-property Annotation.....</b>	<b>32</b>
<b>@jc:location Annotation.....</b>	<b>34</b>
<b>@jc:log Annotation.....</b>	<b>35</b>
<b>@jc:parameter-xml Annotation.....</b>	<b>36</b>
<b>@jc:protocol Annotation.....</b>	<b>37</b>

# Table of Contents

<b>@jc:reliable Annotation.....</b>	<b>38</b>
<b>@jc:return-xml Annotation.....</b>	<b>39</b>
<b>@jc:sql Annotation.....</b>	<b>40</b>
<b>@jc:timer Annotation.....</b>	<b>43</b>
<b>@jc:ws-security-callback Annotation.....</b>	<b>45</b>
<b>@jc:ws-security-service Annotation.....</b>	<b>46</b>
<b>@jc:wSDL Annotation.....</b>	<b>47</b>
<b>Editor-Info Annotations.....</b>	<b>48</b>
<b>@editor-info:code-gen Annotation.....</b>	<b>49</b>
<b>@editor-info:link Annotation.....</b>	<b>50</b>
<b>Java Web Service Annotations.....</b>	<b>51</b>
<b>@jws:context Annotation (Deprecated).....</b>	<b>53</b>
<b>@jws:control Annotation (Deprecated).....</b>	<b>54</b>
<b>@jws:conversation Annotation.....</b>	<b>55</b>
<b>@jws:conversation-lifetime Annotation.....</b>	<b>57</b>
<b>@jws:define Annotation (Deprecated).....</b>	<b>59</b>
<b>@jws:handler Annotation.....</b>	<b>60</b>
<b>@jws:location Annotation.....</b>	<b>62</b>
<b>@jws:message-buffer Annotation (Deprecated).....</b>	<b>63</b>
<b>@jws:operation Annotation (Deprecated).....</b>	<b>64</b>
<b>@jws:parameter-xml Annotation.....</b>	<b>65</b>
<b>@jws:protocol Annotation.....</b>	<b>68</b>
<b>@jws:reliable Annotation.....</b>	<b>72</b>

# Table of Contents

<b>@jws:return-xml Annotation.....</b>	<b>74</b>
<b>@jws:schema Annotation (Deprecated).....</b>	<b>77</b>
<b>@jws:target-namespace Annotation (Deprecated).....</b>	<b>78</b>
<b>@jws:ws-security-callback Annotation.....</b>	<b>79</b>
<b>@jws:ws-security-service Annotation.....</b>	<b>80</b>
<b>@jws:wSDL Annotation.....</b>	<b>81</b>
<b>@jws:xmlns Annotation (Deprecated).....</b>	<b>83</b>
<b>@jpf:action Annotation.....</b>	<b>85</b>
<b>@jpf:catch Annotation.....</b>	<b>90</b>
<b>@jpf:controller Annotation.....</b>	<b>93</b>
<b>@jpf:exception-handler Annotation.....</b>	<b>96</b>
<b>@jpf:forward Annotation.....</b>	<b>98</b>
<b>@jpf:message-resources Annotation.....</b>	<b>104</b>
<b>@jpf:validation-error-forward Annotation.....</b>	<b>106</b>
<b>@jpf:view-properties Annotation.....</b>	<b>111</b>
<b>Custom Control Annotations.....</b>	<b>113</b>
<b>@jcs:control-tags Annotation.....</b>	<b>114</b>
<b>@jcs:ide Annotation.....</b>	<b>115</b>
<b>@jcs:jc-jar Annotation.....</b>	<b>116</b>
<b>@jcs:suppress-common-tags Annotation.....</b>	<b>118</b>
<b>Enterprise JavaBean Annotations.....</b>	<b>119</b>
<b>@ejbgen:automatic-key-generation Annotation.....</b>	<b>122</b>
<b>@ejbgen:cmp-field Annotation.....</b>	<b>124</b>

## Table of Contents

<b>@ejbgen:cmr-field Annotation.....</b>	<b>128</b>
<b>@ejbgen:compatibility Annotation.....</b>	<b>130</b>
<b>@ejbgen:create-default-dbms-tables Annotation.....</b>	<b>131</b>
<b>@ejbgen:ejb-client-jar Annotation.....</b>	<b>132</b>
<b>@ejbgen:ejb-interface Annotation.....</b>	<b>133</b>
<b>@ejbgen:ejb-local-ref Annotation.....</b>	<b>134</b>
<b>@ejbgen:ejb-ref Annotation.....</b>	<b>137</b>
<b>@ejbgen:entity-cache-ref Annotation.....</b>	<b>140</b>
<b>@ejbgen:entity Annotation.....</b>	<b>142</b>
<b>@ejbgen:env-entry Annotation.....</b>	<b>150</b>
<b>@ejbgen:file-generation Annotation.....</b>	<b>152</b>
<b>@ejbgen:finder Annotation.....</b>	<b>156</b>
<b>@ejbgen:foreign-jms-provider Annotation.....</b>	<b>159</b>
<b>@ejbgen:jar-settings Annotation.....</b>	<b>160</b>
<b>@ejbgen:jndi-name Annotation.....</b>	<b>163</b>
<b>@ejbgen:local-home-method Annotation.....</b>	<b>164</b>
<b>@ejbgen:local-method Annotation.....</b>	<b>166</b>
<b>@ejbgen:message-driven Annotation.....</b>	<b>168</b>
<b>@ejbgen:method-isolation-level-pattern Annotation.....</b>	<b>172</b>
<b>@ejbgen:method-permission-pattern Annotation.....</b>	<b>173</b>
<b>@ejbgen:relation Annotation.....</b>	<b>175</b>
<b>@ejbgen:relationship-caching-element Annotation.....</b>	<b>178</b>
<b>@ejbgen:remote-home-method Annotation.....</b>	<b>180</b>

## Table of Contents

<b>@ejbgen:remote-method Annotation.....</b>	<b>182</b>
<b>@ejbgen:resource-env-ref Annotation.....</b>	<b>184</b>
<b>@ejbgen:resource-ref Annotation.....</b>	<b>186</b>
<b>@ejbgen:role-mapping Annotation.....</b>	<b>188</b>
<b>@ejbgen:security-role-ref Annotation.....</b>	<b>189</b>
<b>@ejbgen:select Annotation.....</b>	<b>191</b>
<b>@ejbgen:session Annotation.....</b>	<b>193</b>
<b>@ejbgen:value-object Annotation.....</b>	<b>198</b>

# Annotations Reference

This section provides reference information about WebLogic Workshop annotations, which are formatted like Javadoc tags.

WebLogic Workshop provides custom annotations based on Javadoc technology. Originally developed as a way to embed documentation into source code as comments, Javadoc is extended by WebLogic Workshop through custom annotations that help to define the functionality of a web application component. For example, annotations can be used to define the purpose of a page flow's action method, or a Java control, or a web service.

## Common Annotations

Annotations used commonly in WebLogic Workshop operations.

## Java Control Annotations

Annotations provided by Java controls.

## Java Web Service Annotations

Annotations used in building web services.

## Page Flow Annotations

Annotations used in building page flows.

## Enterprise JavaBean Annotations

Annotations used in building EJBs.



# Common Annotations

These annotations may be available to more than one type of file in WebLogic Workshop.

## Topics Included in This Section

`@common:context` Annotation

Specifies a `JwsContext` object.

`@common:control` Annotation

Specifies a WebLogic Workshop control.

`@common:define` Annotation

Defines inline data that might otherwise be referenced as an external file.

`@common:message-buffer` Annotation

Specifies that there should be a queue between the service's implementation code and the message transport mechanism for the specified method.

`@common:operation` Annotation

Specifies that the associated method is part of the web service's public contract and is available for clients to invoke.

`@common:schema` Annotation

Specifies the schema file whose types are referenced in the `@jws:parameter-xml` and `@jws:return-xml` annotations elsewhere in a JWS or JCX file.

`@common:security` Annotation

Specifies the schema file whose types are referenced in the `@jws:parameter-xml` and `@jws:return-xml` annotations elsewhere in a JWS or JCX file.

`@common:target-namespace` Annotation

Specifies the default XML namespace used for outgoing XML messages and generated WSDL files.

`@common:xmlns` Annotation

Defines an XML namespace prefix for use elsewhere in a JWS file or Web Service control JCX file.

Related Topics

Annotation Reference

# @common:context Annotation

The @common:context annotation specifies that WebLogic Server should create a context for the component (e.g., a web service or a control). The context ensures that conversations between the component and a client are correlated correctly and that the component's state is maintained. The @common:context annotation precedes the declaration of the context object; the type of the context object depends on the container you are using. For web services, the context object is of type JwsContext; for controls, ControlContext. Other components may provide their own context objects.

**Note:** In WebLogic Workshop 7.0, web service context was designated with the @jws:context annotation. This annotation is still supported in JWS files for backward compatibility.

## Syntax

@common:context

## Attributes

None.

## Remarks

The following rules apply to this annotation's use:

- Only one @common:context annotation may appear within a single Javadoc comment block.
- The @common:context annotation must appear on the instance declaration of the context object.

For example, the annotation and instance declaration for the JwsContext object appear as follows:

```
/** @common:context */  
JwsContext context;
```

If the @common:context annotation is not present on the instance declaration, the component that defines it will not function properly.

Related Topics

JwsContext Interface

Structure of a JWS File

# @common:control Annotation

The @common:control annotation indicates that the object annotated by this annotation is a WebLogic Workshop control in a JCX file.

*Note:* In WebLogic Workshop 7.0, controls in web services were designated with the @jws:control annotation. If you are referencing a control that was created in WebLogic Workshop 7.0 and is a file of type CTRL, you must use the @jws:control annotation. If you are referencing a control that was created in WebLogic Workshop 8.1 and is a file of type JCX, you must use the @common:control annotation.

## Syntax

@common:control

## Attributes

None.

## Remarks

The following rules apply to this annotation's use:

- Only one @common:control annotation may appear within a single Javadoc comment block.
- Must appear on each control instance declaration.

If the @common:control annotation is not present on a control instance declaration, the control will function properly. Attempts to invoke control methods will result in Null Pointer Exceptions (NPEs).

## Related Topics

None

# @common:define Annotation

The @common:define annotation defines inline data with the component class that might otherwise be referenced as an external file.

*Note:* In WebLogic Workshop 7.0, inline data in a web service was designated with the @jws:define annotation. This annotation is still supported for backward compatibility.

## Syntax

@common:define

name="nameOfInlineData"

value::

data referred to by

the name attribute

::

## Attributes

name

Required. The name to use when referring to the data.

value

Required. The data referred to by the name. Can be a string that contains a multiline value delimited by :: (two colons) delimiters.

## Remarks

The following rules apply to this annotation's use:

- Optionally may appear in a comment at the end of a control file.

The @common:define annotation is used to define inline data that might otherwise be referenced as an external file.

This is used, for example, to include relevant schema or WSDL files in a control file or JWS file.

## Example

In the example control file below, the `@common:define` annotation defines the name `WorldpProxyWsd1` to refer to the contents of a WSDL file. The entire contents of the WSDL file are included as the value of the `@common:define` annotation. The `@jws:wsdl` annotation then references the name defined by the `@common:define` annotation.

```
/**
 * @jws:wsdl file="#WorldpProxyWsd1"
 */
public interface WorldpProxy extends ServiceControl
{
    ...
}
/** @common:define name="WorldpProxyWsd1" value::
 * <xml version="1.0" encoding="utf-8"?>
 * <definitions xmlns:s="http://www.w3.org/2001/XMLSchema">
 *     ...< remainder of contents of WSDL file>...
 * </definitions>
 * ::
 */
```

### Related Topics

[@jc:wsdl Annotation](#)

[JCX Files: Implementing Controls](#)

[WSDL Files: Web Service Descriptions](#)

[Web Service Control](#)

# @common:message-buffer Annotation

The @common:message-buffer annotation specifies that there should be a queue between the component's implementation code and the message transport wire for the specified method or callback.

*Note:* In WebLogic Workshop 7.0, a message buffer on a method or callback was designated with the @jws:message-buffer annotation. This annotation is still supported in JWS files for backward compatibility.

## Syntax

@common:message-buffer

[enable="true | false"]

[retry-count="number of attempts"]

[retry-delay="duration"]

## Attributes

enable

Required. Specifies whether the messages arriving or departing via the associated method or callback are queued.

retry-count

Optional. The number of times WebLogic Server will attempt to deliver queued messages to the service.

retry-delay

Optional. The amount of time that should pass between message delivery attempts.

## Remarks

The following rules apply to this annotation's use:

- Only one @common:message-buffer annotation may appear within a single Javadoc block.
- Optionally may appear in front of a @common:operation method in a JWS file.
- Optionally may appear in front of a callback method (event declaration) in a JWS file.
- Optionally may appear in front of an event handler method (control\_eventName method) in a JWS file.
- Optionally may appear in front of a method declaration in a control file.

The @common:message-buffer annotation may only be placed on methods or callbacks with return type void.

The @common:message-buffer annotation places a queue between the method and the communication wire. For incoming messages, this means that the message is queued before it is processed. For outgoing messages,

this means that the message is queued before it is sent on the wire.

Note that the queue is transparent to the method that is marked with the `@common:message-buffer` annotation. When multiple messages arrive, they may be queued (or for the client, in the case of a callback) while the method is processing a previous message. Callers invoking a buffered method or callback will always experience an immediate return.

The `retry-delay` attribute can be specified using friendly names for units of time. For example, the default unit is seconds, and can be expressed as 30 seconds. The following are also possible values:

- 5 minutes
- 24 hours
- 3 days

The unit for the `retry-delay` value can also be expressed as a partial name, as follows:

- 30 s
- 30 sec
- 30 seconds

Note that all parameters to methods and callbacks that are marked with the `@common:message-buffer` annotation must be serializable; that is, they must implement `java.io.Serializable`. For an example, see [Using Buffering to Create Asynchronous Methods and Callbacks](#).

## Controlling Retry Behavior

You may control retry behavior of buffered methods at runtime by using `weblogic.jws.RetryException`. See [Controlling Retry Behavior](#) in the topic [Using Buffering to Create Asynchronous Methods and Callbacks](#).

Related Topics

[Designing Asynchronous Interfaces](#)

[Using Buffering to Create Asynchronous Methods and Callbacks](#)

# @common:operation Annotation

Specifies that the associated method or callback on the component (*e.g.*, a web service or a control) is available for clients to invoke.

**Note:** In WebLogic Workshop 7.0, web service operations were designated with the @jws:operation annotation. This annotation is still supported in JWS files for backward compatibility.

## Syntax

@common:operation

## Attributes

None.

## Remarks

The following rules apply to this annotation's use:

- Only one @common:operation annotation may appear within a single Javadoc comment block.
- Must appear on each method or callback that is to be exposed to the client.
- The method or callback must be public.

Related Topics

Structure of a JWS File



# @common:schema Annotation

Specifies the schema file whose types are referenced in the component class.

*Note:* In WebLogic Workshop 7.0, a schema file in a web service was specified with the @jws:schema annotation. This annotation is still supported in JWS files for backward compatibility.

## Syntax

@common:schema

file="schemaToImport"

inline="true | false"

## Attributes

file

Required. The filename or @common:define annotation referring to the schema to import.

inline

Required. A boolean indicating whether the schema is defined within the file (**true**) or externally (**false**). The default is **true**.

## Remarks

The following rules apply to this annotation's use:

- Multiple @common:schema annotations are allowed within a single Javadoc comment block.
- Optionally this annotation may appear in front of a class in a JWS file.
- Optionally this annotation may appear in front of an interface in a control file.

The file attribute should refer to a file or a @common:define annotation that contains a schema file.

Any number of schema files can be imported by using multiple @common:schema annotation declarations. When outputting WSDL files, imported schemas are copied and inserted inline into the WSDL.

Related Topics

@jws:parameter-xml Annotation

@jws:return-xml Annotation

@common:define Annotation

@jc:wsl Annotation

# @common:security Annotation

Specifies the security configuration for a class or an individual method within a web service or Java control class.

Note that the @common:security annotation provides role–mapping with scoped, not global roles, and assumes that the subject has already been authenticated by WebLogic Server's security framework. The role referenced by the @common:security annotation applies to the EJB produced when Workshop compiles the web service or Java control.

## Syntax

@common:security

roles–allowed="space\_separated\_list\_of\_roles\_permitted\_to\_access\_the\_object"

roles–referenced="space\_separated\_list\_of\_roles"

run–as="single\_role\_name"

run–as–principal="single\_principal\_name"

single–principal="true | false"

callback–roles–allowed="space\_separated\_list\_of\_roles\_permitted\_to\_callback\_the\_object"

## Attributes

roles–allowed

Optional. Specify a list of roles permitted to access the object annotated by @common:security. Individual roles listed must be separated by a single space. If @common:security is applied at the class level, then the roles referenced may access all individual methods within the class regardless of any further role restrictions placed on individual methods. The roles allowed are defined in the underlying ejb–jar.xml files and a role–principal mapping, with a principal that is given the same name as the role name, is defined in the underlying weblogic–ejb–jar.xml file. For more information, see Role–Based Security.

roles–referenced

Optional. Specifies a list of roles to which there are programmatic references ( .hasRole("Admin") ) in the class or method code. The annotation causes the generated runtime code to include a reference to the roles in the resulting deployment descriptor.

run–as

Optional. A web resource (a class or method) that includes this attribute assumes the permission–level of the role specified and may access other resources accordingly.

## WebLogic Workshop Annotations Reference

Note that the `run-as` attribute signifies an externally defined role. When `run-as` appears in a JWS file, the EJB deployment descriptor (`weblogic-ejb-jar.xml`) will mark the role with an externally-defined tag: `<externally-defined/>`. To successfully deploy, the role referred to must exist in the target server's security realm.

If `run-as` is present without `run-as-principal`, then the `run-as` value is assumed to be a principal *and* role name.

Note that `run-as` is only applied in a top-level context. It is ignored in a nested context, because `run-as` relies on the generated EJB deployment descriptor, a deployment descriptor possessed only by top-level elements.

### `run-as-principal`

Optional. A web resource (a class or method) that includes this attribute assumes the permission-level of the principal specified and may access other resources accordingly. Note that if you specify the `run-as-principal` attribute, you must also specify the `run-as` attribute.

### `single-principal`

Optional. Takes a boolean value. If true, only the principal who started the conversation can continue and finish the conversation. If false, a conversation can be continued and finished by another (appropriately authorized) user.

### `callback-roles-allowed`

Optional. Specify a list of roles permitted to callback the object annotated by `@common:security`. Individual roles listed must be separated by a single space.

The `callback-roles-allowed` annotation may appear:

(1) On a JWS file, provided that there is a control declared within the JWS file and this control implements `com.bea.control.ExternalCallbackTarget`.

(2) On JCS file, provided that there is a web service control declared within the JCS file and this control implements `com.bea.control.ExternalCallbackTarget`.

(3) Inline on the declaration of the control. Assume that `BankControl` is a JCS control file.

```
/**
 * @common:control
 * @common:security callback-roles-allowed="AccountHolders"
 */
private controls.BankControl bankControl;
```

You cannot place `callback-roles-allowed` on a JCX file.

## Examples

### Example #1

```
/**
 * @common:operation
```

## WebLogic Workshop Annotations Reference

```
* @common:security roles-allowed="friends"
*/
public String hello()
{
    return "Hello Friends!";
}
```

### Example #2

```
/**
 * @common:security single-principal="false"
 */
public class PurchaseSupplies implements com.bea.jws.WebService
{
    /**
     * @common:operation
     * @jws:conversation phase="start"
     */
    public void requestPurchase()
    {
    }

    /**
     * @common:operation
     * @jws:conversation phase="continue"
     */
    public void approvePurchase()
    {
    }

    /**
     * @common:operation
     * @jws:conversation phase="finish"
     */
    public void executePurchase()
    {
    }
}
```

### Related Topics

Security

Role-based Security

@jpf:controller Annotation

@jpf:action Annotation

# @common:target-namespace Annotation

The @common:target-namespace annotation specifies the XML namespace used for outgoing XML messages and generated WSDL files.

**Note:** In WebLogic Workshop 7.0, the namespace in a web service was specified with the @jws:target-namespace annotation. This annotation is still supported in JWS files for backward compatibility.

## Syntax

@common:target-namespace

namespace="defaultXMLNamespace"

## Attributes

namespace

Required. Specifies the default namespace used for outgoing XML messages and generated WSDL files.

## Remarks

The following rules apply to this annotation's use:

- Optionally may appear on the main class in a JWS file.
- Optionally may appear on the main interface in a control file.

If no target namespace is specified (this annotation is not present), the namespace used is <http://www.openuri.org/>.

**Note:** Before deploying web services, you should define a unique default namespace for your organization. If multiple organizations were to leave the default namespace as <http://www.openuri.org/>, it could result in namespace conflicts.

The @common:target-namespace annotation affects the behavior of the following XML generation actions:

- If there is no XML mapping specified for an incoming or outgoing message, a default map is used (the natural mapping). The elements of the resulting XML message must be specified in some namespace. The value of @common:target-namespace is used.
- If a custom XML mapping is present for an incoming or outgoing message but no namespace is specified, the value of @common:target-namespace is used.
- WSDL files generated from JWS files specify a namespace for the elements in the WSDL file. The value of @common:target-namespace is used.

Note that if @common:target-namespace is not defined, the default value is <http://www.openuri.org/>.

For more information on WSDL files, see WSDL Files.

Related Topics

[Introduction to XML](#)

# @common:xmlns Annotation

The @common:xmlns annotation defines an XML namespace prefix for use elsewhere in the component class.

*Note:* In WebLogic Workshop 7.0, the XML namespace prefix was designated by the @jws:xmlns annotation. This annotation is still supported in JWS files for backward compatibility.

## Syntax

```
@common:xmlns
```

```
prefix="namespacePrefix"
```

```
namespace="namespaceURI"
```

## Attributes

prefix

Required. Specifies the XML namespace prefix being defined.

namespace

Required. Specifies the URI that the prefix represents.

## Remarks

The following rules apply to this annotation's use:

- Multiple @common:xmlns annotations may appear within a single Javadoc comment block.
- May appear on the main class declaration in a JWS file or the main interface declaration in a control file.

The following XML namespace prefixes are implicitly defined in WebLogic Workshop:

- xm, with namespace URI <http://www.bea.com/2002/04/xmlmap/>
- xsd, with namespace URI <http://www.w3.org/2001/XMLSchema>

The xm prefix is used in XML map annotations such as <xm:value> and <xm:multiple>.

If the xm or xsd prefixes are explicitly defined in a JWS or control file, the implicit definitions described above are overridden for that file.

To learn about XML namespaces and prefixes, see [Introduction to XML](#).

Related Topics

[Introduction to XML](#)

@common:xmlns Annotation

# Java Control Annotations

The Java control annotations provide information to WebLogic Server about how a control functions.

## Topics Included in this Section

@jc:connection Annotation

Specifies the data source used by a Database control.

@jc:conversation Annotation

Specifies the conversation phase for an operation on the Web Service control.

@jc:ejb Annotation

Specifies the target EJB's home interface for an EJB control.

@jc:handler Annotation

Specifies SOAP message handlers for a Web Service control.

@jc:jms Annotation

Specifies the configuration attributes of a JMS control.

@jc:jms-headers Annotation

Specifies XML maps for headers of messages processed by a JMS control.

@jc:jms-property Annotation

Specifies XML maps for properties of messages processed by a JMS control.

@jc:location Annotation

Specifies the URL at which a Web Service control accepts requests for each supported protocol.

@jc:log Annotation

Specifies the log4j category for a Database control.

@jc:parameter-xml Annotation

Specifies characteristics for marshaling data between XML messages and the data provided to the parameters of a web service operation.

@jc:protocol Annotation



## WebLogic Workshop Annotations Reference

Specifies which protocols and message formats can be accepted by the web service represented by a web service control, and by the operations on that web service.

### @jc:reliable Annotation

Indicates that a web service exposed by a Web Service control uses reliable messaging, and specifies how long messages must be retained by the server in order to perform detection and removal of duplicates.

### @jc:return-xml Annotation

Defines mappings between the return type of the method marked by this annotation and XML message data.

### @jc:sql Annotation

Specifies the SQL statement and associated attributes that correspond to a method in a Database control.

### @jc:timer Annotation

Specifies configuration attributes for a Timer control.

### @jc:ws-security-service Annotation

Specifies the policy file (WSSE file) used to secure (non-callback) traffic between a web service and its clients.

### @jc:ws-security-callback Annotation

Specifies the policy file (WSSE file) used to secure callback traffic between a web service and its clients.

### @jc:wsdl Annotation

Specifies the URL at which a Web Service control accepts requests for each supported protocol.

Related Topics

Annotation Reference

# @jc:connection Annotation

Specifies the data source used by a Database control.

**Note:** The @jc:connection annotation appears in Database controls with the .jcx extension. Database controls with the .ctrl extension, which were created in a previous version of WebLogic Workshop, use the @jws:connection annotation. Controls with the .ctrl extension continue to be supported in WebLogic Workshop.

## Syntax

@jc:connection

data-source-jndi-name="JNDINameofDataSource"

## Attributes

data-source-jndi-name

Required. A string specifying the JNDI name of a defined data source. Default: null.

## Remarks

The following rules apply to this annotation's use:

- Only one @jc:connection annotation may appear within a single Javadoc comment block.
- The @jc:connection annotation must appear in the Javadoc comment on the main interface defined in a Database control's JCX file.

JNDI is the Java Naming and Directory Interface. data-source-jndi-name must be a valid name of a data source as it appears in the local JNDI registry.

To learn how to create, configure and register a new data source, please consult the WebLogic Server documentation on JDBC Data Sources.

Two data sources are pre-configured when WebLogic Workshop is installed. Both operate on the default PointBase database system by default. The pre-configured data source names are:

- cgSampleDataSource: the default data source name that is the initial value for @jc:connection when a new Database control is created. This data source is intended for experimentation. It is used by all Database controls in the samples project.
- cgDataSource: a data source used by WebLogic Workshop for internal bookkeeping. You should not perform any database activities against this data source. When WebLogic Server web services are deployed to a production environment, cgDataSource should be redirected to a more robust database system.

Related Topics

Database Control

@jc:connection Annotation

## WebLogic Workshop Annotations Reference

How Do I: Connect a Database Control to a Database Such as SQL Server or Oracle?

How Do I: Configure WebLogic Workshop to Use a Different Database for Internal State?

How Do I: WebLogic Workshop—Enable an Existing WebLogic Server Domain?

@jc:sql Annotation

# **@jc:conversation Annotation**

The @jc:conversation annotation specifies the role that a control's methods or callbacks play in a conversation. This annotation appears in a web service control that is based on a conversational web service, so it is essentially identical to the corresponding web service annotation, @jws:conversation. For more information on using this annotation, see @jws:conversation.

## **Related Topics**

[@jws:conversation](#)

[Web Service Control](#)

# @jc:ejb Annotation

Specifies the target EJB's home interface for an EJB control.

The home interface set here can be overridden at runtime through the EJB control method `setJndiName()`. The method has the same syntax as the attribute `home-jndi-name` below. For more information see [Creating a New EJB Control](#).

**Note:** The `@jc:ejb` annotation appears in EJB controls with the `.jcx` extension. EJB controls with the `.ctrl` extension, which were created in a previous version of WebLogic Workshop, use the `@jws:ejb` annotation. Controls with the `.ctrl` extension continue to be supported in WebLogic Workshop.

## Syntax

`@jc:ejb`

`(home-jndi-name="JNDInameOfTargetEJB" |`

`ejb-link="RelativeNameOfTargetEJB")1`

## Attributes

`home-jndi-name`

Exactly one attribute of the set must be used. Specifies the JNDI name of the target EJB's home interface. For instance, `"EJBNameHome"`. To access an EJB on a different server in the same domain, use:

`@jc:ejb home-jndi-name="jndi://username:password@host:7001/my.resource.jndi.object"`

For more information, see [Creating a New EJB Control](#)

`ejb-link`

Exactly one attribute of the set must be used. Specifies the name of the target EJB using the application relative path to the EJB JAR. This syntax causes the runtime to use an application scoped name rather than a global JNDI name when locating the referenced EJB. The naming syntax is `BeanName#EJBJAR`, for instance, `CreditCard#CustomerData.jar`.

## Remarks

The following rules apply to this annotation's use:

- Exactly one attribute must be specified for the `@jc:ejb` annotation.
- Only one `@jc:ejb` annotation may appear within a single Javadoc comment block.
- Must appear in front of the main interface definition in an EJB control's JCX file.

The target EJB must be deployed in WebLogic Server in the same domain as WebLogic Workshop project containing the EJB control.

Related Topics

EJB Control

# @jc:handler Annotation

Specifies SOAP message handlers for a Web Service control.

To learn about SOAP message handlers and how to configure them in WebLogic Workshop Web Service controls, see [Specifying SOAP Handlers for a Web Service](#).

## Syntax

@jc:handler

[operation="<handler-spec>"]

[callback="<handler-spec>"]

[file="<handler-config-file-name>"]

## Attributes

operation

Optional. Specifies the handler(s) that will process request and response SOAP messages associated with operations (methods) of a Web Service control. May be a space-separated list of handler class names or the name of a <handler-chain> defined in a handler configuration file specified by the file attribute. Each handler class must implement the `javax.xml.rpc.handler.Handler` interface.

callback

Optional. Specifies the handler(s) that will process request and response SOAP messages associated with callbacks of a Web Service control. May be a space-separated list of handler class names or the name of a <handler-chain> defined in a handler configuration file specified by the file attribute. Each handler class must implement the `javax.xml.rpc.handler.Handler` interface.

file

Optional. Specifies a handler configuration file. A handler configuration file is an XML file that defines handlers, handler chains and handler initialization data. See [Example Handler Configuration File](#), below.

## Remarks

The @jc:handler annotation is used to associate SOAP message handlers with a Web Service control. Each SOAP handler is a class that implements the `javax.xml.rpc.handler.Handler` interface.

The following rules apply to this annotation's use:

- The @jc:handler annotation may only be applied to the main interface in a Web Service control JCX file.
- Only one @jc:handler annotation is allowed per JCX file.

## Example Handler Configuration File

A handler configuration file is an XML file that conforms to the WebLogic Workshop handler configuration schema. The example below illustrates a sample handler configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<hc:wlw-handler-config xmlns:hc="http://www.bea.com/2003/03/wlw/handler/config/">
  <hc:handler-chain name="LoggingHandler">
    <hc:handler handler-name="Logger" handler-class="handler.ConsoleLoggingHandler"/>
      <hc:init-param>
        <hc:description>First Param</hc:description>
        <hc:param-name>param1</hc:param-name>
        <hc:param-value>value1</hc:param-value>
      </hc:init-param>
      <hc:init-param>
        <hc:description>Second Param</hc:description>
        <hc:param-name>param2</hc:param-name>
        <hc:param-value>value2</hc:param-value>
      </hc:init-param>
    <hc:handler handler-name="Auditor" handler-class="handler.AuditHandler"/>
  </hc:handler-chain>
</hc:wlw-handler-config>
```

The handler-class attribute of each <handler> element must refer to a handler class that implements the javax.xml.rpc.handler.Handler interface.

Any <init-param> elements that are present for a given <handler> element will be passed to the handler's init() method when the handler is initialized.

### Related Topics

Specifying SOAP Handlers for a Web Service

@jws:handler Annotation



# @jc:jms Annotation

Specifies the configuration attributes of a JMS control.

**Note:** The @jc:jms annotation appears in JMS controls with the .jcx extension. JMS controls with the .ctrl extension, which were created in a previous version of WebLogic Workshop, use the @jws:jms annotation. Controls with the .ctrl extension continue to be supported in WebLogic Workshop.

## Syntax

@jc:jms

[send-type="queue | topic"]

[send-jndi-name="JNDInameOfSendDestination"]

[receive-type="queue | topic"]

[receive-jndi-name="JNDInameOfReceiveDestination"]

[connection-factory-jndi-name="JNDInameOfConnectionFactory"]

[send-correlation-property="JMSheaderOrPropertyFieldName"]

[receive-correlation-property="JMSheaderOrPropertyFieldName"]

[receive-selector="selectorString"]

[topic-table-datasource="tableName"]

[auto-topic-subscribe="true | false"]

## Attributes

send-type

Optional. Specifies the messaging style used to send a JMS message. Can be queue or topic.

send-jndi-name

Optional. Specifies the JNDI name of the JMS destination (queue or topic) to which messages will be sent.

receive-type

Optional. Specifies the messaging style used to send a JMS message. Can be queue or topic.

receive-jndi-name

## WebLogic Workshop Annotations Reference

Optional. Specifies the JNDI name of the JMS destination (queue or topic) from which messages will be received.

connection-factory-jndi-name

Optional. Specifies the JNDI name of the connection factory used to obtain JMS connections. If not specified the default WebLogic Workshop connection factory is used (weblogic.jws.jms.QueueConnectionFactory).

send-correlation-property

Optional. Specifies the name of the JMS header or property field to which the WebLogic Workshop conversation ID is written in outgoing messages. If not specified, the default is JmsControl.HEADER\_CORRELATIONID. See remarks below.

receive-correlation-property

Optional. Specifies the name of the JMS header or property field from which the WebLogic Workshop conversation ID is read in incoming messages. If not specified, the default is JmsControl.HEADER\_CORRELATIONID. See remarks below.

receive-selector

Optional. Specifies a selector string used to select messages. This syntax is based on a subset of the SQL92 conditional expression syntax. To learn more about how to specify message selector strings, please consult the Filtering Messages section of Programming WebLogic JMS in the WebLogic Server documentation.

topic-table-datasource

Optional. Specifies the data source for the JMS topic subscription table. The default value is specified in the jws-config.properties file for the server.

auto-topic-subscribe

Optional. Provides automatic subscription for a JMS control when the control is initialized. Initialization takes place on the first invocation of a method on the control.

## Remarks

The following rules apply to this annotation's use:

- Only one @jc:jms annotation may appear within a single Javadoc comment block.
- The @jc:jms annotation may appear in the Javadoc comment on the main interface defined in a JMS control's JCX file.
- You must specify either the send-type and send-jndi-name attributes, or the receive-type and receive-jndi-name attributes, in order to use the JMS control. That is, you must specify that the control either sends a message or receives a message, or both.

The send-correlation-property and receive-correlation-property attributes are set to default values that will provide proper operation in most cases. If the application with which the JMS control is communicating already uses the default property name (CorrelationId) for other purposes, another property name should be

used.

Since many web service instances can use JMS controls that listen on the same queue, the underlying dispatching mechanism needs to correlate each message received by the JMS control to the correct web service instance. The conversation ID is used as the correlation ID.

For this to work the application on the other end (the “foreign” application) has to participate in a simple correlation protocol. The JMS control will place the conversation ID in the JMS header identified by the send-correlation-property attribute on every message the JMS control publishes. The foreign application must place the conversation ID into a JMS header that is identified by the receive-correlation-property attribute in all messages that constitute responses.

A web service may not send messages via a JMS control from within a method with a conversation phase of none. These methods have no conversation ID and therefore cannot participate in the protocol described above.

Note that conversational correlation is not supported for JMS topics. All current conversational instances that subscribe to a topic will receive a callback when a message arrives on the topic.

### Related Topics

#### JMS Control

@jc:jms-headers Annotation

@jc:jms-property Annotation

# @jc:jms-headers Annotation

In a JMS control, set and retrieves values for the JMS message headers.

**Note:** The @jc:jms-headers annotation appears in JMS controls with the .jcx extension. JMS controls with the .ctrl extension, which were created in a previous version of WebLogic Workshop, use the @jws:jms-header annotation. Controls with the .ctrl extension continue to be supported in WebLogic Workshop.

## Syntax

@jc:jms-headers

```
JMSCorrelationID="CorrelationID"  
JMSDeliveryMode="DeliveryMode"  
JMSExpiration="Expiration"  
JMSMessageID="MessageID"  
JMSPriority="Priority"  
JMSRedelivered="Redelivered"  
JMSTimestamp="Timestamp"  
JMSType="Type"
```

## Attributes

JMSCorrelationID

Read/write. You can set this header to the conversation ID of the listening service.

JMSDeliveryMode

Read-only. Contains the delivery mode specified when the message was sent.

JMSExpiration

Read-only. Calculated as the sum of the message-time-to-live value when the message was sent plus the current GMT.

JMSMessageID

Read-only. Contains a unique identifier for the message.

JMSPriority

Read-only. Contains the message's priority.

JMSRedelivered

Read-only. Indicates that the message may have been delivered but not acknowledged in the past.

## JMSTimestamp

Read-only. Contains the time that the message was handed off by the provider to be sent.

## JMSType

Read/write. Can be set to an arbitrary value to distinguish the type of message the sender is sending.

## Remarks

The following rules apply to this annotation's use:

- Only one @jc:jms-headers annotation may appear within a single Javadoc comment block.
- Optionally may appear in front of a method in a JCX file defining a JMS control.
- Optionally may appear in front of a callback in a JCX file defining a JMS control.

On a method of a JMS control, sets these headers on the outgoing message. Only JMSCorrelationID and JMSType can be set; the others are read-only.

On a callback of a JMS control, retrieves these headers from the incoming message and passes them as parameters to the callback.

## Example

The example below demonstrates use of the @jc:jms-headers annotation in the callback that receives a message on the JMS control. The header values are retrieved from the message by the control and passed in to the callback:

```
/**
 * @jc:jms-headers
 *   JMSCorrelationID="{CorrelationID}"
 *   JMSDeliveryMode="{DeliveryMode}"
 *   JMSExpiration="{Expiration}"
 *   JMSMessageID="{MessageID}"
 *   JMSPriority="{Priority}"
 *   JMSRedelivered="{Redelivered}"
 *   JMSTimestamp="{Timestamp}"
 *   JMSType="{Type}"
 */
public void receiveUpdate(
    // Message body (XMLBean type)
    AccountTransaction transaction,
    // Headers
    String CorrelationID,
    String DeliveryMode,
    String Expiration,
    String MessageID,
    String Priority,
    String Redelivered,
    long Timestamp,
    String Type
);
```

Related Topics

JMS Control

Specifying Message Headers and Properties

@jc:jms Annotation

@jc:jms-property Annotation

# @jc:jms-property Annotation

In a JMS control, sets and retrieves properties on the message.

*Note:* The @jc:jms-property annotation appears in JMS controls with the .jcx extension. JMS controls with the .ctrl extension, which were created in a previous version of WebLogic Workshop, use the @jws:jms-property annotation. Controls with the .ctrl extension continue to be supported in WebLogic Workshop.

## Syntax

@jc:jms-property

key="identifierString"

value="propertyValue"

## Attributes

key

Required. A string which identifies this property.

value

Required. The property value.

## Remarks

The @jc:jms-property annotation specifies a custom property to be appended to an outgoing JMS message or read from an incoming message. You can add this annotation to a method that sends a JMS message to set properties on the message. You can also add it to a callback that receives a JMS message to retrieve the message properties. The property values are passed to the callback's parameters.

Properties are name/value pairs and can be of type boolean, byte, short, int, long, float, double, or string. You can add as many properties as you wish; each property is specified in a separate @jc:jms-property tag.

## Examples

The following example demonstrates the use of the @jc:jms-property annotation in a method on a JMS control that sends a message to the message service. Note that the value of the transactionType property is hard-coded as a string, but the value of the accountID property is passed to the method by the client, and that value is substituted to set the property.

```
/**
 * @jc:jms-property key="accountIdentifier" value="{accountID}"
 * @jc:jms-property key="transactionType" value="DEPOSIT"
 */
public void deposit(AccountTransaction transaction, String accountID);
```

The example below demonstrates use of the `@jc:jms-property` annotation in the callback that receives a message on the JMS control. The header values are retrieved from the message by the control and passed in to the callback:

```
/**
 * @jc:jms-property key="accountIdentifier" value="{accountID}"
 * @jc:jms-property key="transactionType" value="{transactionType}"
 */
public void receiveUpdate(
    // Message body (XMLBean type)
    AccountTransaction transaction,
    // Properties
    String transactionType,
    String accountID
);
```

### Related Topics

JMS Control

Specifying Message Headers and Properties

`@jc:jms` Annotation

`@jc:jms-headers` Annotation



## **@jc:location Annotation**

The @jc:location annotation specifies the URL at which a web service control accepts requests for each supported protocol. This annotation appears in a web service control that exposes a web service, so it is essentially identical to the corresponding web service annotation, @jws:location. For more information on using this annotation, see @jws:location.

### **Related Topics**

[@jws:location](#)

[Web Service Control](#)

# @jc:log Annotation

The @jc:log annotation indicates the log4j category to use for the Database control. You can specify logging categories in the workshopLogCfg.xml file which is part of your WebLogic Workshop installation. For more information on configuring logging in WebLogic Workshop, see workshopLogCfg.xml Configuration File.

## Syntax

@jc:log

category="categoryName"

## Attributes

category

Optional. Specifies the name of the category under which logging information for this Database control should be written.

## Remarks

By default, the Database control logs errors under a category corresponding to the name of its implementation class (com.bea.wlw.runtime.core.control.DatabaseControlImpl). If you want to override this category, you can specify a new category in the workshopLogCfg.xml file and set the category attribute of the @jc:log tag to the name of that category.

Related Topics

workshopLogCfg.xml Configuration File

## **@jc:parameter+xml Annotation**

The @jc:parameter+xml annotation specifies characteristics for marshaling data between XML messages and the data provided to the parameters of a web service operation. This annotation appears in a web service control that exposes a web service, so it is essentially identical to the corresponding web service annotation, @jws:parameter+xml. For more information on using this annotation, see @jws:parameter+xml.

### **Related Topics**

@jws:parameter+xml

Web Service Control

## **@jc:protocol Annotation**

The @jc:protocol annotation specifies which protocols and message formats can be accepted by the web service represented by a web service control, and by the operations on that web service. This annotation appears in a web service control and is essentially identical to the corresponding web service annotation, @jws:protocol. For more information on using this annotation, see @jws:protocol.

### **Related Topics**

@jws:protocol

Web Service Control

## @jc:reliable Annotation

The @jc:reliable annotation indicates that a web service exposed by a Web Service control uses reliable messaging, and specifies how long messages must be retained by the server in order to perform detection and removal of duplicates. On a method of the web service, the @jc:reliable tag indicates that reliable messaging is enabled or disabled for that method. The @jc:reliable annotation is essentially identical to the @jws:reliable annotation on a web service. For more information, see @jws:reliable.

**Note:** In WebLogic Workshop 8.1, reliable annotations are not propagated into the WSDL for the JWS file. If you generate a Web Service control from a JWS with reliable methods, you must manually modify the Web Service control to match the JWS.

**Note:** Reliable messaging is not supported with web services and controls that use WS–Security (Web Service Security).

Related Topics

@jws:reliable

Web Service Control

## @jc:return-xml Annotation

The @jc:return-xml annotation defines mappings between the return type of the method marked by this annotation and XML message data. This annotation appears in a web service control that exposes a web service, so it is essentially identical to the corresponding web service annotation, @jws:return-xml. For more information on using this annotation, see @jws:return-xml.

### Related Topics

[@jws:return-xml](#)

[Web Service Control](#)

# @jc:sql Annotation

Specifies the SQL statement and associated attributes that correspond to a method in a Database control.

**Note:** The @jc:sql annotation appears in Database controls with the .jcx extension. Database controls with the .ctrl extension, which were created in a previous version of WebLogic Workshop, use the @jws:sql annotation. Controls with the .ctrl extension continue to be supported in WebLogic Workshop.

## Syntax

@jc:sql

statement="sqlStatement"

[iterator-element-type="javaType"]

[array-max-length=integer OR "all"]

[max-rows="integer"]

[command-type="grid | detail | update | delete | insert | templateRow | insertedRow"]

[rowset-name]

## Attributes

statement

Required. The SQL (Structured Query Language) statement that will be executed when the associated Database control method is invoked. The statement may contain substitutions of the form {varName}, where paramName is a parameter of the Database control method (or a member accessible from the parameter).

If the statement is a single line, it may be formatted with and equals sign and quotes, as follows:

```
/**
 * @jc:sql statement="SELECT * FROM sometable"
 */
public HashMap getEverything();
```

If the statement spans multiple lines, it is delimited with :: (two colons), as follows:

```
/**
 * @jc:sql statement::
 *     SELECT name
 *     FROM employees
 *     WHERE name LIKE {partialName}
 * ::
 */
public String[] partialNameSearch(String partialName);
```

## WebLogic Workshop Annotations Reference

For a discussion of parameter substitution in the statement attribute, see [Parameter Substitution in @jc:sql Statements](#).

### iterator–element–type

Required if Database control method return type is `java.util.Iterator`. Specifies the underlying class of the Iterator that will be returned by the Database control method. The class must meet specific criteria that are described in the [Returning an Object](#) section of [Returning a Single Row from a Database Control Method](#).

### array–max–length

Optional. If the return type of the associated Database control method is an array, specifies the maximum size of the array that will be returned. This attribute can be used to set an upper limit on memory usage by a Database control method.

Default value: 1024

Note that if the associated query returns more rows than specified by `array–max–length`, there is no way to access the excess rows.

The special value `all` causes all rows that satisfy the query to be returned. Note that this can possibly exhaust all available memory if not used carefully.

If you wish to limit memory usage, you should return an Iterator or a `ResultSet`. However, these data types cannot be returned directly to a Page Flow or web service file. To learn about returning Iterators and `ResultSets` from a Database control method, see [Returning Multiple Rows from a Database Control Method](#).

### max–rows

Optional. Sets the maximum number of records to be returned by the query. The `max–rows` attribute limits the size of all types of data sets, including Arrays, `RowSets`, etc. This attribute cannot be set dynamically at runtime. It is a fixed value that can be set only at design time. For detailed information on setting the maximum number of records dynamically at runtime, see the [Help](#) topic [Limiting the Size of Returned Data](#).

### command–type

Optional. Possible values are `grid`, `detail`, `update`, `delete`, `insert`, `templateRow`, `insertedRow`.

The `command–type` attribute is used by `RowSet` controls, which are a special type of Database control. To learn more about `RowSet` controls, see [RowSet Control](#).

### rowset–name

Required. This value must match the name of the top level tag of the XML returned by this control file.

The `rowset–name` attribute is used by `RowSet` controls, which are a special type of Database control. To learn more about `RowSet` controls, see [RowSet Control](#).



## Remarks

The following rules apply to this annotation's use:

- The @jc:sql annotation may only occur on a method declaration in an interface that extends com.bea.control.DatabaseControl.
- The @jc:sql annotation may only occur in a Database control file (JCX file).
- The @jc:sql annotation may only appear once per method.

Related Topics

Database Control

Parameter Substitution in @jc:sql Statements

Returning Multiple Rows from a Database Control Method

# @jc:timer Annotation

Specifies configuration attributes for a Timer control.

**Note:** The @jc:timer annotation appears in Timer controls with the .jcx extension. Timer controls with the .ctrl extension, which were created in a previous version of WebLogic Workshop, use the @jws:timer annotation. Controls with the .ctrl extension continue to be supported in WebLogic Workshop.

## Syntax

@jc:timer

[timeout="timeSpecification"]

[repeats—every="timeSpecification"]

[coalesce—events="true" | "false"]

## Attributes

These attributes determine the default behavior of the Timer control. The Timer control may be configured during its lifetime by calling methods of the TimerControl class. To learn more about the TimerControl class, see TimerControl Class.

timeout

Optional. Specifies the time between a call to the Timer control's start or restart method and the Timer control's first onTimeout callback. When the timer expires, the Timer control's onTimeout callback is called.

The timeout attribute expects a relative time specification; for example, 1 second. Specification of an absolute time as the value of the timeout attribute is not supported. Absolute time may only be specified by calling the TimerControl.setTimeoutAt method.

To learn how to specify values for the timeout attribute, see Specifying Time on a Timer Control.

Default: 0 sec. The timer will fire immediately after being started.

repeats—every

Optional. Specifies the interval between subsequent firings of the Timer control after the first expiration. If a valid interval greater than "0 seconds" is specified, the Timer control will continue firing at that interval until stopped.

The repeats—every attribute expects a relative time specification. For example, 1 second. Specification of an absolute time as the value of the repeats—every attribute is not supported.

To learn how to specify values for the repeats—every attribute, see Specifying Time on a Timer Control.

Default: 0 sec. The timer will fire once (at the time specified by the timeout attribute) but will never fire again.

### coalesce-events

Optional. Specifies whether multiple undelivered firing events of a Timer control are delivered as a single onTimeout or as separate callbacks. At times, a Timer control may be unable to deliver one or more callbacks to its referring service. This may occur because the referring service is busy or because high system load delays delivery. A set of undelivered callbacks may accumulate. If the coalesce-events attribute is true, these accumulated callbacks are collapsed into a single callback when the service becomes available. If coalesce-events is false, the accumulated callbacks are delivered individually.

Default: false.

## Remarks

The following rules apply to this annotation's use:

- The @jc:timer annotation may only occur on an declaration of an object of type com.bea.control.TimerControl or of a type that extends TimerControl.
- The @jc:timer annotation is typically found on declaration that also carry the @common:control annotation.

A Timer control callback will never interrupt the current thread of execution of a web service *operation* (a method marked with the @common:operation annotation). If a Timer control expires during execution of a web service operation, the onTimeout callback will be invoked immediately *after* the current web service operation completes.

### Related Topics

Timer Control

Using WebLogic Built-In Controls

# @jc:ws-security-callback Annotation

The @jc:ws-security-callback annotation determines the WS-Security policy file (WSSE file) to be applied to (1) inbound callback SOAP messages sent from the target web service to the control's callback handler and (2) the outbound SOAP messages returned by the control's callback handler to the target web service.

## Syntax

@jc:ws-security-callback

file="[path\_to\_wsse\_file]"

## Attributes

file

Required. Specifies the location of the WS-Security policy file (WSSE file) to apply.

## Example

```
/**
 * @jc:location http-url="http://localhost:7001/Webservices/security/wsse/callback/target/TargetControlPolicy.wsse"
 * @jc:ws-security-callback file="TargetControlPolicy.wsse"
 */
public interface TargetControl extends com.bea.control.ControlExtension, com.bea.control.Service
```

## Related Topics

[Applying WS-Security Policy Files](#)

[@jc:ws-security-service Annotation](#)

[@jws:ws-security-service Annotation](#)

[@jws:ws-security-callback Annotation](#)

# @jc:ws-security-service Annotation

The @jc:ws-security-service annotation determines the WS-Security policy file (WSSE) to be applied to (1) outgoing SOAP invocations of the target web service's methods and (2) the incoming SOAP messages containing the value returned by the target web service's methods.

## Syntax

@jc:ws-security-service

file="[path\_to\_policy\_file]"

## Attributes

file

Required. Specifies the path to the WS-Security policy file (WSSE file) used by the web service.

## Example

```
/**
 * @jc:location http-url="http://localhost:7001/Webservices/security/wsse/callback/target/TargetControlPolicy.wsse"
 * @jc:ws-security-service file="TargetControlPolicy.wsse"
 */
public interface TargetControl extends com.bea.control.ControlExtension, com.bea.control.Service
```

## Related Topics

[Applying WS-Security Policy Files](#)

[@jc:ws-security-callback Annotation](#)

[@jws:ws-security-service Annotation](#)

[@jws:ws-security-callback Annotation](#)

## @jc:wSDL Annotation

The @jc:wSDL annotation specifies a WSDL file that is implemented by a web service exposed through a web service control. This annotation is essentially identical to the corresponding web service annotation, @jws:wSDL. For more information on using this annotation, see @jws:wSDL.

### Related Topics

@jws:wSDL

Web Service Control

# Editor–Info Annotations

These annotations are used by WebLogic Workshop to track information at design time. They are not interpreted by the server at runtime.

## Topics Included in This Section

@editor–info:code–gen Annotation

Specifies code generation behavior for a Java control.

@editor–info:link Annotation

Specifies the association between a Service control and the web service file (JWS file) from which it was generated.

Related Topics

Annotation Reference

# @editor-info:code-gen Annotation

Specifies code generation behavior for a Java control.

## Syntax

```
@editor-info:code-gen  
  control-interface="true | false"
```

## Attributes

control-interface

Optional. true to have WebLogic Workshop generate and maintain the control's interface; false to take responsibility for creating and maintaining this interface. Default is true.

## Remarks

By default, for each JCS file you create to implement a Java control, WebLogic Workshop creates a corresponding control interface. This interface is updated and maintained as you make changes to your Java control. Under nearly all circumstances, this default behavior will satisfy your control building needs. For those rare occasions when you want to create and maintain this file separately, you can set the control-interface to false.

Related Topics

@editor-info:link

Tutorial: Your First Java Control



# @editor-info:link Annotation

Specifies the association between a Service control and the web service file (JWS file) from which it was generated.

## Syntax

@editor-info:link

autogen="true | false"

autogen-style="java"

source="JwsFileName"

## Attributes

autogen

Required. Specifies whether the Service control is automatically updated when the underlying source for the web service is changed.

autogen-style

Required. Specifies the language in which the Service control is auto-generated. At this time, all autogenerated Service controls created in the IDE are generated in Java.

source

Required. Specifies the JWS file on which this control is based.

## Remarks

When you create a Service control based on a JWS file in your application, by default the Service control is auto-generated from that JWS file, so that when you modify the JWS file, the Service control is also updated. If you want to sever the association between the Service control and the JWS, you can set the autogen attribute of the @editor-info:link annotation to false. Once you modify the Service control and save it separately from the JWS file, you cannot re-create the association between the two again; you would need to delete the Service control and recreate it from the JWS file.

Related Topics

@editor-info:code-gen

# Java Web Service Annotations

The Java Web Service annotations used in web services built with WebLogic Workshop provide information to WebLogic Server about how the web service functions.

For overview information about these annotations, and how they are used within web services built with WebLogic Workshop, see [WebLogic Workshop Annotations Reference](#).

## Topics Included in This Section

`@jws:context` Annotation (Deprecated)

Deprecated annotation; use `@common:context` instead.

`@jws:control` Annotation (Deprecated)

Deprecated annotation; use `@common:control` instead.

`@jws:conversation` Annotation

Specifies the role a method or callback plays in a service's conversations.

`@jws:conversation-lifetime` Annotation

Specifies the maximum age and/or the maximum idle time for a service's conversations.

`@jws:define` Annotation (Deprecated)

Deprecated annotation; use `@common:define` instead.

`@jws:handler` Annotation

Specifies SOAP message handlers for a web service.

`@jws:location` Annotation

Specifies the URL at which a service accepts requests for each supported protocol.

`@jws:message-buffer` Annotation (Deprecated)

Deprecated annotation; use `@common:message-buffer` instead.

`@jws:operation` Annotation (Deprecated)

Deprecated annotation; use `@common:operation` instead.

`@jws:parameter-xml` Annotation

Specifies conversion between XML messages and a Java method's parameters.

## WebLogic Workshop Annotations Reference

### @jws:protocol Annotation

Specifies which protocols and message formats a web service can accept or a Web Service control will send to the service it represents.

### @jws:reliable Annotation

Specifies that operations on the web service should use reliable messaging.

### @jws:return-xml Annotation

Specifies conversion between XML messages and a Java method's return value.

### @jws:schema Annotation (Deprecated)

Deprecated annotation; use @common:schema instead.

### @jws:target-namespace Annotation (Deprecated)

Deprecated annotation; use @common:target-namespace instead.

### @jws:ws-security-service Annotation

Specifies the policy file (WSSE file) used to secure (non-callback) traffic between a web service and its clients.

### @jws:ws-security-callback Annotation

Specifies the policy file (WSSE file) used to secure callback traffic between a web service and its clients.

### @jws:wSDL Annotation

Specifies a WSDL file that is implemented by a web service or represented by a Web Service control.

### @jws:smlns Annotation (Deprecated)

Deprecated annotation; use @common:xmlns instead.

## **@jws:context Annotation (Deprecated)**

The @jws:context annotation has been deprecated as of WebLogic Workshop version 8.1. Please refer to the @common:context annotation instead.

## **@jws:control Annotation (Deprecated)**

The @jws:control annotation has been deprecated as of WebLogic Workshop version 8.1. Please refer to the @common:control annotation instead.

# @jws:conversation Annotation

The @jws:conversation annotation specifies the role that an individual method or callback plays in a service's conversations. Note that these roles are different between methods and callbacks.

**Note:** The Web Service control uses the @jc:conversation annotation instead of the @jws:conversation annotation. The functionality is the same.

## Syntax

@jws:conversation

phase="none" | "start" | "continue" | "finish"

## Attributes

phase

One of four values indicating the method or callback's role in conversations. Note that only continue and finish are available for callbacks, because a callback is necessarily part of a conversation, although it cannot start a conversation.

## Remarks

The @jws:conversation annotation sets the conversation phase attribute for a method or callback to specify its role in the service's conversations.

The following rules apply to this annotation's use:

- Only one @jws:conversation annotation can appear within a single Javadoc comment block.
- Optionally the @jws:conversation annotation may appear in front of an @common:operation method in a JWS file.

Possible attribute values are as follows:

- **start:** May be applied to methods only. Specifies that a call to the method starts a new conversation. Each call will create a new conversation context and an accompanying unique conversation ID, save the service's state, and start its idle and age timer.
- **continue:** May be applied to methods and callbacks. Specifies that a call to this method or callback is part of a conversation in progress. WebLogic Server will attempt to use the incoming conversation ID to correlate each call to an existing conversation. Each call to a **continue** method will save the service's state and reset its idle timer.  
Set the phase attribute to **continue** for any method or callback that is likely to be used for communication with a client *in connection with an ongoing request*. In particular, these are methods that are clearly intended to be called subsequent to the conversation's start and before its finish. These include requests for or responses with additional information, requests for progress or status, and so on.
- **finish:** May be applied to methods and callbacks. Specifies that a call to this method or callback finishes an existing conversation. A call will mark the conversation for destruction by WebLogic

## WebLogic Workshop Annotations Reference

Server. At some point after a finish method returns successfully, all data associated with the conversation's context will be removed.

It is also possible to finish a conversation by calling the JwsContext interface finishConversation method. For more information, see Managing Conversation Lifetime.

- ***none***: May be applied to methods only. Specifies that a call to this method does not participate in any conversation. Methods marked none should not attempt to access conversational state via member variables of the JWS class.

The phase attributes default values are as follows:

- When applied to methods, the phase attribute's default value is ***none***.
- When applied to callbacks, the phase attribute's default value is ***continue***.

Related Topics

Overview: Conversations

JwsContext Interface

# @jws:conversation-lifetime Annotation

Specifies the maximum age and/or the maximum idle time for a service's conversations.

## Syntax

@jws:conversation-lifetime

[max-idle-time="duration"]

[max-age="duration"]

## Attributes

max-idle-time

The amount of time that the conversation may remain idle before it is finished by WebLogic Server. Note that only activity between a web service and its client will reset the idle timer; activity between a web service and controls does not reset the idle timer.

Default: 0 seconds. Conversations with 0-length idle timeout will never timeout due to inactivity.

max-age

The amount of time (since it started) that the conversation may remain active before it is finished by WebLogic Server. Default: 1 day (24 hours).

## Remarks

The following rules apply to this annotation's use:

- Use the conversation-lifetime annotation to set initial values for maximum idle time and maximum age.
- Assign values in the form of expressions of time: 2 minutes, 5 days, and so on.
- You may also set values for this annotation using the conversation-lifetime property in the Properties window.
- This annotation must precede any method or class declarations in the JWS file.
- You may also set these values using the setMaxIdleTime and setMaxAge methods of the JwsContext interface. See JwsContext Interface.
- The idle timer may be reset using the resetIdleTime method of the JwsContext interface.
- When a conversation is terminated due to expiration of the idle time or maximum age, the conversation termination never occurs *during* the execution of a web service method. The conversation will actually terminate *after* completion of the method that is executing when the conversation is terminated.
- When a conversation is terminated, the optional JwsContext onFinish callback is invoked. You may implement a handler for this callback to receive notification of conversation expiration. See JwsContext Interface.

Related Topics



## WebLogic Workshop Annotations Reference

@jws:conversation Annotation

Managing Conversation Lifetime

JwsContext Interface

## **@jws:define Annotation (Deprecated)**

The @jws:define annotation has been deprecated as of WebLogic Workshop version 8.1. Please refer to the @common:define annotation instead.

# @jws:handler Annotation

Specifies SOAP message handlers for a web service.

To learn about SOAP message handlers and how to configure them in WebLogic Workshop web services, see [Specifying SOAP Handlers for a Web Service](#).

## Syntax

@jws:handler

[operation="<handler-spec>"]

[callback="<handler-spec>"]

[file="<handler-config-file-name>"]

## Attributes

operation

Optional. Specifies the handler(s) that will process request and response SOAP messages associated with operations (methods) of a web service. May be a space-separated list of handler class names or the name of a <handler-chain> defined in a handler configuration file specified by the file attribute. Each handler class must implement the javax.xml.rpc.handler.Handler interface.

callback

Optional. Specifies the handler(s) that will process request and response SOAP messages associated with callbacks of a web service. May be a space-separated list of handler class names or the name of a <handler-chain> defined in a handler configuration file specified by the file attribute. Each handler class must implement the javax.xml.rpc.handler.Handler interface.

file

Optional. Specifies a handler configuration file. A handler configuration file is an XML file that defines handlers, handler chains and handler initialization data. See [Example Handler Configuration File](#), below.

## Remarks

The @jws:handler annotation is used to associate SOAP message handlers with a web service. Each SOAP handler is a class that implements the javax.xml.rpc.handler.Handler interface.

The following rules apply to this annotation's use:

- The @jws:handler annotation may only be applied to the main class in a web service JWS file.
- Only one @jws:handler annotation is allowed per JWS file.

## Example Handler Configuration File

A handler configuration file is an XML file that conforms to the WebLogic Workshop handler configuration schema. The example below illustrates a sample handler configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<hc:wlw-handler-config xmlns:hc="http://www.bea.com/2003/03/wlw/handler/config/">
  <hc:handler-chain name="LoggingHandler">
    <hc:handler handler-name="Logger" handler-class="handler.ConsoleLoggingHandler"/>
      <hc:init-param>
        <hc:description>First Param</hc:description>
        <hc:param-name>param1</hc:param-name>
        <hc:param-value>value1</hc:param-value>
      </hc:init-param>
      <hc:init-param>
        <hc:description>Second Param</hc:description>
        <hc:param-name>param2</hc:param-name>
        <hc:param-value>value2</hc:param-value>
      </hc:init-param>
    <hc:handler handler-name="Auditor" handler-class="handler.AuditHandler"/>
  </hc:handler-chain>
</hc:wlw-handler-config>
```

The handler-class attribute of each <handler> element must refer to a handler class that implements the javax.xml.rpc.handler.Handler interface.

Any <init-param> elements that are present for a given <handler> element will be passed to the handler's init() method when the handler is initialized.

### Related Topics

Specifying SOAP Handlers for a Web Service

@jc:handler Annotation

# @jws:location Annotation

Specifies the URL at which a web service accepts requests for each supported protocol.

**Note:** The @jc:location annotation appears in controls with the .jcx extension. Controls with the .ctrl extension, which were created in a previous version of WebLogic Workshop, use the @jws:location annotation. Controls with the .ctrl extension continue to be supported in WebLogic Workshop.

**Note:** The Web Service control uses the @jc:location annotation instead of the @jws:location annotation. The functionality is the same.

## Syntax

@jws:location

[http-url="httpEndPoint"]

[jms-url="jmsEndPoint"]

## Attributes

http-url

Optional on control files; prohibited on JWS files. The URL of the target service's HTTP end point. The value may be an empty string represented by "".

jms-url

Optional on control files; prohibited on JWS files. The URL of the target service's JMS end point.

## Remarks

The following rules apply to this annotation's use:

When optionally applied to an interface in a control file:

- Only one @jws:location annotation may be present in the interface's Javadoc comment block.

Values for this annotation's attributes are typically computed automatically when the web service is compiled or deployed. The values are derived from the deployment environment of the web service.

You may wish to change the attribute values if a web service or Web Service control is moved to a different file system location or host.

Related Topics

@jws:protocol Annotation

Web Service Control

@jws:location Annotation

## **@jws:message–buffer Annotation (Deprecated)**

The @jws:message–buffer annotation has been deprecated as of WebLogic Workshop version 8.1. Please refer to the @common:message–buffer annotation instead.

## **@jws:operation Annotation (Deprecated)**

The @jws:operation annotation has been deprecated as of WebLogic Workshop version 8.1. Please refer to the @common:operation annotation instead.

# @jws:parameter-xml Annotation

Specifies characteristics for marshaling data between XML messages and the data in a Java declaration's parameters.

*Note:* The Web Service control uses the @jc:parameter-xml annotation instead of the @jws:parameter-xml annotation. The functionality is the same.

## Syntax

@jws:parameter-xml

[schema-element="schemaPrefix:schemaType"]

[xquery=""]

[xquery-ref=""]

[xml-map=""]

[include-java-types="javaTypes"]

## Attributes

schema-element

Optional. The name of an XML schema type, such as "xsd:string". Schema types are defined in the file(s) specified by the service's @common:schema tag.

xquery

Optional. Specifies an XQuery expression to be run on the values passed as parameters to this operation.

xquery-ref

Optional. Specifies a file containing an XQuery expression to be run on the values passed as parameters to this operation.

xml-map

Optional. Indicates that parameters in the annotated declaration should be mapped to XML message values as described in an XML map or script.

include-java-types

Optional. One or more Java types that should be used for serializing message data.



## Remarks

Use the `@jws:parameter-xml` annotation to define how the data received or sent via parameters should be treated. Each of the annotation's attributes represents a characteristic of the translation between Java data types and XML values. (The `@jws:return-xml` applies the same functionality to a method or callback's return values.)

## Using the `include-java-types` Attribute

The `include-java-types` attribute lists Java data types that should be used when the Java type sent or received in an XML message may be a subclass of a type in the declaration.

```
public void updateAddress(Address newAddressData)
```

To support addresses in the United States and Great Britain, the `Address` object passed to this method would need to support data for the postal code formats of either nation. To enable this support, and to keep things simple, you might implement two subclasses of the `Address` object—one including a field for `usPostalCode`, the other with a field for `gbPostalCode`. Either of these subclasses—`USAddress` and `GBAddress`—are legal as parameters to the `updateAddress` method.

However, as the `updateAddress` method is declared, WebLogic Server has no way of knowing which subclasses of `Address` the method should accept as parameter types. To respect the exact interface you have defined with this declaration, WLS will serialize all incoming compatible objects as `Address`—the type you have specified in the declaration.

To specify that `USAddress` and `GBAddress` (along with data members unique to those types) should be appropriately passed to your method, specify them with the `include-java-types` attribute, as follows:

```
/**
 * @common:operation
 * @jws:parameter-xml include-java-types="com.myCompany.USAddress
 *   com.myCompany.GBAddress"
 */
public void updateAddress(Address newAddressData)
{...}
```

The `include-java-types` attribute is also useful when a Java Collection class is being converted. Java Collection classes contain Objects, which cannot be successfully converted to or from XML without additional information. The `include-java-types` attribute may contain a space-separated list of Java types that are stored in the Collection. For example, if a method accepts `ArrayList` that may contain Integers and Strings, specify `include-java-types` as `"Integer String"`.

## Specifying an XQuery Expression

Use the `xquery` attribute to specify an XQuery expression for mapping an XML message to the operation's parameters. Use the `xquery-ref` attribute to specify a file that contains an XQuery expression.

For more information about XQuery, see Introduction to XQuery Maps.

## Specifying an XML Map

Use the `xml-map` attribute to specify that the parameters of the declaration should be mapped to XML values using the XML map provided. The `xml-map` attribute may include either the text of an XML map or code to invoke one, or code to invoke a script. Note that although XML maps are still supported in WebLogic Workshop 8.1, they are deprecated in favor of XQuery expressions. For more information on XML maps, see [Why Use XML Maps?](#)

## Using the `schema-element` Attribute

Use the `schema-element` attribute to indicate to that your service supports specific complex schema types. You should only use this attribute when your implementation is aware of the type and handles it appropriately.

Related Topics

[@jws:return-xml Annotation](#)

[How Do XML Maps Work?](#)

# @jws:protocol Annotation

Specifies which protocols and message formats a web service can accept or a Web Service control will send to the service it represents.

*Note:* The Web Service control uses the @jc:protocol annotation instead of the @jws:protocol annotation. The functionality is the same.

## Syntax

@jws:protocol

form-get="true | false"

form-post="true | false"

http-soap="true | false"

http-soap12="true | false"

http-xml="true | false"

jms-soap="true | false"

jms-soap12="true | false"

jms-xml="true | false"

soap-style="document | rpc"

## Attributes

form-get

Specifies that messages encoded as HTTP GET requests are understood.

Default: true.

form-post

Specifies that messages encoded as HTTP POST requests are understood.

Default: true.

http-soap

Specifies that SOAP messages conveyed over HTTP are understood.

Default: true.

`http-soap12`

Specifies that SOAP version 1.2 messages conveyed over HTTP are understood.

Default: true.

`http-xml`

Specifies that XML messages conveyed over HTTP are understood.

***Note:*** cannot be used with conversational methods or callbacks. See below.

Default: false.

`jms-soap`

Specifies that SOAP messages conveyed over JMS are understood.

Default: false.

`jms-soap12`

Specifies that SOAP version 1.2 messages conveyed over JMS are understood.

Default: false.

`jms-xml`

Specifies that XML messages conveyed over JMS are understood.

***Note:*** cannot be used with conversational methods or callbacks. See below.

Default: false.

`soap-style`

Specifies which of the two method call styles specified by SOAP is supported. The document method specifies the style commonly referred to as document literal. The rpc method specifies the style commonly referred to as SOAP RPC.

Default: document.

## Remarks

The following rules apply to this annotation's use:

- A single `@jws:protocol` annotation may appear within a single Javadoc comment block.
- Optionally it may appear in front of a class in a JWS file.
- Optionally it may appear in front of an interface in a control file.

The `@jws:protocol` annotation specifies which network protocols and message formats may be used to communicate with a web service.

### When Applied to a Class in a JWS File

When applied to a class in a JWS file, `@jws:protocol` annotation specifies which protocols and messages formats the web service is able to receive. The web service then listens for requests on the listed protocols and understands messages that arrive in the listed formats.

The default behavior for a JWS file is to enable SOAP with document literal formatting of method calls via HTTP POST and HTTP GET.

### When Applied to an Interface in a Control File

A Web Service control must specify a single protocol binding.

The Web Service control represents a web service, referred to as the *target service*. The target service's WSDL description specifies which protocols and message formats the target service accepts. The Web Service control's interface in its control file must contain a `@jws:protocol` annotation that selects exactly one of the protocol/message format combinations the target service is capable of receiving.

### Raw XML Protocols (`http+xml` and `jms+xml`) Do Not Support Conversations

WebLogic Workshop uses SOAP headers to convey conversation ID and other conversational information. Raw XML messages don't carry SOAP headers and therefore cannot participate in conversations.

### Callback Protocol Is Inferred

The protocol and message format used for callbacks is always the same as the protocol and message format used by the start method that started the current conversation. It is an error to attempt to override the protocol or message format of a callback.

## Examples

The `@jws:location` annotation is used to control protocol bindings for a web service. Consider the following example:

```
/**
 * @jws:protocol http-get="true" http-post="false" http-soap="false"
 */
```

A web service with this `@jws:protocol` annotation:

- Accepts form-based GET messages.
- Does not accept form-based POST messages.
- Does not accept HTTP SOAP messages.

## WebLogic Workshop Annotations Reference

In the example control file below, the `@jws:location` annotation specifies a URL for the Bank.jws web service that this control file represents, and the `@jws:protocol` annotation specifies that this Web Service control will communicate with the BankControl using SOAP over HTTP.

```
import com.bea.control.ServiceControl;
/**
 * @jws:location http-url="http://localhost:7001/webapp/Bank.jws"
 * @jws:protocol http-soap="true"
 */
public interface BankControl extends ServiceControl
{
}
```

### Related Topics

[@jws:location Annotation](#)

[WSDL Files: Web Service Descriptions](#)

[Web Service Control](#)

# @jws:reliable Annotation

The @jws:reliable annotation specifies that a web service should use reliable messaging, and specifies how long messages must be retained by the server in order to perform detection and removal of duplicates. Once you have specified that the web service should use reliable messaging, you can then enable or disable it for a given method.

## Syntax

@jws:reliable

message-time-to-live="duration"

enable="true | false"

## Attributes

message-time-to-live

Required. Set for the web service. Specifies how long messages are maintained on the server, in order to detect duplicate messages. The default is 0.

enable

Optional. Set for a method of the web service. Specifies that reliable messaging should be enabled for messages that are received from the client by this method. The default is **false**.

## Remarks

When you enable reliable messaging for a web service, you specify how long the message should be tracked on the server by setting the message-time-to-live attribute. Once the specified interval has passed, the server no longer tracks the message. The client must specify a message time-to-live that is less than or equal to the web service setting.

The message-time-to-live attribute takes a duration as its value. Assign values in the form of expressions of time: 2 minutes, 5 days, and so on. Note that you cannot enable reliable messaging for an operation until you have set the message-time-to-live for the web service.

Reliable messaging is supported only using SOAP over HTTP. The operation for which reliable messaging is enabled cannot support any other protocols. By default, a web service method supports SOAP over HTTP (http-soap), HTTP Form GET (form-get), and HTTP Form POST (form-post). To use reliable messaging for a given operation, you must modify that operation's @jws:protocol annotation so that it supports only http-soap.

A method that uses reliable messaging must return void.

The following example shows a method that has reliable messaging enabled:

```
/**
```

## WebLogic Workshop Annotations Reference

```
* @common:operation
* @jws:protocol http-soap="true" form-get="false" form-post="false"
* @jws:reliable enable="true"
*/
public void usesReliableMessaging()
{
    ...
}
```

### Limitations in This Release

Reliable messaging has the following limitations in WebLogic Workshop 8.1:

- The reliable messaging mechanism does not operate correctly in a cluster. Reliable messaging is only supported in a single server configuration.
- The reliable tag can only be used on operations. Reliable callbacks are not supported in this release.
- Reliable annotations are not currently propagated into the WSDL for the JWS file. A generated Web Service control for a JWS with reliable methods must be manually configured to match the JWS.
- Reliable messaging is not supported with web services and controls that use WS-Security (Web Service Security).

Related Topics

Protocols and Message Formats



# @jws:return-xml Annotation

Defines mappings between the return type of the method marked by this annotation and XML message data.

*Note:* The Web Service control uses the @jc:return-xml annotation instead of the @jws:return-xml annotation. The functionality is the same.

## Syntax

@jws:return-xml

[schema-element="schemaPrefix:schemaType"]

[xquery=""]

[xquery-ref=""]

[xml-map=""]

[include-java-types="javaTypes"]

## Attributes

schema-element

Optional. The name of an XML schema type, such as "xsd:string". Schema types are defined in the file(s) specified by the service's @common:schema tag.

xquery

Optional. Specifies an XQuery expression to be run on the values passed as parameters to this operation.

xquery-ref

Optional. Specifies a file containing an XQuery expression to be run on the values passed as parameters to this operation.

xml-map

Optional. Indicates that parameters in the annotated declaration should be mapped to XML message values as described in an XML map or script.

include-java-types

Optional. One or more Java types that should be used for serializing message data.

## Remarks

Use the `@jws:return-xml` annotation to define how the data received or sent via the return value should be treated. Each of the annotation's attributes represents a characteristic of the translation between Java data types and XML values. (The `@jws:parameter-xml` annotation applies the same functionality to a method or callback's parameter values.)

## Using the `include-java-types` Attribute

The `include-java-types` attribute lists Java data types that should be used when the Java type sent or received in an XML message may be a subclass of a type in the declaration. Consider the following declaration:

```
public Address getCurrentAddress(String employeeID)
{...}
```

To support addresses in the United States and Great Britain, the `Address` object returned by this method would need to support data for the postal code formats of either nation. To enable this support, and to keep things simple, you might implement two subclasses of the `Address` object—one including a field for `usPostalCode`, the other with a field for `gbPostalCode`. Either of these subclasses, `USAddress` and `GBAddress`, are (from the Java perspective) legal as a type to return from the `updateAddress` method because they are subclasses of `Address`.

However, as the `updateAddress` method is declared, WebLogic Server has no way of knowing which subclasses of `Address` the method should be sent. To respect the exact interface you have defined with this declaration, WLS will serialize all outgoing compatible objects as `Address`—the type you have specified in the declaration.

To specify that `USAddress` and `GBAddress` (along with data members unique to those types) should be appropriately returned by your method, specify them with the `include-java-types` attribute, as follows:

```
/**
 * @common:operation
 * @jws:return-xml include-java-types="com.myCompany.USAddress
 *   com.myCompany.GBAddress"
 */
public Address updateAddress(String employeeID)
{...}
```

The `include-java-types` attribute is also useful when a Java Collection class is being converted. Java Collection classes contain Objects, which cannot be successfully converted to or from XML without additional information. The `include-java-types` attribute may contain a space-separated list of Java types that are stored in the Collection. For example, if a method returns an `ArrayList` that may contain `Integers` and `Strings`, specify `include-java-types` as `"Integer String"`.

## Specifying an XQuery Expression

Use the `xquery` attribute to specify an XQuery expression for mapping an XML message to the operation's return value. Use the `xquery-ref` attribute to specify a file that contains an XQuery expression.

For more information about XQuery, see [Introduction to XQuery Maps](#).

## Specifying an XML Map

Use the `xml-map` attribute to specify that the return value of the declaration should be mapped to XML values using the XML map provided. The `xml-map` attribute may include either the text of an XML map or code to invoke one, or code to invoke a script. Note that although XML maps are still supported in WebLogic Workshop 8.1, they are deprecated in favor of XQuery expressions. For more information on XML maps, see [Why Use XML Maps?](#)

## Using the `schema-element` Attribute

Use the `schema-element` attribute to indicate to that this method or callback supports a specific complex schema type. You should only use this attribute when your implementation is aware of the type you and handles it appropriately.

Related Topics

[@jws:parameter-xml Annotation](#)

[How Do XML Maps Work?](#)

## **@jws:schema Annotation (Deprecated)**

The @jws:schema annotation has been deprecated as of WebLogic Workshop version 8.1. Please refer to the @common:schema annotation instead.

## **@jws:target–namespace Annotation (Deprecated)**

The @jws:target–namespace annotation has been deprecated as of WebLogic Workshop version 8.1. Please refer to the @common:target–namespace annotation instead.

# @jws:ws-security-callback Annotation

The @jws:ws-security-callback annotation determines the WS-Security policy file (WSSE file) to be applied to (1) outgoing callback SOAP messages sent from the web service to its clients and to (2) the incoming SOAP messages returned by the client's callback handler method.

## Syntax

@jws:ws-security-callback

file="[path\_to\_wsse\_file]"

## Attributes

file

Required. Specifies the location of the WS-Security file to apply.

## Example

```
/**
 * @jws:ws-security-callback file="MyCompanySecurityPolicy.wsse"
 */
public class MyCompany implements com.bea.jws.WebService
```

## Related Topics

[@jws:ws-security-service Annotation](#)

[@jc:ws-security-service Annotation](#)

[@jc:ws-security-callback Annotation](#)

[Applying WS-Security Policy Files](#)

# @jws:ws-security-service Annotation

The @jws:ws-security-service annotation determines the WS-Security policy file (WSSE) to be applied to (1) incoming SOAP invocations of the web service's methods and (2) the outgoing SOAP messages containing the value returned by the web service's methods.

## Syntax

@jws:ws-security-service

file="[path\_to\_policy\_file]"

## Attributes

file

Required. Specifies the path to the WS-Security policy file (WSSE file) used by the web service.

## Example

```
/**
 * @jws:ws-security-service file="MyCompanySecurityPolicy.wsse"
 */
public class MyCompany implements com.bea.jws.WebService
```

## Related Topics

[@jws:ws-security-callback Annotation](#)

[@jc:ws-security-service Annotation](#)

[@jc:ws-security-callback Annotation](#)

[Applying WS-Security Policy Files](#)

# @jws:wsdl Annotation

Specifies a WSDL file that is implemented by a web service.

**Note:** The @jc:wsdl annotation appears in Web Service controls with the .jcx extension. Web Service controls with the .ctrl extension, which were created in a previous version of WebLogic Workshop, use the @jws:wsdl annotation. Controls with the .ctrl extension continue to be supported in WebLogic Workshop.

**Note:** The Web Service control uses the @jc:wsdl annotation instead of the @jws:wsdl annotation. The functionality is the same.

## Syntax

@jws:wsdl

file="fileName"

## Attributes

file

Required. Specifies the name of a WSDL file. fileName may begin with the # character, in which case the referenced WSDL file is expected to be found inline in the current file as the value of a @common:define annotation with name and value attributes. This arrangement is illustrated in the following example:

```
import com.bea.control.ServiceControl;
/**
 * @jws:location http-url="creditreport/IRS.jws" jms-url="creditreport/IRS.jws"
 * @jws:wsdl file="#IRSwsdl"
 */
public interface IRSControl extends ServiceControl
{
    ...
}
/**
@common:define name="IRSwsdl" value::
    <?xml version=1.0 encoding=utf-8?>
    <definitions ...>
        ...remainder of WSDL here...
    </definitions>
::
*/
```

## Remarks

The following rules apply to this annotation's use:

- When optionally applied to a class in a JWS file, only one @jws:wsdl annotation may be present within the class' Javadoc comment block.
- When optionally applied to an interface in a control file defining a Service control, only one



## WebLogic Workshop Annotations Reference

@jws:location annotation may be present in the interface's Javadoc comment block.

- A WSDL (Web Services Description Language) file conveys the public contract of a web service. When the @jws:wsdl annotation is applied to a class in a JWS file or an interface in a control file, it indicates that the service in the JWS file or the service represented by the control file implements the public contract expressed in the WSDL file.
- WebLogic Workshop validates the actual interface described by the service or Web Service control. Errors are generated at compile time if the defined interface does not comply with that described in the WSDL.
- A Web Service control generated from a WSDL file is always annotated as implementing the WSDL file from which it was generated.

### Related Topics

Web Service Control

WSDL Files: Web Service Descriptions

@common:define Annotation

## @jws:xmlns Annotation (Deprecated)

The @jws:xmlns annotation has been deprecated as of WebLogic Workshop version 8.1. Please refer to the @common:xmlns annotation instead.

## Page Flow Annotations

The page flow annotations used in web applications define the behavior of the page flow controller class and its methods.

### @jpf:action Annotation

Designates an action method. Optionally, this annotation may set login and J2EE role requirements. Without a @jpf:action annotation, an action method will not be recognized by the page flow runtime as an action method. This annotation may be used at the method-level only.

### @jpf:catch Annotation

Provides information used by the page flow runtime to catch exceptions and possibly route them to error handlers. This annotation may be used on action methods and the page flow class, and in the global page flow that resides in WEB-INF/src/global/Global.app.

### @jpf:controller Annotation

Indicates whether the page flow is nested; whether users must be logged in; whether logged-in users must be associated with existing J2EE roles; and whether a Struts XML configuration file should be merged into the page flow. This annotation may be used at the page flow class level only.

### @jpf:exception-handler Annotation

Designates an error handler method. Without a @jpf:exception-handler annotation, the page flow runtime will not recognize an error handler method.

### @jpf:forward Annotation

Describes a location to which the page flow runtime should go to next. The location may be a JSP, a page flow, or an action method. This annotation may be used on action methods and the page flow class, and in the global page flow that resides in WEB-INF/src/global/Global.app.

### @jpf:message-resources Annotation

Specifies a message bundle, allowing you to implement validation in the page flow's form beans. This annotation may be used at the page flow class level only.

### @jpf:validation-error-forward Annotation

Indicates which page should be loaded, or which action should be run, if a form validation error occurs as a result of running the annotated action. This annotation may be used at the method level only.

### @jpf:view-properties Annotation

### @jws:xmlns Annotation (Deprecated)

## WebLogic Workshop Annotations Reference

Describes layout information for your page flow's graphical Flow View. Do not edit this data.

### @common:control Annotation

Indicates that a member variable in a page flow references a WebLogic Workshop control, such as a control or a database control.

### Related Topics

[Guide to Building Page Flows](#)

# @jpf:action Annotation

You can use the @jpf:action annotation to designate an action method in a page flow. Action methods in a page flow perform control logic, such as forwarding the user to a new JSP or another page flow. Optionally, you can cause an action to read and update from a FormData (or ActionForm) member variable; set login requirements for users; or indicate that the action method will not update any member data in the page flow.

For information about the method signatures of an action method, see the @jpf:forward Annotation. That topic also describes the run-time behavior of overloaded actions, in the Remarks section.

## Syntax

@jpf:action

[ form = "<form name>" ]

[ login-required = "{ true | false }" ]

[ read-only = "{ true | false }" ]

[ roles-allowed = "<J2EE role name> [ , <J2EE role name> ] " ]

## Attributes

form

Optional. A Form Bean instance that will be passed to the method. The Form Bean instance that is passed to the , allowing the action to read and update from a FormData (or ActionForm) member variable. For more information, see the Remarks section.

login-required

Optional. A boolean that indicates whether the user must be logged-in to use this action method. If set to login-required="true" the action method can only be run if the user is logged in. That is, the page flow runtime checks to see if request.getUserPrincipal() == null. If it is, then the exception com.bea.wlw.netui.pageflow.NotLoggedInException is thrown and the code in the action method is not executed. You can catch the exception in the page flow or the web project's Global.app, and from there attempt to log the user in.

When you create a new "Web Project" project with WebLogic Workshop, a default Global.app file is created for you in the project's /WEB-INF/src/global directory. Global.app allows you to define actions that can be invoked by any other page flow in a web application. In Global.app, you can catch and handle exceptions that were not caught in your page flow. For more information, see Handling Exceptions in Page Flows.

read-only

Optional. The default is read-only="false". Use this attribute to indicate your intention that this action *will not* update any member data in the page flow. In a WebLogic cluster environment, this designation causes WebLogic Workshop to skip any attempted failover of the pageflow after the action is run. This option may

allow you to increase the performance of the page flow in cluster environments, by making it unnecessary for WebLogic Workshop to communicate portions of this page flow's state data across the nodes in the cluster that pertain to the read-only actions.

roles-allowed

Optional. Specifies the name of one or more security roles that are defined in the web project's /WEB-INF/web.xml file. If more than one security role name is specified, use a comma to separate each name. For more information on security roles, see the Role-Based Security.

If roles-allowed is defined, the use of this action method will be limited to logged-in users who are associated with at least one of the specified roles specified at this method-level, or one of the security roles specified on the class-level annotation @jpf:controller (if used). That is, security roles that are specified at the method level with @jpf:action **add** to any allowable roles defined at the class level with @jpf:controller. If the user is not logged-in, a com.bea.wlw.netui.pageflow.NotLoggedInException will be thrown. If the user is not in an appropriate role, a com.bea.wlw.netui.pageflow.UnfulfilledRolesException will be thrown. For more information, see Handling Exceptions in Page Flows.

validation-error-page (Deprecated)

This attribute on @jpf:action has been deprecated, as of WebLogic Workshop 8.1 Service Pack 2. Instead you can use the @jpf:validation-error-forward Annotation.

## Remarks

The following rules apply to this annotation's use:

- Without an @jpf:action annotation, a method will not be recognized by the page flow runtime as an action method.
- You cannot use the jpf:action annotation at the page flow class level.
- When you drag an action icon onto the palette in a page flow's Flow View, the @jpf:action annotation is automatically added for you in the \*.jpf source code. Initially this looks like the following code fragment, where we provided the method name "shopping" for the action:

```
/**
 * @jpf:action
 */
public Forward shopping()
{
    return new Forward( "success" );
}
```

In Flow View, you can then draw a forward arrow to a destination, such as a JSP or a page flow. This step will add a @jpf:forward annotation in your \*.jpf file. You can also change the Forward object's return name from "success" to another name.

However, if you do, remember to also change the name attribute on the corresponding @jpf:forward annotation for the action method. For more information, see @jpf:forward Annotation.

- You can cause an action to read and update from a FormData (or ActionForm) member variable, by using the @jpf:action form="<form name>" annotation. For example, in the page flow class:

## WebLogic Workshop Annotations Reference

```
/**
 * @jpf:action form="_pageFlowForm"
 */
protected Forward doit( MyForm myForm )
{
    // _pageFlowForm is a member variable
    // Note that myForm == _pageFlowForm

    String name = _pageFlowForm.getFullName();
    ...
}
```

For a related advanced discussion about the behavior seen when you forward directly from one action to another, and the effects of form scoping, see the Remarks section of the topic `@jpf:forward` Annotation and the help topic Page Flow Scopings.

- If you enable `login-required="true"` for an action method, be sure to handle the exception that can occur if a user who is not logged-in raises the action on a JSP page in the page flow. You can add the exception code in the page flow class. For example:

```
import javax.security.auth.login.FailedLoginException;

...

/**
 * @jpf:action
 * @jpf:forward name="success" path="LoginSuccess.jsp"
 * @jpf:catch type="FailedLoginException" method="failedLogin"
 */
protected Forward loginSubmit( LoginForm loginForm )
    throws Exception
{
    login( loginForm.getUsername(), loginForm.getPassword() );
    userName = loginForm.getUsername();
    return new Forward( "success", loginForm );
}

/**
 * @jpf:exception-handler
 * @jpf:forward name="loginPage" path="Login.jsp"
 */
protected Forward failedLogin( FailedLoginException ex, String actionName,
                               String message, FormData form )
{
    return new Forward( "loginPage" );
}
```

You can also use the `Global.app` file, which the Page Flow Wizard provides in your web project's `WEB-INF/src/global` folder. In your page flow, you may notice the line:

```
// protected global.Global globalApp;
```

It is **not** required that you uncomment this line in order to have access to methods in `Global.app`. However, if you do uncomment it, this `globalApp` declaration will give your page flow easy access to public methods and variables in `Global.app`.

In the `Global.app` file, check that you have code such as:

## WebLogic Workshop Annotations Reference

```
/**
 * @jpf:catch type="Exception" method="handleException"
 */
public class Global extends GlobalApp
{
    /**
     * General handler for uncaught exceptions.
     *
     * @jpf:exception-handler
     * @jpf:forward name="errorPage" path="/error.jsp"
     */
    protected Forward handleException( Exception ex, String actionName,
    String message, FormData form )
    {
        System.err.print( "[" + getRequest().getContextPath() + "] " );
        System.err.println( "Unhandled exception caught in Global.app:" );
        ex.printStackTrace();
        return new Forward( "errorPage" );
    }
    ...
}
```

- The roles-allowed attribute lets you limit access to a page flow action method to users who are associated with the J2EE role. This, in turn, limits access from the current context to the web resource that would be loaded as a result of the action.

Note that currently using the following IDE menu option defines a new security role in the *application's* /META-INF/application.xml file:

Tools > Security > Create New Security Role...

However, /META-INF/application.xml is not the XML file that is used for the page flow roles-allowed checking. Enter any J2EE roles in the web project's web.xml file, and associate usernames with the role name in the web project's weblogic.xml file.

### Related Topics

@jpf:catch Annotation

@jpf:controller Annotation

@jpf:exception-handler Annotation

@jpf:forward Annotation

@jpf:message-resources Annotation

@jpf:validation-error-forward Annotation

### Form Bean Scopings

@jpf:action Annotation

## **Samples**

Data Flow Sample



# @jpf:catch Annotation

You can use the @jpf:catch annotation to catch exceptions that occur in the page flow and possibly route them to error handlers.

For more information about using this annotation, see [Handling Exceptions in Page Flows](#).

## Syntax

@jpf:catch

type = "<exception type>"

{ path = "<path>" | method = "<method name>" }

[ message = "<message String>" ]

[ message-key = "<key value>" ]

## Attributes

type

Required. Specifies the fully qualified Java type of the exceptions that should be caught.

path

Either this attribute or the method attribute is required. Specifies the resource to which the runtime should forward when it catches an exception of the type specified in the type attribute. The path can map to one of the following:

- A JSP or another page flow in this web application.
- An action, such as /help/helpPage.do.

method

Either this attribute or the path attribute is required. Specifies the name of an exception handler method that has been designated with a @jpf:exception-handler annotation. That method will be invoked when the exception of the type specified in the type attribute is caught.

message

Optional. Specify a string that provides useful information about the exception. This message will be passed to the exception-handler method, if the method attribute is used.

message-key

Optional. Specify a key that is used to select a message from a message map. Typically this key is used to enable internationalization of messages, which are passed to the exception-handler method, and can also be

displayed using the `<netui:errors/>` tag. The key refers to a message resource that is defined by the `@jpf:message-resources` Annotation.

## Remarks

The following rules apply to this annotation's use:

- The `@jpf:catch` annotation may be used on action methods and the page flow class, and in the `/WEB-INF/src/global/Global.app` file.
- If the type specified is a base type (for example, `Exception` is a base type of `NullPointerException`), this annotation will apply to any subtype that is thrown, unless there is a `@jpf:catch` that applies more specifically to the subtype.

For example, given the following annotations, an `IllegalStateException` will be routed to "error1.jsp", while a `NullPointerException` will be routed to "error2.jsp". (Both `IllegalStateException` and `NullPointerException` are subtypes of `Exception`).

```
* @jpf:catch type="Exception" path="error1.jsp"
* @jpf:catch type="NullPointerException" path="error2.jsp"
```

- The path value can be relative to the page flow within the local file hierarchy, or relative to the root of the web application.
- In a page flow class, you can use the `@jpf:catch` annotation with, or without, the `@jpf:exception-handler` annotation. If you use `@jpf:catch` without `@jpf:exception-handler` in the page flow, you can only catch errors and take action such as forwarding to a generic `errors.jsp` page. For example:

```
/**
 * @jpf:action
 * @jpf:forward name="readyForNextCandidate" path="name.jsp"
 * @jpf:catch type="com.acme.WorkflowException" path="errors.jsp"
 */
public Forward confirmationPage_hire(HireForm form)
throws Exception
{
    hiringService.hire(firstName, lastName, title, startDate);
    .
    .
    .
}
```

- Note that `@jpf:catch` annotations **cannot** be used to catch exceptions on exception-handler methods. The following code is invalid and results in a compilation error:

```
/**
 * Invalid code:
 * @jpf:exception-handler
 * @jpf:catch type="Exception" path="foo.jsp"
 */
public Forward bad( Exception e, String msg, String msgKey, ActionForm form )
```

For information about using the `@jpf:catch` and `@jpf:exception-handler` annotations together, to both catch and handle exceptions, see `@jpf:exception-handler` annotation. Also see [Handling Exceptions in Page Flows](#).

## WebLogic Workshop Annotations Reference

### Related Topics

[@jpf:exception-handler Annotation](#)

[Handling Exceptions in Page Flows](#)

[@jpf:action Annotation](#)

[@jpf:controller Annotation](#)

[@jpf:forward Annotation](#)

[@jpf:message-resources Annotation](#)

[@jpf:validation-error-forward Annotation](#)

[Getting Started with Page Flows](#)

# @jpf:controller Annotation

You can use the @jpf:controller annotation at the class-level only to designate the following characteristics of the page flow:

- Is this page flow "nested," meaning, can control return to the calling page flow after this page flow completes its work?
- Is the use of this page flow limited to logged-in users?
- Is the use of this page flow limited to logged-in users who are associated with a J2EE role, as defined in the application's web.xml file?
- Will a Struts configuration file be merged with the page flow's configuration file?

**Note:** At the class level, you have the option of also using a @jpf:catch annotation, a global @jpf:forward annotation, or both. These annotations may be used at the class level, with or without a @jpf:controller annotation.

## Syntax

@jpf:controller

[ nested = "true | false" ]

[ login-required = "true | false" ]

[ roles-allowed = "<J2EE-role-name> [ , <J2EE-role-name> ] " ]

[ struts-merge = "<Struts-file>.xml" ]

## Attributes

nested

Optional. A boolean that indicates whether this is a nested page flow that can be called from another page flow. If the @jpf:controller annotation is used, but the nested attribute is not, the default is nested="false".

login-required

Optional. A boolean that indicates whether the user must be logged-in to use this page flow. If the @jpf:controller annotation is used, but both the login-required and roles-allowed attributes are not used, the default is login-required="false". However, if only the roles-allowed attribute is used, then login-required="true" is implied.

The JPF compiler will issue an error if you use both the roles-allowed attribute and the login-required attribute.

roles-allowed

Optional. Specifies the name of one or more J2EE roles that are defined in the web project's /WEB-INF/web.xml file. If more than one J2EE role name is specified, use a comma to separate each name.

If defined, the use of all action methods in this page flow is limited to users who are associated with the specified role(s). Note that users who are granted the roles specified may access all methods in the page flow, regardless of any further role restrictions applied to individual methods by the @jpf:action annotation.

You can, optionally, also use the @jpf:action annotation to **add** to the authorized role(s) that are declared at the class-level by @jpf:controller. The method-level @jpf:action annotation also has a roles-allowed attribute. For more information, see @jpf:action annotation.

### struts-merge

Optional. Specifies a standard Struts configuration XML file that you want the WebLogic Workshop compiler to merge into the generated configuration file for the page flow:

/WEB-INF/jpf-struts-<pageflow-name>.xml. The Struts XML file, by convention, should reside in the web project's /WEB-INF directory. After the compilation step that creates the merged /WEB-INF/jpf-struts-<pageflow-name>.xml, functionality that was enabled by the Struts configuration is available to the page flow controller.

The purpose of the "Struts Merge" feature is to enable you to override page flow defaults, or to specify settings for the page flow that are not provided by page flow annotations or their attributes. The Struts merge files should typically be small and only modify the page flow and its actions and form beans. You can also add Struts tags that are not supported by page flows, such as the <plug-in> tag. While you could, for example, add action mappings in the Struts merge file, BEA does not recommend this practice. Struts action mappings should be declared in the page flow's .jpf file with @jpf annotations, and then (if necessary) modified with the Struts merge file. For details, see Merging Struts Artifacts Into Page Flows.

## Remarks

The following rules apply to this annotation's use:

- The @jpf:controller annotation is optional and can be used only to qualify the page flow class. This annotation cannot be used on a method within the page flow class.
- A nested page flow is useful when you want to perform some processing and then return (with or without the results) to the calling page flow. For example, a vacation scheduling page flow could call a nested page flow that lists the company holidays. After the companyHolidays page flow is used, a method that exists in the nested page flow could return the user to the calling vacationScheduling page flow.
- The roles-allowed attribute on this @jpf:controller annotation lets you limit access to this page flow's action methods to users who are associated with the security role. This, in turn, limits access from the current context to the web resource that would be loaded as a result of the action. Again, you can add to the authorized role(s) declared at the class-level with @jpf:controller by using the roles-allowed attribute on a method-level @jpf:action annotation.
- Note that the following IDE menu option defines a new security role in the *application's* /META-INF/application.xml file:

Tools > Security > Create New Security Role...

However, /META-INF/application.xml is not the XML file that is used for the page flow scoped roles-allowed checking. Enter any security roles in the web project's web.xml file, and associate principals with the role name in the web project's weblogic.xml file. For more information, see the Role-Based Security.

## WebLogic Workshop Annotations Reference

### Related Topics

[@jpf:action Annotation](#)

[@jpf:catch Annotation](#)

[@jpf:forward Annotation](#)

[@jpf:exception-handler Annotation](#)

[@jpf:message-resources Annotation](#)

[@jpf:validation-error-forward Annotation](#)

[Getting Started with Page Flows](#)

# @jpf:exception-handler Annotation

You can use the @jpf:exception-handler annotation to mark an error handler method that is specified by a @jpf:catch annotation in a page flow. Without a @jpf:exception-handler annotation, the page flow runtime will not route exceptions to an error handler method.

For more information about using this annotation, see Handling Exceptions in Page Flows.

## Syntax

@jpf:exception-handler

[ read-only = "{ true | false }" ]

## Attributes

read-only

Optional. The default is read-only="false". Use this attribute to indicate your intention that this method *will not* update any member data in the page flow. In a WebLogic cluster environment, this designation causes WebLogic Workshop to skip any attempted failover of the pageflow after the action is run. This option may allow you to increase the performance of the page flow in cluster environments, by making it unnecessary for WebLogic Workshop to communicate portions of this page flow's state data across the nodes in the cluster that pertain to the read-only actions.

## Remarks

The following rules apply to this annotation's use:

- The @jpf:exception-handler annotation must be used with a @jpf:catch annotation. For example:

```
package login2;

import com.bea.wlw.netui.pageflow.PageFlowController;
import com.bea.wlw.netui.pageflow.Forward;
import com.bea.wlw.netui.pageflow.FormData;
import javax.servlet.http.HttpServletRequest;
import javax.security.auth.login.FailedLoginException;

/**
 * @jpf:controller nested="true"
 */
    public class Login extends PageFlowController
    {
        public String userName;
    }
/**
 * @jpf:action
 * @jpf:forward name="success" path="LoginSuccess.jsp"
 * @jpf:catch type="FailedLoginException" method="failedLogin"
 */
```

## WebLogic Workshop Annotations Reference

```
protected Forward loginSubmit( LoginForm loginForm )
    throws Exception
{
    login( loginForm.getUsername(), loginForm.getPassword() );
    userName = loginForm.getUsername();
    return new Forward( "success", loginForm );
}

/**
 * @jpf:exception-handler
 * @jpf:forward name="loginPage" path="Login.jsp"
 */
protected Forward failedLogin( FailedLoginException ex, String actionName,
                               String message, FormData form )
{
    LoginForm loginForm = ( LoginForm ) form;
    loginForm.setMessage( "Login incorrect. Do you need to create an account?" );
    return new Forward( "loginPage" );
}

.
.
.
```

- Note that @jpf:catch annotations cannot be used to catch exceptions on exception-handler methods. The following code is invalid and results in a compilation error:

```
/**
 * Invalid code:
 * @jpf:exception-handler
 * @jpf:catch type="Exception" path="foo.jsp"
 */
public Forward bad( Exception e, String msg, String msgKey, ActionForm form )
```

### Related Topics

[@jpf:catch Annotation](#)

[Handling Exceptions in Page Flows](#)

[@jpf:action Annotation](#)

[@jpf:controller Annotation](#)

[@jpf:forward Annotation](#)

[@jpf:message-resources Annotation](#)

[@jpf:validation-error-forward Annotation](#)

[Getting Started with Page Flows](#)



# @jpf:forward Annotation

You can use the @jpf:forward annotation to describe the location the page flow runtime should go to next. The location may be a JSP, a page flow, a specific page flow action, or an external URL. This annotation may be used on action methods and the page flow class, and in the /WEB-INF/src/global/Global.app. Using the @jpf:forward at the class level results in a global forward that can be used by any action method in the page flow class.

The Remarks section includes a discussion about what happens if you use overloaded actions. Also, the Remarks section contains a description about the behavior seen when you forward directly from one action to another, and the effects of form scoping.

## Syntax

@jpf:forward

name = "<forward name>"

{ path = "<path>" | return-action = "<action name>" | return-to = { "currentPage" | "previousPage" | "previousAction" } }

{ return-form = "<form name>" | return-form-type = "<form type>" }

[ redirect = { "true" | "false" } ]

## Attributes

name

Required. Specifies the name of the forward, which causes the navigation to occur. Note that @jpf:forward names on exception-handlers can conflict with @jpf:forward names on actions and at the class level. The @jpf:forward names that you specify must be unique in the page flow.

***You must specify only one of the following: the path, return-to, or the return-action attributes.***

path

The path attribute is a string that maps to one of the following entities:

- A JSP or another page flow in this web application
- An action within a page flow in this web application
- A resource somewhere on the web, in this web application or a different web application.

If the path begins with a protocol such as "http:" the page flow runtime will look outside of this web application for the resource, and it will automatically cause a redirect (rather than a server forward) to the resource. If the path begins with a forward slash, "/", the runtime will start at the web application's root directory to locate the resource. If the path omits the forward slash, "/", the reference is relative to the page flow's directory.

### return-to

The return-to attribute always applies to the current page flow, whether it is a nested page flow or the main page flow. To return to an action in the nesting page flow use the return-action attribute.

The value for the return-to attribute must be a keyword, either "currentPage", or "previousPage", or "previousAction".

**Note:** To clarify the purpose of the return-to attribute, the keywords "page" and "action" were deprecated as of WebLogic Workshop 8.1 Service Pack 2. Instead of "page", the equivalent function is "previousPage". Instead of "action", the equivalent function is "previousAction".

If the value is "currentPage", the same JSP page is rendered again by the server, along with any updated data that occurred as a result of executing the annotated action method.

If the value is "previousPage", the page that was shown before the current page is rendered.

If the value is "previousAction", the previous action in the current page flow is run.

### return-action

The return-action attribute, which is only valid in a *nested* page flow, causes control to return to the calling (or "nesting") page flow (leaving the current page flow), and then causes the specified action to be raised on the calling page flow.

#### return-form

Optional. The return-form attribute, which is only valid when used with the return-action attribute, causes the given page flow member variable to be attached to the returned Forward automatically.

#### return-form-type

Optional. The return-form-type attribute, which is only valid when used with the return-action attribute, is used in conjunction with the Forward that is returned. Forward takes the actual FormData instance as the second argument to its constructor. This attribute declares the type of the form bean that will be returned to the calling page flow.

**Note:** If multiple actions with the same name (such as "success") exist in the calling page flow, either the return-form-type value or the return-form member variable type will be used to determine the appropriate action.

### redirect

Optional. A boolean, true or false. Default is false. When set to true, navigation causes a browser redirect to the specified destination. Redirecting is useful when you want to clear out any data that was attached to a request, or when it is important that the user's URL bar reflect the actual page that is displayed (instead of the action name). For details, see the Remarks section.

## Action Method Signatures

An action method has several possible signatures. One signature takes no parameters. For example:

```
public Forward shopping()
```

The second type of signature uses a form. For example:

```
public Forward shopping(CheckoutForm form)
```

The form bean is a class that extends `com.bea.wlw.netui.pageflow.FormData`. You can define the form bean as an inner class within the page flow class, or separately in the web project.

If you are using the `<netui-data:declarePageInput>` tag in your JSP, you may use a third type of signature, where you return a new Forward that contains three parameters. For example:

```
/**
 * @jpf:action
 * @jpf:forward name="success" path="next.jsp"
 */
public Forward next()
    throws SQLException
{
    Item[] items = itemsDB.getAllItems();
    return new Forward("success", "items", items);
}
```

In the new Forward, the parameters in `new Forward("success", "items", items)` for this example are as follows:

- "success" is the Forward name, and its value matches the name attribute's value on the `@jpf:forward` annotation. This match causes navigation to the `next.jsp` page when the action is run.
- "items" is the name of the page input, as defined in a JSP that is part of this page flow.
- items is the name of the actual object from which data will be obtained. In this case, items is a table referenced by a database control called by the page flow. For more information, see the `<netui-data:declarePageInput>` Tag Sample.

If you need to add more than one page input, you can first create the Forward object and then call its `addPageInput` method. For example:

```
/**
 * @jpf:action
 * @jpf:forward name="success" path="next.jsp"
 */
public Forward next()
    throws SQLException
{
    Forward fwd = new Forward("success");
    fwd.addPageInput("products", productsDB.getAllProducts());
    fwd.addPageInput("items", itemsDB.getAllItems());
    return fwd;
}
```

## Remarks

The following rules apply to this annotation's use:

- Optionally, you can add a `@jpf:catch` annotation for exception handling.
- The forward's name and its capitalization ***must match exactly*** in the `@jpf:forward` annotation and in the return argument of a Forward object. For example:

```
/**
 * @jpf:action
 * @jpf:forward name="goShopping" path="cart.jsp"
 */
public Forward shopping()
{ return new Forward( "goShopping" );
```

Also note that `@jpf:forward` names on exception-handlers can conflict with `@jpf:forward` names on actions and at the class level. The `@jpf:forward` names that you specify must be unique in the page flow.

- You could pass a `FormData` instance as the second argument to the Forward constructor, which has to take a `@jpf:forward` **name** as its first argument. For example:

```
/**
 * @jpf:action
 * @jpf:forward name="success" path="DonePage.jsp"
 */
protected Forward PrefsPageSubmit( PrefsForm form )
{
    userInfo.setFavColor( form.getFavColor() );
    DoneForm doneForm = new DoneForm();
    doneForm.setUsername( userInfo.getUsername() );
    doneForm.setPassword( userInfo.getPassword() );
    doneForm.setFavColor( userInfo.getFavColor() );
    return new Forward( "success", doneForm );
}
```

- You can use the `@jpf:forward` as a class-level annotation, resulting in a global forward for the page flow. A global forward is useful for pages that are often forwarded to, such as error pages. When you use a global forward, you do not have to define the forward on every action method. For example:

```
/**
 * @jpf:forward name="errorPage" path="error.jsp"
 */
public class WelcomeFlow extends PageFlowController
{
    ...

    /**
     * @jpf:action
     */
    protected Forward loginSubmit( LoginForm loginForm )
    {
        if ( userManagerBusinessControl.isOutOfService() )
        {
            return new Forward( "errorPage" );
        }
        else
        {
            ...
        }
    }
}
```

- ```

    }
  }
}

```
- If your page flow uses overloaded actions, you must be aware of the run-time behavior to avoid unintended results. Here are the rules when an action forwards to an overloaded action:
    - ◆ The choice of overload is determined strictly by the form that was passed in the Forward object. If there was no form passed in the Forward, the action with no form is used.
    - ◆ If you do not put a form on a Forward object, then there must be an action that takes no form; otherwise, it will fail.
    - ◆ If a nested page flow returns, the choice of overload is determined strictly by the return-form or return-form-type attributes on the returning @jpf:forward in the nested page flow. If neither of these attributes are present, the choice of overload is determined strictly by the form that was passed in the Forward.
    - ◆ If a page raises an action, the overload without a form will be chosen first. If all overloads take a form, then it will choose the one whose full classname comes first alphabetically.
  - By default, an instance of a Form Bean passed to an action method is scoped to the request. You can alternatively scope a Form Bean instance to the life of the Controller file. For information about pageflow-scoped forms, see the Remarks section of the @jpf:action Annotation topic and the help topic Form Bean Scopings. If you merge in Struts actions, you can scope a Form Bean instance to the session object, as explained in Merging Struts Artifacts Into Page Flows.

The following list describes advanced behavior that is important to know only for cases when you are forwarding between two actions. This presents scenarios when action A forwards to action B:

- ◆ If A is pageflow-scoped and B is not, B gets a new instance of the Form Bean.
- ◆ If A is pageflow-scoped and B is not, **and** if A explicitly passes its Form Bean on the Forward to B, B will use A's instance of the Form Bean (the member variable).
- ◆ If A is not pageflow-scoped and B is pageflow-scoped, B will not use the Form Bean instance that A created.
- ◆ If A is not pageflow-scoped and B is pageflow-scoped, **and** if A explicitly passes its Form Bean instance on the Forward to B, B will not only use A's Form Bean instance, but the member variable will be set to the instance that A passed.
- ◆ If A is pageflow-scoped and B is pageflow-scoped, then B will use A's Form Bean instance (the member variable).
- When an action method causes navigation to a page, it forwards the request to the URL for that page. Any attributes that were attached to the original request are available for the next page to use. This means that the user can send a request for an action, such as "GoToLogin.do". The page flow can attach data to the request and forward to a page, such as "login.jsp", which is sent back to the browser. The user's URL address bar might read "http://server/welcomeFlow/GoToLogin.do". However, the page displayed is actually "http://server/welcomeFlow/login.jsp".

An alternative method of navigation is by using the redirect attribute in the @jpf:forward annotation. In this case, the server can respond to the browser and tell it to navigate somewhere else. If the user requests ("GoToLogin.do"), the page flow could redirect to "login.jsp", which would cause the browser to navigate directly to "http://server/welcomeFlow/login.jsp". The latter URL is shown in the user's URL bar. Thus the redirect attribute is useful for two scenarios:

- ◆ When you want to clear out any data that was attached to a request.
- ◆ When it is important that the user's URL bar reflect the actual page that is displayed.

## WebLogic Workshop Annotations Reference

### Related Topics

[@jpf:action Annotation](#)

[@jpf:catch Annotation](#)

[@jpf:controller Annotation](#)

[@jpf:exception-handler Annotation](#)

[@jpf:message-resources Annotation](#)

[@jpf:validation-error-forward Annotation](#)

[Form Bean Scopings](#)

[Data Flow Sample](#)

# @jpf:message-resources Annotation

You can use the @jpf:message-resources annotation at the page flow class level to specify a message bundle. Using message resources allows you to internationalize your web applications by *not hard-coding* labels in your JSP pages. For example, an ErrorMessage.properties resource file could contain messages to display when form validation errors occur.

## Syntax

```
@jpf:message-resources
```

```
resources = "<property-resources>"
```

```
[ key = "<bundle-key-name>" ]
```

## Attributes

resources

Required. A resource that contains message properties, such as an Errors.properties file that you create separately and place in a subfolder under the page flow's /WEB-INF/classes folder. For example:

```
@jpf:message-resources resources="validation.ErrorMessage"
```

The page flow runtime will look for the messages in a subfolder of the web project's /WEB-INF folder, either:

- /WEB-INF/classes/validation/ErrorMessage.properties
- Or in a JAR file in /WEB-INF/lib

key

Optional. You can specify the name of the ServletContext attribute in which to store the Bundle read from the <property-resources>. In most cases you can omit this attribute and accept the default of org.apache.struts.Globals.MESSAGE\_KEY, which works with the <netui:error> and <netui:errors> JSP tags provided by WebLogic Workshop.

## Remarks

The following rules apply to this annotation's use:

- The message property resource cannot reside in the WEB-INF/... directory of another web project. The \*.properties file must reside in a /WEB-INF/classes/<folder>/ directory, or in a JAR in /WEB-INF/lib (in a JAR), for this web project.
- For a sample page flow that demonstrates form validation and the use of the message properties, see the following page flow. Also see the <netui:error> tag used in this sample page flow's JSP files.

```
<WEBLOGIC_HOME>\samples\workshop\SamplesApp\WebApp\validation\Controller.jsp
```

## WebLogic Workshop Annotations Reference

- For sample message properties files, see:

`<WEBLOGIC_HOME>\samples\workshop\SamplesApp\WebApp\WEB-INF\classes\validation`

and:

`<WEBLOGIC_HOME>\samples\workshop\SamplesApp\WebApp\WEB-INF\classes\validation`

### Related Topics

[@jpf:catch Annotation](#)

[@jpf:controller Annotation](#)

[@jpf:exception-handler Annotation](#)

[@jpf:forward Annotation](#)

[@jpf:validation-error-forward Annotation](#)

[Getting Started with Page Flows](#)



# @jpf:validation-error-forward Annotation

You can use the @jpf:validation-error-forward annotation to indicate which page should be loaded, or which action should be run, if a form validation error occurs as a result of running the annotated action.

**Note:** Starting in WebLogic Workshop 8.1 Service Pack 2, this annotation replaces the deprecated validation-error-page attribute that was available on the @jpf:action annotation. The @jpf:validation-error-forward annotation's attributes provide a more flexible validation forwarding behavior; specifically, they allow you to indicate return-to="..." options, and they allow forwarding to non-local files.

## Syntax

@jpf:validation-error-forward

name = "<forward name>"

{ path = "<path>" | return-action = "<action name>" | return-to = { "currentPage" | "previousPage" | "previousAction" } }

{ return-form = "<form name>" | return-form-type = "<form type>" }

[ redirect = { "true" | "false" } ]

## Attributes

name

Required. Specifies the name of the forward, which causes the navigation to occur.

path

You must specify only one of the following: this path attribute, or the return-action attribute, or one of the return-to attributes. The path attribute is a string that maps to one of the following entities:

- A JSP or another page flow in this web application
- An action within a page flow in this web application
- A JSP somewhere on the web, in this web application or a different web application. Currently the URL must be full URL, such as path="http://someOtherServer.com/foo/bar/valError.jsp".

If the path begins with a protocol such as "http:" the page flow runtime will look outside of this web application for the resource, and it will automatically cause a redirect (rather than a server forward) to the resource. If the path begins with a forward slash, "/", the runtime will start at the web application's root directory to locate the resource. If the path omits the forward slash, "/", the reference is relative to the page flow's directory.

return-to

## WebLogic Workshop Annotations Reference

The return-to attribute always applies to the current page flow, whether it is a nested page flow or the main page flow. To return to an action in the nesting page flow use the return-action attribute.

The value for the return-to attribute must be a keyword, either "currentPage", or "previousPage", or "previousAction".

**Note:** To clarify the purpose of the return-to attribute, the keywords "page" and "action" were deprecated as of WebLogic Workshop 8.1 Service Pack 2. Instead of "page", the equivalent function is "previousPage". Instead of "action", the equivalent function is "previousAction".

If the value is "currentPage", the same JSP page is rendered again by the server, along with any updated data that occurred as a result of executing the annotated action method.

If the value is "previousPage", the page that was shown before the current page is rendered.

If the value is "previousAction", the previous action in the current page flow is run.

### return-action

The return-action attribute, which is only valid in a *nested* page flow, causes control to return to the calling (or "nesting") page flow (leaving the current page flow), and then causes the specified action to be raised on the calling page flow.

### return-form

Optional. The return-form attribute, which is only valid when used with the return-action attribute, causes the given page flow member variable to be attached to the returned Forward automatically.

### return-form-type

Optional. The return-form-type attribute, which is only valid when used with the return-action attribute, is used in conjunction with the Forward that is returned. Forward takes the actual FormData instance as the second argument to its constructor. This attribute declares the type of the form bean that will be returned to the calling page flow.

**Note:** If multiple actions with the same name (such as "success") exist in the calling page flow, either the return-form-type value or the return-form member variable type will be used to determine the appropriate action.

### redirect

Optional. A boolean, true or false. Default is false. When set to true, navigation causes a browser redirect to the specified destination. Redirecting is useful when you want to clear out any data that was attached to a request, or when it is important that the user's URL bar reflect the actual page that is displayed (instead of the action name). For details, see the Remarks section below.

## Remarks

The following rules apply to this annotation's use:

## WebLogic Workshop Annotations Reference

- If you used the deprecated `validation-error-page` attribute on the `@jpf:action` annotation, use this more flexible `@jpf:validation-error-forward` annotation instead.
- The `@jpf:validation-error-forward` annotation is designed to let you handle form input validation errors. In that sense, this annotation is not related to the `@jpf:catch` and `@jpf:exception-handler` annotations, which are designed to catch and handle exceptions.
- The JPF compiler in WebLogic Workshop will issue an error if there is more than one `@jpf:validation-error-forward` annotation on an action.
- The target value for the `path` attribute can be local or remote, in the same page flow, or a different page flow, or a page on a different server. For the `return-to` attribute, the target value refers to an entity in the current page flow. A few examples:

```
* <!-- Same page flow: -->
* @jpf:validation-error-forward name="failure" path="foo.jsp"

* <!-- Different page flow: -->
* @jpf:validation-error-forward name="failure" path="/foo/bar.jsp"

* <!-- Different server: -->
* @jpf:validation-error-forward name="failure"
*   path="http://someOtherServer.com/foo/bar/etc.jsp"

* <!-- Current page in the current page flow: -->
* @jpf:validation-error-forward name="failure"
*   return-to="currentPage"

* <!-- Previous page in the current page flow: -->
* @jpf:validation-error-forward name="failure"
*   return-to="previousPage"

* <!-- Previous action in the current page flow -->
* @jpf:validation-error-forward name="failure"
*   return-to="previousAction"
```

- Here is a portion of the validation sample that is installed with WebLogic Workshop. For the full sample, see `<WEBLOGIC_HOME>\samples\workshop\SamplesApp\WebApp\validation\...`

```
/**
 * @jpf:action
 * @jpf:forward name="success" path="success.jsp"
 * @jpf:validation-error-forward name="failure" return-to="currentPage"
 */
public Forward submitForm( Form form )
{
    return new Forward( "success" );
}

...

/**
 * This form bean does validation manually.
 */
public static class Form extends FormData
{
    private String _email;
```

## WebLogic Workshop Annotations Reference

```
private String _zipCode;

public String getEmail()
{
    return _email;
}

public void setEmail( String email )
{
    _email = email;
}

public String getZipCode()
{
    return _zipCode;
}

public void setZipCode( String zipCode )
{
    _zipCode = zipCode;
}

public ActionErrors validate( ActionMapping mapping, HttpServletRequest request )
{
    ActionErrors errs = new ActionErrors();

    int at = _email.indexOf( '@' );
    int dot = _email.lastIndexOf( '.' );

    if ( at == -1 || at == 0 || dot == -1 || at > dot )
    {
        errs.add( "email", new ActionError( "badEmail" ) );
    }

    if ( _zipCode.length() != 5 )
    {
        errs.add( "zipCode", new ActionError( "badZip", new Integer( 5 ) ) );
    }

    return errs;
}
```

### Related Topics

[Validating User Input](#)

[Form Validation Sample](#)

[@jpf:catch Annotation](#)

[@jpf:controller Annotation](#)

[@jpf:exception-handler Annotation](#)

[@jpf:forward Annotation](#)

[@jpf:message-resources Annotation](#)

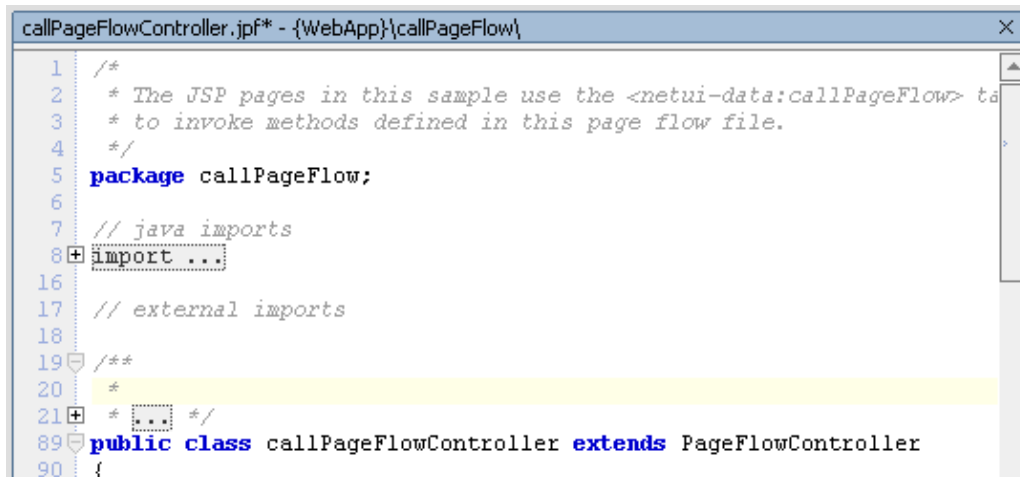
[@jpf:validation-error-forward Annotation](#)

Getting Started with Page Flows

# @jpf:view-properties Annotation

Describes layout information for your page flow's graphical Flow View. This content is generated automatically for you when you create or modify the page flow in the IDE. Do not edit this data.

By default the code that comprises the view properties is contained in a collapsed set of lines; in Source View, this looks similar to the following sample, where lines 21 through 88 are collapsed:

A screenshot of an IDE window titled 'callPageFlowController.jsp\* - {WebApp}\callPageFlow\'. The code is in Java/JSP syntax. Lines 1-4 are comments. Line 5 is 'package callPageFlow;'. Line 7 is '// java imports'. Line 8 is 'import ...' with a collapsed block icon. Line 16 is '// external imports'. Line 19 is '/\*\*'. Line 20 is a comment. Line 21 is a collapsed block icon. Line 89 is 'public class callPageFlowController extends PageFlowController'. Line 90 is '{'.

```
1  /*
2   * The JSP pages in this sample use the <netui-data:callPageFlow> tag
3   * to invoke methods defined in this page flow file.
4   */
5  package callPageFlow;
6
7  // java imports
8  import ...
9
16 // external imports
17
18
19 /**
20  *
21  * ... */
22
89 public class callPageFlowController extends PageFlowController
90 {
```

## Syntax

@jpf:view-properties

## Attributes

None.

## Remarks

Do not edit the automatically generated values that appear in the @jpf:view-properties comments.

If you add one of the other @jpf annotations at the class level, such as @jpf:controller, @jpf:catch, or @jpf:forward, you should add the new annotation in the existing comment block. Do not start a new comment block, as this may result in an extra copy of commented-out (and safely ignored) view properties in the JPF file.

## Related Topics

@jpf:action Annotation

@jpf:catch Annotation

@jpf:controller Annotation

@jpf:forward Annotation

@jpf:view-properties Annotation

## WebLogic Workshop Annotations Reference

@jpf:exception-handler Annotation

@jpf:message-resources Annotation

@jpf:validation-error-forward Annotation

Getting Started with Page Flows

# Custom Control Annotations

The custom control annotations are used in custom control implementation files.

## Topics Included in This Section

`@jcs:control-tags` Annotation

Specifies the XML file that contains definitions for the control's properties and attributes.

`@jcs:ide` Annotation

Specifies whether a control method or callback should be hidden in the IDE.

`@jcs:jc-jar` Annotation

Specifies characteristics of the control as it appears in the Weblogic Workshop IDE.

`@jcs:suppress-common-tags` Annotation

Provides a means to suppress certain annotations in a custom control implementation (JCS file).

Related Topics

WebLogic Workshop Annotations Reference



# @jcs:control-tags Annotation

Specifies the XML file that contains definitions for the control's properties and attributes.

## Syntax

```
@jcs:control-tags  
  file="pathToAnnotationXMLFile"
```

## Attributes

file

Required. The path to the annotation XML that defines properties for this control. This path must be relative to the JCS file.

Remarks

None.

Related Topics

[How Do I: Define Properties for a Java Control](#)

[Control Property Schema Reference](#)

[Tutorial: Java Control](#)

# @jcs:ide Annotation

Specifies whether a control method or callback should be hidden in the IDE.

## Syntax

```
@jcs:ide  
  hide="true" | "false"
```

## Attributes

hide

Optional. true to hide the method or callback; false to display it.

## Remarks

None.

Related Topics

None.

# @jcs:jc-jar Annotation

Specifies characteristics of the control as it appears in the Weblogic Workshop IDE. These include icons associated with the control, its name on menus and palettes, and so on.

## Syntax

```
@jcs:jc-jar
  description="descriptionInPropertyEditor"
  display-in-palette="true | false"
  group-priority="numberIndicatingGroupPriority"
  group-name="nameOfSubmenu"
  icon-16="pathToGifFile"
  icon-32="pathToGifFile"
  insert-wizard-class="fullyQualifiedNameOfWizardClass"
  label="nameOnMenusAndPalettes"
  palette-priority="numberIndicatingPalettePriority"
  requires-extension="true | false"
  resource-file="nameOfLocalizationResource"
  version="versionNumber"
```

## Attributes

description

Optional. The text that will appear in the Description pane when the control is selected in Design View.

display-in-palette

Optional. Defines whether this control should be shown in the palette.

group-priority

Optional. A number indicating where in the Insert menu the group named in the group-name attribute should appear. Higher numbers (relative to other groups) place the group higher in the menu. Default is 0.

group-name

Optional. The name of the submenu under which this control should appear. If the group-name attribute value ends with the string "Menu" (such as "MyGroupMenu"), then the groupname, stripped of the word Menu, always becomes a submenu name, and all controls sharing that group appear on the submenu. If the group name does not end in the word Menu, then controls in that group appear as top-level items in the Insert Controls menu, unless there are more than five controls in that group. In that case, the group name becomes the top level menu name and all controls in that group appear as sub-menu items.

icon-16

Optional. A path (relative to the JCS file) to a .gif file that should be used as the control's icon on menus, palettes, in Design View, and so on.

icon-32

Optional. A path (relative to the JCS file) to a .gif file that should be used as the control's large icon.

insert-wizard-class

Optional. A fully-qualified name for a class that implements a custom insert dialog box.

label

Optional. The name used for the control on menus and palettes.

palette-priority

Optional. A number that suggests a priority for this control when the IDE is ordering controls in the palette.

requires-extension

Optional. Defines whether this control must be customized. Usually set to false.

resource-file

Optional. Name of the resource to use in loading localization strings.

version

Optional. The control's version number.

## Remarks

None.

Related Topics

Tutorial: Java Control

How Do I: Specify IDE Characteristics for a Java Control?

# @jcs:suppress-common-tags Annotation

The @jcs:suppress-common-tags annotation provides a means to suppress certain annotations in a custom control implementation (JCS file). The annotations that may be suppressed are @common:define, @common:schema, @common:target-namespace, and @common:xmlns. You can use the @jcs:suppress-common-tags annotation to suppress these annotations if they are not appropriate for the control you are building. The annotations which are suppressed are subsequently not available to JCX controls which extend the custom control: the suppressed annotations do not appear in the Property Editor for the control in Design View; and in Source View, adding a suppressed annotation will generate a compile-time error.

## Syntax

@jcs:suppress-common-tags

tags="annotationList"

## Attributes

tags

Optional. Specifies the list of @common annotations, separated by spaces, that should be suppressed for this control. Note that you do not need to include the @common prefix, only the annotation name.

## Remarks

The following @common annotations appear by default on a custom control, but can be suppressed using the @jcs:suppress-common-tags annotation:

- @common:define
- @common:schema
- @common:target-namespace
- @common:xmlns

The following fragment shows how to use the @jcs:suppress-common-tags annotation to suppress the @common:define, @common:schema, @common:target-namespace, and @common:xmlns annotations:

```
/**
 * @jcs:suppress-common-tags tags="define schema target-namespace xmlns"
 */
```

## Related Topics

Tutorial: Java Control

Building Custom Java Controls

# Enterprise JavaBean Annotations

When you develop Enterprise JavaBeans in WebLogic Workshop, you will notice special Javadoc tags with the prefix `ejbgen` in the source view of the bean class. Instead of using separate classes for the various interfaces and instead of directly changing the deployment descriptor, `ejbgen` tags reduces to one the number of EJB files you need to edit and maintain during development. When you build an Enterprise JavaBeans, these tags are used to generate the various 'J2EE-standard' files for remote and home interfaces, and the appropriate deployment descriptors.

## Topics Included in This Section

`@ejbgen:automatic-key-generation` Annotation

In a CMP entity bean, specifies how primary keys are auto-generated.

`@ejbgen:cmp-field` Annotation

In a CMP entity bean, marks a CMP field.

`@ejbgen:cmr-field` Annotation

In a CMP entity bean, marks a CMR field.

`@ejbgen:compatibility` Annotation

In a CMP entity bean, specifies compatibility settings with prior versions of WebLogic.

`@ejbgen:create-default-dbms-tables` Annotation

This tag is deprecated. Please use the `@ejbgen:jar-settings` attribute `create-tables`.

`@ejbgen:ejb-client-jar` Annotation

This tag is deprecated; please use `@ejbgen:jar-settings` attribute `ejb-client-jar`

`@ejbgen:ejb-interface` Annotation

This tag specifies the EJB interface the bean implements. In most cases this tag does not need to be used.

`@ejbgen:ejb-local-ref` Annotation

This tag maps a JNDI reference used within a bean to the referenced bean's local interfaces.

`@ejbgen:ejb-ref` Annotation

This tag maps a JNDI reference used within a bean to the referenced bean's remote interfaces.

`@ejbgen:entity-cache-ref` Annotation

This tags specifies a reference to an application-level entity cache.

### @ejbgen:entity Annotation

This tag defines the class-scope properties of an entity bean.

### @ejbgen:env-entry Annotation

This tag specifies environment entries for an EJB.

### @ejbgen:file-generation Annotation

This tag specifies the interface, compound primary key, and value classes that are to be auto-generated during build.

### @ejbgen:finder Annotation

This tag specifies a finder method for a CMP entity bean.

### @ejbgen:foreign-jms-provider Annotation

This tag specifies a non-BEA JMS provider for a message-driven bean.

### @ejbgen:jar-settings Annotation

In an EJB project, specifies JAR and table management settings.

### @ejbgen:jndi-name Annotation

This tag specifies the local and remote JNDI names of an EJB.

### @ejbgen:local-home-method Annotation

This tag defines the methods for an entity bean's local home interface.

### @ejbgen:local-method Annotation

This tag defines the methods for an entity or session bean's local (business) interface.

### @ejbgen:message-driven Annotation

This tag defines the class-scope properties of a message-driven bean.

### @ejbgen:method-isolation-level-pattern Annotation

This tag specifies the transaction isolation level for methods matching a given pattern.

### @ejbgen:method-permission-pattern Annotation

This specifies security roles authorized to invoke the methods matching a given pattern.

### @ejbgen:relation Annotation

## WebLogic Workshop Annotations Reference

This tag specifies an entity relationship between two CMP entity beans.

@ejbgen:relationship-caching-element Annotation

This tag specifies an eager relationship loading definition.

@ejbgen:remote-home-method Annotation

This tag defines the methods for the entity or session bean's local home interface.

@ejbgen:remote-method Annotation

This tag defines the methods for an entity bean's remote home interface.

@ejbgen:resource-env-ref Annotation

This tag maps a JNDI reference used within a bean to an administered object associated with an external resource.

@ejbgen:resource-ref Annotation

This tag maps a JNDI reference used within a bean to an external resource connection factory.

@ejbgen:role-mapping Annotation

This tag defines an EJB-scoped security role used by the beans in an EJB Project.

@ejbgen:security-role-ref Annotation

This tag maps a reference to a security role used in the bean code to an EJB-scoped security role defined in the EJB project.

@ejbgen:select Annotation

This tag specifies a select method for a CMP entity bean.

@ejbgen:session Annotation

This tag defines the class-scope properties of a session bean.

@ejbgen:value-object Annotation

This tag specifies whether EJB object references in a generated value object should be represented as a value object or a local interface.

Related Topics

WebLogic Workshop Annotations Reference



# @ejbgen:automatic-key-generation Annotation

For a CMP entity bean, specifies how primary keys are auto-generated. Primary keys can be auto-generated using Oracle or SQL server specific methods or a vendor-neutral named sequence table. A summary of the syntax is provided below. For detailed explanations and examples, see the related topics at the bottom of the page.

## Scope

Class tag for a CMP entity bean.

## Syntax

**@ejbgen:automatic-key-generation**

type="keyGeneratorType"

[name="nameKeyGenerator"]

[cache-size="keyCacheSize"]

## Attributes

type

Required. The following key generator keywords are available:

- ◆ *Oracle*. Use an Oracle SEQUENCE database table.
- ◆ *SQLServer*. Use SQL Server's IDENTITY column.
- ◆ *SQLServer2000*. Use SQL Server 2000's IDENTITY column.
- ◆ *NamedSequenceTable*. Use a named sequence table.

**Note.** The SQLServer2000 option is the same as SQLServer, except that SQLServer uses @@IDENTITY column to get the generated key value and SQLServer2000 uses the SCOPE\_IDENTITY() function instead.

cache-size

Optional. Specifies the size of the key cache.

name

Optional. Specifies the name of the Oracle SEQUENCE table or the name of the named sequence table.

## Remarks

The following rules apply to this tag's use:

- When you specify the Oracle **type**, you must specify the **name** of the SEQUENCE table and the

## WebLogic Workshop Annotations Reference

***cache-size***. The ***cache-size*** must match the value of the increment.

- When you specify the `SQLServer(2000)` ***type***, you can optionally specify the ***cache-size***. The ***cache-size*** determines how many keys are fetched from the named sequence table by the EJB container. A large cache size lowers the number of trips to the table.
- When you specify the `NamedSequenceTable` ***type***, you must specify the ***name*** of the named sequence table and the ***cache-size***. The ***cache-size*** determines how many keys are fetched from the named sequence table by the EJB container. A large cache size lowers the number of trips to the named sequence table.

Related Topics

Automatic Primary Key Generation

Automatic Primary Key Generation Sample

# @ejbgen:cmp-field Annotation

In a CMP entity bean, marks a virtual container-managed persistence (CMP) field.

## Scope

Method tag for a CMP entity bean.

## Syntax

**@ejbgen:cmp-field**

column="tableName"

[column-type="OracleClob/OracleBlob"]

[exclude-from-value-object="True/False"]

[group-names="GroupName"]

[ordering-number="0...N"]

[primkey-field="True/False"]

[read-only-in-value-object="True/False"]

[table-name="TableName"]

## Attributes

column

Required. Specifies the column in a database table to which this CMP field will be mapped.

column-type

Optional. Specify OracleClob or OracleBlob to map the CMP field to these database column types. When you use this field, you must declare the CMP field to either be a byte array or a class that implements the Serializable interface. If you want to serialize a byte array that is mapped to an OracleBlob, you must set the serialize-byte-array-to-oracle-blob attribute of the ejbgen:compatibility tag to true (default is false).

exclude-from-value-object

Optional. Specify true if you don't want this field to be included in the auto-generated value object (data transfer object) class. When left unspecified, this attribute defaults to false. For more information, see @ejbgen:file-generation Annotation.

group-names

Optional. This attribute defines how CMP fields of an entity bean are stored in memory. CMP fields and CMR fields that have the same `group-name` attribute specified are stored together. When this property is not used, all CMP fields and CMR fields are stored together in memory. Breaking up the memory signature into two or more groups can improve performance. For more details, see *Accelerating Entity Bean Data Access*.

`ordering-number`

Optional. Specify a number to enforce where this field will appear relative to other fields in signatures and constructors of generated compound primary key and value object (data transfer object) classes. In order for this ordering to work, all CMR and CMP fields must have this attribute set with a distinct numeric value. See the remark below.

`primkey-field`

Optional. Sets whether this field is the primary key field, or part of the compound primary key. The default is `false`.

`read-only-in-value-object`

Optional. Specify `True` if you want this field to be read-only in a generated value object class.

`table-name`

Optional. Specify the table(s) where this field should be mapped to. If the `table-name` is not specified, the CMP field is mapped to the table specified in the `ejbgen:entity` tag. You can also use this attribute to map primary key fields to multiple tables within the same database. For more information, see the example below.

## Remarks

The following comments apply to this tag's use:

- Various attributes (`exclude-from-value-object`, `ordering-number`, and `read-only-in-value-object`) affect value and primary key classes generated during the build process. These attributes do not affect explicit definitions of dependent value classes and/or method definitions in EJBs in the EJB Project folder.
- For more information on the (disabling of) automatic creation of value objects, see `@ejbgen:file-generation` Annotation.
- If you specify `ordering-number` to all CMP and CMR fields, it will only affect compound primary key and value object classes that are generated during the build process. For instance, if you define three primary key fields `FieldA`, `FieldB`, and `FieldC`, and need to enforce that the compound primary key class constructor uses these arguments in the given order, you can set the `ordering-number` of these attributes (to 1, 2, and 3). This will ensure this order, even if the `ejbCreate` method that returns the compound primary key object calls these arguments in a different order, for instance:

```
public my.customer.UntitledPK ejbCreate(int FieldC, java.lang.String FieldA, java.lang.S
```

Also notice that the `ordering-number` attribute does not enforce the order of arguments in the methods in the `ejb` file itself or in any dependent value classes you have explicitly defined in your EJB project.

## Multiple Table Mapping Example

The following example demonstrates the use of the table-name property. The MultipleTablesBean entity bean by default maps to the table OneTable as defined in the ejbgen:entity tag. The DefaultTable\_field CMP field has no table-name property on its ejbgen:cmp-field tag and therefore maps to this table. The OtherTable\_field CMP field has the table-name property set to OtherTable and therefore maps to the table OtherTable. The setMultipleTables\_ID primary key field is set to both tables. Only primary keys can map to multiple tables:

```
/**
 * @ejbgen:entity prim-key-class="java.lang.Integer"
 *   ejb-name="MT"
 *   data-source-name="cgSampleDataSource"
 *   table-name="OneTable"
 *   abstract-schema-name = "Divided"
 *   ...
 */
public abstract class MultipleTablesBean extends GenericEntityBean implements EntityBean
{

    /**
     * @ejbgen:cmp-field column="DefaultTable_field"
     * @ejbgen:local-method
     */
    public abstract String getDefaultTable_field();

    /**
     * @ejbgen:local-method
     */
    public abstract void setDefaultTable_field(String arg);

    /**
     * @ejbgen:cmp-field table-name="OtherTable" column="OtherTable_field"
     * @ejbgen:local-method
     */
    public abstract String getOtherTable_field();

    /**
     * @ejbgen:local-method
     */
    public abstract void setOtherTable_field(String arg);

    /**
     * @ejbgen:cmp-field table-name="OneTable, OtherTable" primkey-field="true" column="Divided"
     * @ejbgen:local-method
     */
    public abstract Integer getMultipleTables_ID();

    /**
     * @ejbgen:local-method
     */
    public abstract void setMultipleTables_ID(Integer arg);

    public java.lang.Integer ejbCreate(java.lang.Integer MultipleTables_ID, java.lang.String Ot
    {
```

## WebLogic Workshop Annotations Reference

```
    setMultipleTables_ID(MultipleTables_ID);  
    setOtherTable_field(OtherTable_field);  
    setDefaultTable_field(DefaultTable_field);  
  
    return null; // FIXME return PK value  
}  
  
public void ejbPostCreate(java.lang.Integer MultipleTables_ID, java.lang.String OtherTable_  
{  
}  
}
```

### Related Topics

[How Do I: Define a Container-Managed Persistence \(CMP\) Field?](#)

[Accelerating Entity Bean Data Access](#)

[@ejbgen:entity Annotation](#)

[@ejbgen:file-generation Annotation](#)

[@ejbgen:cmr-field Annotation](#)

[ejbgen:compatibility Annotation](#)

# @ejbgen:cmr-field Annotation

In a CMP entity bean, marks a container-managed relationship field. The entity relation that uses this CMR field is described in an @ejbgen:relation tag.

## Scope

Method tag for a CMP entity bean.

## Syntax

**@ejbgen:cmr-field**

[exclude-from-value-object="True/False"]

[group-names="GroupName"]

[ordering-number="0...N"]

[read-only-in-value-object="True/False"]

## Attributes

exclude-from-value-object

Optional. Specify True if you don't want this field to be included in the auto-generated value object (data transfer object) class. When left unspecified, this attribute defaults to false. For more information, see @ejbgen:file-generation Annotation.

group-names

Optional. This attribute defines how CMR fields of an entity bean are stored in memory. CMP fields and CMR fields that have the same group-name attribute specified are stored together. When this property is not used, all CMP fields and CMR fields are stored together in memory. Breaking up the memory signature into two or more groups can improve performance. For more details, see Accelerating Entity Bean Data Access.

ordering-number

Optional. Specify a number to enforce where this field will appear relative to other fields in signatures and constructors of generated compound primary key and value object (data transfer object) classes. In order for this ordering to work, all CMR and CMP fields must have this attribute set with a distinct numeric value.

read-only-in-value-object

Optional. Specify True if you want this field to be read-only in a generated value object class.

## Remarks

For more information on these attributes, see the identical attributes for @ejbgen:cmp-field Annotation. For more information on entity relations, see the @ejbgen:relation Annotation.

### Related Topics

How Do I: Add a Relation to an Entity Bean?

Accelerating Entity Bean Data Access

@ejbgen:cmp-field Annotation

@ejbgen:entity Annotation

@ejbgen:file-generation Annotation

@ejbgen:relation Annotation



# @ejbgen:compatibility Annotation

In a CMP entity bean, specifies compatibility settings with prior versions of WebLogic.

## Scope

Class tag for a CMP entity bean.

## Syntax

**@ejbgen:compatibility**

`serialize-byte-array-to-oracle-blob="True/False"`

## Attributes

`serialize-byte-array-to-oracle-blob`

Optional. Specify True for this attribute if, for a CMP field, you want to serialize a byte array that is mapped to an OracleBlob. For more information about OracleBlobs, see `ejbgen:cmp-field`. In prior versions of WebLogic, this was the default behavior but starting with WebLogic 8.1 SP2, you must set this attribute to True to obtain this behavior.

Related Topics

`@ejbgen:cmp-field` Annotation

# **@ejbgen:create-default-dbms-tables Annotation**

This tag is deprecated. Please use the @ejbgen:jar-settings attribute create-tables.

Related Topics

[@ejbgen:jar-settings Annotation](#)

## **@ejbgen:ejb-client-jar Annotation**

This tag is deprecated; please use @ejbgen:jar-settings attribute ejb-client-jar.

Related Topics

[@ejbgen:jar-settings Annotation](#)

# @ejbgen:ejb-interface Annotation

Specifies the EJB interface the bean implements. In most cases this tag does not need to be used; When you define a new EJB using the IDE, the bean definition will automatically implement the correct interface. For more information, see [How Do I: Create an Enterprise JavaBean?](#)

## Scope

Class tag for a an EJB.

## Syntax

**@ejbgen:ejb-interface**

ejb-type="InterfaceName"

## Attributes

ejb-type

Required. One of the following interfaces can be selected:

- ◆ javax.ejb.MessageDrivenBean
- ◆ javax.ejb.EntityBean
- ◆ javax.ejb.SessionBean

Related Topics

[How Do I: Create an Enterprise JavaBean?](#)

# @ejbgen:ejb-local-ref Annotation

Maps a JNDI reference used by a bean to the referenced bean's local interfaces.

## Scope

Class tag on the EJB that references the other EJB through its local interfaces.

## Syntax

**@ejbgen:ejb-local-ref**

[home="LocalHomeInterFace"]

[id="TagID"]

[jndi-name="JNDIName"]

[link="EJBName"]

[local="LocalInterface"]

[name="ReferenceName"]

[type="Entity/Session"]

## Attributes

home

Optional. Specifies the local home interface of the referenced EJB.

id

Optional. Specifies the ID of the tag. For more information, see EJBGen Tag Inheritance.

jndi-name

Optional. Specifies the local JNDI name of the referenced EJB.

link

Optional. Specifies the (descriptive) name of the referenced bean, that is the name given in the ejb-name property located in the *General Settings* section of the *Property Editor*.

local

Optional. Specifies the local (business) interface of the referenced EJB.

name

Optional. The name used to reference the other bean.

type

Optional. Specifies the EJB type of the referenced bean. Valid values are Entity and Session.

## Remarks

To use this tag, specify its attributes in one of the following ways:

- Specify link to map the reference.
- Specify name, jndi-name, home, local, and type.

## Example

In this example, a Band EJB references a Recording EJB. Here is the partial code for the Band EJB, with the reference definition and use shown in bold:

```
/**
 * @ejbgen:ejb-local-ref link="Recording"
 * ...
 */
abstract public class BandBean extends GenericEntityBean implements EntityBean
{
    ...
    private RecordingHome recordingHome;

    public void setEntityContext(EntityContext c) {
        ctx = c;
        try {
            javax.naming.Context ic = new InitialContext();
            recordingHome = (RecordingHome)ic.lookup("java:/comp/env/ejb/Recording");
        }
        catch(Exception e) {
            System.out.println("Unable to obtain RecordingHome: " + e.getMessage());
        }
    }
    ...
}
```

The name of the Recording bean is given in the definition of that bean:

```
/**
 * @ejbgen:entity prim-key-class="bands.RecordingBeanPK"
 * ejb-name = "Recording"
 * data-source-name="cgSampleDataSource"
 * table-name="recording_Sample"
 * abstract-schema-name = "Recording"
 *
 * ...
 */
abstract public class RecordingBean extends GenericEntityBean implements EntityBean
```

```
{  
    ...  
}
```

The Band bean defines the local reference to the Recording bean by specifying the Recording bean's name in the link attribute. It uses this information to locate and reference the Recording bean via the InitialContext's lookup method. To learn more about JNDI naming, see the tutorial [Getting Started: Enterprise JavaBeans](#). To see an example of a local reference tag that specifies name, jndi-name, home, local, and type, see [@ejbgen:ejb-ref Annotation](#).

### Related Topics

[@ejbgen:ejb-ref Annotation](#)

[@ejbgen:jndi-name Annotation](#)

[@ejbgen:file-generation Annotation](#)

[How Do I: Add a Relation to an Entity Bean?](#)

[Tutorial: Enterprise JavaBeans](#)

# @ejbgen:ejb-ref Annotation

Maps a JNDI reference used by a bean to the referenced bean's remote interfaces.

## Scope

Class tag on the EJB that references the other EJB through its remote interfaces.

## Syntax

**@ejbgen:ejb-ref**

[home="RemoteHomeInterFace"]

[id="TagID"]

[jndi-name="JNDIName"]

[link="EJBName"]

[remote="RemoteInterface"]

[name="ReferenceName"]

[type="Entity/Session"]

## Attributes

home

Optional. Specifies the remote home interface of the referenced EJB.

id

Optional. Specifies the ID of the tag. For more information, see EJBGen Tag Inheritance.

jndi-name

Optional. Specifies the remote JNDI name of the referenced EJB.

link

Optional. Specifies the (descriptive) name of the referenced bean, that is the name given in the ejb-name property located in the *General Settings* section of the *Property Editor*.

remote

Optional. Specifies the remote (business) interface of the referenced EJB.



name

Optional. The name of used to reference another bean.

type

Optional. Specifies the EJB type of the referenced bean. Valid values are Entity and Session.

## Remarks

To use this tag, specify its attributes in one of the following ways:

- Specify link to map the reference.
- Specify name, jndi-name, home, remote, and type.

If the two beans are co-located in the same EJB container, consider using a local reference to allow the beans to interact without the overhead of a distributed object protocol, thereby improving performance. For more information on a local reference, see `@ejbgen:ejb-local-ref` Annotation.

## Example

In this example, a Band EJB references a Recording EJB. Here is the partial code for the Band EJB, with the reference definition and use shown in bold:

```

* @ejbgen:ejb-ref type="Entity" remote="RecordingRemote" home="RecordingRemoteHome" jndi-name=
* name="ejb/recordLink"
* ...
*/
abstract public class BandBean extends GenericEntityBean implements EntityBean
{
    ...
    private RecordingRemoteHome recordingRemoteHome;

    public void setEntityContext(EntityContext c) {
        ctx = c;
        try {
            javax.naming.Context ic = new InitialContext();
            recordingRemoteHome = (RecordingRemoteHome)ic.lookup("java:/comp/env/ejb/recordLink");
        }
        catch(Exception e) {
            System.out.println("Unable to obtain RecordingHome: " + e.getMessage());
        }
    }
    ...
}

```

The Band bean defines the reference to the Recording bean by specifying the Recording bean's type, remote home interface, remote business interface, and remote JNDI name, and it names the local reference `ejb/recordLink`. It uses this reference name to locate and reference the Recording bean via the `InitialContext`'s `lookup` method. To learn more about JNDI naming, see the tutorial *Getting Started: Enterprise JavaBeans*.

The name of the Recording bean is given in the definition of that bean:

```
/**
```

## WebLogic Workshop Annotations Reference

```
* @ejbgen:entity prim-key-class="bands.RecordingBeanPK"
*   ejb-name = "Recording"
*   data-source-name="cgSampleDataSource"
*   table-name="recording_Sample"
*   abstract-schema-name = "Recording"
*
* @ejbgen:jndi-name remote="ejb.RecordingRemoteHome"
*
* @ejbgen:file-generation ...
* remote-home-name = "RecordingRemoteHome"
* remote-class-name = "RecordingRemote"
* ...
*/
abstract public class RecordingBean extends GenericEntityBean implements EntityBean
{
    ...
}
```

To see an example of a reference tag that specifies link only, see @ejbgen:ejb-local-ref Annotation.

### Related Topics

@ejbgen:ejb-local-ref Annotation

@ejbgen:jndi-name Annotation

@ejbgen:file-generation Annotation

How Do I: Add a Relation to an Entity Bean?

Tutorial: Enterprise JavaBeans

# @ejbgen:entity-cache-ref Annotation

This tag specifies a reference to an application-level entity cache. If the `ejbgen:entity-cache-ref` is not specified, a separate cache is used for each entity bean.

## Scope

Class tag on an entity bean

## Syntax

`@ejbgen:entity-cache-ref`

`concurrency-strategy="ConcurrencyOption"`

`name="CacheName"`

`[cache-between-transactions="True/False"]`

`[estimated-bean-size="ByteSize"]`

## Attributes

`concurrency-strategy`

Required. Defines the concurrency strategy for the entity bean. Valid values are `ReadOnly`, `Exclusive`, and `Database`. When left unspecified, the default is `Database`.

`name`

Required. Names the application-level cache.

`cache-between-transactions`

Optional. Specifies whether to cache the persistent data of an entity bean across (between) transactions. When left unspecified, the default is `false`.

`estimated-bean-size`

Optional. Specifies the estimated average size of an instance of an entity bean, in bytes. When left unspecified, the default is 100.

## Remarks

The following rules apply to this tag's use:

- When entity-cache related attributes are set using the `@ejbgen:entity` Annotation, such as `cache-between-transactions`, `idle-timeout-seconds`, and `max-beans-in-cache`, the

## WebLogic Workshop Annotations Reference

`ejbgen:entity-cache-ref` tag is ignored.

- Using application-level entity bean caching, and referencing an application-level cache with this tag is discussed in detail in [Accelerating Entity Bean Data Access](#).

### Related Topics

[Accelerating Entity Bean Data Access](#)

[@ejbgen:entity Annotation](#)

# @ejbgen:entity Annotation

Defines the class-scope properties of an entity bean.

## Scope

Class tag for an entity bean.

## Syntax

**@ejbgen:entity**

ejb-name="NameCMPBean"

prim-key-class="PKClassName"

[abstract-schema-name="AbstractSchemaName"]

[cache-between-transactions="True/False"]

[check-exists-on-method="True/False"]

[clients-on-same-server="True/False"]

[concurrency-strategy="ConcurrencyStrategyOption"]

[data-source-name="DataSourceName"]

[database-type="DatabaseOption"]

DEPRECATED [db-is-shared="True/False"]

[default-dbms-tables-ddl="DDLFileName."]

[default-transaction="DefaultTransactionOption"]

[delay-database-insert-until="ejbCreate/ejbPostCreate"]

[delay-updates-until-end-of-tx="True/False"]

[disable-warning="WarningsList"]

[dispatch-policy="DispatchPolicyQueue"]

[enable-batch-operations="True/False"]

[enable-call-by-reference="True/False"]

[enable-dynamic-queries="True/False"]

@ejbgen:entity Annotation

[finders-load-bean="True/False"]

[home-call-router-class-name="ClassName"]

[home-is-clusterable="True/False"]

[home-load-algorithm="LoadAlgorithmOption"]

[idle-timeout-seconds="TimeoutValue"]

[invalidation-target="ReadOnlyBean"]

[lock-order="number"]

[max-beans-in-cache="number"]

[optimistic-column="columnName"]

[order-database-operations="True/False"]

[persistence-type="CMP/BMP"]

[prim-key-class-nogen="True/False"]

[read-timeout-seconds="Seconds"]

[reentrant="True/False"]

[run-as="RoleName"]

[run-as-identity-principal="PrincipalName"]

[table-name="TableName"]

[trans-timeout-seconds="Seconds"]

[use-caller-identity="True/False"]

[use-select-for-update="True/False"]

[validate-db-schema-with="MetaData/TableQuery"]

[verify-columns="VerificationOption"]

[verify-rows="Read/Modified"]

## Attributes

ejb-name

## WebLogic Workshop Annotations Reference

Required. Specifies the (descriptive) name of the entity bean. When you create a new entity bean with the file name <name>Bean.ejb or <name>.ejb, the ejb-name will be <name> by default (but you can change this if desired). For more information, see [How Do I: Create an Enterprise JavaBean?](#)

prim-key-class

Required. Specifies the Java class of the primary key. When a compound primary key is used, this class will be auto-generated during the build process. When you use the ejbCreate dialog in Design view to create one or more ejbCreate methods, the primary-key-class will be automatically specified. For more information, see [How Do I: Add a Create Method to an Entity Bean?](#)

abstract-schema-name

Optional. Gives the abstract schema name for the CMP entity bean. When you create a new entity bean, by default the ejb-name and the abstract-schema-name are the same, but you can change this if desired. For more information on default naming, see the ejb-name attribute above. A unique abstract-schema-name is required when the entity bean definition contains query methods. For more information, see [Query Methods and EJB QL](#).

cache-between-transactions

Optional. Specifies whether to cache the persistent data of an entity bean across (between) transactions. When left unspecified, it defaults to false. Caching options are discussed in detail in [Accelerating Entity Bean Data Access](#).

check-exists-on-method

Optional. Valid values are true and false. When left unspecified, the default is true. When true, the EJB container checks the database to verify that the CMP entity bean persistent record still exists at the end of a business method invocation, provided that this method did not already access any of the bean instance's persistent data.

clients-on-same-server

Optional. Specifies whether all clients are co-located with the entity bean on the same server. Valid values are true and false. When this attribute is left unspecified, it defaults to false. If set to True, the server instance will not multicast JNDI announcements for the EJB when it is deployed, that is, it turns off JNDI replication so that the EJB's JNDI name is only bound to the local server, hence reducing the startup time for large clusters.

concurrency-strategy

Optional. Defines the concurrency strategy for the entity bean. Valid values are ReadOnly, Exclusive, Optimistic, and Database. When left unspecified, the default is Database. Concurrency options are discussed in detail in [Accelerating Entity Bean Data Access](#).

data-source-name

Optional. Specifies the data source that holds the database table(s) the entity bean uses.

database-type

@ejbgen:entity Annotation

## WebLogic Workshop Annotations Reference

Optional. Specifies the database type of the underlying Database Management System (DBMS). This attribute needs to be specified for WebLogic features that require database-specific implementations, such as eager loading of relationships (see Accelerating Entity Bean Data Access). Valid values are DB2, Informix, Oracle, SQLServer, Sybase, and PointBase.

db-is-shared

This feature is deprecated. Use cache-between-transactions instead.

default-dbms-tables-ddl

Optional. When you specify a file name for this attribute, the EJB container will write an exact copy of the table creation scripts it uses to enable the automatic table creation feature for the EJB to this file. For more information, see the create-tables attribute of @ejbgen:jar-settings Annotation.

default-transaction

Optional. Specifies the default transaction attribute to be applied to all methods that do not have a transaction attribute setting. Valid values are NotSupported, Supports, Required, RequiresNew, Mandatory, and Never. When the default-transaction is left unspecified, it defaults to Supports. When the transaction-type is set to Bean, the default-transaction attributed is ignored. For more information, see EJBs and Transactions.

delay-database-insert-until

Optional. The delay-database-insert-until attribute specifies the precise time when a new CMP bean is inserted into the database. Valid values are ejbCreate and ejbPostCreate. This attribute interacts with the order-database-operations attribute (see the Remarks below).

delay-updates-until-end-of-tx

Optional. Specifies whether updates will be executed at commit time at the end of a transaction. When left unspecified, this attribute defaults to True. This attribute interacts with the order-database-operations attribute (see the Remarks below).

disable-warning

Optional. Provide a comma-separated list of warnings that will be suppressed during deployment of this entity bean. The following warnings can be suppressed (for more information on deployment errors and warning messages, see the WebLogic Server documentation topic EJB Subsystem Messages). To suppress the warning, specify the warning message code in this attribute:

- ◆ BEA-010054. The <ejbName> has a class <className> that is in the classpath. This class should only be located in the ejb-jar file.
- ◆ BEA-010202. Call-by-reference is not enabled for the EJB <ejbName>. The server will have better performance if it is enabled. To enable call-by-reference, set the enable-call-by-reference property, located in the *Tuning* section of the *Property Editor*, to True.
- ◆ BEA-010001. While deploying EJB <ejbName>, class <className> was loaded from the system classpath. As a result, this class cannot be reloaded while the server is running. To prevent this behavior in the future, make sure the class is not located in the server classpath.



## WebLogic Workshop Annotations Reference

- ◆ BEA-010200. The EJB Module <moduleName> has a class <className> that contains a member. To update this class or any of its superclasses, you must redeploy the entire EJB Module. You cannot update these classes using the dynamic EJB impl class update feature.

**Note.** To disable deployment warnings for all EJBs in the project, use the `disable-warning` attribute of `ejbgen:jar-settings`.

### `dispatch-policy`

Optional. Specifies the dispatch policy queue for this bean. If this attribute is not specified, the server's default execute thread pool is used.

### `enable-batch-operations`

Optional. This attribute specifies whether to allow the EJB container to perform batch operations, including batch inserts, batch updates and batch deletes. When the attribute is not specified, it defaults to `True`. This attribute interacts with the `order-database-operations` attribute (see the Remarks below).

### `enable-call-by-reference`

Optional. Specifies whether the container will call this EJB by reference. When left unspecified this attribute defaults to `False`. Setting this attribute to `True` will improve performance, but this can only be done under certain circumstances. For more information, see [Accelerating Entity Bean Data Access](#).

### `enable-dynamic-queries`

Optional. Specifies whether dynamic queries are enabled. When left unspecified, the attribute defaults to `False`. For more information, see [Query Methods and EJB QL](#).

### `finders-load-bean`

Optional. Specifies whether the bean's method will indeed honor `field-group` and `relationship-caching` settings. When left unspecified, this attribute defaults to `true`. When set to `false`, query methods will just load the primary key values of the beans returned by the query. For more information about eager loading for query methods and relationships, see [Accelerating Entity Bean Data Access](#).

### `home-call-router-class-name`

Optional. Specifies the name of a custom class to use for routing bean method calls. This class must implement `weblogic.rmi.cluster`.

### `home-is-clusterable`

Optional. Specifies whether this bean can be deployed from multiple servers in a cluster.

### `home-load-algorithm`

Optional. Specifies the algorithm to use for load-balancing between replicas of the EJB home. Valid values are `RoundRobin`, `Random`, `WeightBased`, `RoundRobinAffinity`, `RandomAffinity`, and `WeightBasedAffinity`.

### idle-timeout-seconds

Optional. Specifies the maximum duration in seconds that an EJB will stay in the cache before it is removed. When left unspecified, there is no timeout. For more information, see [Accelerating Entity Bean Data Access](#).

### invalidation-target

Optional. Specifies the ejb-name of a read-only entity bean to be invalidated when the current CMP bean has been modified. For more information, see [Accelerating Entity Bean Data Access](#).

### lock-order

Optional. Specifies the relative database locking order of this bean relative to other beans during a cascade delete operation. Valid values are 0...n. When left unspecified, the EJB container determines lock order. Typically you should only use this attribute if the lock order used by the EJB container results in a database deadlock.

### max-beans-in-cache

Optional. Specifies the maximum number of beans in the entity bean cache. When left unspecified, the default is 1000. If 0 is specified, there is no maximum. This attribute only applies to the entity bean's cache, and is ignored when an application-level cache is used instead for this bean. If this maximum has been reached, existing instances might be passivated to make room for new instances. For more information, see [Accelerating Entity Bean Data Access](#).

### optimistic-column

Optional. Specifies the column that holds the timestamp or version number for optimistic concurrency. For more information, see [Accelerating Entity Bean Data Access](#).

### order-database-operations

Optional. Specifies whether to sort the database operations – insert, update, delete – at commit time to take into account data dependencies. When left unspecified, this attribute defaults to True. This attribute interacts with various other attributes (see the Remarks below).

### persistence-type

Optional. Specifies the type of the entity bean. Valid values are CMP and BMP. When left unspecified, the default is CMP.

### prim-key-class-nogen

Optional. Set this attribute to True if you have defined the compound primary key class for this entity bean and don't want WebLogic to use a auto-generated primary key class. When not specified, this attribute defaults to False.

### read-timeout-seconds

Optional. Specifies the number of seconds before a read-only bean's cached data times out and needs to be refreshed. When left unspecified, the default is 600. To never time out, set this attribute to 0. For more

information, see Accelerating Entity Bean Data Access.

### reentrant

Optional. Specifies whether an entity bean is reentrant (can be used concurrently). If not specified, this attribute defaults to False.

### run-as

Optional. Specifies the EJB-scoped security role this bean should run under. To use this attribute, `use-caller-identity` must be False. Also make sure that the security role is defined using the `@ejbgen:role-mapping` Annotation. For more information, see Role-Based Security.

### run-as-identity-principal

Optional. Specifies the name of the principal to run as in case the security role specified with the `run-as` attribute maps to several principals. To specify this attribute you must also specify the `run-as` attribute. You must specify the `run-as-identity-principal` attribute when the `run-as` security role is externally defined. For more information, see Role-Based Security.

### table-name

Optional. Specifies the table to which this entity bean is mapped. By default the same name as the `ejb-name` is used. For more information on default naming, see the `ejb-name` attribute above.

### trans-timeout-seconds

Optional. Specifies the transaction timeout (in seconds). If not specified, this attribute defaults to 30 seconds.

### use-caller-identity

Optional. Specifies whether or not this entity bean runs using the caller's security identity. When not specified, this attribute defaults to True. You must set this attribute to False to use the `run-as` and `run-as-identity-principal` attributes. For more information, see Role-Based Security.

### use-select-for-update

Optional. When set to true, this attribute causes *SELECT ... FOR UPDATE* to be used when the bean is loaded from the database. This locks the bean's record(s) in the database for the duration of the transaction it is involved in.

### validate-db-schema-with

Optional. This attribute specifies the method used to validate the table(s) created by the EJB container for the CMP entity bean. Valid values are `MetaData` and `TableQuery`. If you specify `MetaData`, WebLogic uses the JDBC metadata to validate the schema. If you specify `TableQuery`, WebLogic queries the tables directly to verify that they match the definition of the CMP bean. When this attribute is left unspecified, `TableQuery` is used. Also see the section Automatic Table Creation of the `@ejbgen:jar-settings` Annotation.

### verify-columns

Optional. Specifies how optimistic concurrency verifies that the columns have or have not been modified during transactions. Valid values are Read, Modified, Version, and Timestamp. In order to use this attribute, the concurrency-strategy attribute needs to be set to Optimistic. For more information, see Accelerating Entity Bean Data Access.

verify-rows

Optional. Specifies how optimistic concurrency verifies that the rows have or have not been modified during transactions. Valid values are Read and Modified. In order to use this attribute, the concurrency-strategy attribute needs to be set to Optimistic. For more information, see Accelerating Entity Bean Data Access.

## Remarks

The following comments apply to this tag's use:

- By default, the order-database-operations and enable-batch-operations attributes are set to True. This means that database operations – inserts, updates, deletes – are not actually executed until commit time at the end of a transaction. Related database operations are grouped and executed in batch, typically resulting in performance enhancement. For more information, see Accelerating Entity Bean Data Access. When your database driver does not support batch operations, set the enable-batch-operations attribute to False.
- When the order-database-operations attribute is set to True, the delay-updates-until-end-of-tx and delay-database-insert-until attributes are ignored.
- When the order-database-operations attribute is set to False, insert and delete database operations are executed at the end of each method while update operations are performed during commit time. To execute database updates at the end of each method, set the delay-updates-until-end-of-tx attribute to False.
- When the order-database-operations attribute is set to False, you can delay database inserts until ejbPostCreate by specifying this value in the delay-database-insert-until attribute. If you set a relationship field in the ejbPostCreate method, the delay-database-insert-until attribute should be set to ejbCreate. For more information, see Entity Relationships.

### Related Topics

[@ejbgen:file-generation Annotation](#)

[@ejbgen:cmr-field Annotation](#)

[@ejbgen:cmp-field Annotation](#)

[@ejbgen:jar-settings Annotation](#)

[Getting Started with Entity Beans](#)

[Accelerating Entity Bean Data Access](#)

[Role-Based Security](#)

# @ejbgen:env-entry Annotation

This tag specifies environment entries for an EJB.

*Note.* An environment entry is a read-only EJB-specific variable that can be changed at deployment time. It allows you to modify the business logic of the EJB without having to change and recompile your code. For example, you can use an environment entry to set a maximum credit amount for a credit card, specify a company name, and so forth. An example is given below. For more general information on environment entries, see your favorite J2EE documentation.

## Scope

A class tag on an EJB.

## Syntax

**@ejbgen:env-entry**

[id="TagID"]

name="EnvEntryName"

type="TypeOption"

value="Value"

## Attributes

id

Optional. Specifies the ID of the tag. For more information, see EJBGen Tag Inheritance.

name

Required. Specifies the name of the environment entry.

type

Required. Specifies the type of the environment entry. The type can be String or any of the primitive wrappers, including Character, Short, Integer, Long, Double, Float, Byte, and Boolean.

value

Required. Specifies the value for this environment entry.

## Example

In this example an Order bean has an addProduct method to add a certain number of items to a shopping cart.

## WebLogic Workshop Annotations Reference

This method uses a environment entry to limit the quantity that can be ordered:

```
/**
 *
 * @ejbgen:env-entry value="5" type="java.lang.Integer" name="maxQuantity"
 *
 * ...
 */
public class OrderBean ...
{
    ...
    public void addProduct(ProductInfo theProduct, int requestedQuantity)
    {
        int maximumQuantity;

        try {
            InitialContext ic = new InitialContext();
            Integer maxQuantity = (Integer) ic.lookup("java:comp/env/maxQuantity");
            maximumQuantity = maxQuantity.intValue();
        }
        catch (NamingException ne) {
            ...
        }

        if(requestedQuantity > maximumQuantity) {
            // don't allow purchase
            ...
        }
        else
        {
            // allow purchase
            ...
        }
        ...
    }
}
```

### Related Topics

EJBGen Tag Inheritance

# @ejbgen:file-generation Annotation

This tag specifies the interface, compound primary key, and value classes that are to be auto-generated during build.

## Scope

Class tag on an entity or session bean.

## Syntax

@ejbgen:file-generation

[local-class="True/False"]

[local-class-name="NameLocalInterface"]

[local-home="True/False"]

[local-home-name="NameLocalHomeInterface"]

[local-home-package="NameLocalHomeInterfacePackage"]

[local-package="NameLocalInterfacePackage"]

[pk-class="True/False"]

[remote-class="True/False"]

[remote-class-name="NameRemoteInterface"]

[remote-home="True/False"]

[remote-home-name="NameRemoteHomeInterface"]

[remote-home-package="NameRemoteHomeInterfacePackage"]

[remote-package="NameRemoteInterfacePackage"]

[value-class="True/False"]

[value-class-name="NameValueClass"]

## Attributes

local-class

Optional. Specifies whether to generate the local interface of the EJB. Valid values are True and False. When left unspecified, it defaults to False. You can set this attribute and local-home together by using the checkbox

field **Local EJB** in the *Naming Section* of the *Property Editor*.

local-class-name

Optional. Specifies the name of the local interface. You can set this attribute either directly in Source View or by specifying the Local EJB's **Bean class name** field in the *Naming Section* of the *Property Editor*.

local-home

Optional. Specifies whether to generate the local home interface of the EJB. Valid values are True and False. When left unspecified, it defaults to False. You can set this attribute and local-class together by using the checkbox field **Local EJB** in the *Naming Section* of the *Property Editor*.

local-home-name

Optional. Specifies the name of the local home interface. You can set this attribute either directly in Source View or by specifying the Local EJB's **Home class name** field in the *Naming Section* of the *Property Editor*.

local-home-package

Optional. Specifies the package for the local home interface. When left unspecified, the same package is used as defined in the bean's .ejb file.

local-package

Optional. Specifies the package for the local interface. When left unspecified, the same package is used as defined in the bean's .ejb file.

pk-class

Optional. For an entity bean, specifies whether to generate the primary key class. Valid values are True and False. When left unspecified, it defaults to True. The primary key class name is stored in the prim-key-class attribute of the ejbgen:entity tag.

remote-class

Optional. Specifies whether to generate the remote interface of the EJB. Valid values are True and False. When left unspecified, it defaults to False. You can set this attribute and remote-home together by using the checkbox field **Remote EJB** in the *Naming Section* of the *Property Editor*.

remote-class-name

Optional. Specifies the name of the remote interface. You can set this attribute either directly in Source View or by specifying the Remote EJB's **Bean class name** field in the *Naming Section* of the *Property Editor*.

remote-home

Optional. Specifies whether to generate the remote home interface of the EJB. Valid values are True and False. When left unspecified, it defaults to False. You can set this attribute and remote-class together by using the checkbox field **Remote EJB** in the *Naming Section* of the *Property Editor*.



remote-home-name

Optional. Specifies the name of the remote home interface. You can set this attribute either directly in Source View or by specifying the Remote EJB's *Home class name* field in the *Naming Section* of the *Property Editor*.

remote-home-package

Optional. Specifies the package for the remote home interface. When left unspecified, the same package is used as defined in the bean's .ejb file.

remote-package

Optional. Specifies the package for the remote interface. When left unspecified, the same package is used as defined in the bean's .ejb file.

value-class

Optional. For an entity bean, specifies whether to auto-generate the value class. Valid values are true and false. When left unspecified, it defaults to true. Auto-generated value object classes

value-class-name

Optional. For an entity bean, specifies the name of the value class.

## Remarks

The following rules apply to this tag's use:

- When you create a new entity bean with the file name <name>Bean.ejb or <name>.ejb, the following interface and JNDI name related attributes are set by default:

```
* @ejbgen:jndi-name local = "ejb.<name>LocalHome"
*
* @ejbgen:file-generation local-class = "True" local-class-name = "<name>"
* local-home = "True" local-home-name = "<name>Home"
* remote-class = "False" remote-home = "False"
* remote-home-name = "<name>RemoteHome"
* remote-class-name = "<name>Remote"
* value-class = "False"
* value-class-name = "<name>Value"
* pk-class = "True"
```

Notice that for entity beans, only the local interfaces are auto-generated by default. Also, new CMP fields, CMR fields, component methods, home methods, and finder methods (except for findByPrimaryKey) are by default created in the local interfaces. These default settings encourage the use of local interfaces only to interact with entity beans.

**Note.** EJB Create methods and the findByPrimaryKey method are always defined in the home interfaces that are auto-generated during build. For instance, if both the local and remote home interface is auto-generated, these methods will be exposed through both interfaces.

## WebLogic Workshop Annotations Reference

- When you create a new session bean with the file name <name>Bean.ejb or <name>.ejb, the following interface and JNDI name related attributes are set by default:

```
* @ejbgen:jndi-name remote = "ejb.<name>RemoteHome"
*
* @ejbgen:file-generation remote-class = "true" remote-class-name = "<name>Remote"
* remote-home = "true" remote-home-name = "<name>Home"
* local-class = "false" local-class-name = "<name>Local"
* local-home = "false" local-home-name = "<name>LocalHome"
```

Notice that for session beans, only the remote interfaces are auto-generated by default. Also, component methods are by default created in the remote interface only. These default settings encourage the use of session beans as the intermediary between a client application and entity beans.

- Value objects, also known as data transfer objects, are plain (non-EJB) Java classes that are used to transport persistent data across the network in bulk, and are typically discarded once data has been read. These are not to be confused with *dependent* value objects whose life cycle is entirely managed by a CMP entity bean, that is these are actively invoked in the definition of the entity bean. For more information about using an auto-generated value class, see the Value Object Sample.

### Related Topics

@ejbgen:jndi-name Annotation

@ejbgen:entity Annotation

@ejbgen:local-method Annotation

@ejbgen:local-home-method Annotation

@ejbgen:remote-method Annotation

@ejbgen:remote-home-method Annotation

How Do I: Create an Enterprise JavaBean?

# @ejbgen:finder Annotation

This tag specifies a finder method for a CMP entity bean. This tag is not used for the `findByPrimaryKey` method; this method is automatically created by WebLogic during build. For more information, see the *Remarks* section of the `@ejbgen:file-generation` Annotation.

## Scope

A class tag on a CMP entity bean.

## Syntax

`@ejbgen:finder`

`[caching-name="CacheName"]`

`[comment="CommentText"]`

`ejb-ql="QueryLanguage"`

`[generate-on="Local/Remote"]`

`[group-name="GroupName"]`

`[id="TagID"]`

`[include-updates="True/False"]`

`[isolation-level="IsolationLevelOption"]`

`[max-elements="MaxNumber"]`

`[signature="ClassSignature"]`

DEPRECATED `[sql-select-distinct="True/False"]`

`[transaction-attribute="TransactionOption"]`

`[weblogic-ejb-ql="Query"]`

## Attributes

`caching-name`

Optional. Specifies the name of an eager relationship caching definition. For more details, see the `ejbgen:relationship-caching-element` Annotation and Accelerating Entity Bean Data Access.

`comment`

## WebLogic Workshop Annotations Reference

Optional. Specifies a comment that will be added to the auto-generated finder Java method in the home interface(s).

`ejb-ql`

Required. Specifies the EJB QL statement for this query.

`generate-on`

Optional. Specifies in which home interface this finder method will be defined. Valid values are Local and Remote. If this attribute is not specified, the finder method will be auto-generated on both. For more information, see the Remarks below.

`group-name`

Optional. This attribute defines which group-name will be eagerly loaded when returning the results of the query. For more details, see Accelerating Entity Bean Data Access.

`id`

Optional. Specifies the ID of the tag. For more information, see EJBGen Tag Inheritance.

`include-updates`

Optional. Specifies whether updates made during the current transaction must be reflected in the result of a query. Valid values are True and False. When this attribute is left unspecified, it defaults to true. For more details, see Accelerating Entity Bean Data Access.

`isolation-level`

Optional. Specifies the transaction isolation level for this method. Valid values are TransactionSerializable, TransactionReadCommitted, TransactionReadUncommitted, and TransactionRepeatableRead. If not specified, the isolation level specified by an `ejbgen:method-isolation-level-pattern` Annotation might apply. Otherwise, the default isolation level of the underlying database is used.

`max-elements`

Optional. Specifies the maximum number of elements that should be returned by a query. When left unspecified, no maximum is set.

`signature`

Optional. Specifies the signature of the finder method as it will be defined in the home interface. Exceptions that might be thrown by this method should not be specified, as they are added automatically during build. Any arguments should be fully qualified (such as `java.lang.String`).

`sql-select-distinct`

This attribute is deprecated. Please use the `DISTINCT` keyword in the `EJB-QL` instead.

`transaction-attribute`

`@ejbgen:finder` Annotation

Optional. Specifies the transaction attribute for this local method. Valid values are NotSupported, Supports, Required, RequiresNew, Mandatory, and Never. If not specified, the default transaction of the session or entity bean will be used.

`weblogic-ejb-ql`

Optional. Specifies the WebLogic QL statement for this query.

## Remarks

The following rules apply to this tag's use:

- The `ejbgen:finder` tag defines the entire finder method. In other words, unlike the `ejbgen:select` tag, the tag is not added to a method definition. To verify this, select a finder method in Design View and notice that in Source View the finder method *equals* the `ejbgen:finder` tag.
- When building an EJB, the `ejbgen:file-generation` tag specifies whether the local and/or remote business interface will be generated. If so, the finder method will be defined in this interface if the `auto-generate` attribute specifies this.
- Use either the `ejb-ql` or the `weblogic-ejb-ql` for the query executed for this finder method. When you specify queries for both attributes, the query specify for the `ejb-ql` attribute will be ignored.

### Related Topics

[@ejbgen:file-generation Annotation](#)

[How Do I: Add a Finder Method to an Entity Bean?](#)

[Finder Methods Sample](#)

[Query Methods and EJB-QL](#)

[@ejbgen:select Annotation](#)

# @ejbgen:foreign-jms-provider Annotation

This tag specifies a non-BEA JMS provider for a message-driven bean.

## Scope

Class tag on a message-driven bean.

## Syntax

**@ejbgen:foreign-jms-provider**

[connection-factory-jndi-name="FactoryJNDIName"]

[initial-context-factory="InitialContext"]

provider-url="URL"

## Attributes

connection-factory-jndi-name

Optional. Specifies the connection factory JNDI name, that is, the name of the connection factory object stored in JNDI.

initial-context-factory

Optional. Specifies the initial JNDI context factory, that is, the name of the class to perform lookup.

provider-url

Required. Specifies the provider URL, that is, the location of the directory information needed by the initial context factory.

## Remarks

Use this tag to specify the foreign JMS provider, and use the @ejbgen:message-driven Annotation to specify the destination JNDI name. For detailed examples of using various foreign JMS providers with message-driven beans, see the white paper Using Foreign JMS Providers with WebLogic Server.

Related Topics

[@ejbgen:message-driven Annotation](#)

[Getting Started with Message-Driven Beans](#)

# @ejbgen:jar-settings Annotation

In an EJB project, specifies JAR and table management settings. Typically the attributes on this tag are set in the *JAR Settings* and *Deployment* sections of the *Property Editor*. The various settings are stored in the build.properties file. Some other settings stored in this file are discussed below.

## Scope

The EJB Project.

## Syntax

@ejbgen:jar-settings

[create-tables="CreateTableOption"]

[disable-warning="IgnoreWarningsList"]

[ejb-client-jar="ClientJarList"]

[enable-bean-class-redeploy="True/False"]

## Attributes

create-tables

Optional. Set this attribute to enable automatic table creation for CMP entity beans. For details, see Automatic Table Creation below.

disable-warning

Optional. Provide a comma-separated list of warnings that will be suppressed during EJB deployment. The following warnings can be suppressed (for more information on deployment errors and warning messages, see the WebLogic Server documentation topic EJB Subsystem Messages). To suppress the warning, specify the warning message code in this attribute:

- ◆ BEA-010054. The <ejbName> has a class <className> that is in the classpath. This class should only be located in the ejb-jar file.
- ◆ BEA-010202. Call-by-reference is not enabled for the EJB <ejbName>. The server will have better performance if it is enabled. To enable call-by-reference, set the enable-call-by-reference property, located in the *Tuning* section of the *Property Editor*, to True.
- ◆ BEA-010001. While deploying EJB <ejbName>, class <className> was loaded from the system classpath. As a result, this class cannot be reloaded while the server is running. To prevent this behavior in the future, make sure the class is not located in the server classpath.
- ◆ BEA-010200. The EJB Module <moduleName> has a class <className> that contains a member. To update this class or any of its superclasses, you must redeploy the entire EJB Module. You cannot update these classes using the dynamic EJB impl

class update feature.

`ejb-client-jar`

Optional. Specify the name of the client JAR to be generated. An EJB client JAR contains the EJB's home and remote interface classes and any other classes necessary for a client program to access the EJB.

`enable-bean-class-redeploy`

Optional. Specifies whether the EJBs in the EJB project (that is, the EJB JAR file) can be redeployed without redeploying the entire application (that is, the EAR file). When left unspecified, this settings defaults to False.

## Automatic Table Creation

When you are developing a new entity bean, you can make iterative development easier by enabling automatic table creation. Any changes a database table are made immediately after a build when the EJBs are redeployed. The following options are available for the `create-tables` attribute:

- **Unspecified.** When you don't specify the attribute, the EJB container does not create or alter tables.
- **Disabled.** The EJB container does not create or alter tables.
- **CreateOnly.** For each CMP bean in the project, if there are no table(s) in the database to persist the bean data, the EJB container attempts to create the table(s) based on the definition in the EJB file. If necessary, join tables, and sequences tables required for automatic primary key generation are also created. If an EJB stores CMP fields in multiple tables all tables are created.
- **DropAndCreate.** For each CMP bean in the project, if the changes to the definition of the EJB do not require changes to the table definition, no action is taken, and the existing data is preserved. Otherwise, the table is deleted and recreated. Any data in the deleted table will be lost.
- **DropAndCreateAlways.** For each CMP bean in the project, the associated tables are dropped and recreated whenever the beans are redeployed.
- **AlterOrCreate.** For each CMP bean in the project, if the associated tables do not exist, this setting behaves as **CreateOnly**. If the table exists, the EJB container attempts to alter the table schema using the "ALTER TABLE" SQL command, and saves the data present in the table.

**Note.** Do not use the **AlterOrCreate** setting when you have changed the definition of a primary key. For instance, if you added a primary key CMP field and use this setting, the updated table will not accurately represent the added column as a primary key. You must **DropAndCreate** the table instead.

## Build.Properties File

The `build.properties` file stores additional build settings in addition to the attributes of the `@ejbgen:jar-settings` annotation. Some of these settings are:

- **ejb.tmpdir.** Use this attribute to set the folder name of the temporary directory used to save files generated during the build process. An example of use is `ejb.tmpdir=${java.io.tmpdir}/my_tmp_EJBs`.
- **ejbgen.source.** By default, WebLogic Workshop 8.1 uses JDK1.4 to check and build Java code. If you want to use JDK1.3 to build the EJBs of a project, add `ejbgen.source=1.3` to the project's `build.properties` file. Note that this will cause the code to be build with JDK1.3, although the realtime code checking done during development in the IDE will still be using JDK1.4. In other words, you might see red and green squiggles in your code that indicate errors and warnings relative to JDK1.4,



## WebLogic Workshop Annotations Reference

although your code is valid relative to JDK1.3 and will build without error.

### Related Topics

[Getting Started with Entity Beans](#)

[How Do I: Add an Existing Enterprise JavaBean to an Application?](#)

# @ejbgen:jndi-name Annotation

This tag specifies the local and remote JNDI names of an EJB, that is, the JNDI names associated with the local and remote interfaces of an EJB. You can set this tag's attributes either directly in Source View or in the *Naming Section* of the *Property Editor*.

## Scope

Class tag on an entity or session bean.

## Syntax

@ejbgen:jndi-name

[local="LocalJNDIName"]

[remote="RemoteJNDIName"]

## Attributes

local

Optional. Specifies the local JNDI name of the EJB.

remote

Optional. Specifies the remote JNDI name of the EJB.

## Remarks

The local or remote JNDI name of an EJB is typically used to reference the local or remote interface of an EJB, for instance by an EJB Control, or by other EJBs using a (remote) @ejbgen:ejb-ref or (local) @ejbgen:ejb-local-ref tag, although it is possible in the latter case to reference an EJB without using its JNDI name. For more information, see the description of these two ejbgen tags.

When you create a new entity or session bean, JNDI names and related EJB interfaces are set by default. For more default, see @ejbgen:file-generation.

Related Topics

@ejbgen:file-generation Annotation

@ejbgen:ejb-ref Annotation

@ejbgen:ejb-local-ref Annotation

# @ejbgen:local-home-method Annotation

This tag defines the home methods for an entity bean's local home interface.

## Scope

Method tag on an entity bean's home method.

## Syntax

**@ejbgen:local-home-method**

[is-idempotent="True/False"]

[ordering-number="Number"]

[roles="RolesList"]

[transaction-attribute="TransactionOption"]

## Attributes

is-idempotent

Optional. Specifies whether this method is idempotent. Valid values are True and False. An idempotent method does not alter the state of the system, that is, produces the same results on subsequent invocations if the same arguments are provided.

ordering-number

Optional. Specify a number to enforce in what order this home method will be defined, relative to the other home methods, in the auto-generated local home interface. Valid values are 0...n. In order for this ordering to work, all local-home-method tags used in this EJB must have this attribute set with a distinct numeric value. If the ordering-number attribute is not used, the home methods will be defined in alphabetical order. The definition of the home methods always follows the definition of the findByPrimaryKey and create methods. For more information on interface generation, see `ejbgen:file-generation` Annotation.

roles

Optional. Defines a comma-separated list of EJB-scoped security roles that are allowed to invoke this method. If not specified, the roles specified by an `ejbgen:method-permission-pattern` Annotation might apply. These EJB-scoped security roles must be defined using the `@ejbgen:role-mapping` Annotation. For more information, see Role-Based Security.

transaction-attribute

Optional. Specifies the transaction attribute for this local home method. Valid values are NotSupported, Supports, Required, RequiresNew, Mandatory, and Never. If not specified, the default transaction of the entity bean will be used. For more information, see EJBs and Transactions.

## Remarks

The following rules apply to this tag's use:

- Home methods can be defined in the local and/or the remote home interfaces. In Design View you can easily create these definitions by right-clicking the diamond on the arrow to the left of the method name, and selecting the desired interface option.
- When building an EJB, the `ejbgen:file-generation` tag specifies whether the local home business interface will be generated. If so, the home methods with an `ejbgen:local-home-method` tag will be defined in this interface.

### Related Topics

[@ejbgen:file-generation Annotation](#)

[@ejbgen:local-method Annotation](#)

[@ejbgen:remote-home-method Annotation](#)

[How Do I: Add a Home Method to an Entity Bean?](#)

[Home Methods Sample](#)

# @ejbgen:local-method Annotation

This tag defines the methods for an entity or session bean's local (business) interface.

## Scope

Method tag on an entity bean's CMP accessor methods, CMR accessor methods, and component methods;  
Method tag on a session bean's component methods.

## Syntax

**@ejbgen:local-method**

[is-idempotent="True/False"]

[isolation-level="IsolationLevelType"]

[ordering-number="Number"]

[roles="RolesList"]

[transaction-attribute="TransactionOption"]

## Attributes

is-idempotent

Optional. Specifies whether this method is idempotent. Valid values are True and False. An idempotent method does not alter the state of the system, that is, produces the same results on subsequent invocations if the same arguments are provided.

isolation-level

Optional. Specifies the transaction isolation level for this method. Valid values are TransactionSerializable, TransactionReadCommitted, TransactionReadUncommitted, and TransactionRepeatableRead. If you are using an Oracle database, you can also specify TransactionReadCommittedForUpdate or TransactionReadCommittedForUpdateNoWait. If not specified, the isolation level specified by an `ejbgen:method-isolation-level-pattern` Annotation might apply. Otherwise, the default isolation level of the underlying database is used. For more information, see EJBs and Transactions.

ordering-number

Optional. Specify a number to enforce in what order this method will be defined, relative to the other methods, in the auto-generated local interface. Valid values are 0...n. In order for this ordering to work, all `local-method` tags used in this EJB must have this attribute set with a distinct numeric value. If the `ordering-number` attribute is not used, the methods will be defined in alphabetical order. For more information on interface generation, see `ejbgen:file-generation` Annotation.

roles

Optional. Defines a comma-separated list of EJB-scoped security roles that are allowed to invoke this method. If not specified, the roles specified by an `ejbgen:method-permission-pattern` Annotation might apply. These EJB-scoped security roles must be defined using the `@ejbgen:role-mapping` Annotation. For more information, see [Role-Based Security](#).

`transaction-attribute`

Optional. Specifies the transaction attribute for this local method. Valid values are `NotSupported`, `Supports`, `Required`, `RequiresNew`, `Mandatory`, and `Never`. If not specified, the default transaction of the session or entity bean will be used. For more information, see [EJBs and Transactions](#).

## Remarks

The following rules apply to this tag's use:

- Component methods and, for entity beans, CMP and CMR accessor methods can be defined in the local and/or the remote interfaces. In Design View you can easily create these definitions by right-clicking the diamonds in front of the field/method names and selecting the desired interface option.
- Finder methods are not defined in the local interface with this tag, but with the `generate-on` attribute on the `ejbgen:finder` tag.
- When building an EJB, the `ejbgen:file-generation` tag specifies whether the local business interface will be generated. If so, the methods with an `ejbgen:local-method` tag will be defined in this interface.

### Related Topics

[@ejbgen:file-generation Annotation](#)

[@ejbgen:remote-method Annotation](#)

[@ejbgen:local-home-method Annotation](#)

[How Do I: Define a Container-Managed Persistence \(CMP\) Field?](#)

[How Do I: Add a Component Method to an Entity or Session Bean?](#)

[How Do I: Add a Relation to an Entity Bean?](#)

# @ejbgen:message-driven Annotation

This tag defines the class-scope properties of a message-driven bean.

## Scope

Class tag on a message-driven bean.

## Syntax

**@ejbgen:message-driven**

destination-jndi-name="JNDIName"

destination-type="javax.jms.Queue/javax.jms.Topic"

ejb-name="NameMessageDrivenBean"

[acknowledge-mode="auto-acknowledge/dups-ok-acknowledge"]

[clients-on-same-server="True/False"]

[create-as-principal-name="PrincipalName"]

[default-transaction="DefaultTransactionOption"]

[durable="True/False"]

[initial-beans-in-free-pool="InitialNumber"]

[max-beans-in-free-pool="MaxNumber"]

[message-selector="JMSSelector"]

[passivate-as-principal-name="PrincipalName"]

[remove-as-principal-name="PrincipalName"]

[run-as="RoleName"]

[run-as-identity-principal="PrincipalName"]

[trans-timeout-seconds="Seconds"]

[transaction-type="Bean/Container"]

[use-caller-identity="True/False"]

## Attributes

### destination-jndi-name

Required. Specifies the JNDI name of the destination from which the message-driven bean receives messages.

### destination-type

Required. Specifies the destination type from which the message-driven bean receives messages. Valid values are `javax.jms.Queue` and `javax.jms.Topic`.

### ejb-name

Required. Specifies the (descriptive) name of the message-driven bean. When you create a new message-driven bean with the file name `<name>Bean.ejb` or `<name>.ejb`, the `ejb-name` will be `<name>` by default.

### acknowledge-mode

Optional. Specifies the acknowledgement mode of the message-driven bean. Valid values are `auto-acknowledge` and `dups-ok-acknowledge`. When not specified, the default is `auto-acknowledge`. The `auto-acknowledge` setting tells the EJB container that an acknowledgement must be sent to the JMS provider when an instance of this message-driven bean has received the message. The `dups-ok-acknowledge` setting allows for a delay of acknowledgement. The latter setting should only be used when duplicate messages can be processed correctly.

The `acknowledge-mode` attribute setting is important when the `transaction-type` attribute is set to `Bean` (reflecting bean-managed transactions), or when the `transaction-type` is set to `Container` and the `default-transaction` attribute is set to `NotSupported`. In all other cases this setting is ignored.

### clients-on-same-server

Optional. Specifies whether all clients are co-located with the session bean on the same server. Valid values are `true` and `false`. When this attribute is left unspecified, it defaults to `false`. If set to `True`, the server instance will not multicast JNDI announcements for the EJB when it is deployed, that is, it turns off JNDI replication so that the EJB's JNDI name is only bound to the local server, hence reducing the startup time for large clusters.

### create-as-principal-name

Optional. Specifies the name of the principal to be used when running the bean's `ejbCreate` method. For more information, see [Role-Based Security](#).

### default-transaction

Optional. Specifies the default transaction attribute to be applied to all methods that do not have a transaction attribute setting. Valid values are `NotSupported` and `Required`. When the attribute is left unspecified, it defaults to `NotSupported`, assuming container-managed transaction. When the `transaction-type` is set to `Bean`, the `default-transaction` attributed is ignored. When set to `Required` (assuming container-manager



## WebLogic Workshop Annotations Reference

transactions), the acknowledge-mode attribute is ignored. For more information on transactions and message-driven beans, see EJBs and Transactions.

### durable

Optional. Specifies whether the bean's subscription to a `javax.jms.Topic` (as specified by the `destination-type`) is durable, that is, can outlast the EJB container's connection to the JMS provider. Valid values are `true` and `false`. When left unspecified, the default is `true`. This attribute is not relevant for queues.

### initial-beans-in-free-pool

Optional. Specifies the initial number of beans in the free pool. If not specified, the default is computed using the formula  $\min(\text{number of threads in the execute queue} * .5) + 1, \text{max-beans-in-free-pool}$ . To learn more about execute queues, see the WebLogic Server documentation at <http://edocs.bea.com>.

### max-beans-in-free-pool

Optional. Specifies the maximum number of beans in the free pool. When left unspecified, the default is 1000. If 0 is specified, there is no maximum.

### message-selector

Optional. Specifies a message selector for this message-driven bean to be more selective about the messages it receives. For more information, see the Message-Driven Bean Sample.

### passivate-as-principal-name

Optional. Specifies the name of the principal to be used when running the bean's `ejbPassivate` method.

### remove-as-principal-name

Optional. Specifies the name of the principal to be used when running the bean's `ejbRemove` method.

### run-as

Optional. Specifies the EJB-scoped security role this bean should run under. To use this attribute, `use-caller-identity` must be `False`. Also make sure that the security role is defined using the `@ejbgen:role-mapping` Annotation. For more information, see Role-Based Security.

### run-as-identity-principal

Optional. Specifies the name of the principal to run as in case the security role specified with the `run-as` attribute maps to several principals. To specify this attribute you must also specify the `run-as` attribute. You must specify the `run-as-identity-principal` attribute when the `run-as` security role is externally defined. For more information, see Role-Based Security.

### trans-timeout-seconds

Optional. Specifies the transaction timeout (in seconds). If not specified, this attribute defaults to 30 seconds.

### transaction-type

## @ejbgen:message-driven Annotation

## WebLogic Workshop Annotations Reference

Optional. Specifies who manages the transactions for this EJB. Default values are Bean and Container. When left unspecified, the default is Container, meaning that the message-driven bean executes within its own transaction context. The setting of this attribute affect the default-transaction and acknowledge-mode attributes. For more information, see above. For more information on transactions and message-driven beans, see EJBs and Transactions.

use-caller-identity

Optional. Specifies whether or not this EJB uses the caller's identity. When not specified, this attribute defaults to True. You must set this attribute to False to use the run-as and run-as-identity-principal attributes. For more information, see Role-Based Security.

Related Topics

Getting Started with Message-Driven Beans

Message-Driven Bean Sample

EJB Tutorial, Part Three: Adding a Message-Driven Bean

Role-Based Security

EJBs and Transactions

# @ejbgen:method-isolation-level-pattern Annotation

This tag specifies the transaction isolation level for methods matching a given pattern.

## Scope

Class tag on session and entity beans.

## Syntax

**@ejbgen:method-isolation-level-pattern**

[id="TagID"]

isolation-level="IsolationLevelType"

pattern="Pattern"

## Attributes

id

Optional. Specifies the ID of the tag. For more information, see EJBGen Tag Inheritance.

isolation-level

Required. Specifies the transaction isolation level for the methods matching this pattern, provided that the transaction isolation level for a method has not been explicitly set. Valid values are TransactionSerializable, TransactionReadCommitted, TransactionReadUncommitted, and TransactionRepeatableRead. If you are using an Oracle database, you can also specify TransactionReadCommittedForUpdate or TransactionReadCommittedForUpdateNoWait. For more information, see EJBs and Transactions.

pattern

Required. Specifies the method to which the specified isolation level applies. To make this applicable to all methods, use the '\*' wildcard. It is not possible to use a combination of wildcard and letters.

Related Topics

@ejbgen:finder Annotation

@ejbgen:local-method Annotation

@ejbgen:remote-method Annotation

# @ejbgen:method-permission-pattern Annotation

This specifies security roles authorized to invoke the methods matching a given pattern.

## Scope

Class tag on session and entity beans.

## Syntax

**@ejbgen:method-permission-pattern**

[id="TagID"]

[interface="InterfaceOption"]

pattern="Pattern"

roles="RolesList"

## Attributes

id

Optional. Specifies the ID of the tag. For more information, see EJBGen Tag Inheritance.

interface

Optional. Specifies the interface containing the methods that will be tested against the pattern. Valid values are Home, Remote, LocalHome, and Local. In other words, you can for instance specify that the pattern given in this attribute should only be matched against the methods defined in the local (business) interface. If not specified, all methods are considered.

pattern

Required. Specifies the method to which the specified roles applies. To make these applicable to all methods, use the '\*' wildcard. It is not possible to use a combination of wildcard and letters.

roles

Required. Defines a comma-separated list of EJB-scoped security roles that are allowed to invoke the methods matching the pattern and interface attributes. These EJB-scoped security roles must be defined using the @ejbgen:role-mapping Annotation. For more information, see Role-Based Security.

## Remarks

Security roles are not set on finder methods, and are not considered by the @ejbgen:method-permission-pattern tag during pattern matching.

### Related Topics

#### Role-Based Security

@ejbgen:local-home-method Annotation

@ejbgen:local-method Annotation

@ejbgen:remote-home-method Annotation

@ejbgen:remote-method Annotation

# @ejbgen:relation Annotation

This tag specifies an entity relationship between two CMP entity beans.

## Scope

Class tag on an entity bean.

## Syntax

**@ejbgen:relation**

[db-cascade-delete="True/False"]

[cascade-delete="True/False"]

[cmr-field="CMRFieldName"]

[fk-column="ForeignKeyColumnName"]

[foreign-key-table="ForeignKeyTableName"]

[id="TagID"]

[joint-table="JointTableName"]

multiplicity="One/Many"

name="RelationshipName"

[primary-key-table="PKTableName"]

[role-name="RoleName"]

[target-ejb="TargetEJBName"]

## Attributes

db-cascade-delete

Optional. Specifies whether a cascade delete will use the built-in cascade delete facilities of the underlying DBMS. Valid values are true and false. When left unspecified, the default is false. All of the major databases have a built-in cascade delete function. Please verify that the underlying database supports cascade delete before enabling this feature.

cascade-delete

Optional. Specifies whether deletion of the entity bean should trigger a cascade delete for this entity relation. Valid values are true and false. When left unspecified, the default is false.

### cmr-field

Optional. Specifies the CMR field where this relationship will be stored. This attribute is only specified for certain relationships. For more details, see Entity Relationships.

### fk-column

Optional. Specifies the column(s) in the database table for the entity bean that holds the primary key(s) values of the other EJB engaged in the entity relation. This attribute is only specified for certain relationships. For more details, see Entity Relationships.

### foreign-key-table

Optional. Specifies the name of the database table that contains the foreign key. This attribute only needs to be specified when this entity bean is mapped to multiple tables.

### id

Optional. Specifies the ID of the tag. For more information, see EJBGen Tag Inheritance.

### joint-table

Optional. Specifies the name of the joint table that is used to hold the primary keys values of the two entity beans involved in this entity relation. This attribute is only specified for Many-to-Many relationships. For more details, see Entity Relationships.

### multiplicity

Required. Specifies the multiplicity of the relationship. Valid values are One and Many.

### name

Required. Specifies the name of the relationship. The name is used to identify the relationship, so the same must be used by both entity beans in this relationship.

### primary-key-table

Optional. Specifies the name of the database table that contains the primary key. This attribute only needs to be specified when this entity bean is mapped to multiple tables.

### role-name

Optional. Specifies the name of this role.

### target-ejb

Optional. Specifies the name of the other EJB name in this relationship.

## Remarks

When you use the Add Relation Wizard to specify an entity relation, the correct ejbgen:relation tags and attributes are added to both EJBs in the entity relation, reducing development time and the likelihood of errors.

Related Topics

Entity Relationships

How Do I: Add a Relation to an Entity Bean?



# @ejbgen:relationship-caching-element Annotation

This tag specifies an eager relationship loading definition. This definition can be used by a query method to enable eager loading of a entity bean's related bean(s), as defined by an entity relationship.

## Scope

Class tag on a CMP entity bean.

## Syntax

**@ejbgen:relationship-caching-element**

caching-name="EagerLoadingDefinitionName"

cmr-field="CMRFieldList"

[group-name="FieldGroupList"]

[id="TagID"]

[parent-id="TagIDOfParent"]

## Attributes

caching-name

Required. Specifies the name of this eager relationship loading definition.

cmr-field

Required. Specifies a comma-separated list of one or more CMR field names holding the related entity beans that need to be loaded.

group-name

Optional. Specifies a comma-separated list of one or more field group names to be loaded for the entity beans specified in the cmr-field attribute.

id

Optional. Specifies the ID of this eager relationship loading definition.

parent-id

Optional. Specifies a parent for this eager relationship loading definition.

## Remarks

This tag specifies one set of CMR fields, and optionally field-groups, that can be used by this entity bean's finder or select method to load related beans as part of the query results. An entity bean can have multiple tags to specify different eager loading relationships as required. Also, eager relationship loading can be nested such that for an eagerly loaded bean, its entity relations are loaded as well. For more details, see [Accelerating Entity Bean Data Access](#).

Related Topics

[Accelerating Entity Bean Data Access](#)

[Entity Relationships](#)

[@ejbgen:select Annotation](#)

[@ejbgen:finder Annotation](#)

[@ejbgen:relation Annotation](#)

# @ejbgen:remote-home-method Annotation

This tag defines the entity methods for an entity bean's remote home interface.

## Scope

Method tag on an entity bean's home method.

## Syntax

**@ejbgen:remote-home-method**

[is-idempotent="True/False"]

[ordering-number="Number"]

[roles="RolesList"]

[transaction-attribute="TransactionOption"]

## Attributes

is-idempotent

Optional. Specifies whether this method is idempotent. Valid values are True and False. An idempotent method does not alter the state of the system, that is, produces the same results on subsequent invocations if the same arguments are provided.

ordering-number

Optional. Specify a number to enforce in what order this home method will be defined, relative to the other home methods, in the auto-generated remote home interface. Valid values are 0...n. In order for this ordering to work, all remote-home-method tags used in this EJB must have this attribute set with a distinct numeric value. If the ordering-number attribute is not used, the home methods will be defined in alphabetical order. The definition of the home methods always follows the definition of the findByPrimaryKey and create methods. For more information on interface generation, see `ejbgen:file-generation` Annotation.

roles

Optional. Defines a comma-separated list of EJB-scoped security roles that are allowed to invoke this method. If not specified, the roles specified by an `ejbgen:method-permission-pattern` Annotation might apply. These EJB-scoped security roles must be defined using the `@ejbgen:role-mapping` Annotation. For more information, see Role-Based Security.

transaction-attribute

Optional. Specifies the transaction attribute for this remote home method. Valid values are NotSupported, Supports, Required, RequiresNew, Mandatory, and Never. If not specified, the default transaction of the session or entity bean will be used. For more information, see EJBs and Transactions.

## Remarks

The following rules apply to this tag's use:

- Home methods can be defined in the local and/or the remote home interfaces. In Design View you can easily create these definitions by right-clicking the diamond on the arrow to the left of the method name, and selecting the desired interface option.
- When building an EJB, the `ejbgen:file-generation` tag specifies whether the remote home business interface will be generated. If so, the home methods with an `ejbgen:remote-home-method` tag will be defined in this interface.

### Related Topics

[@ejbgen:file-generation Annotation](#)

[@ejbgen:remote-method Annotation](#)

[@ejbgen:local-home-method Annotation](#)

[How Do I: Add a Home Method to an Entity Bean?](#)

[Home Methods Sample](#)

# @ejbgen:remote-method Annotation

This tag defines the methods for an entity or session bean's remote (business) interface.

## Scope

Method tag on an entity bean's CMP accessor methods, CMR accessor methods, and component methods;  
Method tag on a session bean's component methods.

## Syntax

@ejbgen:remote-method

[is-idempotent="True/False"]

[isolation-level="IsolationLevelType"]

[ordering-number="Number"]

[roles="RolesList"]

[transaction-attribute="TransactionOption"]

## Attributes

is-idempotent

Optional. Specifies whether this method is idempotent. Valid values are True and False. An idempotent method does not alter the state of the system, that is, produces the same results on subsequent invocations if the same arguments are provided.

isolation-level

Optional. Specifies the transaction isolation level for this method. Valid values are TransactionSerializable, TransactionReadCommitted, TransactionReadUncommitted, and TransactionRepeatableRead. If you are using an Oracle database, you can also specify TransactionReadCommittedForUpdate or TransactionReadCommittedForUpdateNoWait. If not specified, the isolation level specified by an @ejbgen:method-isolation-level-pattern Annotation might apply. Otherwise, the default isolation level of the underlying database is used. For more information, see EJBs and Transactions.

ordering-number

Optional. Specify a number to enforce in what order this method will be defined, relative to the other methods, in the auto-generated remote interface. Valid values are 0...n. In order for this ordering to work, all remote-method tags used in this EJB must have this attribute set with a distinct numeric value. If the ordering-number attribute is not used, the methods will be defined in alphabetical order. For more information on interface generation, see @ejbgen:file-generation Annotation.

roles

Optional. Defines a comma-separated list of EJB-scoped security roles that are allowed to invoke this method. If not specified, the roles specified by an `ejbgen:method-permission-pattern` Annotation might apply. These EJB-scoped security roles must be defined using the `@ejbgen:role-mapping` Annotation. For more information, see [Role-Based Security](#).

`transaction-attribute`

Optional. Specifies the transaction attribute for this remote method. Valid values are `NotSupported`, `Supports`, `Required`, `RequiresNew`, `Mandatory`, and `Never`. If not specified, the default transaction of the session or entity bean will be used. For more information, see [EJBs and Transactions](#).

## Remarks

The following rules apply to this tag's use:

- Component methods and, for entity beans, CMP and CMR accessor methods can be defined in the local and/or the remote interfaces. In Design View you can easily create these definitions by right-clicking the diamonds in front of the field/method names and selecting the desired interface option.
- Finder methods are not defined in the remote interface with this tag, but with the `generate-on` attribute on the `ejbgen:finder` tag.
- When building an EJB, the `ejbgen:file-generation` tag specifies whether the remote business interface will be generated. If so, the methods with an `ejbgen:remote-method` tag will be defined in this interface.

### Related Topics

[@ejbgen:file-generation Annotation](#)

[@ejbgen:local-method Annotation](#)

[@ejbgen:remote-home-method Annotation](#)

[How Do I: Define a Container-Managed Persistence \(CMP\) Field?](#)

[How Do I: Add a Component Method to an Entity or Session Bean?](#)

[How Do I: Add a Relation to an Entity Bean?](#)

# @ejbgen:resource-env-ref Annotation

This tag maps a JNDI reference used within a bean to an administered object associated with an external resource. Common examples of such connection factories are (JMS) `javax.jms.Queue` and (JMS) `javax.jms.Topic`.

## Scope

Class tag on all EJBs.

## Syntax

```
@ejbgen:resource-env-ref
```

```
[id="TagID"]
```

```
[jndi-name="JNDIName"]
```

```
name="ReferenceName"
```

```
type="ObjectType"
```

## Attributes

`id`

Optional. Specifies the ID of the tag. For more information, see EJBGen Tag Inheritance.

`jndi-name`

Optional. Specifies the JNDI Name of the administered object.

`name`

Required. Specifies the name of the reference to the administered object as used in the bean definition.

`type`

Required. Specifies the type of the resource. Examples are `javax.jms.Queue`, and `javax.jms.Topic`.

## Remarks

The `ejbgen:resource-env-ref` tag is used in conjunction with the `ejbgen:resource-ref` tag. The latter tag maps the external resource used while the `ejbgen:resource-env-ref` tag maps any administered object associated with that queue. For example, the `ejbgen:resource-ref` tag maps a reference to a `javax.jms.TopicConnectionFactory`, while the `ejbgen:resource-env-ref` tag maps to `javax.jms.Topic`.

For a code example, see the `@ejbgen:resource-ref` tag or Step 12 of Tutorial: Enterprise JavaBeans.

Related Topics

[@ejbgen:resource-ref Annotation](#)

[Tutorial: Enterprise JavaBeans](#)



# @ejbgen:resource-ref Annotation

This tag maps a JNDI reference used within a bean to an external resource connection factory. Common examples of such connection factories are (JDBC) `javax.sql.DataSource`, (JMS) `javax.jms.QueueConnectionFactory`, and (JMS) `javax.jms.TopicConnectionFactory`.

## Scope

Class tag on all EJBs.

## Syntax

`@ejbgen:resource-ref`

`auth="Application/Container"`

`[id="TagID"]`

`jndi-name="JNDIName"`

`name="ReferenceName"`

`[sharing-scope="Shareable/Unshareable"]`

`type="ResourceType"`

## Attributes

`auth`

Required. Specifies who is responsible for authentication to use the resource. Valid values are `Application` and `Container`.

`id`

Optional. Specifies the ID of the tag. For more information, see `EJBGen Tag Inheritance`.

`jndi-name`

Required. Specifies the JNDI Name of the referenced external resource.

`name`

Required. Specifies the name of the reference to the external resource as used in the bean definition.

`sharing-scope`

Optional. Specifies whether the resource connections can be shared by multiple EJBs in the same transaction or 'unit of work'. Valid values are `Shareable` and `Unshareable`. When left unspecified the default `Shareable` is

used.

type

Required. Specifies the type of the resource. Examples are `javax.sql.DataSource`, `javax.jms.QueueConnectionFactory`, and `javax.jms.TopicConnectionFactory`.

## Example

The following example shows the use of external resource and administered object references, and the mapping of these references using `ejbgen:resource-ref` and `ejbgen:resource-env-ref` tags in an EJB:

```
/**
 *
 * @ejbgen:resource-ref auth="Container"
 *   jndi-name = "weblogic.jws.jms.QueueConnectionFactory"
 *   name = "jms/QueueConnectionFactory"
 *   type="javax.jms.QueueConnectionFactory"
 *
 * @ejbgen:resource-env-ref
 *   jndi-name="MyQueue"
 *   name="jms/MyQueue"
 *   type = "javax.jms.Queue"
 *
 * ...
 */
public class SomeBean extends GenericSessionBean implements SessionBean
{
    private QueueConnectionFactory factory;
    private Queue queue;

    ...

    try {
        javax.naming.Context ic = new InitialContext();
        factory = (QueueConnectionFactory) ic.lookup("java:comp/env/jms/QueueConnectionFactory");
        queue = (Queue) ic.lookup("java:comp/env/jms/MyQueue");
    }
    catch (NamingException ne) {
        ...
    }
    ...
}
```

### Related Topics

[@ejbgen:resource-env-ref Annotation](#)

[Tutorial: Enterprise JavaBeans](#)

# @ejbgen:role-mapping Annotation

This tag defines an EJB-scoped security role used by the beans in an EJB Project. In addition, the tag defines the role-principal mapping or refers to an externally defined security role for the role-principal mapping definition. An externally defined role can be a security role with application-scope or it can be a global security role defined in the domain's security realm. For more information on security roles, see Role-Based Security.

## Scope

Class tag on session and entity beans.

## Syntax

@ejbgen:role-mapping

[global-role="True/False"]

[id="TagID"]

[principals="PrincipalsList"]

role-name="NameRole"

## Attributes

global-role

Optional. Specifies whether this security role is externally defined. Specify True if the role is externally defined, otherwise do not specify this attribute.

id

Optional. Specifies the ID of the tag. For more information, see EJBGen Tag Inheritance.

principals

Optional. Specifies the names of the principals in this role (separated by commas). When the security role is EJB-scoped, this attribute must be specified.

role-name

Required. Specifies the name of the role.

Related Topics

Role-Based Security

# @ejbgen:security-role-ref Annotation

This tag maps a reference to a security role used in the bean code to an EJB-scoped security role defined in the EJB project.

## Scope

Class tag on entity and session beans.

## Syntax

@ejbgen:security-role-ref

[id="TagID"]

[role-link="RoleLink"]

role-name="RoleName"

## Attributes

id

Optional. Specifies the ID of the tag. For more information, see EJBGen Tag Inheritance.

role-link

Optional. References a EJB-scoped security role. If you do not specify this attribute, deployment of the bean is not possible until the role name is manually mapped the link to a security role by the deployer (administrator).

role-name

Required. Specifies the name of the security role used in the code.

## Remarks

When you use this tag, make sure that the EJB scoped security role is defined using the @ejbgen:role-mapping Annotation.

## Example

The following example shows the use of a manager role in the EntityContext's isCallerInRole method in a session bean. The ejbgen:security-role-ref tag at the top of the class definition maps this reference to managerLevel3. The ejbgen:role-mapping tag defines the managerLevel3 security role.

```
* @ejbgen:role-mapping global-role="true" role-name="managerLevel3"  
* @ejbgen:security-role-ref role-link="managerLevel3" role-name="manager"
```

## WebLogic Workshop Annotations Reference

```
* ...
*/
public class RoleCheckerBean extends GenericSessionBean implements SessionBean
{
    SessionContext context;

    public void ejbCreate() {
        // Your code here
    }

    /**
     * @ejbgen:local-method
     */
    public String approveExpense(double amount)
    {
        if(context.isCallerInRole("manager") == false) {
            // not allowed to approve this expense
            ...
        }
        ...
    }
}
```

### Related Topics

#### Role-Based Security

#### @ejbgen:role-mapping Annotation

# @ejbgen:select Annotation

This tag specifies an select method for a CMP entity bean.

## Scope

A method tag on a CMP entity bean.

## Syntax

@ejbgen:select

[caching-name="CacheName"]

[ejb-ql="QueryLanguage"]

[group-name="GroupName"]

[include-updates="True/False"]

[max-elements="MaxNumber"]

[ordering-number="0...N"]

[result-type-mapping="Local/Remote"]

DEPRECATED [sql-select-distinct="True/False"]

[weblogic-ejb-ql="WebLogicEJBQueryLanguage"]

## Attributes

caching-name

Optional. Specifies the name of an eager relationship caching definition. For more details, see the `ejbgen:relationship-caching-element` Annotation and Accelerating Entity Bean Data Access.

ejb-ql

Optional. Specifies the EJB QL statement for this query.

group-name

Optional. This attribute defines which group-name will be eagerly loaded when returning the results of the query. For more details, see Accelerating Entity Bean Data Access.

include-updates

## WebLogic Workshop Annotations Reference

Optional. Specifies whether updates made during the current transaction must be reflected in the result of a query. Valid values are True and False. When this attribute is left unspecified, it defaults to true. For more details, see Accelerating Entity Bean Data Access.

max-elements

Optional. Specifies the maximum number of elements that should be returned by a query. When left unspecified, no maximum is set.

ordering-number

Optional. (0..n) Specifies the number where this method must appear in the generated class.

result-type-mapping

Optional. Specifies whether the returned objects are mapped to javax.ejb.EJBLocalObject or javax.ejb.EJBObject, that is, the remote or local (business) interface. Valid values are Remote and Local. If left unspecified, returned objects are mapped to the local interface.

sql-select-distinct

This attribute is deprecated. Please use the DISTINCT keyword in the EJB-QL instead.

weblogic-ejb-ql

Optional. Specifies the WebLogic QL statement for this query.

## Remarks

The following rules apply to this tag's use:

- The ejbgen:select tag is added to an ejbSelect method. In other words, unlike the ejbgen:finder tag, the tag is added to a method definition. To verify this, click a select method in Design View and notice that in Source View the tag is added to an ejbSelect method.
- Use either the ejb-ql or the weblogic-ejb-ql for the query executed for this finder method. When you specify queries for both attributes, the query specify for the ejb-ql attribute will be ignored.

Related Topics

How Do I: Add a Select Method to an Entity Bean?

Select Methods Sample

Query Methods and EJB-QL

@ejbgen:finder Annotation

# @ejbgen:session Annotation

This tag defines the class-scope properties of a session bean.

## Scope

Class tag on a session bean.

## Syntax

**@ejbgen:session**

ejb-name="NameSessionBean"

[allow-concurrent-calls="True/False"]

[allow-remove-during-transaction="True/False"]

[bean-load-algorithm="NameAlgorithm"]

[cache-type="NRU/LRU"]

[call-router-class-name="ClassName"]

[clients-on-same-server="True/False"]

[create-as-principal-name="PrincipalName"]

[default-transaction="DefaultTransactionOption"]

[dispatch-policy="DispatchPolicyQueue"]

[enable-call-by-reference="True/False"]

[home-call-router-class-name="ClasName"]

[home-is-clusterable="True/False"]

[home-load-algorithm="LoadAlgorithmOption"]

[idle-timeout-seconds="TimeoutValue"]

[initial-beans-in-free-pool="InitialNumber"]

[is-clusterable="True/False"]

[max-beans-in-cache="MaxNumber"]

[max-beans-in-free-pool="MaxNumber"]

@ejbgen:session Annotation



[methods-are-idempotent="True/False"]

[passivate-as-principal-name="PrincipalName"]

[persistent-store-dir="DirectoryName"]

[remove-as-principal-name="PrincipalName"]

[replication-type="InMemory/None"]

[run-as="RoleName"]

[run-as-identity-principal="PrincipalName"]

[trans-timeout-seconds="Seconds"]

[transaction-type="Bean/Container"]

[type="Stateless/Stateful"]

[use-caller-identity="True/False"]

## Attributes

ejb-name

Required. Specifies the (descriptive) name of the session bean. When you create a new session bean with the file name <name>Bean.ejb or <name>.ejb, the ejb-name will be <name> by default.

allow-concurrent-calls

Optional. Specifies whether to allow concurrent calls for a stateful session bean. Valid values are true and false. When left unspecified, it defaults to false. If a second method call occurs while the session bean is still processing the previous method, and this attribute is set to true, the second method call will wait. If the waiting period exceeds the transaction time out (see below), a LockTimedOutException is thrown. If the attribute is not set to true, this exception is thrown immediately. Concurrent calls typically occur when a web user reloads/refreshes a page before the request initiated through this page has been completed by the session bean.

allow-remove-during-transaction

Optional. Specifies whether the remove method can be invoked during a transaction. Valid values are true and false. When the attribute is left unspecified, the default is false.

bean-load-algorithm

Optional. Specifies the algorithm to be used for load-balancing among replicas of a stateless session bean. Valid values are round-robin, weight-based, random, round-robin-affinity, weight-based-affinity, and random-affinity. When this attribute is left unspecified, the default is round-robin.

### cache-type

Optional. Specifies the cache type for passivation of a stateful session bean. Valid values are NRU (not recently used, or eager passivation) and LRU (least recently used, or lazy passivation). If this attribute is left unspecified, the default is NRU. For more information, see *The Life Cycle of a Session Bean*.

### call-router-class-name

Optional. Specifies the class name to be used for routing home method calls. This class must implement `weblogic.rmi.cluster`.

### clients-on-same-server

Optional. Specifies whether all clients are co-located with the session bean on the same server. Valid values are true and false. When this attribute is left unspecified, it defaults to false. If set to True, the server instance will not multicast JNDI announcements for the EJB when it is deployed, that is, it turns off JNDI replication so that the EJB's JNDI name is only bound to the local server, hence reducing the startup time for large clusters.

### create-as-principal-name

Optional. Specifies the name of the principal to be used when running the bean's `ejbCreate` method. For more information, see *Role-Based Security*.

### default-transaction

Optional. Specifies the default transaction attribute to be applied to all methods that do not have a transaction attribute setting. Valid values are NotSupported, Supports, Required, RequiresNew, Mandatory, and Never. When the default-transaction is left unspecified, it defaults to Supports. When the transaction-type is set to Bean, the default-transaction attributed is ignored. For more information, see *EJBs and Transactions*.

### dispatch-policy

Optional. Specifies the dispatch policy queue for this bean. If this attribute is not specified, the server's default execute thread pool is used.

### enable-call-by-reference

Optional. Specifies whether the container will call this EJB by reference. When left unspecified this attribute defaults to False.

### home-call-router-class-name

Optional. Specifies the name of a custom class to use for routing bean method calls.

### home-is-clusterable

Optional. Specifies whether this bean can be deployed from multiple servers in a cluster. Valid values are true and false. If this attribute is not specified, it defaults to true.

### home-load-algorithm

## @ejbgen:session Annotation

## WebLogic Workshop Annotations Reference

Optional. Specifies the algorithm to use for load-balancing between replicas of the EJB home. Valid values are RoundRobin, Random, WeightBased, RoundRobinAffinity, RandomAffinity, and WeightBasedAffinity.

idle-timeout-seconds

Optional. Specifies the maximum duration in seconds that an stateful session bean will stay in the cache before it is removed. The default is 600.

initial-beans-in-free-pool

Optional. Specifies the initial number of beans in the free pool. The default is 0.

is-clusterable

Optional. Specifies whether this bean is clusterable. Valid values are true and false. If this attribute is not specified, it defaults to true.

max-beans-in-cache

Optional. Specifies the maximum number of beans in cache. The default is 1000.

max-beans-in-free-pool

Optional. Specifies the maximum number of beans in the free pool.

methods-are-idempotent

Optional. Specifies whether the methods for a stateless session bean are idempotent. Valid values are true and false. When left unspecified, the default is false.

passivate-as-principal-name

Optional. Specifies the name of the principal to be used when running the bean's ejbPassivate method. For more information, see Role-Based Security.

persistent-store-dir

Optional. Specifies the directory in which to store passivated stateful beans.

remove-as-principal-name

Optional. Specifies the name of the principal to be used when running the bean's ejbRemove method. For more information, see Role-Based Security.

replication-type

Optional. Specifies whether to replicate stateful session beans in a cluster. Valid values are InMemory and None. When this attribute is left unspecified, the default is None.

run-as

## WebLogic Workshop Annotations Reference

Optional. Specifies the EJB-scoped security role this bean should run under. To use this attribute, `use-caller-identity` must be `False`. Also make sure that the security role is defined using the `@ejbgen:role-mapping` Annotation. For more information, see [Role-Based Security](#).

`run-as-identity-principal`

Optional. Specifies the name of the principal to run as in case the security role specified with the `run-as` attribute maps to several principals. To specify this attribute you must also specify the `run-as` attribute. You must specify the `run-as-identity-principal` attribute when the `run-as` security role is externally defined. For more information, see [Role-Based Security](#).

`trans-timeout-seconds`

Optional. Specifies the transaction timeout (in seconds). If not specified, this attribute defaults to 30 seconds.

`transaction-type`

Optional. Specifies who manages the transactions for this EJB. Valid values are `Bean` and `Container`. When not specified, this attribute defaults to `Container`. For more information, see [EJBs and Transactions](#).

`type`

Optional. Valid values are `Stateless` and `Stateful`. When not specified, this attribute defaults to `Stateless`.

`use-caller-identity`

Optional. Specifies whether or not this EJB uses the caller's identity. When not specified, this attribute defaults to `True`. You must set this attribute to `False` to use the `run-as` and `run-as-identity-principal` attributes. For more information, see [Role-Based Security](#).

Related Topics

[Getting Started with Session Beans](#)

[Role-Based Security](#)

[EJBs and Transactions](#)

# @ejbgen:value-object Annotation

This tag specifies whether EJB object references in a generated value object should be represented as a value object or a local interface. That is, when a value object is generated for a CMP entity bean, it contains all the EJB's fields. If one of these fields is another EJB – in most cases this would be a CMR field used to model an entity relation – you can choose to have this field represented as a value object or a local interface.

## Scope

Class tag on entity beans.

## Syntax

**@ejbgen:value-object**

reference="Local/Value"

## Attributes

reference

Required. Specifies how EJB object references in a value object class should be represented. Valid values are Local and Value.

## Remarks

The following rules apply to this tag's use:

- When this tag is not specified for an entity bean, EJB object references in a value object are represented as value objects.
- When EJB object references are represented as value objects, the referred entity beans must define or auto-generate its value object class.

## Related Topics

@ejbgen:file-generation Annotation