BEA WebLogic Integration™

Annotations Reference

Version 8.1 Service Pack 3
Document Date: June 2004

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

Http Control Interfaces and Annotations

JMS Control Annotations

Message Broker Control Annotations

MQSeries Control Interfaces and Annotations

Process Control Annotations

RosettaNet Control Annotations

Service Broker Control Annotations

Worklist Control Annotations

Business Process Annotations

Data Transformation Annotations

CHAPTER 1

Annotations Reference

This section provides reference information about WebLogic Integration-specific 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, in WebLogic Integration, annotations can be used to define the purpose of an Integration control or a business process.

To learn about Workshop annotations not specific to WebLogic Integration (that is, Web service annotations, page flow annotations, and so on), see the WebLogic Workshop Reference.

Topics Included in This Section

CommonAnnotations Reference

Provides reference information for *common* annotations. Common annotations are those annotations that are available to more than one type of file (JWS, JXCX, JPD, and so on) in WebLogic Workshop.

Integration Controls Annotations Reference

This section provides reference information for WebLogic Integration controls annotations.

Business Process Annotations Reference

This section provides reference information for WebLogic Integration business process (JPD) annotations (that is, annotations that are of the following format: @jpd:*name_of_annotation*).

Data Transformation Annotations

This section provides reference information for WebLogic Integration data transformation (DTF) annotations (that is, annotations that are of the following format: @dtf:*name_of_annotation*).

CHAPTER 2

Common Annotations

This section provides reference information for common annotations. Common annotations are available to more than one type of file in WebLogic Workshop.

Topics Included in This Section

@common:context Annotation

Specifies to WebLogic Server the type of component for which it creates a context. The components include business processes, controls, Web services and so on.

@common:control Annotation

Specifies that the object annotated by this annotation is a WebLogic Workshop control in a JCX file.

@common:context Annotation

The @common:context annotation specifies that WebLogic Server should create a context for the component, such as a business process 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 business processes, the context object is of type JpdContext; for controls, ControlContext.

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 JpdContext object appear as follows:

```
/** @common:context */

   JpdContext context;
```

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

Related Topics

jpdContext Interface

@common:control Annotation

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

Syntax

`@common:control`

Attributes

None.

Remarks

The following rules apply to this annotation's use:

- Only one `@common:control` annotation can be specified 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 not function properly; attempts to invoke the control's methods will result in Null Pointer Exceptions (NPEs).

Related Topics

Using Integration Controls

CHAPTER 3

Integration Controls Annotations

This section provides reference information for WebLogic Integration controls annotations.

Topics Included in This Section

Chapter 12, "Process Control Annotations"
Describes the Process control annotations.

Chapter 13, "RosettaNet Control Annotations"
Describes the RosettaNet control annotations.

Chapter 14, "Service Broker Control Annotations"
Describes the Service Broker control annotations.

Chapter 15, "Worklist Control Annotations"
Describes the Worklist control (Task control and Task Worker control) annotations.

Note: There is a TPM Control, but this control has no annotations.

CHAPTER 4

Application View Control Annotations

The Application View control annotations provide information to WebLogic Server about how the control functions. This section contains information about the Application View control annotations, including the syntax to use and the available attributes that can be set for the control.

Topics Included in This Section

@jc:av-identity Annotation
Specifies the target Application View for an Application View control.

@jc:av-service Annotation
Specifies the Application View service associated with a method of an Application View control.

@jc:av-identity Annotation

Specifies the target Application View for an Application View control.

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

Syntax

@jc:av-identity

name="applicationViewName"
app="applicationName"
namespaceEnforcementEnabled="true | false"

Attributes

name

Required. Specifies the name of the target Application View. The name is fully qualified and dot separated. The Application View must be deployed before the Application View control will function.

app

Required. Specifies the name of the associated WebLogic Workshop application. This is usually the same as the current WebLogic Workshop application.

This parameter allows you to reuse an application view between WebLogic Workshop applications. First, define the application view in the context of the primary application. Then, define an Application View control in a process within a second application and specify the primary application in the app parameter. When reusing an application view from another application, all services are accessed synchronously.

namespaceEnforcementEnabled

If true, indicates that the client requires response instances and event instances to declare the namespace indicated in the response/event definitions, and force the proper namespace declaration onto the response/event if needed. Some legacy adapters do not provide responses/events using the namespaces declared in the response/event schema. This option allows clients that perform schema-based XML checking to use such adapters. The option is disabled by default, because it can have serious performance implications for adapters that return their responses/events as raw XML text (not parsed). In this case, this option forces a parse of the XML text in order to inject the proper namespace declaration.

Remarks

The following rules apply to this annotation's use:

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

Related Topics

Application View Control

@jc:av-service Annotation

@jc:av-service Annotation

Specifies the Application View service associated with a method of an Application View control.

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

Syntax

@jc:av-service
name="avServiceName"
async="true | false"

Attributes

name

Required. Specifies the name of the service in the target Application View with which this Application View control method is associated.

async

Required. Specifies whether or not the service is asynchronoous. The default value is false (synchronous).

Remarks

The following rules apply to this annotation's use:

- The @jc:av-service annotation may only occur on a method declaration in an interface that extends weblogic.jws.control.ApplicationViewControl.
- The @jc:av-service annotation may only occur in a JCX file.
- The @jc:av-service annotation may only appear once per method.

Related Topics

Application View Control

@jc:av-identity Annotation

ebXML Control Annotations

The ebXML control enables WebLogic Workshop business processes to exchange business messages and data with trading partners via ebXML. The ebXML control supports both the ebXML 1.0 and ebXML 2.0 messaging services. You use ebXML controls in *initiator* business processes to manage the exchange of ebXML business messages with participants.

This section contains information about ebXML control annotations, including the syntax to use and the available attributes that can be set for the control.

Topics Included in This Section

@jc:ebxml Annotation

Specifies JCX class-level annotations for the ebXML control.

Note: For most attributes, annotations can also be specified at the instance and method level. The order of precedence is:

1. JCX method level.
2. JCX instance level.
3. JCX class level.

Syntax

```
jc:ebxml
   [from="initiatorID"] | [from-selector="{xquery-expression}"]
   [to="participantID"] | [to-selector="{xquery-expression}"]
   [ebxml-service-name="ebxml-service-name"]
   [ebxml-action-mode="default | non-default"]
```

Attributes

from

Business ID of the initiator. Must match the business ID for the trading partner as defined in the TPM repository.

from-selector

XQuery expression that selects the business ID of the initiator. To learn how to specify the initiator business ID dynamically, see "Dynamically Specifying Business IDs" in Using an ebXML Control.

Note: This attribute is not available for all methods at the control type level in the control definition file (JCX file). It only applies to the send method in the control definition or to control instance declarations in the business process file (JPD file).

to

Business ID of the participant. Must match the business ID for the trading partner as defined in the TPM repository.

to-selector

XQuery expression that selects the business ID of the participant. To learn how to specify the participant business ID, see "Dynamically Specifying Business IDs" in Using an ebXML Control.

Note: This attribute is not available for all methods at the control type level in the control definition file (JCX file). It only applies to the send method in the control definition or to control instance declarations in the business process file (JPD file).

ebxml-service-name

Name of an ebXML service. For initiator and participant business processes that participate in the same conversation, the settings for **`ebxml-service-name`** must be identical. This service name corresponds to the `eb:Service` entry in the ebXML message envelope.

ebxml-action-mode

Action mode for this ebXML control. Determines the value specified in the `eb:Action` element in the message header of the ebXML message, which becomes important in cases where multiple message exchanges occur within the same conversation. One of the following values:

- `default`—Sets the `eb:Action` element to `SendMessage` (default name).

- `non-default`—Sets the `eb:Action` element to the name of the method (on the ebXML control) that sends the message in the initiator business process. For sending a message from the initiator to the participant, this name must match the method name of the **Client Request** node in the corresponding participant business process. For sending a message from the participant to the initiator, the method name in the callback interface for the client callback node in the participant business process must match the method name (on the ebXML control) in the control callback interface in the initiator business process. Using `non-default` is recommended to ensure recovery and high availability.

If unspecified, the `ebxml-action-mode` is set to `non-default`.

Related Topics

@jc:ebxml-method Annotation

@jc:ebxml-method Annotation

Specifies method-level annotations for the ebXML control.

Syntax

```
jc:ebxml-method
   [to-selector="{xquery-expression}"]
   [envelope="{env}"]
```

Attributes

to-selector

XQuery expression that selects the recipient business ID. To learn how to specify the business ID dynamically using selectors in the **Property Editor**, see "Dynamically Specifying Business IDs" in Using an ebXML Control.

envelope

Used with a callback method to assign the ebXML envelope of an incoming message.

Note: You can rename the default value (`env`) as long as it matches the name of the parameter specified in the method.

Related Topics

@jc:ebxml Annotation

CHAPTER 6

Email Control Annotations

This section describes the Email control annotations.

Topics Included in This Section

@jc:email Annotation

Specifies class- and method-level configuration attributes for the Email control.

Syntax

```
jc:email
      [from-address="from-address"]
      [from-name="from-name"]
      [smtp-address="smtp-address"]
      [reply-to-address="reply-to-address"]
      [reply-to-name="reply-to-name"]
      [smtp-username="smtp-username"]
      [smtp-password="smtp-password"]
      [smtp-password-alias="smtp-password-alias"]
      [header-encoding="header-encoding"]
```

Attributes

These attributes determine the default behavior of the Email control. The Email control may be configured during its lifetime by calling methods of the EmailControl class.

from-address
A string containing the originating e-mail address. This attribute is required if the `from-name` attribute is present.

from-name
A string containing the display name for the originating e-mail address. This attribute is optional.

smtp-address
A string containing the address of the SMTP server in *`host:port`* or *`host`* form. If the port is not specified, the standard SMTP port of 25 is used. This attribute is required.

reply-to-address
A string containing the e-mail address to reply to. This attribute is required if the `reply-to-name` attribute is present.

reply-to-name
A string containing the display name for the reply-to-address. This attribute is optional.

smtp-username
A string containing the username for server's that require authentication to send. This attribute is optional.

smtp-password
A string containing the associated password. This attribute is optional.

smtp-password-alias
A string containing the password alias. The alias is used to look up the password in the password store. This attribute is optional and is mutually exclusive with the `smtp-password` attribute.

header-encoding
A string specifying the encoding to be used for the mail headers as specified by `from-name`, `reply-to-name`, `to`, `bc`, `bcc`, `subject`, and `attachments`. If no header encoding is specified, the system default encoding is used.

Related Topics

Email Control

@jc:send-email Annotation

Specifies class- and method-level configuration attributes for the Email control.

Syntax

```
jc:send-email
      [to="To-recipients"]
      [cc="CC-recipients"]
      [bcc="BCC-recipients"]
      [subject="subject"]
      [body="body"]
      [content-type="content-type"]
      [attachments="file-list"]
```

Attributes

These attributes determine the default behavior of the Email control. The Email control may be configured during its lifetime by calling methods of the EmailControl class.

Parameter substitution can be used for any of the following method attributes. Substitutions are allowed in the middle of the subject or body. For example, upon receiving an order, you can send the following e-mail:

```
"Thanks for your order. Your order number is {orderNumber}.
Please reference this number in all your future correspondence."
```

to

The list of To recipients. This attribute takes a comma separated list of Strings. This attribute is required.

cc

The list of CC recipients. This attribute takes a comma separated list of Strings. This attribute is optional.

bcc

The list of BCC recipients. This attribute takes a comma separated list of Strings. This attribute is optional.

subject

A string containing the subject of the e-mail. This attribute is optional.

body

A string containing the body of the e-mail. This attribute is optional.

content-type

A string containing the content-type of the body. If not specified, the default is `text/plain` for String bodies and `text/xml` for XmlObject bodies. Aside from the default `text/plain`, expected content types include `text/html`, `text/xml`, and `application/xml`. This attribute is optional.

attachments

The list of files to send as attachments. This attribute takes a comma separated list of Strings. This attribute is optional.

Unqualified paths specifying attachment locations are relative to the location of the domain's `startWeblogic` command file. Because the domain root may be deep in the directory structure, we recommend the use of absolute paths for specifying attachment locations.

The `to` and `cc` recipient lists can include display names as shown in the following examples:

```
Joe User <joe.user@myorg.com>, Jane User <jane.user@myorg.com>
```

```
"Joe A. User" <joe.user@myorg.com>, "Jane B. User" <jane.user@myorg.com>
```

Related Topics

Email Control

@jc:email Annotation

CHAPTER 7

File Control Annotations

This section describes the File control annotations.

Topics Included in This Section

@jc:file Annotation

Specifies the annotations for the File control.

Syntax

```
@jc:file
      [directory-name="directory name"]
      [file-mask="file name or file mask"]
      [suffix-name="file name suffix"]
      [suffix-type="timestamp or index"]
      [create-mode="over-write or rename-old"]
      [ftp-host-name="ftp host name"]
      [ftp-username-name="ftp user name"]
      [ftp-password="password"]
      [ftp-password-alias="password alias"]
      [ftp-local-directory="local directory name"]
```

Attributes

These attributes determine the default behavior of the File control. The File control can be configured during its lifetime by calling methods of the `FileContol` class. To learn more about the `FileControl` class, see the javadoc for the File Control.

directory-name

A directory name is the absolute path name for the directory. In other words, it includes the drive specification as well as the path specification. For example, following are valid directory names:

```
C:\directory (Windows)

/directory (Unix)

\\servername\sharename\directory (Win32 UNC)
```

The *`directory-name`* attribute is required. Leaving the *`directory-name`* attribute unspecified results in an error.

file-mask

The *file-mask* attribute can specify either a file name or a file mask. If the file-mask contains a wild-card character (such as "*") it will be treated as a file mask. Typically, a wild-card character is specified to get the list of files in a directory. It is illegal to specify a wild-card character for any other operation.

File names are used for read, write and append operations.

suffix-name

This suffix will be used along with a timestamp or incrementing index for creating the file names. The default *suffix-name* will be "_". For example:

```
file_01, file_02, file_0809021230123
```

suffix-type

This option specifies if a timestamp or an incrementing index should be used as a suffix for the file names. The allowed options are: `index` and `timestamp`.

create-mode

This option specifies what needs to be done when a write operation is creating a new file and a file with the same name already exists. The allowed options are: `over-write` and `rename-old`.

When you use `create-mode="rename=old"` to rename a file, make sure that you mention the `suffix-name` and the `suffix-type` attributes for the new file name. If the suffix attributes are not indicated, then the File control overwrites the old file, instead of renaming it.

ftp-host-name
This option specifies the name of the FTP host, for example, `ftp://ftp.bea.com`.

ftp-user-name
This option specifies the name of the FTP user.

ftp-password
This option specifies the FTP user's password. If you specify this attribute, you cannot specify the *`ftp-password-alias`* attribute.

ftp-password-alias
This option specifies the alias for a user's password. The alias is used to look up a password in a password store. If you specify this attribute, you cannot specify the *`ftp-password`* attribute.

ftp-local-directory
This option specifies the directory used for transferring files between the remote file system and the local file system. When reading a remote file, the file is copied from the remote system to the local directory and then read. Similarly, when writing to a remote file system, the file is written to the local directory and then copied to the remote system.

Related Topics

File Control

@jc:file-operation Annotation

@jc:file-operation Annotation

Specifies configuration attributes for a File control.

Syntax

```
@jc:file-operation
      [io-type="read, readline, write or append"]
      [file-content="file content description"]
      [record-size="number of bytes per record"]
      [encoding="character set encoding"]
```

Attributes

These attributes determine the default behavior of the File control. The File control can be configured during its lifetime by calling methods of the `FileContol` class. To learn more about the `FileControl` class, see the javadoc for the File Control.

io-type

This attribute specifies the type of operation. The valid values are: `read`, `write`, `append` and `readline`. (To learn about the readline value, see record-size).

file-content

This attribute indicates the contents of the identified variable which will be written to the file.

record-size

This option is used with methods of type `@jc:file-operation io-type="readline"`. The record size, a positive integer, is expressed in bytes.

The `record-size` attribute is valid for methods with a return type of `RawData` and `String`, but not `XmlObject`. If this attribute is not specified, the default platform-specific line delimiters, such as carriage returns or line feeds, are used.

The following code illustrates the use of the `record-size` attribute:

```
/**
* @jc:file-operation io-type="readline" record-size="80"
*/
RawData readLine();
```

encoding

This option is used to specify the character set encoding for the file. The file type must be String or XMLObject. This option can not be used if large files are being processed.

Related Topics

File Control

@jc:file Annotation

CHAPTER 8

Http Control Interfaces and Annotations

This section describes the Http control interfaces and annotations and provides a sample of an extended Http control.

Topics Included in This Section

Http Control Annotations
A reference for Http control annotations.

Http Control Interface

Sample Extended Http Control

The following code shows an example of an extended control. This code is automatically created by the control wizard.

```
package processes;

import com.bea.control.*;

import com.bea.wli.control.httpResponse.ResponseDocument;

import com.bea.wli.control.httpParameter.ParametersDocument;

import com.bea.xml.XmlObject;

     /*

     * A custom Http control.
```

```
     */
    /**
     * @jc:httpsend-data url-name="edocs.bea.com"
     */
public interface dddd extends HttpControl, com.bea.control.ControlExtension
 {
    /*
    * A version number for this JCX. This will be incremented in new versions
of
     * this control to ensure that conversations for instances of earlier
     * versions were invalid.
     */
     static final long serialVersionUID = 1L;
     ResponseDocument sendDataAsHttpGet(ParametersDocument
parameters,String charset);
     byte[] getResponseBodyData();
}
```

Http Control Annotations

This section includes information on Http control annotations. It includes the following topics:

@jc:httpsend-data

@jc:httpsend-data

Specifies the URL to which an Http message is to be sent, and from which response is to be received.

Syntax

```
@jc:httpsend-data url-name
      [url-name="name of the URL"]
```

JMS Control Annotations

The JMS Control annotations provide information to WebLogic Server about how the control functions.

The WLI JMS control is an extension of the JMS control. The following JMS control annotations also apply to the WLI JMS control:

- @jc:jms-headers Annotation
- @jc:jms-property Annotation

Related Topics

WLI JMS Control

CHAPTER 10

Message Broker Control Annotations

This section describes the Message Broker Publish and Subscription control annotations.

Topics Included in This Section

Message Broker Publish Control Annotations

@jc:mb-publish-control Annotation
Defines class level attributes for the Publish control.

@jc:mb-publish-method Annotation
Defines method level attributes for the Publish control.

Message Broker Subscription Control Annotations

@jc:mb-subscription-control Annotation
Defines class level attributes for the Subscription Control.

@jc:mb-subscription-method Annotation
Defines method level attributes for the Subscription Control.

@jc:mb-subscription-callback Annotation
Defines callback attributes for the Subscription Control.

@jc:mb-publish-control Annotation

This section describes the class attributes supported for the Publish control.

Syntax

```
@jc:mb-publish-control
      [channel-name="channel name"]
      [message-metadata="message metadata"]
```

Attributes

channel-name

The name of the Message Broker channel to which the MB Publish control publishes messages.

message-metadata

By default, this XML header is included in messages published with this control. Valid values include a string containing XML.

Related Topics

Message Broker Controls

@jc:mb-publish-method Annotation

This section describes the method attributes supported for the Publish control.

Syntax

```
@jc:mb-publish-method
      [message-metadata="message metadata"]
      [message-body="message body"]
```

Attributes

message-metadata

XML header to include in messages published with the control method to which it is associated. Valid values include a string containing XML, or a method parameter in curly braces. For example: `{parameter1}`.

message-body

Valid values include a string containing text that is used as the message body in the published message, or a method parameter in curly braces. For example: `{parameter2}`.

Related Topics

Message Broker Controls

@jc:mb-subscription-control Annotation

This section describes the class attributes supported for the Subscription control.

Syntax

```
@jc:mb-subscription-control
      [channel-name="channel name"]
      [xquery="xquery"]
```

Attributes

channel-name

The name of the Message Broker channel to which the control subscribes. This is a required class-level annotation that cannot be overridden.

xquery

The XQuery expression that is evaluated for each message published to a subscribed channel. Messages that do not satisfy this expression are not dispatched to a subscribing business process. This is an optional class-level annotation that cannot be overridden.

Related Topics

Message Broker Controls

@jc:mb-subscription-method Annotation

This section describes the method attributes supported for the Subscription control.

Syntax

```
@jc:mb-subscription-method
      [filter-value-match="filter value match"]
```

Attributes

filter-value-match

The *filter-value* that the XQuery expression results must match for the message to be dispatched to a subscribing business process. This is an optional method-level annotation. Valid values for the *filter-value-match* annotation include a string constant that is compared directly to the XQuery results, or a method parameter in curly braces. For example: `{parameter1}`

Related Topics

Message Broker Controls

@jc:mb-subscription-callback Annotation

This section describes the callback attributes supported for the Subscription control.

Syntax

```
@jc:mb-subscription-callback
      [message-metadata="message metadata"]
      [message-body="message body"]
```

Attributes

message-metadata

The name of a parameter in the callback method that receives the metadata from the message that triggered the subscription. This parameter can be of an `XmlObject` or typed XML.

message-body

The name of a parameter in the callback method that receives the body from the message that triggered the subscription. This parameter must be of type `XmlObject` (or a typed XBean), `String`, `RawData`, or a non-XML MFL class (a subclass of `MflObject`).

Related Topics

Message Broker Controls

CHAPTER 11

MQSeries Control Interfaces and Annotations

This section describes the MQSeries control interfaces and annotations and provides a sample of an extended MQSeries control.

Topics Included in This Section

MQSeries Control Interface
Describes the MQSeries control interface with an example of an extended MQSeries control.

MQSeries Control Annotations
A reference for MQSeries control annotations.

MQSeries Control Interface

The MQSeries control supports the sending and receiving of messages to and from MQSeries queues. The supported message types are Bytes, String and XML.

The MQSeries control supports two types of connections, that is, TCP and Bindings. The connection options, such as Queue Manager Name, Queue Name and so on, for the MQSeries control can be specified while configuring the MQSeries control.

The following are the MQSeries control methods:

```
public interface MQControl extends Control

{

    /**
```

```
     * Begins a MQ transaction
     * @exception ResourceException if transaction state is invalid
     */
    void begin() throws javax.resource.ResourceException;
    /**
     * Rolls back a MQ transaction
     * @exception ResourceException if transaction state is invalid
     */
    void rollback() throws javax.resource.ResourceException;
    /**
     * Commits a MQ transaction
     * @exception ResourceException if transaction state is invalid
     */
    void commit() throws javax.resource.ResourceException;
    /**
     * Sets the dynamic properties for the control.
     * @param mqDynPropsDoc the MQDynamicProperties document containing
     * the dynamic properties to be set
     */
    void
setDynamicProperties(com.bea.wli.control.mqDynamicProperties.MQDynamicProperti
esDocument mqDynPropsDoc);


    /**
     * Gets a byte array(binary) message from the queue This function calls the
     * generic getMessage function
     * @param queue the queue from which the message is to be got
    * @param mqmd the MQMDHeaders document containing the MQMD attributes based
on which the message is to be got
```

```
     * @return byte[] the byte array representing the message got

     * @exception ResourceException if any exception occurs while get

     */

    byte[] getMessageAsBytes(java.lang.String queue,
com.bea.wli.control.mqmdHeaders.MQMDHeadersDocument mqmd)
    throws javax.resource.ResourceException;

    /**

     * Gets a String(text) message from the queue This function calls the generic

     * getMessage function

     * @param queue the queue from which the message is to be got

    * @param mqmd the MQMDHeaders document containing the MQMD attributes based
on which the message is to be got

     * @return String the String representing the message got

     * @exception ResourceException if any exception occurs while get

     */

    java.lang.String getMessageAsString(java.lang.String queue,
com.bea.wli.control.mqmdHeaders.MQMDHeadersDocument mqmd)
    throws javax.resource.ResourceException;

    /**

     * Gets a XmlObject(xml) message from the queue This function calls the

     * generic getMessage function

     * @param queue the queue from which the message is to be got

    * @param mqmd the MQMDHeaders document containing the MQMD attributes based
on which the message is to be got

     * @return XmlObject the XmlObject representing the message got

     * @exception ResourceException if any exception occurs while get

     */

    com.bea.xml.XmlObject getMessageAsXml(java.lang.String queue,
com.bea.wli.control.mqmdHeaders.MQMDHeadersDocument mqmd)
    throws javax.resource.ResourceException;

    /**
```

```
     * Puts a XmlObject(xml) message into the queue This function calls the
     * generic putMessage function
     * @param message the xml message to be put into the queue
     * @param queue the queue to which the message is to be put
     * @param mqmd the MQMDHeaders document containing the MQMD attributes of
the message to be put
    * @return MQMDHeadersDocument representing the attributes of the message put
     * @exception ResourceException if any exception occurs while put
     */
    com.bea.wli.control.mqmdHeaders.MQMDHeadersDocument
putMessageAsXml(com.bea.xml.XmlObject message, java.lang.String queue,
    com.bea.wli.control.mqmdHeaders.MQMDHeadersDocument mqmd) throws
javax.resource.ResourceException;
    /**
     * Puts a String(text) message into the queue This function calls the generic
     * putMessage function
     * @param message the String message to be put into the queue
     * @param queue the queue to which the message is to be put
     * @param mqmd the MQMDHeaders document containing the MQMD attributes of
the message to be put
    * @return MQMDHeadersDocument representing the attributes of the message put
     * @exception ResourceException if any exception occurs while put
     */
    com.bea.wli.control.mqmdHeaders.MQMDHeadersDocument
putMessageAsString(java.lang.String message, java.lang.String queue,
    com.bea.wli.control.mqmdHeaders.MQMDHeadersDocument mqmd) throws
javax.resource.ResourceException;
    /**
     * Puts a byte array(binary) message into the queue This function calls the
     * generic putMessage function
     * @param message the byte array message to be put into the queue
```

```
     * @param queue the queue to which the message is to be put

     * @param mqmd the MQMDHeaders document containing the MQMD attributes of
the message to be put

    * @return MQMDHeadersDocument representing the attributes of the message put

     * @exception ResourceException if any exception occurs while put

     */

    com.bea.wli.control.mqmdHeaders.MQMDHeadersDocument
putMessageAsBytes(byte[] message, java.lang.String queue,
    com.bea.wli.control.mqmdHeaders.MQMDHeadersDocument mqmd) throws
javax.resource.ResourceException;

}
```

Sample Extended MQSeries Control

The following code shows an example of an extended control. This code is automatically created by the control wizard.

```
package processes;

import com.bea.control.*;

import com.bea.xml.XmlCursor;

import com.bea.control.MQControl;

import com.bea.wli.control.mqmdHeaders.MQMDHeadersDocument;

import com.bea.wli.control.mqDynamicProperties.MQDynamicPropertiesDocument;

import javax.resource.ResourceException;

import com.bea.xml.XmlObject;


    /*

     * A custom MQ control.

     */

    /**

     * @jc:MQConnectionType connectionType="Bindings"

     * @jc:MQConnectionPoolProps mqPoolSize="20"
```

```
     * @jc:ConnectionPoolTimeout conTimeout="3600"
     * @jc:MQQueueManager queueManager="QM_bea_aruna"
     * @jc:MQAuthorization requireAuthorization="No"
     * @jc:TCPSettings host=""
                       port="1414"
                       channel=""
                       ccsid="819"
                       user=""
                       password=""
                       sendExit=""
                       receiveExit=""
                       securityExit=""
     * @jc:DefaultQueue defaultQueueName="default"
     * @jc:ImplicitTransaction implicitTransactionRequired="true"
     */
public interface newjcx extends MQControl, com.bea.control.ControlExtension
 {
    /*
    * A version number for this JCX. This will be incremented in new versions of
     * this control to ensure that conversations for instances of earlier
     * versions were invalid.
     */
    static final long serialVersionUID = 1L;
}
```

MQSeries Control Annotations

This section includes information on MQSeries control annotations.

Topics Included in This Section

@jc:MQConnectionType

Specifies the connection type for an MQSeries control.

Syntax

```
@jc:MQConnectionType
      [connectionType="TCP|Bindings"]
```

Attributes

This attribute determines the type of connection to an MQSeries queue manager.

`connectionType`

The connection type can be TCP or Bindings. When the Bindings type of connection is used, WebLogic Workshop application and the MQSeries server are running on the same machine. When the TCP connection type is used, WebLogic Workshop application and the MQSeries server may be running on different machines.

@jc:MQConnectionPoolProps

Specifies the MQSeries connection pool properties for the MQSeries control.

Syntax

```
@jc:MQConnectionPoolProps
      [mqPoolSize="pool size value"]
```

Attributes

This attribute determines the MQSeries connection pool size.

`connectionType`
The mqPoolSize is a positive integer greater than 0, representing the MQSeries connection pool size.

@jc:ConnectionPoolTimeout

Specifies the MQSeries connection pool timeout in seconds.

Syntax

```
@jc:ConnectionPoolTimeout
      [conTimeout="connection timeout value in seconds"]
```

Attributes

This attribute determines the MQSeries connection pool timeout value in seconds.

conTimeout
The connection timeout value should be a positive integer greater than 0.

@jc:MQQueueManager

Specifies the name of the queue manager for connection.

Syntax

```
@jc:MQQueueManager
      [queueManager="queue manager name"]
```

Attributes

This attribute determines the MQSeries queue manager to which connection is to be obtained.

queueManager
The queue manager name should be a String value, representing the queue manager to which connection is to be obtained.

@jc:MQAuthorization

Specifies the MQSeries authorization property for the MQSeries control.

Syntax

```
@jc:MQAuthorization
      [requireAuthorization="Yes|No"]
```

Attributes

This attribute determines whether MQSeries authorization is required or not.

requireAuthorization
The requireAuthorization attribute value should be either Yes or No.

@jc:TCPSettings

Specifies the TCP connection settings for the MQSeries control.

Syntax

```
@jc:TCPSettings
      [host="host name"]
      [port="port number"]
      [channel="server connection channel name"]
      [ccsid="Coded Character Set Id"]
      [user="User name"]
      [password="Password"]
      [sendExit="Send Exit class name"]
      [receiveExit="Receive Exit class name"]
      [securityExit="Security Exit class name"]
```

Attributes

These attributes determine the TCP connection settings while connecting to the queue manager using the TCP connection mode.

host
This represents the host name of the machine where the queue manager is running.

port
This represents the port number of the queue manager.

channel
This represents the server connection channel of the queue manager through which the connection is to be obtained.

ccsid
This represents the Coded Character Set Id to be used while connecting to the queue manager.

user
This represents the user who is connecting to the queue manager.

password
This represents the password of the user connecting to the queue manager.

sendExit
This represents the fully qualified name of the class implementing the MQSeries MQSendExit interface.

receiveExit
This represents the fully qualified name of the class implementing the MQSeries MQReceiveExit interface.

securityExit
This represents the fully qualified name of the class implementing the MQSeries MQSecurityExit interface.

@jc:DefaultQueue

Specifies the default queue name to be used for sending and retrieving messages.

Syntax

```
@jc:DefaultQueue
      [defaultQueueName="name of the default queue"]
```

This attribute determines the name of the default queue to be used for sending and retrieving messages.

`defaultQueueName`
: The default queue name attribute value should be a String value representing a valid MQSeries queue name present in the queue manager to which connection is to be obtained.

@jc:ImplicitTransaction

Specifies the transaction mode of the MQSeries control.

Syntax

```
@jc:ImplicitTransaction
      [implicitTransactionRequired="true|false"]
```

Attributes

This attribute determines whether the implicit transaction mode is required or not, while sending and receiving messages.

`implicitTransactionRequired`
: The implicit transaction required attribute value can be either True or False.

Process Control Annotations

This section describes the process control annotations.

The Process control extends WebLogic Workshop controls and uses some of the same annotations. For more information, see the following annotations:

- @common:message-buffer Annotation
- @jc:conversation Annotation
- @jc:location Annotation

Related Topics

Process Control

RosettaNet Control Annotations

This section describes the RosettaNet control annotation.

@jc:rosettanet Annotation

@jc:rosettanet Annotation

Specifies the JCX class-level annotations for the RosettaNet control.

Note: Annotations can be specified at the JCX class level, at the JCX instance level, and at the JCX method level, in increasing order of precedence.

Syntax

```
jc:rosettanet
   [from="initiatorID"] | [from-selector="{xquery-expression}"]
   [to="participantID"] | [to-selector="{xquery-expression}"]
   [="rnif-version"]
   [="pip"]
   [="pip-version"]
   [="from-role"]
   [="to-role"]
```

Attributes

The following attributes specify class- and method-level configuration attributes for the RosettaNet control:

from

DUNS of the initiator. Must match the business ID for the trading partner as defined in the TPM repository.

from-selector

XQuery expression that selects the business ID of the initiator. To learn how to specify the initiator business ID dynamically, see "Dynamically Specifying Business IDs" in Using a RosettaNet Control.

Note: This attribute is not available at the control type level in the control definition file (JCX file). It only applies to control instance declarations in the business process file (JPD file).

to

DUNS of the participant. Must match the business ID for the trading partner as defined in the TPM repository.

to-selector

XQuery expression that selects the business ID of the participant. To learn how to specify the recipient business ID dynamically, see "Dynamically Specifying Business IDs" in Using a RosettaNet Control.

Note: This attribute is not available at the control type level in the control definition file (JCX file). It only applies to control instance declarations in the business process file (JPD file).

rnif-version

Version of the RosettaNet Implementation Framework. Must be either 1.1 or 2.0.

pip

RosettaNet PIP code, such as `3B2`. Must be a valid PIP code as defined in `http://www.rosettanet.org`.

pip-version

RosettaNet PIP version. Must be a valid version number associated with the PIP.

from-role

RosettaNet role name for the sender as defined in the PIP specification, such as Buyer, Seller, Supplier, Receiver, Shipper, and so on. A PIP request might be rejected if an incorrect value is specified.

to-role

RosettaNet role name for the recipient as defined in the PIP specification, such as Buyer, Seller, Supplier, Receiver, Shipper, and so on. A PIP request might be rejected if an incorrect value is specified.

CHAPTER 14

Service Broker Control Annotations

This section describes the service control annotations.

The Service Broker control extends WebLogic Workshop controls and uses some of the same tags and annotations. For more information, see the following annotations:

- @common:define Annotation
- @jc:conversation Annotation
- @jc:location Annotation
- @jc:wsdl Annotation

Related Topics

Service Broker Control

Worklist Control Annotations

This section includes information on Task control and Task Worker control annotations.

These annotations along with their attributes determine the default behavior of the Worklist controls. The Task and Task Worker controls may be configured during their lifetimes by calling methods in the `TaskControl` and `TaskWorkerControl` classes. The information contained in the annotations include the following:

- The object type that you must create to pass or return from methods at run time.
- The object type on which you must base the formatting of the text or the specified `java.lang.String` value you provide.
- All worklist annotation tags can receive string values in the format of the relevant object type.
- Some annotations accept enumerations, limited to a choice of defined values.
- Each annotation specifies which Task or Task Worker control it uses.
- Values that may be arrays show the base class with a suffix of [], for example, String[].

Topics Included in This Section

@jc:select Annotation
Accepts values to search for Tasks, including `TaskSelector` objects, and returns a set of Task IDs.

@jc:task Annotation
Assigns a Task to the Assignees List.

@jc:task-abort Annotation
Change the state of a task to ABORTED.

@jc:task-assign Annotation
Assigns a Task to the Assignees List.

@jc:task-claim Annotation
Sets a default user as having put a Task in a claimed state, as a claimant.

@jc:task-complete Annotation
Creates Worklist control methods that place Tasks in a completed state.

@jc:task-create Annotation
Creates Tasks.

@jc:task-delete Annotation
Creates Worklist control methods that delete Tasks.

@jc:task-event Annotation
Provides Task information for implementing callback method interfaces.

@jc:task-get-info Annotation
Creates Worklist control methods that return information from Tasks.

@jc:task-get-property Annotation
Creates methods that return the value of a Task property as a String.

@jc:task-get-property-name Annotation
Creates Worklist control methods that return Task property names.

@jc:task-get-request Annotation
Creates Worklist control methods that return Task request data.

@jc:task-get-response Annotation
Creates Worklist control methods that return Task response data.

@jc:task-remove-property Annotation
Remove properties with the name you specify from Tasks.

@jc:task-resume Annotation
Creates Worklist control methods that remove Tasks from a suspended state.

@jc:task-return Annotation
Create Worklist control methods that place Tasks in an assigned state using the original Assignees List.

@jc:task-set-property Annotation
Sets the value of a single Task property.

@jc:task-start Annotation
Creates Worklist control methods that place Tasks in a started state.

@jc:task-stop Annotation
Creates Worklist control methods that change Tasks from started to claimed states.

@jc:task-suspend Annotation
Creates Worklist control methods that place Tasks in a suspended state.

@jc:task-update Annotation
Updates one or more Task properties at a time for a method.

@jc:task-worker Annotation
Specifies that the control is a Task Worker control.

Related Topics

TaskControl Interface

Worklist Controls

@jc: advanced Annotation

Notations for advanced options.

Used by the Task control.

Syntax

```
ajc:advanced
      [can-be-reassigned="true|false"]
      [can-be-returned="true|false"]
      [can-be-aborted="true|false">]
      [claim-due-business-date="the business due date"]
      [completion-due-business-date="the completion business due date"]
```

```
[completion-user-calendar="the user completion due date calendar"]
[claim-user-calendar="the user claim due date calendar"]
[completion-calendar="the completion due date calendar"]
[claim-calendar="the claim due date calendar"]
```

Attributes

can-be-reassigned

A Boolean that determines whether the Task can be reassigned to a different Assignees List. If the value is set to `true`, the Task can be reassigned. If the value is set to `false`, the Task cannot be reassigned by the assignee or claimant of the Task.

can-be-returned

A Boolean that determines whether the Task can be returned to an assigned state, which specifies the users who are allowed to become the claimant. If the value is set to `true`, a Task can be returned to an assigned state. If the value is set to `false`, the Task cannot be returned by the assignee or claimant of the Task.

can-be-aborted

A Boolean that determines whether a Task can be aborted. If the value is set to `true`, a Task can be aborted. If the value is set to `false`, the Task cannot be aborted by the assignee or claimant of the Task.

claim-due-business-date

A string that sets due date for a Task to be claimed, using a business time duration.

completion-due-business-date

A string for setting the business duration that a Task is due to be completed.

completion-user-calendar

A string that directs a Task to use the calendar of the user whose name you specify for the completion date.

claim-user-calendar

A string that directs the Task to use the calendar of the user whose name you specify for the claim due date.

completion-calendar

A string that sets the calendar for the date that a Task is due to be completed.

claim-calendar

A string that indicates which business calendar to use for setting the date that a Task should be in a claimed state.

@jc:assignee Annotation

Assigns user and groups to Tasks.

Used by the Task control.

Syntax

```
@jc:assignee
      [user="user1,user2,user3,...,userN"]
      [group="group1,group2,group3,...,groupN"]
      [algorithm=["ToUser" | "ToUserInGroup" | "ToUsersAndGroups"]
```

Attributes

user

A string or string array. This setting consists of one or more user names for the Assignees List. You must separate each group name with a comma.

group

A string or string array. This annotation consists one or more group names for the Assignees List. You must separate each group name with a comma.

algorithm

A string for determining how to assign Tasks from users on the Assignees List. It must be one of the following values:

`ToUser`—assigns the Task to a user by name.

`ToUserInGroup`—assigns the Task to the user in the group that has the least work to perform.

`ToUsersAndGroups`—assigns the Task to any list of users, list of groups, or combination of users and groups.

@jc:select Annotation

Accepts values to search for Tasks, including `TaskSelector` objects, and returns a set of Task IDs.

Used by the Task Worker control.

Syntax

```
@jc:select
      [assigned-group="group(s) for assignee list"]
      [assigned-user="user(s) for assignee list"]
      [claimant="the claimant user"]
      [claim-due-date-after="search for Tasks with claim due dates after
      this value"
      [claim-due-date-before="search for Tasks with claim due dates before
      this value"
      [comment="the Task comment"]
      [completion-due-date-after="search for Tasks with completion due dates
      after this value"]
      [completion-due-date-before="search for Tasks with completion due
      dates before this value"]
      [creation-date-after="search for Tasks with creation dates after this
      value"]
      [creation-date-before="search for Tasks with creatoin dates before
      this value"]
      [max-priority="return Tasks with priorities less than or equal to this
      value"]
      [min-priority="return Tasks with priorities greater than or equal to
      this value"]
      [owner="search by Task owner"]
      [property-name="search by property name"]
      [property-value="search by the value of the setting for
      property-name"]
      [selector="a TaskSelector object to use for searching Tasks"]
      [states="search by Task state"]
      [task-id="search by Task ID"]
```

Attributes

assigned-group

A string or string array (`string []`) that specifies a search by groups on the Assignees List for a Task.

assigned-user

A string or string array (`string []`) that specifies a search by users on the Assignees List for a Task.

claimant

A string or string array (`string []`) that specifies a search by the claimant for a Task.

claim-due-date-after
Specifies a search by Tasks with a due date after the value you provide (`java.lang.Date`).

claim-due-date-before
Specifies a search by Tasks with a claim due date before the value you provide (`java.lang.Date`).

comment
A string that specifies a search by Task comments.

completion-due-date-after
Specifies a search by Tasks with a completion due date after the value you provide (`java.lang.Date`).

completion-due-date-before
Specifies a search by Tasks with a completion due date before the value you provide (`java.lang.Date`).

creation-date-after
Specifies a search by Tasks with a creation date after the value you provide (`java.lang.Date`).

creation-date-before
Specifies a search by Tasks with a creation date before the value you provide (`java.lang.Date`).

max-priority
Specifies a search by Tasks with no greater priority than the value you provide (`java.lang.Long`).

min-priority
Specifies a search by Tasks with no lesser priority of the value you provide (`java.lang.Long`).

owner
A string or string array (`string []`) that specifies a search by the Task owner.

property-name
A string that specifies a search by Tasks with a given property.

property-value
A string that specifies a search by Tasks with a given value for the property defined by `property-name`.

selector
A `TaskSelector` that specifies a search by the configuration of the `TaskSelector` object you provide for this value.

states

Specifies searching of Tasks by state. Values can be as follows:

A string or string array of valid state types, such as `completed` or `assigned`.

An integer or integer array representation of state types (`java.lang.Long`).

A `com.bea.wli.worklist.api.StateType` or `StateType` array.

task-id

A string or string array (`string []`) that specifies a search by the unique Task ID.

assigned-group

A string or string array (`string []`) that specifies a search by the groups on the Assignees Lists for a Task.

@jc:task Annotation

Assigns a Task to the Assignees List.

Used by the Task control.

Syntax

```
@jc:task
      [name="task name"]
      [owner="task owner user"]
      [description="task description"]
```

Attributes

name

A string specifying the Task name displayed in WebLogic Workshop.

owner

A string specifying the name of the Task owner.

description

A string for describing a Task to provide information.

@jc:task-abort Annotation

Change the state of a task to ABORTED.

Used by the Task and Task Worker controls.

Syntax

```
@jc:task-abort
      [enabled="true"]
```

Attributes

enabled

When set to `true`, this Boolean creates Worklist control methods that place Tasks in an aborted state.

@jc:task-assign Annotation

Assigns a Task to the Assignees List.

Used by Task and Task Worker controls.

Syntax

```
@jc:task-assign
      user="user1,user2,user3,...,userN"]
      [group="group1,group2,group3,...,groupN"]
      [algorithm=["ToUser" | "ToUserInGroup" | "ToUsersAndGroups"]
```

Attributes

algorithm

A string for determining how to assign or claim Tasks from users on the Assignees List. It must be one of the following values:

`ToUser`—claims the Task to a user by name.

`ToUserInGroup`—assigns the Task to the user in the group that has the least work to perform.

`ToUsersAndGroups`—assigns the Task to any list of users, list of groups, or combination of users and groups.

group

A string or string array. This annotation consists one or more group names for the Assignees List. You must separate each group name with a comma.

user

A string or string array. This setting consists of one or more user names for the Assignees List. You must separate each group name with a comma.

@jc:task-claim Annotation

Sets a default user as having put a Task in a claimed state, as a claimant.

Used by the Task Worker control.

Syntax

```
@jc:task-claim
      [enabled="true"]
      [claimant="user"]
```

Attributes

enabled

When set to `true`, this Boolean creates Worklist control methods that place Tasks in a claimed state.

claimant

A string that specifies the name of the claimant user.

@jc:task-complete Annotation

Creates Worklist control methods that place Tasks in a completed state.

Used by the Task Worker Control.

Syntax

```
@jc:task-complete
      [enabled="true"]
```

Attributes

enabled

When set to `true`, this Boolean creates Worklist control methods that place Tasks in a completed state.

@jc:task-create Annotation

Creates Tasks.

Used by the Task and Task Worker controls.

Syntax

```
@jc:task-create
      name="the Task name"
      [description="task description"]
      [comment="the text of a comment"]
      [request-mime-type="the mime type of the request"]
      [request=the data of the request]
      [response-mime-type="the mime type of the response"]
      [response=the data of the response]
      [priority="an integer for priority"]
      [owner="the new task owner"]
      [can-be-reassigned="true|false"]
      [can-be-aborted="true|false">]
      [can-be-returned="true|false"]
      [claim-due-business-date="the business due date"]
      [claim-due-date="the claim due date"]
      [completion-user-calendar="the user completion due date calendar"]
      [completion-calendar="the completion due date calendar"]
      [claim-user-calendar="the user claim due date calendar"]
      [claim-calendar="the claim due date calendar"]
      [completion-due-date="the completion due date"]
      [completion-due-business-date="the completion business due date"]
```

Attributes

can-be-aborted

A Boolean that determines whether a Task can be aborted. If the value is set to `true`, a Task can be aborted. If the value is set to `false`, the Task cannot be aborted by the assignee or claimant of the Task.

can-be-reassigned
A Boolean that determines whether the Task can be reassigned to a different Assignees List. If the value is set to `true`, the Task can be reassigned. If the value is set to `false`, the Task cannot be reassigned by the assignee or claimant of the Task.

can-be-returned
A Boolean that determines whether the Task can be returned to an assigned state, which specifies the users who are allowed to become the claimant. If the value is set to `true`, a Task can be returned to an assigned state. If the value is set to `false`, the Task cannot be returned by the assignee or claimant of the Task.

claim-calendar
A string that indicates which business calendar to use for setting the date that a Task should be in a claimed state.

claim-due-business-date
A string that sets due date for a Task to be claimed, using a business time duration.

claim-due-date
Sets the date that a Task is due to be in a claimed state (`java.lang.Date`).

claim-user-calendar
A string that directs the Task to use the calendar of the user whose name you specify for the claim due date.

comment
A string for setting for comments about the Task.

completion-calendar
A string that sets the calendar for the date that a Task is due to be completed.

completion-due-business-date
A string for setting the business duration that a Task is due to be completed.

completion-due-date
Sets the date that a Task is due to be completed (`java.lang.Date`).

completion-user-calendar
A string that directs a Task to use the calendar of the user whose name you specify for the completion date.

description
A string for describing a Task to provide information.

name
A string specifying the Task name displayed in WebLogic Workshop.

owner
A string specifying the name of the Task owner.

priority
This integer sets the magnitude of the priority of the Task. The default is 1.

request
The data of the Task request. Can be any type or format that can accept any serializable Java object that is imported and accessible to your control.

request-mime-type
A string that specifies the mime type of the `request` data. This annotation exists for information purposes only and does not provide any handling of data or validation.

@jc:task-delete Annotation

Creates Worklist control methods that delete Tasks.

Used by the Task Worker control.

Syntax

```
@jc:task-delete
      [enabled="true"]
```

Attributes

enabled
When set to `true`, this Boolean creates a Worklist control method for deleting Tasks.

@jc:task-event Annotation

Provides Task information for implementing callback method interfaces.

Used by the Task control.

Syntax

```
@jc:task-event
      event-type=
          "["abort"|"claim"|"claimExpire"|"complete"|"expire"
           |"resume"|"return"|"start"|"stop"|"suspend"]"
      [response="the Task response data"]
      [time="the time of the event"]
      [user="the user who changed the Task state"]
```

Attributes

`event-type`

A string (enumeration) that specifies the possible events that can trigger a callback method. It must be one of the following values:

`abort`
`claim`
`claimExpire`
`complete`
`expire`
`resume`
`return`
`start`
`stop`
`suspend`

response

A parameter for the response data of the Task. Can be any type or format that can accept any serializable Java object that is imported and accessible to your control.

time

Specifies text in `java.lang.Date` format that represents the time of the event listed for the `event-type`.

user

A string that specifies the active user who triggers the event listed for the `event-type`.

@jc:task-get-info Annotation

Creates Worklist control methods that return information from Tasks.

Used by the Task and Task Worker controls.

Syntax

```
@jc:task-get-info
      [enabled="true"]
```

Attributes

enabled

When set to `true`, this Boolean creates Worklist control methods that return information from Tasks.

Remarks

Methods in the Task Worker control with this annotation can have the following return types:

```
String, String[]
TaskInfo, TaskInfo[]
TaskInfoXML, TaskInfoXML[]
```

The `jc:task-get-info` annotation is used in correlation with `@jc:task-get-info`.

The return type determines the value returned by the method:

```
String → the taskId
TaskInfo → the TaskInfo object
TaskInfoXML → the taskInfoXML
```

If you need to select more than one Task with `@jc:task-get-info`, use an array instead. If you specify a return type for a single task and multiple tasks are selected, a run-time exception occurs when the code executes. If you are unsure, it is better to return an array, as shown in the following example:

```
/**
     * @jc:task-get-info enabled="true"
     * @jc:select task-id="{taskId}"
     */

    public TaskInfo getTaskInfo(String taskId);

    /**
     * @jc:task-get-info enabled="true"
     * @jc:select task-id="{taskIds}"
     */
```

```
public TaskInfoXMLDocument[] getTasksInfoXML(String[] taskIds);
```

@jc:task-get-property Annotation

Creates methods that return the value of a Task property as a string.

Used by the Task and Task Worker controls.

Syntax

```
@jc:task-get-property
      name="the property name"
```

Attributes

name

Creates methods that return the value of a Task property as a string.

@jc:task-get-property-name Annotation

Creates Worklist control methods that return Task property names.

Used by the Task Worker control.

Syntax

```
@jc:task-get-property-name
      [enabled="true"]
```

Attributes

enabled

When set to `true`, this Boolean creates Worklist control methods that return Task property names.

@jc:task-get-request Annotation

Creates Worklist control methods that return Task request data.

Used by the Task and Task Worker controls.

Syntax

```
@jc:task-get-request
      [enabled="true"]
```

Attributes

enabled

When set to `true`, this Boolean creates Worklist control methods that return Task request data.

@jc:task-get-response Annotation

Creates Worklist control methods that return Task response data.

Used by the Task and Task Worker controls.

Syntax

```
@jc:task-get-response
      [enabled="true"]
```

Attributes

enabled

When set to `true`, creates Worklist control methods that return Task response data.

@jc:task-remove-property Annotation

Removes a property with the name you specify from Tasks.

Used by the Task and Task Worker controls.

Syntax

```
@jc:task-remove-property
      [name="the name of a property to remove"]
```

Attributes

name

A string that removes properties with the name you specify from Tasks.

@jc:task-resume Annotation

Creates Worklist control methods that remove Tasks from a suspended state.

Used by the Task and Task Worker control.

Syntax

```
@jc:task-resume
      [enabled="true"]
```

Attributes

enabled

When this Boolean is set to `true`, creates Worklist control methods that remove Tasks from a suspended state.

@jc:task-return Annotation

Create Worklist control methods that place Tasks in an assigned state using the original Assignees List.

Used by the Task Worker control.

Syntax

```
@jc:task-return
      [enabled="true"]
```

Attributes

enabled

When set to `true`, this Boolean creates Worklist control methods that place Tasks in an assigned state using the original Assignees List.

@jc:task-set-property Annotation

Sets the value of a single Task property. To set the values for more than one property at a time, use `@jc:task-update Annotation`.

Used by the Task control.

Syntax

```
@jc:task-set-property
      [name="the name of a property to set"]
      [value="the value to set the property"]
```

Attributes

name

A string that specifies the name of the property.

value

A string that specifies the value to assign the property.

@jc:task-start Annotation

Creates Worklist control methods that place Tasks in a started state.

Used by the Task Worker control.

Syntax

```
@jc:task-start
      [enabled="true"]
```

Attributes

enabled

When set to `true`, this Boolean creates Worklist control methods that place Tasks in a started state.

@jc:task-stop Annotation

Creates Worklist control methods that change Tasks from started to claimed states.

Used by the Task Worker control.

Syntax

```
@jc:task-stop
      [enabled="true"]
```

Attributes

enabled

When set to `true`, this Boolean creates Worklist control methods that change Tasks from started to claimed states.

@jc:task-suspend Annotation

Creates Worklist control methods that place Tasks in a suspended state.

Used by the Task and Task Worker controls.

Syntax

```
@jc:task-suspend
      [enabled="true"]
```

Attributes

enabled

When set to true, this Boolean creates Worklist control methods that place Tasks in a suspended state.

@jc:task-update Annotation

Updates one or more Task properties at a time for a method. It offers the same attributes that are available through the `task-create` attribute, except that it offers response-related attributes and it does not offer the `description` attribute.

Used by the Task and Task Worker controls.

Syntax

```
@jc:task-update
      [comment="the text of a comment"]
      [request-mime-type="the mime type of the request"]
      [request=the data of the request]
      [response-mime-type="the mime type of the response"]
      [response=the data of the response]
      [priority="an integer for priority"]
      [owner="the new task owner"]
      [can-be-reassigned="true|false"]
      [can-be-aborted="true|false">]
```

```
[can-be-returned="true|false"]
[name="the Task name"]
[claim-due-business-date="the business due date"]
[claim-due-date="the claim due date"]
[completion-user-calendar="the user completion due date calendar"]
[completion-calendar="the completion due date calendar"]
[claim-user-calendar="the user claim due date calendar"]
[claim-calendar="the claim due date calendar"]
[completion-due-date="the completion due date"]
[completion-due-business-date="the completion business due date"]
```

Attributes

For a list of other available attributes, see `@jc:task-create Annotation`.

response

Specifies the data the Task sends back to the calling process. Can be any type or format that can accept any serializable Java object that is imported and accessible to your control.

response-mime-type

A string that provides a comment for the person implementing the control that indicates the response data type.

@jc:task-worker Annotation

Specifies that the control is a Task Worker control.

Used by the Task Worker control.

Syntax

`@jc:task-worker`

This annotation uses no attributes.

CHAPTER 16

Business Process Annotations

This section provides reference information for WebLogic Integration business process annotations (@jpd:*name_of_annotation*). A number of Web Service annotations are also supported in business processes (JPD files). Web Service annotations are of the format @jws:*name_of_annotation*. This section provides reference to the Web Service annotations supported in JPDs.

Topics Included in This Section

@jpd:ebxml Annotation
Specifies settings for participant business processes involved in exchanging ebXML business messages.

@jpd:ebxml-method Annotation
Specifies settings for methods in participant business processes involved in exchanging ebXML business messages.

@jpd:mb-static-subscription Annotation
Specifies the subscription parameters for a business process.

@jpd:process Annotation
Specifies settings for a business process.

@jpd:rosettanet Annotation
Specifies settings for participant business processes involved in exchanging RosettaNet business messages.

@jpd:selector Annotation
Precedes an XQuery definition in a business process (JPD) file. The XQuery definition can specify the dynamic callback properties for a Client Response node, or in the case of a Process or a Service Broker control, specifies the dynamic selection of subprocesses to call at run time.

@jpd:transform Annotation
Annotates a WebLogic Integration transformation control instance, which is instantiated automatically at run time.

@jpd:unexpected-message Annotation
Specifies settings that allow a business process to ignore a message received before the process flow encounters the node at which the message is expected.

@jpd:version Annotation
Specifies how to invoke sub processes when different versions of the parent process exist.

@jpd:xml-list Annotation
Annotates business process variable of Untyped XML—XmlObjectList.

@jpd:xquery Annotation
Precedes the global XQuery definitions in a business process (JPD) file.

@common Annotations
Describes the @common annotations supported by WebLogic Integration business processes.

@jws (Web Service) Annotations
Describes the @jws annotations supported by WebLogic Integration business.

General Properties
Describes a set of general properties that are displayed in the Property Editor for all the nodes in a business process.

Variable Properties
Specifies the properties for business process variables.

Control Properties
Displays the properties for business process controls: **Control Send**, **Control Receive**, **Control Send with Return**.

@jpd:ebxml Annotation

Specifies annotations for participant business processes involved in exchanging ebXML business messages.

Syntax

```
@jpd:ebxml
      protocol-name="ebXML"

      ebxml-service-name="serviceName"

      ebxml-action-mode="default" | "non-default"
```

Attributes

protocol-name

The protocol name, which is always `ebXML`.

ebxml-service-name

The name of the ebXML service associated with this business process and defined in the TPM repository. Defaults to the name of the business process file. The name specified here must match the service name specified on the initiator side (for example, in the `ebxml-service-name` annotation on the ebXML control in the initiator business process). You provide this service name to your trading partners.

Note: This service name corresponds to the `eb:Service` entry in the ebXML message envelope.

ebxml-action-mode

Action mode for this business process. Determines the value specified in the `eb:Action` element in the message header of the ebXML message, which becomes important in cases where multiple message exchanges occur within the same conversation. One of the following values:

- `default`—Sets the `eb:Action` element to `SendMessage` (default name).
- `non-default`—Sets the `eb:Action` element to the name of the method (on the ebXML control) that sends the message in the initiator business process. For one-way conversations, this name must match the method name of the **Client Request** node in the corresponding participant business process. For round trip conversations, the method name in the callback interface for the Client Callback node in the participant business process must match the method name (on the ebXML control) in the control

callback interface in the initiator business process. Using `non-default` is recommended to ensure recovery and high availability.

Note: If unspecified, the `ebxml-action-mode` is set to `non-default`.

Remarks

None.

Related Topics

ebXML Control

WebLogic Workshop Reference

Introducing Trading Partner Integration at
`http://edocs.bea.com/wli/docs81/tpintro/index.html`

@jpd:ebxml-method Annotation

Specifies settings for methods in participant business processes involved in exchanging business messages via ebXML.

Syntax

```
@jpd:ebxml-method
      envelope="{env}"
```

Attributes

envelope

Represents the message envelope in an incoming ebXML business message. You can rename the default value (`env`) as long as it matches the name of the parameter specified in the method.

Remarks

Use this annotation with the `request` method in a client request nodes to assign the ebXML envelope of an incoming message.

Example

The following example code shows an implementation of `request` in a participant business process that retrieves the ebXML envelope from a business message.

```
/**
*@jpd:ebxml-method envelope="{env}"
*/
public void request(XmlObject payload, XmlObject env) {
}
```

Related Topics

ebXML Control

WebLogic Workshop Reference

Introducing Trading Partner Integration at
`http://edocs.bea.com/wli/docs81/tpintro/index.html`

@jpd:mb-static-subscription Annotation

Specifies the subscription parameters for a business process that is started as the result of receiving a message from a Message Broker channel to which the process is subscribed.

Business processes are started by messages. The first activity in a business process (that is, the first child of the `<process>` tag) must be either a Client Request node or an Event Choice node. When the client invokes this operation, an instance of the business process is started.

A special case of message-started processes is when the Message Broker starts the process as a result of a subscription to a Message Broker channel. The subscription parameters are defined by annotating the starting clientRequest method to indicate that this process is invoked by the Message Broker when it delivers the message.

See Note About Static and Dynamic Subscriptions.

Syntax

```
@jpd:mb-static-subscription
      channel-name="/prefix/xxxx/Name"
      [xquery="xquery"]
```

```
[filter-value-match="myvalue"]
[message-metadata="{x1}"] [message-body="{x0}"]
```

Attributes

channel-name

Required. The name of the Message Broker channel to which the business process subscribes. Channel files (*filename*.channel) define the Message Broker channels available in a WebLogic Integration application. Channel files must be placed in a **Schemas** project in your application. To learn how to create Message Broker channels, see How Do I: Create Message Broker Channels?

xquery

Optional. Specifies the xquery to use for filtering. Element variables are named by Filter-body and Filter-header. The element variables are named by the message-body and message-metadata attributes.

filter-value-match

Optional string constant. (Required if xquery is specified.) The value of the filter-value-match attribute is compared against the results of the xquery.

message-metadata

Optional string constant. This attribute maps a named parameter in the xquery to the SOAP headers of an incoming message.

message-body

Optional string constant. This attribute maps a named parameter in the xquery to the XML body of an incoming message.

Note About Static and Dynamic Subscriptions

There are two categories of subscriptions to a Message Broker channel for WebLogic Integration business processes:

- You can create an instance of a Message Broker Subscription control and design the subscription to a Message Broker channel using the control.
- For the special case of a process that is started as a result of a subscription to a Message Broker channel, you design the subscription on the process' Start node. (See How Do I: Subscribe to Message Broker Channels?)

Subscriptions on a **Start** node in a business process are called ***static*** subscriptions because WebLogic Integration is aware of the subscription when the business process (JPD) is deployed. At deployment time, WebLogic Integration updates its in-memory tables with the subscription information. Tables remain updated until the business process is undeployed.

Subscriptions to Message Broker channels that are defined at a **Control** node, which communicates with a Message Broker Subscription control, are called ***dynamic*** subscriptions. This means that WebLogic Integration is not aware of the subscription when the business process (JPD) is deployed (because the subscription is embedded in the JPD code). When the Message Broker Subscription control node is executed at run time, WebLogic Integration update the tables with the subscription information. The dynamic part for dynamic subscriptions refers to:

- The subscription state (subscribed or not)
- The filter value for the filter expression

Related Topics

Message Broker Control Annotations

Subscription Start (Asynchronous)

Subscription Start (Synchronous)

Event Choice Start

How Do I: Create Message Broker Channels?

@jpd:process Annotation

Contains the process logic settings for a business process.

Syntax

```
@jpd:process [binding="webservice | ebxml | rosettanet"] process::
      <process name="processname" [freezeOnFailure="true | false”]
      [onSyncFailure="rethrow | rollback"] [retryCount="<count>"]
      [retryDelay="<delay>"]>
```

Note: See also the stateless attribute in the following section.

Attributes

binding

This property specifies whether the business process uses the Web service, ebXML, or RosettaNet protocol. The default value is **webservice**. If your business process is an **ebXML** or a **RosettaNet** process, select **ebxml** or **rosettanet**. In keeping with your selection in the **Property Editor**, an attribute is written to the `@jpd: process` annotation in the source code. For example:

```
@jpd:process binding="rosettanet" process::
```

To learn about ebXML and RosettaNet business processes, see Building ebXML Participant Business Processes and Building RosettaNet Participant Business Processes.

name

This is the name of your business process, which is displayed throughout the WebLogic Workshop application, including the WebLogic Integration Administration Console. You can change the name by clicking on this property in the Property Editor and entering a new name.

freeze on failure

When a business process fails and there is no exception handler configured to handle the exception thrown, the business process is placed into an aborted state and no recovery is possible. However, if the business process is configured to *freeze on failure*, the business process rolls back to the last commit point and the state is persisted if it fails. The process can then be restarted from the WebLogic Integration Administration Console. To configure a business process to freeze on failure: select **true** from the **freeze on failure** drop-down menu.

To learn how to unfreeze business processes in the WebLogic Integration Administration Console, see Process Instance Monitoring in *Managing WebLogic Integration Solutions* at `http://edocs.bea.com/wli/docs81/manage/processmonitoring.html`.

on sync failure

This property only applies to your process if it is configured to be a synchronous subprocess; it is ignored for any other business processes. If a synchronous subprocess fails, the default behavior is to mark it as **rollback**, which causes both the subprocess and the parent process to rollback. However, if the **on sync failure** property is set to **rethrow**, only the subprocess is rolled back. To learn more about synchronous subprocesses and the **on sync failure** property, see "Working with Subprocesses" in Building Synchronous and Asynchronous Business Processes.

retry count

Specify how many times, after the first attempt, the process engine should try to execute the business process.

If your business process contains an asynchronous client request node or multiple client request nodes, any one of which is asynchronous, then you can set the **retry count** for the business process. You cannot set the **retry count** property for business processes that contain *only* synchronous client request nodes (that is, **Client Request with Return** nodes).

retry delay

Specify the amount of time (in seconds) you want to pass before a retry is attempted.

If your business process contains an asynchronous client request node or multiple client request nodes, any one of which is asynchronous, then you can set the **retry delay** for the business process. You cannot set the **retry delay** property for business processes that contain *only* synchronous client request nodes (that is, **Client Request with Return** nodes).

stateless

This property is read only; it cannot be edited. It specifies whether your business process is stateless (property displays **true**) or stateful (property displays **false**). To learn more about stateless and stateful business processes, see Building Stateless and Stateful Business Processes.

@jpd:rosettanet Annotation

Specifies settings for participant business processes involved in exchanging RosettaNet business messages.

Syntax

```
@jpd:rosettanet
      protocol-name="rosettanet"
      protocol-version="1.1" | "2.0"
      pip-name="pipName"
      pip-version="pipVersion"
      pip-role="pipRole"
```

Attributes

protocol-name
The protocol name, which is always `rosettanet`.

protocol-version
RNIF (RosettaNet Implementation Framework) version. One of the following values:
- `1.1`
- `2.0`

pip-name
RosettaNet PIP code, such as `3B2`. Must be a valid PIP code as defined in `http://www.rosettanet.org`.

pip-version
RosettaNet PIP version. Must be a valid version number associated with the PIP.

pip-role
RosettaNet role name for the recipient as defined in the PIP specification, such as Seller, Supplier, Shipper, and so on. A PIP request might be rejected if an incorrect value is specified.

Remarks

None.

Related Topics

RosettaNet Control

WebLogic Workshop Reference

Introducing Trading Partner Integration at
`http://edocs.bea.com/wli/docs81/tpintro/index.html`

@jpd:selector Annotation

The @jpd:selector annotation precedes an XQuery definition in a business process (JPD) file. The XQuery definition can specify the dynamic callback properties for a Client Response node, or in the case of a Process or a Service Broker control, the XQuery allows dynamic selection of subprocesses at run time. In other words, configuration of the business process at run time allows one of multiple subprocesses to be called.

Syntax

```
@jpd:selector xquery::
```

Remarks

The XQuery definition can specify the dynamic callback properties for a Client Response node, or in the case of a Process or a Service Broker control, the XQuery allows dynamic selection of subprocesses at run time.

For a Process control or a Service Broker control, the dynamic selector allows you to configure a lookup property based on a LookupControl or TPM function. You can then configure your business process in the WebLogic Integration Administration Console such that at run time, the security of the callback to the client is handled differently, based on the value of the lookup property that you specified in the dynamic selector.

To learn about adding dynamic callback properties for Client Response nodes in a business process, see "Adding Dynamic Callback Properties" in Sending Messages to Clients.

To design a dynamic selector for a Process control, see Editing and Testing a Dynamic Selector.

To design a dynamic selector for a Service Broker control, see Editing and Testing a Dynamic Selector.

@jpd:transform Annotation

The @jpd:transform annotation annotates a WebLogic Integration data transformation control instance, which is instantiated automatically at run time.

Syntax

```
@jpd:transform
```

Remarks

None.

Related Topics

Guide to Data Transformation

Tutorial: Building Your First Data Transformation

Adding Instances of Controls to Your Business Process Project

@jpd:unexpected-message Annotation

Allows a business process to ignore messages received by a business process for a Control Receive node or a Client Request node (in positions in the business process other than at the Start node) before the process flow encounters the node at which the message is expected.

Syntax

```
@jpd:unexpected-message
      action = "save | discard"
```

Attributes

action

Specifies the action to be taken for messages that are received by a business process before the process flow encounters the node at which the message is expected. Possible values are `save` (the default) and `discard`.

Remarks

Business processes can include Control Receive or Client Request nodes, at which the process flow waits at run time for delivery of a message before continuing. By default, messages that arrive before they are expected—that is, before the business process encounters the Control Receive or Client Request node in the process flow—are buffered and are delivered when the process is ready to receive them (that is, when the Control Receive or Client Request node is encountered). You can design your process such that any such *unexpected* (early) messages are discarded. This enables the Control Receive and Client Request nodes to ignore messages that arrived and were buffered by the business process, but that are no longer relevant to the process. The `jpd:unexpected-message` annotation is available for Control Receive nodes and Client Request nodes in positions other than the Start node. This annotation gives you the ability to control this behavior on a node-by-node basis at design time.

To Specify the jpd:unexpected-message Annotation

1. Switch your view of the business process to the Source View.
2. Click on the Control Receive or Client Request node's method header. The **Property Editor** displays the **unexpected-message** property. Note that the `action` attribute is specified as the default value: `save`.

3. In the Property Editor, change the specification from **save** to **discard**. The annotation is written into the JPD source code, immediately preceding the method header:

```
/**
* @jpd:unexpected-message action="discard"
*/
```

Related Topics

Interacting With Clients

Interacting With Resources Using Controls

@jpd:version Annotation

Specifies how to invoke sub processes when different versions of the parent process exist.

Syntax

```
@jpd:version
      strategy="loosely-coupled |tightly-coupled"
```

Attributes

loosely-coupled

Select **loosely-coupled** if you want the subprocess version to be set at the time that the sub process is invoked.

tightly-coupled

Select **tightly-coupled** if you want the subprocess version to be set at the time the parent process is invoked.

Remarks

When a business process is displayed in the Design View or the Source View, the Property Editor displays the version property. By default the strategy is specified as loosely coupled. You can change the specification from the strategy drop-down menu in the Property Editor. The annotation is written into the JPD source code—for example:

```
::
* @jpd:version strategy="loosely-coupled"
*/
```

Related Topics

Versioning Business Processes

@jpd:xml-list Annotation

The @jpd:xml-list annotation annotates business process variables of Untyped XML, specifically variables of type XmlObjectList.

Syntax

```
@jpd:xml-list
```

Remarks

An XMLObjectList data type represents a sequence of non-typed XML format data. In other words, this data type represents a sequence of XML elements (a set of repeating elements) that is not valid against an XML Schema.

Related Topics

Business Process Variables and Data Types

For an example of working with XmlObjectList data types, see Step 9. Create Quote Document in the *Tutorial: Building Your First Business Process*.

@jpd:xquery Annotation

Precedes the global XQuery definitions in a business process (JPD) file. The definitions are in scope for all XQueries in the business process. Namespaces are declared in the xquery prologue.

Syntax

```
@jpd:xquery prologue ::
      xquery_namespaces_and_function_definitions
      ::
```

Where *xquery_namespaces_and_function_definitions* represents the XQuery namespaces and function definitions specified in the annotation.

Remarks

In WebLogic Workshop, expand the region of code labeled **/** Process Language*/** in the Source View for a business process (JPD) to see the Java code that describes the business process you created in the Design View. XQuery statements are written to the JPD file in this region.

The XQuery statements are preceded by the following annotation:

```
@jpd:xquery prologue::
```

For example, when you select a repeating XML node using the **For Each** node builder, as described in Designing For Each Nodes, an XQuery expression is created in your JPD file. The expression returns the set of XML elements over which the **For Each** node iterates. XQuery expressions are also written in your JPD file when you create conditions on **Decision** nodes. XQuery expressions also define the transformations you create between disparate data types using the mapping tool.

Related Topics

To learn about XQueries, see *XQuery Reference*.

To learn about **For Each** nodes and **Decision** nodes, see the following topics:

Looping Through Items in a List

Defining Conditions For Branching

To learn data transformations, see the following topics:

Guide to Data Transformations

Tutorial: Building Your First Data Transformation

@common Annotations

Common annotations (those annotations available to more than one type of file in WebLogic Workshop) are also supported in JPD files. They are of the format @common:*name_of_annotation*.

The following links provide reference to the common annotations used in business processes:

Common Annotations
include . . .

@common:control	Indicates that the object annotated by this annotation is a WebLogic Workshop control in a JCX file.
@common:message-buffer	Specifies that there should be a queue between the component's implementation code and the message transport wire for the specified method or callback.
@common:security	Specifies the security configuration for a class or an individual method within a class.
@common:xmlns	Defines an XML namespace prefix for use elsewhere in the component class.

To learn about all the @common annotations, see Common Annotations.

@jws (Web Service) Annotations

A number of Web Service annotations are supported in WebLogic Integration business processes (JPD files). Web Service annotations are of the format @jws:*name_of_annotation*.

The following links provide reference to the Web Service annotations used in business processes:

Java Web Service Annotations
include . . .

@jws:conversation-lifetime	Specifies the maximum age and the maximum idle time for a service's conversations. See also, Managing Conversations
@jws:handler	Specifies SOAP message handlers for a web service.
@jws:location	Specifies the URL at which a Web service control accepts requests for each supported protocol.
@jws:protocol	Specifies the protocols and message formats a Web service can accept or a Web Service control will send to the service it represents.

@jws:reliable	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.
@jws:wsdl	Specifies a WSDL file that is implemented by a web service.

To learn about all the @jws annotations, see Web Service Annotations.

Note: WS-Security policy (WSSE) files are not supported for business processes (JPDs). Therefore, the following annotations are not supported for JPD files: @jws:ws-security-callback and @jws:ws-security-service. To learn more about WSSE in Web services, see WS-Security Policy File Reference.

General Properties

General Properties are displayed in the **Property Editor** for all the nodes in a business process.

For a given node, the following properties are displayed:

name

This is the name of the node selected in WebLogic Workshop. You can change the name of the associated node by clicking on this property and entering a new name. Note that the name is changed in the **Design View** in keeping with the change you make in the **Property Editor**.

Note that for the scenario in which the Start node of a business process is selected in the **Design View**, the name of the business process (JPD file) is displayed.

notes

Enter any notes that you want associated with your business process by clicking on this property to open the Property Text Editor. Notes entered in the editor will also be displayed in the WebLogic Integration Administration Console at run time.

Related Topics

Guide to Building Business Processes

Variable Properties

When you click on any business process variable in the **Data Palette**, its **Variable Properties** are displayed in the **Property Editor**.

For a given variable, the following properties are displayed:

name
Displays the name of the variable. You can double click the name and change it in the Property Editor.

type
Displays the data type for the variable. This is a read-only field.

value

Displays the initial value for the variable if one has been assigned. Click **...** in the **value** field to access the **Source View** for the variable in question—you can edit the initial value assigned to a given variable in the **Source View**.

Related Topics

Business Process Variables and Data Types

Creating Variables

Control Properties

When you click any control node of a business process (**Control Send**, **Control Receive**, **Control Send with Return**) in the **Design View**, its **Properties** are displayed in the **Property Editor** in a **control** group. The properties in the **Property Editor** are read-only.

The properties that are displayed in the **Property Editor** depend on which type of control you select in the **Design View**:

- Control Send
- Control Receive
- Control Send with Return

Control Send

target control
The control to which this node is sending a message.

target method
The method being invoked on the target control.

Control Receive

source control
The control from which this node is receiving a message.

callback method
The callback method on the source control.

Control Send with Return

target control
The control to which this node is sending a message.

target method
The method being invoked on the target control.

Related Topics

@common:control Annotations

Using Integration Controls

@common:context Annotations

CHAPTER 17

Data Transformation Annotations

This section provides reference information for WebLogic Integration data transformation annotations (@dtf:*name_of_annotation*).

Topics Included in This Section

@dtf:xquery Annotation

The `@dtf:xquery` annotation with the `prologue` attribute defines global XQuery functions that can be used by the queries specified in a DTF file. (The call to the function is made in the XQ file that contains the query.) In addition, the `@dtf:xquery` annotation with the `prologue` attribute specifies XQuery namespaces that can be used within the scope of the prologue of the DTF file.

Note: The `@dtf:xquery` annotation is optional.

Syntax

The syntax of the `@dtf:xquery` annotation is dependant on the number of lines in the annotation as described in the following options:

- For XQuery namespace and function definitions on multiple lines, the following syntax is used:

```
@dtf:xquery prologue ::
      xquery_namespaces_and_function_definitions
      ::
```

  Where *xquery_namespaces_and_function_definitions* defines the XQuery namespace and function definitions specified on multiple lines in the annotation.

- For XQuery namespaces and function definitions on a single line, the following syntax can be used:

```
@dtf:xquery prologue="xquery_namespace_and_function_definition"
```

  Where *xquery_namespace_and_function_definition* defines the XQuery namespace and function definitions specified on a single line in the annotation.

Note: The only attribute supported for the `@dtf:xquery` annotation is the `prologue` attribute.

Remarks

The following example `@dtf:xquery` annotation defines a namespace and a function on multiple lines:

```
/**
 * @dtf:xquery prologue::
 * declare namespace env = "http://www.example.org/envelope"
 *
 * define function wrapInEnvelope(element $e) returns element
 * {
```

```
*     <env:envelope>
*       {$e}
*     </env:envelope>
* }
* ::
*/
```

Caution: The namespaces definitions must be listed before the function definitions in the annotation.

Note: The namespace(s) defined in the prologue can only be used in the scope of the prologue (as shown in the preceding example) and cannot be referred to from the query (stored in the XQ file) while function definitions can be used in queries (stored in the XQ file).

You can add the `@dtf:prologue` annotation using one of the following procedures:

- Using the **Property Editor**:
  a. Open the DTF file in the **Design View**.
  b. Select the DTF name in the **Design View**.
  c. Click **prologue** in the **Property Editor**.
  d. Click **...** next to the **prologue** field.

     The **Property Test Editor** is displayed.
  e. In the **Property Test Editor**, enter the desired namespaces and function definitions and click **OK**.
- Open the DTF file in the **Source View** and enter the `@dtf:prologue` annotation after the last `import` statements but before the class definition statement for the DTF as shown highlighted in bold in the following example code listing:

```
package examples;
import com.bea.xml.XmlObject;
import com.bea.xml.XmlObjectList;
/**
 * @dtf:xquery prologue::
 * define function debugFunction(xs:string $str) returns xs:string {
 * concat("DTF XQuery: ", $str)
 * }
 * ::
 */
public abstract class examplesimplements
com.bea.transform.TransformSource
```

```
{
    static final long serialVersionUID = 1L;
    /**
     * @dtf:transform xquery-ref="myTransMethod.xq"
     * @dtf:schema-validate return-value="false" parameters="false"
     */
    public abstract stockquotes.PriceQuoteDocument
myTransMethod(stockquotes.PriceQuoteDocument PriceQuoteDoc);
    /**
     * @dtf:transform xquery-ref="mySecondTransMethod.xq"
     * @dtf:schema-validate return-value="false" parameters="false"
     */
     public abstract org.example.quote.QuoteDocument
mySecondTransMethod(noNamespace.SupplierDocument SupplierDoc);
}
```

The `@dtf:xquery` annotation is a class level annotation. A function defined in the DTF class can be used by all the queries defined in the DTF file. In the preceding example, the `debugFunction` function can be used in both the `myTransMethod` query (stored in the `myTransMethod.xq` file) and the `mySecondTransMethod` query (stored in the `mySecondTransMethod.xq` file).

Related Topics

To learn more see XQuery Prologs.

To learn about the XQuery language, see the XQuery 1.0: An XML Query Language Specification - W3C Working Draft 16 August 2002 available from the W3C web site at the following URL:

`http://www.w3.org/TR/2002/WD-xquery-20020816/`

To learn data transformations, see the following topics:

Guide to Data Transformations

Tutorial: Building Your First Data Transformation

@dtf:transform Annotation

The `@dtf:transform` annotation annotates the XQuery and XSLT abstract methods in a DTF file. During run time, a business process (JPD) invokes the abstract method which in turn invokes the associated XSLT or query (written in the XQuery language.)

Syntax

The `@dtf:transform` annotation can specify one and only one of the following attributes:

- xquery-ref Attribute
- xquery Attribute
- xslt-ref Attribute
- xslt Attribute

xquery-ref Attribute

```
@dtf:transform
      xquery-ref="myTrans.xq"
```

Where *myTrans* specifies the XQ file that contains the query written in the XQuery language. During run time, if the abstract method is invoked by a business process, the associated query stored in the XQ file is invoked by the XQuery engine.

The following example code segment shows a `@dtf:transform` annotation specifying the `xquery-ref` attribute:

```
/**
 * @dtf:transform xquery-ref="myXQueryMethod.xq"
 * @dtf:schema-validate return-value="false" parameters="false"
 */
public abstract noNamespace.SupplierDocument
myXQueryMethod(noNamespace.PurchaseOrderDocument PurchaseOrderDoc);
```

During run time, if the `myXQueryMethod` abstract method is invoked by a business process, the query stored in the `myXQueryMethod.xq` is invoked by the XQuery engine.

A `dtf:transform` annotation with the `xquery-ref` attribute is generated when you create a transformation method in a DTF file and configure the transformation method to be a XQuery method. To learn more, see Creating a Transformation File and a Transformation Method in the *Guide to Data Transformation*.

xquery Attribute

```
@dtf:transform xquery ::
      xquery_code
      ::
```

Where *xquery_code* specifies the inline query (written in the XQuery language) to be invoked by the business process during run time.

The following example code segment shows a `@dtf:transform` annotation specifying the `xquery` attribute:

```
/**
 * @dtf:transform xquery ::
 * <Supplier>
 *  <id>{ data($PurchaseOrderDoc/partId) }</id>
 * </Supplier>
 * ::
 */
public abstract noNamespace.SupplierDocument
myXQueryInlineMethod(noNamespace.PurchaseOrderDocument PurchaseOrderDoc);
```

The unbound source variables used in the XQuery code must correspond in name and type to the Java transformation (abstract) method parameters defined in the DTF file. (An unbound source variable is a variable used in the XQuery code that is not first defined by an XQuery `let` statement.) As shown highlighted in bold in the preceding example code listing, the source `$PurchaseOrderDoc` variable used in the inline XQuery code is defined as the parameter in the `myXQueryInlineMethod` transformation method as type: `PurchaseOrderDocument`.

You can add the `@dtf:transform` annotation with the `xquery` attribute using one of the following procedures:

- Open the DTF file in the **Source View** and enter the `@dtf:transform` annotation with the `xquery` attribute as shown in the presiding example code listing.
- Using the **Property Editor**:
  a. Open the DTF file in the **Design View**.
  b. In the **Design View**, select a Transformation (abstract) method.
  c. In the **Property Editor**, under **transform**, click the **xquery** attribute.

     The field to the right of the **xquery** attribute becomes active.
  d. Paste the query into the **xquery** field.

xslt-ref Attribute

```
@dtf:transform
      xslt-ref="myXSL.xsl"
```

Where *myXSL* specifies the XSL file that contains the eXtensible Stylesheet Language code. XSLT is an older language defined by the W3C that supports the use of style sheets for the conversion of XML data.

The following example code segment shows a `@dtf:transform` annotation specifying the `xslt-ref` attribute:

```
/**
 * @dtf:transform xslt-ref="../myXSLT.xsl"
 */
public abstract com.bea.xml.XmlObject myXsltMethod(com.bea.xml.XmlObject
source);
```

During run time, if the `myXsltMethod` abstract method is invoked by a business process, the associated XSLT stored in the `myXSLT.xsl` is invoked by the XSLT processor.

The return type and the first parameter of the method must be `XmlObject` or a typed XMLBeans class. The remaining parameters are treated as XSLT parameters.

A `dtf:transform` annotation with the `xslt-ref` attribute is generated when you create a transformation method in a DTF file and configure the transformation method to be a XSLT method. To learn more, see Transforming Data Using XSLTs in the *Guide to Data Transformation*.

xslt Attribute

```
@dtf:transform xslt ::
      XSL_Code
      ::
```

Where *XSL_code* specifies the inline XSL code to be invoked by the business process during run time.

The return type and the first parameter of the method must be `XmlObject` or a typed XMLBeans class. The remaining parameters are treated as XSLT parameters.

You can add the `@dtf:transform` annotation with the `xslt` attribute using one of the following procedures:

1. Open the DTF file in the **Source View** and enter the `@dtf:transform` annotation with the `xslt` attribute above the Java code defining the XSLT transformation method.
2. In the **Property Editor**, open the DTF file in the **Design View**.
3. In the **Design View**, select a Transformation (abstract) method.

4. In the **Property Editor**, under **transform**, click the **xslt** attribute.
5. The field to the right of the **xslt** attribute becomes active.
6. Paste the XSL code into the **xslt** field.

Related Topics

Guide to Data Transformation

Tutorial: Building Your First Data Transformation

@dtf:schema-validate Annotation

Specifies if the source parameters and/or the return value should be schema validated. The schema-validate annotation is an optional annotation associated with `@dtf:transform` methods.

Syntax

```
@dtf:schema-validate
      return-value="true" | "false"
      parameters="true" | "false"
```

Attributes

return-value

If this parameter is set to `true`, during run time, if the return type has an associated schema, it will be validated against its schema type after the transformation is executed. A typed XML return value will be schema validated against its XML Schema and a typed non-XML return value will be validated against the schema in the MFL file. A return types which is untyped or is a Java primitive will not be validated because it does not have an associated schema.

parameters

If this parameter is set to `true`, during run time, the source parameters that have an associated schema will be validated against their schema types before the transformation is executed. All typed XML parameters will be schema validated against their XML Schema and typed non-XML parameters will be validated against the schema in the MFL file. Source parameters which are untyped or are Java primitives will not be validated because they do not have an associated schema.

Remarks

In the following example, the `@dtf:schema-validate` annotation defines that during run time, when the transformation method `myXQueryMethod` is executed, the return value will be schema validated but the source parameters of the transformation method will not, as shown highlighted in bold in the following code segment:

```
/**
 * @dtf:transform xquery-ref="myXQueryMethod.xq"
 * @dtf:schema-validate return-value="true" parameters="false"
 */
public abstract noNamespace.SupplierDocument
myXQueryMethod(noNamespace.PurchaseOrderDocument PurchaseOrderDoc);
```

Related Topics

See "Schema Validating During Run Time" in Validating in the *Guide to Data Transformation*.

@dtf:transform Annotation

@dtf:xquery-function Annotation

Specifies that a user-defined Java method (non-abstract) method in a DTF file can be invoked from queries (written in the XQuery language).

Syntax

```
@dtf:xquery-function
```

Remarks

In the following example, the `@dtf:xquery-function` annotation defines that the Java method `calculateTotalPrice` that can be invoked from a query during run time:

```
/**
 * @dtf:xquery-function
 */
public float calculateTotalPrice(float taxRate, int quantity, float price,
boolean fillOrder)
{
       float totalTax, costNoTax, totalCost;
```

```
        if (fillOrder)
        {
               // Calculate the total tax
               totalTax = taxRate * quantity * price;
               // Calculate the total cost without tax
               costNoTax = quantity * price;
               // Add the tax and the cost to get the total cost
               totalCost = totalTax + costNoTax;
        }
        else
        {
               totalCost = 0;
        }
        return totalCost;
}
```

Related Topics

See “Invoking Functions or Operators in a Query” in Modifying Links Using the Target Expression Tab in the *Guide to Data Transformation*.

See “Create a User-Defined Java Method to Invoke From the Join Query” in Step 4: Mapping Repeating Elements—Creating a Join in the *Tutorial: Building Your First Data Transformation*.

General Properties

General Properties are displayed in the **Property Editor** for all the methods in a DTF file.

For a given method, the **name** property is displayed. The **name** property defines the name of the selected DTF method. You can change the name of the associated method by clicking on this property and entering a new name. Note that the name is changed in the **Design View** in keeping with the change you make in the **Property Editor**.

Related Topics

Guide to Data Transformation

@dtf:transform Annotation