BEA WebLogic Integration™

Reference

Version 8.1 Service Pack 2
Document Date: December 2003

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

1 Reference

This section provides reference information for WebLogic Integration.

Topics Included in This Section

Chapter 2, "Java Class Conversion"
Describes which fields of Java classes can be used in transformations and variables.

Chapter 3, "JpdContext Interface"
Describes the execution context for business processes.

Chapter 4, "Business Process Annotations"
Describes the `@jpd:` annotations used in business processes.

Chapter 5, "File Control Interfaces and Annotations"
Describes the File control interface and the annotations used in business processes.

Chapter 6, "Email Control Interfaces and Annotations"
Describes the Email control interface and the annotations used in business processes.

Chapter 7, "WLI JMS Control Interface and Annotations"
Describes the WLI JMS control interface and includes links to the annotations used in business processes.

Chapter 8, "Message Broker Controls Interfaces and Annotations"
Describes the Message Broker Publish and Subscribe interfaces.

Chapter 9, “Application View Control Interface”
Describes the Application View control interface and includes links to the annotations used in business processes as documented in Application View Control Annotations.

Chapter 10, “Process Control Interface”
Describes the Process control interface and includes links to the annotations used in business processes.

Chapter 11, “Service Broker Control Interface”
Describes the Service Broker control interface and includes links to the annotations used in business processes.

Chapter 12, “Message Attachments”
Describes the API for accessing attachments in ebXML and RosettaNet business messages.

Chapter 13, “Worklist Control Interfaces and Annotations”
Describes the Worklist control interface and the annotations used in business processes.

Chapter 14, “TPM Control Interface”
Describes the TPM control interface.

Chapter 15, “ebXML Control Interface and Annotations”
Describes the ebXML control interfaces and annotations.

Chapter 16, “RosettaNet Control Interface and Annotations”
Describes the RosettaNet control interface and annotations.

Application View Control Annotations
Describes the Application View control annotations.

2 Java Class Conversion

Java classes can be used as input and output types for transformations. Also, business process variables can be created from Java classes. However, not all the fields that can make up a Java class are supported. This section describes which fields of a Java class are supported.

When you use a Java class or primitive in a transformation, WebLogic Integration converts it into an internal representation of XML Schema data types. Java fields that are not supported are ignored. WebLogic Integration then uses this internal representation to transform your Java classes to and from XML and non-XML (MFL) documents

Note: This internal representation of the XML Schema is not exposed in WebLogic Workshop.

Note: The mapper functionality of WebLogic Workshop will only display the fields of a Java class that are supported.

For a Java class to be used in WebLogic Integration, it must first be available in the WebLogic Workshop project. To learn more about including Java class in your project, see Using Existing Applications.

Java classes are converted using the following procedure:

1. If the Java class is selected as an input type for a DTF, by default the name of the class becomes the name of the variable. (For example, if the class `Book` is selected as an input type, by default, the `_BookDoc` variable is created by the mapper.) If the Java class is selected as the output type of a DTF, the name of the class becomes the name of the XML document's root element.

   All the Java fields that can be mapped are converted to the appropriate XML Schema elements in the new XML Schema. If a field cannot be mapped it is ignored. To learn more, see the following table.

2. For each field of the Java class, this conversion procedure is applied recursively to determine the type of each sub-element. Fields that are themselves Java classes will be sub-elements in the internal XML Schema. These sub-elements will have the same name in the XML schema as their name in the Java class. For an example, see Using Java Classes in Transformations.

3. Java fields of the class are converted according to the following rules:

   - Public instance member variables of supported Java types are converted.

     The following table describes the supported fields of Java classes and how these fields are converted to the internal XML Schema representation of the Java class. Also, the XML Schema data type can be converted back to the Java field unless otherwise noted.

Java Field	Converts to this XML Schema Data Type . . .
Java primitive types	equivalent XML data types For example, Java primitive float type is converted to a XML Schema float type.
`java.lang.String`	`xsd:string`
`java.util.Date`	`xsd:dateTime`
`java.util.Calendar`	`xsd:dateTime`
XML Bean classes	To learn more see, XMLBeans Support for Built-In Schema Types.
MflObject classes	The type of the corresponding XML Schema
Java arrays	`xsd:sequence` of specified data type For example, an array of Java integers is converted to a XML sequence of integers.
`javax.sql.RowSet`	`xsd:anyType` **Note:** You cannot convert from XML back to a RowSet.

Java Field	Converts to this XML Schema Data Type . . .
`java.util.Collection` and classes that implement the interface	`xsd:anySimplType*` **Note:** You cannot convert from XML back to a Collection because `java.util.Collection` is only an interface. You can, however, convert back to a class that implements the Collection interface.

Note: Static member variables are not converted.

Note: Final instance member variables (constants) are not converted.

- Private fields are converted if JavaBean style `get` and `set` property methods for that field are available in the class. For example, the following private member field: `testBool` is converted to an XML schema type because the class that contains `testBool` has `set` and `get` methods for this private field as show in the following code segment:

```
public class TestSimpleTypes
{
   private boolean testBool;
   public boolean getTestBool(){
      return testBool;
   }
   public void setTestBool(boolean b){
      testBool = b;
   }
}
```

- Interface type fields will be converted if JavaBean style get and set property methods for the interface are available in the class. You cannot convert from XML back to an interface. You can, however, convert back to a class that implements a interface.

Related Topics

To learn more about creating Java class variables in business processes, see Creating Variables.

To lean more about using Java classes as input or output parameters in transformations, see Using Java Classes in Transformations.

CHAPTER

3 JpdContext Interface

Represents the execution context of a Business Process. Methods in this interface can be used to access container services for JPD (Process Definition for Java) files.

A JpdContext instance named `context` is included by default in each business process you create with WebLogic Workshop.

To learn about building Business Processes, see Guide to Building Business Processes.

Syntax

```
public interface JpdContext extends JwsContext
```

Package

```
com.bea.jpd
```

Nested Class Summary

`public ExceptionInfo getExceptionInfo()`
Returns a handle to the exception handler information.

`public interface ExceptionInfo`
See ExceptionInfo Interface

```
public interface Callback extends JwsContext.Callback
```

Method Summary

Return Type	Description
`void`	`trackData(XmlObject value)` Logs the specified XML value to the log tables such that it can be correlated to a node in your business process.
`void`	`trackData(String value)` Logs the specified String value to the log tables such that it can be correlated to a node in your business process.
`void`	`trackData(RawData value)` Logs the specified RawData value to the log tables such that it can be correlated to a node in your business process.
`boolean`	`isTransactionRetried()` Indicates whether the current transaction is executing a retry.
`void`	`setProcessLabel(String label)` Associates the specified label with this instance of the business process. The label can be used as a business handle for an instance of a business process. For example, you can use an order number, customer number, DUNS number, or some other value of use in auditing.
`String`	`getProcessLabel()` Retrieves the current process label associated with this business process.
`String`	`getInstanceId()` Retrieves the instance id of the associated business process.

Related Topics

ExceptionInfo Interface

ExceptionInfo Interface

Syntax

public interface ExceptionInfo

Method Summary

Return Type	Description
`Exception`	`Exception getException() throws IllegalStateException` Returns the current process exception. This method is only valid in an exception handler. Throws `IllegalStateException` if not currently in an exception handler.
`String`	`getExceptionNodeName() throws IllegalStateException` Returns the name of the node in the business process that threw the exception; returns `null` if the node is undefined. This method is only valid in an exception handler. Throws `IllegalStateException` if not currently in an exception handler.
`int`	`getRetriesRemaining() throws IllegalStateException` Returns the number of retries remaining for the current exception block. This method is only valid within an exception handler, or in a block of code that contains an exception handler. Throws `IllegalStateException` if not currently in an exception block or handler.

Return Type	Description
`int`	`getRetryCount() throws IllegalStateException` Returns the number of times the current exception block has been retried. This method is only valid within an exception handler, or in a block of code that contains an exception handler. Throws `IllegalStateException` if not currently in an exception block or handler.

Related Topics

JpdContext Interface

JwsContext Interface

ControlContext Interface

WebLogic Workshop Reference

4 Business Process Annotations

Business process annotations provide information to WebLogic Server about how a business process operates.

Syntax

Business process annotations have the following syntax:

```
@jpd:element attribute:setting,[attribute:setting,]...
```

Summary of Business Process Annotations

The following table provides a summary of business process annotations:

Annotation Name	Description
@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.

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

@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.

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

Remarks

None.

Related Topics

ebXML Control

WebLogic Workshop Reference

Introducing Trading Partner Integration

@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

@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/pipdirectory.

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

5 File Control Interfaces and Annotations

This section describes the File control interfaces and annotations, and provides samples of an extended File control.

Topics Included in This Section

File Control Interface
: Describes the File control interface and an example of an extended File control.

File Control Annotations
: A reference for File control annotations.

File Control Interface

The File control supports read, write and append operations on files using the Extensible mechanism. You set up a JCX file specifying the `io-type` attribute to be either `read`, `write`, or `append`. The supported types for the file content are: XmlObject, RawData and String. Only String and RawData types are supported for append operations.

The File control also support copy, rename and delete file operations. Typically, these operations are used to manipulate large files, without having to process them in any way. You can also generate a list of files in the specified directory.

The following are the File control methods:

```
/**
 * File Control base interface
 */
public interface FileControl extends Control
{
    /**
    * Returns Iterator for the File objects matching the specified file mask in the
     * current directory.
     *
     * @return an iterator for the File objects for the files
     *         in the current directory matching the file mask
     */
     FileControlFileListDocument getFiles();

    /**
     * Set the dynamic properties for the control
     *
     * @param xmlProp the dynamic properties for the control
     * @exception IllegalArgumentException if invalid properties are specified
     */
     public void setProperties(FileControlPropertiesDocument xmlProp) throws
IllegalArgumentException;

    /**
     * Get the dynamic properties for the control
     *
     * @return a FileControlPropertiesDocument object
     */
     public FileControlPropertiesDocument getProperties();

    /**
     * Rename the current file to the specified file name.
     *
     * @param fileName name of the file to be renamed to
     * @exception IllegalArgumentException if invalid file name is specified
     * @exception FileNotFoundException if the file is not present
     * @exception IOException reading the file
     * @exception SecurityException reading the file
     */
     public void rename(String fileName) throws FileNotFoundException,
                                                java.io.IOException,
                                                SecurityException,
```

```
                                                IllegalArgumentException;


    /**
     * Delete the current file.
     *
     * @exception IllegalArgumentException if invalid file name is specified
     * @exception FileNotFoundException if the file is not present
     * @exception IOException reading the file
     * @exception SecurityException reading the file
     */
    public void delete() throws FileNotFoundException,
                                              java.io.IOException,
                                              SecurityException;

    /**
     * Copy the current file to the specified file name.
     *
     * @param fileName name of the file name to be copied to
     * @exception IllegalArgumentException if invalid file name is specified
     * @exception FileNotFoundException if the file is not present
     * @exception IOException reading the file
     * @exception SecurityException reading the file
     */
     public void copy(String fileName) throws   FileNotFoundException,
                                                java.io.IOException,
                                                SecurityException,
                                                IllegalArgumentException;

    /**
     * Reset the file control by closing any in-progress operations such as:
     * readLine, readRecord and append.
     */
     public void reset();
}
```

Sample Extended File Control

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

```
package jc.fileControl;

import weblogic.jws.*;
import com.bea.control.*;
import java.io.*;
```

```
import com.bea.data.RawData;
import com.bea.xml.XmlObject;
/**
 * A custom File Control.
 *
 * @jc:file directory-name="c:\some\directory"
 *             file-mask="filetooperateon"
 *             create-mode="over-write"
 */
public interface MyFileControl extends FileControl,
                                com.bea.control.ControlExtension
{
    /**
     * @jc:file-operation io-type ="read"
     */
    XmlObject readPO();
    /**
     * @jc:file-operation io-type ="write"
     */
    FileControlPropertiesDocument writePO(XmlObject po);
}
```

File Control Annotations

This section includes information on File control annotations.

@jc:file Annotation

Specifies configuration attributes for a 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"]
@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 may be configured during its lifetime by calling methods of the `FileContol` class. To learn more about the `FileControl` class, see File Control Interface.

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`.

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.

io-type

This attribute specifies the type of operation. The valid values are: `read`, `readline`, `write`, and `append`.

file-content

This option specifies a description of the contents of 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 Interface

File Control

6 Email Control Interfaces and Annotations

This section describes the Email control interfaces and annotations, and provides samples of an extended Email control.

Topics Included in This Section

Email Control Interface

The Email control supports send e-mail operations using the Extensible mechanism. You set up a JCX file specifying the `send-email` attributes. The supported types for the content of the e-mail body are XmlObject and String.

The following are the Email control methods:

```
package com.bea.control;

/**
 * Email control interface
 */
public interface EmailControl extends Control, Extensible
{
    /**
     * Sets the dynamic properties for the control
     * @param xmlProp  the dynamic properties for the control
     */
     public void
         setProperties(EmailControlPropertiesDocument xmlProp);


    /**
     * Get the properties set for this control.
     * @return the dynamic properties for the control.
     * NULL if the control has not been initialized yet.
     */

     public EmailControlPropertiesDocument getProperties();
}
```

Sample Extended Email Control

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

```
package jc.emailControl;

import weblogic.jws.*;
import com.bea.control.*;
import com.bea.xml.XmlObject;

/**
 * A custom Email control.
 * @jc:email  smtp-address="myorg.mymailserver.com:25"
 *             from-address="joe.user@myorg.com"
 *             from-name="Joe User"
 *             reply-to-address=""
 *             reply-to-name=""
 *             header-encoding=""
```

```
 *              username=""
 *              password=""
 *
 */
public interface myFileControl extends
 EmailControl,com.bea.control.ControlExtension
{
   /**
    * @jc:send-email to="{to}"
    *                cc="{cc}"
    *                bcc="{bcc}"
    *                subject="{subject}"
    *                body="{body}"
    *                attachments="{attachments}"
    *                content-type="text/plain"
    */
   void sendEmail(String to, String cc, String bcc, String subject,
                  String body, String attachments);
}
```

Email Control Annotations

This section includes information on Email control annotations.

@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"]

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.

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.

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 Interface

Email Control

7 WLI JMS Control Interface and Annotations

This section describes the WLI JMS control interface, and provides samples of an extended WLI JMS control.

Topics Included in This Section

WLI JMS Control Interface

The WLI JMS control enables WebLogic Workshop business processes to easily interact with messaging systems that provide a JMS implementation. A specific WLI JMS control is associated with particular facilities of the messaging system. The WLI JMS control is an extension of the JMS control.

The following are the WLI JMS control methods:

```
package com.bea.control;

public interface WliJMSControl extends com.bea.control.JMSControl {
    /**
     *  This method sets the control properties dynamically.
     *  Properties set by this method over-ride the static
     *  annotations.
     * @param jmsProps control properties
     */
    public void setProperties(JMSControlPropertiesDocument
       jmsProps) throws ControlException;

    /**
     * This method returns the WliJMSControl's properties.
     * @return JMSControlPropertiesDocument WliJMSControl
     * dynamic properties
     */
    public JMSControlPropertiesDocument getControlProperties();

    public interface Callback extends
       com.bea.control.JMSControl.Callback {}
}
```

Sample Extended WLI JMS Control

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

```
package FunctionDemo;

import com.bea.control.*;
import com.bea.xml.*;
import java.io.Serializable;
```

```
/**
 *   @jc:jms send-type="queue" send-jndi-name="myqueue.async.request"
 *   receive-type="queue"  receive-jndi-name="myqueue.async.response"
 *   connection-factory-jndi-name="weblogic.jws.jms.QueueConnFactory"
 */
public interface MyJMS extends
 WliJMSControl,com.bea.control.ControlExtension
{
    /**
     * this method will send a javax.jms.TextMessage to send-jndi-name
     */
    public void sendTextMessage(XmlObject payload);
    /**
     * If your control specifies receive-jndi-name, that is your JWS expects to
     * receive messages from this control, you will need to implement
     * callback handlers.
     */
    interface Callback extends WliJMSControl.Callback
    {
        /**
         * Define only 1 callback method here.
         *
         * This method defines a callback that can handle text messages
         * from receive-jndi-name
         */
        public void receiveTextMessage(XmlObject payload);
    }
}
```

WLI JMS Control Annotations

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 Interface

WLI JMS Control

8 Message Broker Controls Interfaces and Annotations

This section describes the Message Broker Publish and Subscription control interfaces, and provides samples of extended Message Broker controls.

Topics Included in This Section

Related Topics

Message Broker Controls

Message Broker Publish Control Interface

The Message Broker Publish control is used to publish messages to Message Broker channels. You bind the Message Broker channel to the Publish control when you declare the control, but it can be overridden dynamically. You can add additional methods to your extension (subclass) of the Message Broker Publish control.

The following are the Message Broker Publish control methods:

```
package com.bea.control;

import com.bea.wli.control.dynamicProperties.PublishControlPropertiesDocument;
import org.w3c.dom.Element;
import weblogic.jws.control.Control;

/**
 * Message Broker Publish control base interface
 */

public interface PublishControl extends Control {

    /**
     * Temporarily sets the message headers for the next publish operation
     * @param headers headers to set
     */

    void setOutputHeaders(Element[] headers);

    /**
     * Sets the dynamic properties for the control
     * @param props the dynamic properties for the control
     */

     void setProperties(PublishControlPropertiesDocument props);
    /**
     * Sets the dynamic properties for the control
     * @return the current properties for the control
     */
```

```
    PublishControlPropertiesDocument getProperties();
}
```

Sample Extended Publish Control

The following code shows an example of an extended control.

```
/*
 * @jc:mb-publish-control channel-name="/controls/potopic"
 */

public interface MyPublishControl extends
PublishControl,com.bea.control.ControlExtension {
    /**
     * @jc:mb-publish-method
     *      message-metadata="<custom-header>ACME
Corp</custom-header>"
     *      message-body="{myMsgToSend}"
     */

    void publishPO(XmlObject myMsgToSend);
}
```

The following code shows how a JPD uses the control:

```
/*
 * @common:control
 */
private MyPublishControl pubCtrl;

// publish a message
void sendIt(XmlObject myMsgToSend) {
    pubCtrl.publishPO(myMsgToSend);
}
```

Message Broker Subscription Control Interface

The Message Broker Subscription control is used to dynamically subscribe to channels and receive messages. You bind the channel and optionally, an XQuery expression for filtering messages, when you create an instance of the control for your business process. The bindings cannot be overridden dynamically.

The following are the Message Broker Subscription control methods:

```
package com.bea.control;

import weblogic.jws.control.Control;

/**
 * Message Broker Subscription control base interface
 */

public interface SubscriptionControl extends Control
{

/**
     * Subscribes the control to the message broker. If the subscription
     * uses a filter expression, then the default filter value will be
     * used. If no default filter value is defined in the annotations,
     * then a <tt>null</tt> filter value will be used, meaning that any
     * filter result will trigger a callback.
     */

    void subscribe();

    /**
     * Unsubscribes the control from the message broker, stopping
     * further events (messages) from being delivered to the control.
     */

    void unsubscribe();

    interface Callback {
        /**
         * Internal callback method used to implement user-defined callbacks.
         * JPDs cannot and should not attempt to implement this callback method.
         *
         * @param msg  the message that triggered the subscription
```

```
         * @throws Exception
         *
        void _internalCallback(Object msg) throws Exception;
         */
    }
}
```

Sample Extended Subscription Control

The following code shows an example of an extended control.

```
/*
 * @jc:mb-subscription-control
 *         channel-name="/controls/stocks"
 *         xquery="$message/StockSymbol/text()"
 */
interface MySubscriptionControl extends SubscriptionControl, ControlExtension {
    /**
     * @jc:mb-subscription-method
     *      filter-value-match="BEA"
     */
    void subscribeToBEA();
    /**
     * @jc:mb-subscription-method
     *      filter-value-match="{symbol}"
     */
    void subscribeToSymbol(XmlObject symbol);
    interface Callback {
        /**
         * @jc:mb-subscription-callback message-body="{myMsgReceived}"
         */
        onXMLMessage(XmlObject myMsgReceived);
   }
}
/*
 * @common:control
 */
 MySubscriptionControl subCtrl;
// subscribe to a message

void subscribeIt() {
      subCtrl.subscribeToBEA();
}
// receive a message after subscribing
subCtrl_onXMLMessage(XmlObject myMsgReceived)
{
}
```

Message Broker Publish Control Annotations

This section includes information on Message Broker Publish control annotations.

Topics Included in This Section

@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.

@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}`.

Message Broker Subscription Control Annotations

This section includes information on Message Broker Subscription control annotations.

Topics Included in This Section

@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-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.

@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}`

@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`).

9 Application View Control Interface

This section describes the Application View control interface and provides a sample of a sub-classed Application View control.

Topics Included in This Section

Application View Control Interface

This topic defines the base interface for all Application View controls.

```
package com.bea.wlai.control;

import com.bea.control.XMLControl;

/**
 * ApplicationView Control base interface
```

```
 */
public interface ApplicationViewControl
    extends XMLControl
{
    // -------------------------
    // Local transaction methods
    // -------------------------

    public void beginLocalTransaction()
        throws Exception;

    public void commitLocalTransaction()
        throws Exception;

    public void rollbackLocalTransaction()
        throws Exception;


    // -------------------------
    // Async methods
    // -------------------------

    public boolean isAsyncRequestActive();

    public String getActiveAsyncRequestID();

    public interface Callback

}
```

The methods of this interface may be invoked by any web service or business process with an Application View control instance.

The local transaction methods are not used in a container that controls transactions, such as the process container for WebLogic Integration or the web service container for WebLogic Workshop. This is because the WebLogic Server Transaction Manager (TM) automatically enlists any local resource (a maximum of one) into the active XA transaction. Thus, using application views and adapters that support only local transactions appears the same as using application views and adapters that support XA transactions.

You can have only one local-only resource in the transaction. The business process designer must ensure that only one local resource gets enlisted anywhere within a given set of explicit transaction tags (or the implicit outer transaction tags if none are given explicitly). If a second local-only resource is required, you must separate the first and second resources across transaction tags in the process definition language.

For containers that do not control the outer XA transaction (for example, a WebLogic Portal page group), you can use the local transaction methods. However, you can also start an XA transaction using the `javax.transaction.UserTransaction` interface, and allow the WebLogic Server Transaction Manager to automatically enlist the local resource.

`public void beginLocalTransaction() throws Exception`
: Begin a local transaction on this control. This will begin a local transaction on the underlying ApplicationView instance. All work done by this control instance between this call and a call to `commitLocalTransaction`() or `rollbackLocalTransaction`() will be committed or rolled back, respectively, as a unit.
If the underlying adapter used by the application view for this control does not support local transactions, an exception is thrown.

`public void commitLocalTransaction() throws Exception`
: Commit the active local transaction for this control. All work done since the last call to `beginLocalTransaction`() will be committed into the EIS's permanent state.
If the underlying adapter used by the application view for this control does not support local transactions, an exception is thrown.

`public void rollbackLocalTransaction() throws Exception`
: Rollback the active local transaction for this control. All work done since the last call to `beginLocalTransaction`() will be discarded.
If the underlying adapter used by the application view for this control does not support local transactions, an exception is thrown.

`public boolean isAsyncRequestActive()`
: Tests to determine whether the control instance is currently processing an async request.

`public String getActiveAsyncRequestID()`
: Return the ID of the active asynchronous request being processed by this control instance, or null if no async request is active.

`public interface Callback`
: The callback interface defines the generic events that can be delivered to any web service or business process hosting this control interface.

Note: Any asynchronous service for the JCX is represented as a callback on the JCX. This allows the extension of the base control for callbacks. Asynchronous response callbacks use the XML type to represent the response document.

Sample Sub-Classed Application View Control

The following code shows an example of a sub-classed control. This code is automatically created by the control wizard.

```
package FunctionDemo;

import weblogic.jws.*;
import com.bea.wlai.control.ApplicationViewControl;
import com.bea.xml.XmlObject;
/**
 * This ApplicationView provides some simple services to
 * create/get/update customers in the sample CUSTOMER_TABLE table.
 * It also defines an event
 * indicating a customer record has been updated.
 * @jc:av-identity name="FunctionDemo.CustomerMgmt"
 * app="sampleApp"
 */
public interface CustomerMgmtAppView extends
 com.bea.control.ControlExtension, ApplicationViewControl
{

    /**
     * Get a customer record given first and last name.
     * @jc:av-service name="GetCustomer" async="false"
     */
public wlai.functionDemo.
customerMgmtGetCustomerResponse.RowsDocument
GetCustomer(wlai.functionDemo.customerMgmtGetCustomerRequest.
InputDocument request)
        throws Exception;

    /**
     * Update the customer's email address.
     * @jc:av-service name="UpdateCustomer" async="false"
     */

public wlai.functionDemo.customerMgmtUpdateCustomerResponse.
RowsAffectedDocument UpdateCustomer(wlai.functionDemo.
customerMgmtUpdateCustomerRequest.InputDocument request)
        throws Exception;

    /**
     * Select all customers in the customer table.
     * @jc:av-service name="GetAllCustomers" async="true"
     */
    public void GetAllCustomers()
        throws Exception;
```

```
    /**
     * Create a new customer given first and last name,
     * and date of birth.
     * @jc:av-service name="CreateCustomer" async="false"
     */
public wlai.functionDemo.
customerMgmtCreateCustomerResponse.RowsAffectedDocument
CreateCustomer(wlai.functionDemo.
customerMgmtCreateCustomerRequest.InputDocument request)
        throws Exception;


    public interface Callback extends
 ApplicationViewControl.Callback

    {

        /**
         * Async response callback method for
         * GetAllCustomers service.
         */
public void onGetAllCustomersResponse(wlai.functionDemo.
customerMgmtGetAllCustomersResponse.RowsDocument response);

        /**
         * Callback to handle errors with async service requests.
         * Note that only one async request can be in flight
         * for a given control instance at any given time.
         *
         * @param errorMsg
         * The error message text for the failed async request.
         */
        public void onAsyncServiceError(String errorMsg);
    }
}
```

Application View Control Annotations

You can customize an Application View control using annotations on the following properties:

- `av-identity`
  For more information, see @jc:av-identity Annotation.
- `av-service`
  For more information, see @jc:av-service Annotation.

Related Topics

Application View Control Interface

Application View Control

10 Process Control Interface

This section describes the Process control interface, and provides a sample of a sub-classed Process control.

Topics Included in This Section

Process Control Interface

The Process control is used to send requests to and receive callbacks from another business process. The Process control is typically used to call a subprocess from a parent process.

The following are the Process control methods:

```
package com.bea.control;

import com.bea.control.ServiceProxy;
import com.bea.control.Control;
import com.bea.control.ControlException;
import com.bea.wli.control.dynamicProperties.ProcessControlPropertiesDocument;
import java.net.URI;

/**
 * The process control is used to call a sub-process from a parent process.
 * Sub-process callbacks are routed to the caller process. The sub-process must
 * be in the same domain as the caller. The invocation is dispatched
 * internally over RMI without going through http. If the target process is
 * versioned, the actual version invoked depends on
 * the version strategy of the caller (as specified by the jpd:version annotation).
 *
 * The process control runtime properties can be configured externally through
 * dynamic properties and xquery selectors.
 */
public interface ProcessControl extends Control, ServiceProxy, Asynchronous
{
    /**
     * Manual control over conversation ID
     * @param conversationID
     */
    public void setConversationID(String conversationID);

    /**
     * Returns the conversation ID currently associated with this
     * control or null.
     * @return conversation id
     */
    public String getConversationID();

    /**
     * Manual control over the target uri.
     * @param uri
     * @throws ControlException if the caller process has "tightly-coupled"
versioning semantics.
     */
    public void setTargetURI(URI uri) throws ProcessControlException;

    /**
     * Returns the target URI or null if not assigned.
     * @return target uri
     */
    public URI getTargetURI();

    /**
```

```
     * Sets credential information.
     * @param username
     */
    public void setUsername(String username);

    /**
     * Sets credential information.
     * @param password
     */
    public void setPassword(String password);

    /**
     * Gets credential information.
     * @return username
     */
    public String getUsername();

    /**
     * Resets the conversational state of the proxy; this could result
     * in dropping an existing conversation.
     */
    public void reset();

    /**
     * Sets the dynamic properties for the control
     * @param props the dynamic properties for the control
     */
    void setProperties(ProcessControlPropertiesDocument props) throws
ProcessControlException;

    /**
     * Returns the current control properties. Note that this
     * does not include username/password.
     * @return the properties
     */
    ProcessControlPropertiesDocument getProperties();
}
```

Sample Sub-Classed Process Control

The following code shows an example of a sub-classed control. This code was automatically created by the control wizard based on a business process used in the WebLogic Integration samples.

```
package FunctionDemo;

/**
 * @jc:location
 uri="/ApplicationIntegration/FunctionDemo/CustomerMgmt.jpd"
 * @editor-info:link source="CustomerMgmt.jpd" autogen="true"
 */
public interface CustomerMgmtPControl extends
com.bea.control.ProcessControl, com.bea.control.ControlExtension
{
    public interface Callback
    {
        /**
         * @common:message-buffer enable="true"
         */
        public void sendSummary(java.lang.String summary);
    }

    /**
     * @jc:conversation phase="start"
     */
    public void getCustomerInfo(java.lang.String firstName,
 java.lang.String lastName, java.lang.String email);
}
```

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

11 Service Broker Control Interface

This section describes the Service Broker control interface, and provides a sample of a sub-classed Service Broker control.

Topics Included in This Section

Service Broker Control Interface
: Describes the Process control interface and an example of a sub-classed Process control.

Service Broker Control Annotations
: Provides links to information on annotations used with the Service Broker control.

Service Broker Control Interface

The Service Broker control allows a business process to send requests to and receive callbacks from another business process, a web service, or a web service or business process defined in a WSDL file.

The Service Broker control lets you dynamically set control attributes. This allows you to reconfigure control attributes without having to redeploy the application.

The following are the Service Broker control methods:

```
package com.bea.control;

import com.bea.wli.control.dynamicProperties.
       ServiceBrokerControlPropertiesDocument;

public interface ServiceBrokerControl extends ServiceControl {

     /**
      *  This method sets the control properties dynamically.
      *  Properties set by this method over-ride the static
      *  annotations.
      * @param props control properties
      */

    void setProperties(ServiceBrokerControlPropertiesDocument
props) throws Exception;
}

    /**
     * Returns the current control properties. Note that this
     * includes all properties except security-related settings
     * like username/password, keyAlias/keyPassword and
     * keyStoreLocation/keyStorePassword.
     * @return the properties
     */
    ServiceBrokerControlPropertiesDocument getProperties();
```

Sample Sub-Classed Service Broker Control

The following code shows an example of a sub-classed control. This code was automatically created by the control wizard based on a business process used in the WebLogic Integration samples. The XML schema follows the `@common:define` annotation.

```
package FunctionDemo;

/**
 * @jc:location http-url="CustomerMgmt.jpd"
jms-url="CustomerMgmt.jpd"
 * @jc:wsdl file="#CustomerMgmtWsdl"
 * @editor-info:link source="CustomerMgmt.jpd" autogen="true"
 */
public interface CustomerMgmtSBControl extends
```

```
 com.bea.control.ControlExtension,
com.bea.control.ServiceBrokerControl
{
    public static class StartHeader
        implements java.io.Serializable
    {
        public java.lang.String conversationID;
        public java.lang.String callbackLocation;
    }

    public static class ContinueHeader
        implements java.io.Serializable
    {
        public java.lang.String conversationID;
    }

    public static class CallbackHeader
        implements java.io.Serializable
    {
        public java.lang.String conversationID;
    }


    public interface Callback extends
com.bea.control.ServiceControl.Callback
    {
        public void sendSummary (java.lang.String summary);

    }

    /**
     * @jc:conversation phase="start"
     */
    public void getCustomerInfo (java.lang.String firstName,
java.lang.String lastName,
 java.lang.String email);

    static final long serialVersionUID = 1L;
}

/** @common:define name="CustomerMgmtWsdl" value::
    <?xml version="1.0" encoding="utf-8"?>
.
>
>
 * ::
 */
```

Service Broker 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

CHAPTER

12 Message Attachments

Business processes can exchange business messages with trading partners via ebXML or RosettaNet. These business messages include one or more *attachments* containing XML or non-XML data.

Note: For ebXML messages, each attachment represents a single *payload* in the ebXML message.

Attachments can be any of the following Java types:

Type	Description
`XmlObject`	Represents non-typed XML format data.
`XmlObject[]`	Used for ebXML only—an array containing one or more `XmlObject` elements.
`RawData`	Represents any non-XML structured or unstructured data for which no MFL file (and therefore no known schema) exists.
`RawData[]`	Used for ebXML only—an array containing one or more `RawData` elements
`MessageAttachment[]`	Represents either non-typed XML or non-XML data in a message attachment. Used for payloads in business messages that contain both non-typed XML and non-XML data.

Attachments can also be typed XML or typed MFL data as long as you specify the corresponding XML Bean or MFL class name in the parameter.

If you use arrays as attachment type, certain restrictions apply to the order of your arguments. For more informations, see "Specifying XmlObject and RawData Array Payloads" in Using an ebXML Control

For business messages containing both non-typed XML and non-XML data, the message payload is represented as an array of `MessageAttachment` objects:

```
MessageAttachment[]
```

The following APIs in the `com.bea.data` package provide access to individual `MessageAttachment` objects within the array:

Object	Description
MessageAttachment Interface	Represents part of a message attachment in an ebXML or RosettaNet business message. Provides methods for retrieving untyped XML or non-XML data from an attachment.
MessageAttachment.Factory Class	Factory for creating `MessageAttachment` instances. Provides methods for creating `MessageAttachment` instances from untyped XML or non-XML data.

Related Topics

Guide to Building Business Processes

ebXML Control

RosettaNet Control

Introducing Trading Partner Integration

WebLogic Workshop Reference

MessageAttachment Interface

Represents part of an attachment in an ebXML or RosettaNet business message. Provides methods for retrieving untyped XML or non-XML data from an attachment.

Syntax

```
public interface MessageAttachment
```

Extends

```
Serializable
```

Package

```
com.bea.data
```

Nested Class Summary

```
public static final class MessageAttachment.Factory
```

See MessageAttachment.Factory Class.

Field Summary

```
public static final long serialVersionUID
```

Version. Internal use only.

Method Summary

Return Type	Description
`com.bea.xml.XmlObject`	`getXmlObject()` Retrieves this portion of the attachment as an `XmlObject`. Call `isXmlObject()` first to verify that the data is untyped XML.
`com.bea.data.RawData`	`getRawData()` Retrieves this portion of the attachment as non-XML (raw) data. Call `isRawData()` first to verify that the data is raw data.
`boolean`	`isXmlObject()` Returns `true` if this portion of the attachment contains untyped XML data.
`boolean`	`isRawData()` Returns `true` if this portion of the attachment contains non-XML (raw) data.
`String`	`toString()` Converts this portion of the attachment to a `String`. Overrides: `toString` in class `Object`.

Related Topics

MessageAttachment.Factory Class

Guide to Building Business Processes

ebXML Control

RosettaNet Control

Introducing Trading Partner Integration

WebLogic Workshop Reference

MessageAttachment.Factory Class

Static factory class for creating new `MessageAttachment` instances.

Syntax

```
public static final class MessageAttachment.Factory
```

Extends

```
Object
```

Package

```
com.bea.data
```

Constructor

```
public MessageAttachment.Factory()
```

Method Summary

Return Type	Description
`MessageAttachment`	`newInstance(RawData rd)` Creates a `MessageAttachment` instance from a `RawData` object containing non-XML (raw) data for which there is no MFL file.

Return Type	Description
`MessageAttachment`	`newInstance(com.bea.xml.XmlObject xo)` Creates a `MessageAttachment` instance from an `XmlObject` containing XML data.

Related Topics

MessageAttachment Interface

Guide to Building Business Processes

ebXML Control

RosettaNet Control

Introducing Trading Partner Integration

WebLogic Workshop Reference

CHAPTER

13 Worklist Control Interfaces and Annotations

This section describes the Task Control and Task Worker Control Interfaces, and provides samples of extended Task and Task Worker controls.

Topics Included in This Section

Task Control Interface
: Describes the Task Control Interface and an example of an extended Task Control.

Task Worker Control Interface
: Describes the Task Worker Control Interface and an example of an extended Task Worker Control.

Worklist Control Annotations
: A reference for Worklist Control annotations.

Task Control Interface

The Task control enables a business process to create a single Task instance, manage its state and data, and provide callback methods that report status. Each Task control operates on a single active Task instance. You can extend this control.

The following are the Task control methods:

```
package com.bea.control;

import weblogic.jws.control.Control;
import com.bea.control.Extensible;
import com.bea.wli.worklist.api.TaskInfo;
import com.bea.wli.worklist.xml.TaskCreationXMLDocument;
import com.bea.wli.worklist.xml.TaskInfoXMLDocument;
import com.bea.wli.worklist.xml.TaskUpdateXMLDocument;

/**
 * Task control interface.
 */

public interface TaskControl extends Control {

    /**
     * the control jc tag.
     */

    static final String TAG_TASK = "jc:task";

    /**
     * the control advanced options jc tag.
     */

    static final String TAG_ADVANCED = "jc:advanced";

    /**
     * the control assignement jc tag.
     */

    static final String TAG_ASSIGNEE = "jc:assignee";

    /**
     * Set the TaskId the control is linked to. The taskId can only be set once.
     * When a task is created, its taskId is set to the control if no taskId
     * has been set before or return an exception otherwise.
     * If you want to work with more than one taskId you can use multiple
     * TasksControls, a TaskControl factory or the WorklistControl.
```

```
     */

    public void setTaskId(String taskId);

    public String getTaskId();

    /**
     * Create and optionally assigns a new task using an XML interface
     * @param doc the task parameters
     * @return the taskId
     */

    public String createTask(TaskCreationXMLDocument doc);

    /**
     * The @jc:task-create tag is used to annotate a JBCX method that
     * creates a new task. This method can at most be called once by control
     *and returns the taskId.

     * The callbacks are registered to the new task created

     */

    static final String TAG_TASK_CREATE                 = "jc:task-create";

    static final String ATTR_CREATE_NAME                = "name";

   static final String ATTR_CREATE_DESCRIPTION       = "description"; // optional

    static final String ATTR_CREATE_COMMENT             = "comment"; //optional

    static final String ATTR_CREATE_PRIORITY           = "priority"; // optional

    static final String ATTR_CREATE_CLAIM_DUE_DATE      = "claim-due-date"; //
optional

   static final String ATTR_CREATE_COMPLETION_DUE_DATE   = "completion-due-date";
// optional, if no expiration date is specified the task will never expire.

    static final String ATTR_CREATE_CLAIM_DUE_BUSINESS_DATE =
"claim-due-business-date"; //optional

   static final String ATTR_CREATE_CLAIM_USER_CALENDAR = "claim-user-calendar";
//optional

    static final String ATTR_CREATE_CLAIM_CALENDAR      = "claim-calendar";
//optional

    static final String ATTR_CREATE_COMPLETION_DUE_BUSINESS_DATE =
"completion-due-business-date"; //optional

    static final String ATTR_CREATE_COMPLETION_USER_CALENDAR =
"completion-user-calendar"; //optional
```

```
   static final String ATTR_CREATE_COMPLETION_CALENDAR = "completion-calendar";
//optional

    static final String ATTR_CREATE_OWNER               = "owner";

    static final String ATTR_CREATE_CAN_BE_REASSIGNED   = "can-be-reassigned";
// optional: by default true

    static final String ATTR_CREATE_CAN_BE_RETURNED     = "can-be-returned";
// optional by default false

    static final String ATTR_CREATE_CAN_BE_ABORTED      = "can-be-aborted";
// optional by default true

    static final String ATTR_CREATE_REQUEST             = "request";
// optional, can be mapped to any serializable object.

    static final String ATTR_CREATE_REQUEST_MIME_TYPE   = "request-mime-type";
//optional

    /**
     * The @jc:task-assign tag is used to annotate a JBCX method that
     * assign or reassign a task. The algorithm name is used to determine the way
     * a task is assigned. If no name is pecified the default algorithm is used
     * and the args must be of type String[];
     * In any case the attribute "args" is required and must be of type Object[].
     *
     */

    static final String TAG_TASK_ASSIGN          = "jc:task-assign";

    static final String ATTR_ASSIGN_USER         = "user";

    static final String ATTR_ASSIGN_GROUP        = "group";

    static final String ATTR_ASSIGN_ALGORITHM    = "algorithm";

    static final String TAG_TASK_RESUME          = "jc:task-resume";

    static final String TAG_TASK_SUSPEND         = "jc:task-suspend";

    public static final String TAG_TASK_ABORT    = "jc:task-abort";

    public TaskInfo getTaskInfo();

    public TaskInfoXMLDocument getTaskInfoXMLDocument();

    public String[] getTaskPropertiesNames();

    /**
     *  The return type of the method JBCX is used to determine
```

```
     *  what is returned.
     */

    public static final String TAG_TASK_GET_REQUEST     = "jc:task-get-request";

    //**
     *  The return type of the method JBCX is used to determine
     *  what is returned.
     */

    public static final String TAG_TASK_GET_RESPONSE    = "jc:task-get-response";

    //**
     *  The return type of the method JBCX is used to determine
     *  what is returned.
     */

    public static final String TAG_TASK_GET_PROPERTY    = "jc:task-get-property";

    public static final String ATTR_GET_PROPERTY_NAME   = "name";

    public static final String TAG_TASK_SET_PROPERTY    = "jc:task-set-property";

    public static final String ATTR_SET_PROPERTY_NAME   = "name";

    public static final String ATTR_SET_PROPERTY_VALUE  = "value";

    public static final String ATTR_SET_PROPERTY_MAP    = "map";

    public static final String TAG_TASK_REMOVE_PROPERTY =
"jc:task-remove-property";

    public static final String ATTR_REMOVE_PROPERTY_NAME= "name";

    /**
     * The @jc:task-update tag is used to annotate a JBCX method that
     * update the task properties. You can modify one or more properties at the
same time.
     */

    static final String TAG_TASK_UPDATE                 = "jc:task-update";

   static final String ATTR_UPDATE_CAN_BE_ABORTED        = "can-be-aborted"; //
optional

   static final String ATTR_UPDATE_CAN_BE_REASSIGNED     = "can-be-reassigned";
// optional

    static final String ATTR_UPDATE_CAN_BE_RETURNED        = "can-be-returned";
// optional

    static final String ATTR_UPDATE_COMMENT             = "comment"; //optional
```

```
    static final String ATTR_UPDATE_PRIORITY            = "priority"; // optional

    static final String ATTR_UPDATE_CLAIM_DUE_DATE      = "claim-due-date";
//optional

   static final String ATTR_UPDATE_COMPLETION_DUE_DATE  = "completion-due-date";
//optional

    static final String ATTR_UPDATE_CLAIM_DUE_BUSINESS_DATE =
"claim-due-business-date"; //optional

   static final String ATTR_UPDATE_CLAIM_USER_CALENDAR = "claim-user-calendar";
//optional

    static final String ATTR_UPDATE_CLAIM_CALENDAR      = "claim-calendar";
//optional

    static final String ATTR_UPDATE_COMPLETION_DUE_BUSINESS_DATE =
"completion-due-business-date"; //optional

    static final String ATTR_UPDATE_COMPLETION_USER_CALENDAR =
"completion-user-calendar"; //optional

   static final String ATTR_UPDATE_COMPLETION_CALENDAR = "completion-calendar";
//optional

    static final String ATTR_UPDATE_OWNER               = "owner"; //optional

    static final String ATTR_UPDATE_REQUEST             = "request"; //optional

    static final String ATTR_UPDATE_REQUEST_MIME_TYPE   = "request-mime-type";
//optional

   static final String ATTR_UPDATE_RESPONSE             = "response"; //optional

   static final String ATTR_UPDATE_RESPONSE_MIME_TYPE    = "response-mime-type";
//optional

    public void updateTask(TaskUpdateXMLDocument doc);

    static final Integer CALENDAR_TYPE_SYSTEM = new Integer(0);

    static final Integer CALENDAR_TYPE_USER_OR_GROUP = new Integer(1);

    static final Integer CALENDAR_TYPE_NAME = new Integer(2);


    public interface Callback

    {
```

```
    /**
     * Internal callback method used to implement user-defined callbacks. When a
     * task is set to the control instance, all the callback are registered to
     * the task.
     *
     */

    static final String TAG_TASK_EVENT               = "jc:task-event";

    // event-types are: "completed, aborted, claimed, suspended, resumed,
    // assigned, task-expired, task-claim-expired"

    static final String ATTR_TASK_EVENT_TYPE         = "event-type";
    static final String ATTR_TASK_EVENT_TIME         = "time";//optional

    // optional, if a user has performed an action changing the state
    static final String ATTR_TASK_EVENT_USER        = "user"; // optional

    // task response passed along "completed" with the event
    static final String ATTR_TASK_EVENT_RESPONSE       = "response"; // optional

    }
}
```

Sample Extended Task Control

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

```
package processes;

import weblogic.jws.*;
import com.bea.control.TaskControl;
import com.bea.xml.XmlObject;
import java.util.Date;

/**
 * @jc:task
 */

public interface task1 extends TaskControl, com.bea.control.ControlExtension
{
    /**
    * @jc:task-create
    *   name="{name}"
    */

    public String createTaskByName(String name);
```

```
/**
 * @jc:task-assign
 * user="{users}"
 * group="{groups}"
 * algorithm="ToUsersAndGroups"
 */

public void assignTaskToUsersAndGroups(String[] users, String[] groups);

/**
 * @jc:task-assign
 * user="{user}"
 * algorithm="ToUser"
 */

public void assignTaskToUser(String user);

/**
 * @jc:task-assign
 * group="{group}"
 * algorithm="ToUserInGroup"
 */

public void assignTaskToUserInGroup(String group);

/**
 * @jc:task-update
 * request={xml}
 */

public void setRequest(XmlObject xml);

/**
 * @jc:task-update
 * response={xml}
 */

public void setResponse(XmlObject xml);

/**
 * @jc:task-update
 * comment={comment}
 */

public void setComment(String comment);

/**
 * @jc:task-update
 * priority={priority}
 */
```

```
    public void setPriority(Integer priority);

    /**
     * @jc:task-update
     * owner={owner}
     */

    public void setOwner(String owner);

    /**
     * @jc:task-update can-be-aborted="{aborted}" can-be-returned="{returned}"
     *can-be-reassigned="{reassigned}"
     */

    public void setPermissions(Boolean aborted, Boolean returned, Boolean
reassigned);

    /**
     * @jc:task-update completion-due-date="{date}"
     *
     */

    public void setCompletionDueDate(Date date);

    /**
     * @jc:task-update completion-due-business-date={duration}
     * completion-calendar={calendarID}
     */

   public void setCompletionDueBusinessDate(String duration, String calendarID);

    /**
     * @jc:task-update claim-due-business-date={duration}
     * claim-calendar={calendarID}
     */

    public void setClaimDueBusinessDate(String duration, String calendarID);

   /**
     * @jc:task-update claim-due-business-date={duration}
     */

    public void setClaimDueBusinessDateSystemCalendar(String duration);

    /**
     * @jc:task-update completion-due-business-date={duration}
     */

    public void setCompletionDueBusinessDateSystemCalendar(String duration);
```

```
/**
 * @jc:task-update claim-due-date="{date}"
 */

public void setClaimDueDate(Date date);

/**
 * @jc:task-abort enabled="true"
 */

public void abortTask();

/**
 * @jc:task-resume enabled="true"
 */

public void resumeTask();

/**
 * @jc:task-suspend enabled="true"
 */

public void suspendTask();

/**
 * @jc:task-get-request enabled="true"
 */

public XmlObject getRequest();

/**
 * @jc:task-get-response enabled="true"
 */

public XmlObject getResponse();

/**
 * @jc:task-get-property name="{name}"
 */

public String getProperty(String name);

/**
 * @jc:task-set-property name="{name}" value="{value}"
 */

public void setProperty(String name, String value);

/**
 * @jc:task-remove-property name="{name}"
 */
```

```
    public void removeProperty(String name);


    public interface Callback extends TaskControl.Callback {

       /**
        * @jc:task-event event-type="complete" response="{response}"
        */

        void onTaskCompleted(XmlObject response);

        /**
        * @jc:task-event event-type="abort" response="{response}"
        */

        void onTaskAborted(XmlObject response);

        /**
        * @jc:task-event event-type="expire" time="{time}"
        */

        void onTaskOverdue(Date time);

    }
}
```

Task Worker Control Interface

The Task Worker control enables a business process or UI to assume ownership of Tasks, work on them, and complete them. It offers administrative operations, such as operations to start, stop, delete, and assign. The Task Worker controls allow operations on several Task instances simultaneously. You can extend this control.

The following are the Task Worker control methods:

```
package com.bea.control;

import weblogic.jws.control.Control;
import com.bea.control.Extensible;
import com.bea.wli.worklist.api.TaskInfo;
import com.bea.wli.worklist.xml.TaskCreationXMLDocument;
```

```
/**
 * Task control interface.
 */

public interface TaskWorkerControl extends Control {

    /**
     * the control jc tag.
     */

    public static final String TAG_TASK_WORKER = "jc:task-worker";

    /**
     * Create and optionally assigns a new task using an XML interface=
     * @param doc the task parameters
     * @return the taskId
     */

    public String createTask(TaskCreationXMLDocument doc);

    /**
     * The @jc:task-create tag is used to annotate a JBCX method that
     * creates a new task. This method returns the taskId of the created task.
     */

    static final String TAG_TASK_WORKER_CREATE  = "jc:task-create";

    static final String ATTR_CREATE_NAME                = "name";

   static final String ATTR_CREATE_DESCRIPTION       = "description"; // optional

    static final String ATTR_CREATE_COMMENT             = "comment"; //optional

    static final String ATTR_CREATE_PRIORITY           = "priority"; // optional

    static final String ATTR_CREATE_CLAIM_DUE_DATE      = "claim-due-date"; //
optional

   static final String ATTR_CREATE_COMPLETION_DUE_DATE   = "completion-due-date";
    // optional, if no expiration date is specified the task will never expire.

    static final String ATTR_CREATE_CLAIM_DUE_BUSINESS_DATE =
    "claim-due-business-date"; //optional

   static final String ATTR_CREATE_CLAIM_USER_CALENDAR = "claim-user-calendar";
//optional

    static final String ATTR_CREATE_CLAIM_CALENDAR      = "claim-calendar";
//optional

    static final String ATTR_CREATE_COMPLETION_DUE_BUSINESS_DATE =
    "completion-due-business-date"; //optional
```

```
    static final String ATTR_CREATE_COMPLETION_USER_CALENDAR =
    "completion-user-calendar"; //optional

   static final String ATTR_CREATE_COMPLETION_CALENDAR   = "completion-calendar";
//optional

    static final String ATTR_CREATE_OWNER                = "owner";

    static final String ATTR_CREATE_CAN_BE_REASSIGNED    = "can-be-reassigned";
 // optional: by default true

    static final String ATTR_CREATE_CAN_BE_RETURNED      = "can-be-returned";
// optional by default false

    static final String ATTR_CREATE_CAN_BE_ABORTED       = "can-be-aborted";
// optional by default true

    static final String ATTR_CREATE_REQUEST              = "request";
// optional, can be mapped to any serializable object.

    static final String ATTR_CREATE_REQUEST_MIME_TYPE    = "request-mime-type";
//optional

    public static final String TAG_TASK_WORKER_ABORT     = "jc:task-abort";

    public static final String TAG_TASK_WORKER_CLAIM     = "jc:task-claim";

    public static final String ATTR_CLAIM_CLAIMANT       = "claimant";
// to claim the task on behalf of a user.

    public static final String TAG_TASK_WORKER_COMPLETE = "jc:task-complete";

    public static final String TAG_TASK_WORKER_DELETE    = "jc:task-delete";

    public static final String TAG_TASK_WORKER_RESUME    = "jc:task-resume";

    public static final String TAG_TASK_WORKER_RETURN    = "jc:task-return";

    public static final String TAG_TASK_WORKER_START     = "jc:task-start";

    public static final String TAG_TASK_WORKER_STOP      = "jc:task-stop";

    public static final String TAG_TASK_WORKER_SUSPEND   = "jc:task-suspend";

    /**
     * The @jc:task-assign tag is used to annotate a JBCX method that
     * assign or reassign a task. The algorithm name is used to determine the way
     * a task is assigned. If no name is pecified the default algorithm is used
     * and the args must be of type String[];
     * In any case the attribute "args" is required and must be of type Object[].
     */

    public static final String TAG_TASK_WORKER_ASSIGN    = "jc:task-assign";
```

```
    static final String ATTR_ASSIGN_USER         = "user";

    static final String ATTR_ASSIGN_GROUP        = "group";

    static final String ATTR_ASSIGN_ALGORITHM    = "algorithm";

    /**
     *  The return type of the method JBCX is used to determine
     *  what is returned:
     *  TaskInfo[]
     *  TaskInfoXML[]
     */

   public static final String TAG_TASK_WORKER_GET_INFO      = "jc:task-get-info";

    /**
     *  The return type of the method JBCX is used to determine
     *  what is returned.
     */

    public static final String TAG_TASK_WORKER_GET_REQUEST     =
"jc:task-get-request";

    /**
     *  The return type of the method JBCX is used to determine
     *  what is returned.
     */

    public static final String TAG_TASK_WORKER_GET_RESPONSE     =
"jc:task-get-response";

    public static final String TAG_TASK_WORKER_GET_PROPERTY_NAME =
"jc:task-get-property-name";

    public static final String TAG_TASK_WORKER_GET_PROPERTY    =
"jc:task-get-property";

    public static final String ATTR_GET_PROPERTY_NAME   = "name";

    public static final String TAG_TASK_WORKER_SET_PROPERTY    =
"jc:task-set-property";

    public static final String ATTR_SET_PROPERTY_NAME   = "name";

    public static final String ATTR_SET_PROPERTY_VALUE  = "value";

    public static final String TAG_TASK_WORKER_REMOVE_PROPERTY =
"jc:task-remove-property";

    public static final String ATTR_REMOVE_PROPERTY_NAME= "name";
```

```
    /**
     * The @jc:task-update tag is used to annotate a JBCX method that
     * update the task properties. You can modify one or more properties at the
same time.
     */

   public static final String TAG_TASK_WORKER_UPDATE         = "jc:task-update";

    public static final String ATTR_UPDATE_CAN_BE_ABORTED         =
"can-be-aborted"; // optional

    public static final String ATTR_UPDATE_CAN_BE_REASSIGNED      =
"can-be-reassigned"; // optional

    public static final String ATTR_UPDATE_CAN_BE_RETURNED        =
"can-be-returned"; // optional

    public static final String ATTR_UPDATE_CLAIM_DUE_DATE         =
"claim-due-date"; //optional

    public static final String ATTR_UPDATE_CLAIM_DUE_BUSINESS_DATE =
"claim-due-business-date"; //optional

    public static final String ATTR_UPDATE_COMPLETION_DUE_DATE     =
"completion-due-date"; //optional

   static final String ATTR_UPDATE_CLAIM_USER_CALENDAR = "claim-user-calendar";
//optional

    static final String ATTR_UPDATE_CLAIM_CALENDAR      = "claim-calendar";
//optional

    static final String ATTR_UPDATE_COMPLETION_DUE_BUSINESS_DATE =
"completion-due-business-date"; //optional

    static final String ATTR_UPDATE_COMPLETION_USER_CALENDAR =
"completion-user-calendar"; //optional

   static final String ATTR_UPDATE_COMPLETION_CALENDAR = "completion-calendar";
//optional

   public static final String ATTR_UPDATE_COMMENT         = "comment"; //optional

    public static final String ATTR_UPDATE_OWNER           = "owner"; //optional

   public static final String ATTR_UPDATE_PRIORITY      = "priority"; // optional

   public static final String ATTR_UPDATE_REQUEST         = "request"; //optional

    public static final String ATTR_UPDATE_REQUEST_MIME_TYPE           =
"request-mime-type"; //optional
```

```
    public static final String ATTR_UPDATE_RESPONSE            = "response";
//optional

    public static final String ATTR_UPDATE_RESPONSE_MIME_TYPE     =
"response-mime-type"; //optional

    /**
     * The @jc:select tag is used in correlation with the other method tags of
this control.
     * It is used to determine the tasks concerned by the operation.
     */

    public static final String TAG_TASK_WORKER_SELECT            = "jc:select";

   public static final String ATTR_SELECT_TASK_ID         = "task-id"; //optional

    public static final String ATTR_SELECT_TASK_NAME            = "task-name";
//optional

    public static final String ATTR_SELECT_TASK_COMMENT           = "comment";
//optional

    public static final String ATTR_SELECT_TASK_OWNER             = "owner";
//optional

    public static final String ATTR_SELECT_CLAIMANT               = "claimant";
//optional

   public static final String ATTR_SELECT_ASSIGNED_USER       = "assigned-user";
//optional

    public static final String ATTR_SELECT_ASSIGNED_GROUP         =
"assigned-group"; //optional

   public static final String ATTR_SELECT_MIN_PRIORITY         = "min-priority";
//optional

   public static final String ATTR_SELECT_MAX_PRIORITY         = "max-priority";
//optional

    public static final String ATTR_SELECT_STATES                 = "states";
//optional

    public static final String ATTR_SELECT_COMPLETION_DUE_DATE_BEFORE =
"completion-due-date-before"; //optional

    public static final String ATTR_SELECT_COMPLETION_DUE_DATE_AFTER =
"completion-due-date-after"; //optional

    public static final String ATTR_SELECT_CLAIM_DUE_DATE_BEFORE  =
"claim-due-date-before"; //optional
```

```
    public static final String ATTR_SELECT_CLAIM_DUE_DATE_AFTER   =
"claim-due-date-after"; //optional

    public static final String ATTR_SELECT_CREATION_DATE_BEFORE   =
"creation-date-before"; //optional

    public static final String ATTR_SELECT_CREATION_DATE_AFTER    =
"creation-date-after"; //optional

   public static final String ATTR_SELECT_PROPERTY_NAME       = "property-name";
//optional

    public static final String ATTR_SELECT_PROPERTY_VALUE         =
"property-value"; //optional

    public static final String ATTR_SELECT_SELECTOR               = "selector";
    //optional to use a java TaskSelector instead

    /**
     * Archive the tasks that are in the state aborted or completed
     */

    public void archiveTasks();

    /**
     * Purge the tasks that are ready to be purged:
     * completed or aborted time > purgDelay.
     * if archiver is on the task must be archived first.
     */

    public void purgeTasks();
}
```

Sample Extended Task Worker Control

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

```
package processes;

import weblogic.jws.*;
import com.bea.control.TaskWorkerControl;
import com.bea.wli.worklist.api.*;
import com.bea.wli.worklist.xml.*;
import com.bea.xml.XmlObject;
import java.util.Date;
```

```
/**
 * @jc:task-worker
 */

public interface taskworker1 extends TaskWorkerControl,
com.bea.control.ControlExtension

{
    /**
    * @jc:task-create
    *   name="{name}"
    */

    public String createTaskByName(String name);

    /**
     * @jc:task-assign
     * user="{users}"
     * group="{groups}"
     * algorithm="ToUsersAndGroups"
     * @jc:select task-id="{taskIds}"
     */

    public void assignTaskToUsersAndGroups(String[] users, String[] groups,
String[] taskIds);

    /**
     * @jc:task-assign
     * user="{user}"
     * algorithm="ToUser"
     * @jc:select task-id="{taskIds}"
     */

    public void assignTaskToUser(String user, String[] taskIds);

   /**
     * @jc:task-assign
     * group="{group}"
     * algorithm="ToUserInGroup"
     * @jc:select task-id="{taskIds}"
     */

    public void assignTaskToUserInGroup(String group, String[] taskIds);

    /**
     * @jc:task-update request-mime-type="{type}"
     * request={xml}
     * @jc:select task-id="{taskIds}"
     */

    public void setRequest(XmlObject xml, String type, String[] taskIds);
```

```
    /**
     * @jc:task-update response-mime-type="{type}"
     * response={xml}
     * @jc:select task-id="{taskIds}"
     */

    public void setResponse(XmlObject xml, String type, String[] taskIds);

    /**
     * @jc:task-update
     * comment={comment}
     * @jc:select task-id="{taskIds}"
     */

    public void setComment(String comment, String[] taskIds);

    /**
     * @jc:task-update
     * priority={priority}
     * @jc:select task-id="{taskIds}"
     */

    public void setPriority(Integer priority, String[] taskIds);

    /**
     * @jc:task-update
     * owner={owner}
     * @jc:select task-id="{taskIds}"
     */

    public void setOwner(String owner, String[] taskIds);

    /**
     * @jc:task-update can-be-aborted="{aborted}" can-be-returned="{returned}"
can-be-reassigned="{reassigned}"
     * @jc:select task-id="{taskIds}"
     */

    public void setPermissions(Boolean aborted, Boolean returned, Boolean
reassigned, String[] taskIds);

    /**
     * @jc:task-update completion-due-date="{date}"
     * @jc:select task-id="{taskIds}"
     */

    public void setCompletionDueDate(Date date, String[] taskIds);

    /**
     * @jc:task-update completion-due-business-date={duration}
     * completion-calendar={calendarID}
```

```
     * @jc:select task-id="{taskIds}"
     */

   public void setCompletionDueBusinessDate(String duration, String calendarID,
String[] taskIds);

   /**
     * @jc:task-update claim-due-business-date={duration}
     * @jc:select task-id="{taskIds}"
     */

    public void setClaimDueBusinessDateSystemCalendar(String duration, String[]
taskIds);

    /**
     * @jc:task-update completion-due-business-date={duration}
     * @jc:select task-id="{taskIds}"
     */

    public void setCompletionDueBusinessDateSystemCalendar(String duration,
String[] taskIds);

    /**
     * @jc:task-update claim-due-business-date={duration}
     * claim-calendar={calendarID}
     * @jc:select task-id="{taskIds}"
     */

    public void setClaimDueBusinessDate(String duration, String calendarID,
String[] taskIds);

    /**
     * @jc:task-update claim-due-date="{date}"
     * @jc:select task-id="{taskIds}"
     */

    public void setClaimDueDate(Date date, String[] taskIds);

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

    public void abortTask(String[] taskIds);

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

    public void resumeTask(String[] taskIds);
```

```
/**
 * @jc:task-suspend
 * @jc:select task-id="{taskIds}"
 */

public void suspendTask(String[] taskIds);

/**
 * @jc:task-claim enabled="true" claimant="{claimant}"
 * @jc:select task-id="{taskIds}"
 */

public void claimTaskOnBehalfOf(String claimant, String[] taskIds);

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

public void claimTask(String[] taskIds);

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

public void completeTask(String[] taskIds);

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

public void returnTask(String[] taskIds);

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

public void startTask(String[] taskIds);

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

public void stopTask(String[] taskIds);

/**
 * @jc:task-get-request enabled="true"
```

```
 * @jc:select task-id="{taskIds}"
 */

public XmlObject[] getRequest(String[] taskIds);

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

public XmlObject[] getResponse(String[] taskIds);

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

public TaskInfo[] getTaskInfo(String[] taskIds);

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

public TaskInfoXMLDocument[] getTaskInfoXML(String[] taskIds);

/**
 * @jc:task-get-property name="{name}"
 * @jc:select task-id="{taskIds}"
 */

public String[] getProperty(String name, String[] taskIds);

/**
 * @jc:task-set-property name="{name}" value="{value}"
 * @jc:select task-id="{taskIds}"
 */

public void setProperty(String name, String value, String[] taskIds);

/**

 * @jc:task-remove-property name="{name}"
 * @jc:select task-id="{taskIds}"
 */

public void removeProperty(String name, String[] taskIds);

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

```
    public void deleteTask(String[] taskIds);
}
```

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:advanced Annotation
: Notations for advanced options.

@jc:assignee Annotation
: Assigns user and groups to Tasks.

@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

Task Control 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

14 TPM Control Interface

The TPM (trading partner management) control provides WebLogic Workshop business processes and web services with query (read-only) access to trading partner and service information stored in the TPM repository.

This section describes the TPM control interface.

Topics Included in This Section

TPM Control Interface
: Describes the TPM control interface.

Related Topics

TPM Control

Introducing Trading Partner Integration

TPM Control Interface

The `TPMControl` interface extends `com.bea.control` and includes the following methods for retrieving information from the TPM repository:

Method Name	Description
`getBasicProperties`	Returns the basic properties of the specified trading partner.
`getDefaultEbxmlBinding`	Returns the default ebXML binding for the specified trading partner and ebXML version.
`getDefaultPartner`	Returns the default trading partner.
`getDefaultRosettanetBinding`	Returns the default RosettaNet binding for the specified trading partner and RosettaNet version.
`getExtendedPropertySet`	Returns the specified extended property set of the specified trading partner.
`getPartnerById`	Returns the trading partner associated with the specified business ID.
`getPartnerByName`	Returns the trading partner associated with the specified name.
`getService`	Returns the service with the specified name.
`getServiceProfile`	Returns the service profile associated with the specified service and trading partners.
`getServiceProfileEbxmlBinding`	Returns the ebXML binding of the service profile associated with the specified service name and trading partners.
`getServiceProfileRosettanetBinding`	Returns the RosettaNet binding of the service profile associated with the specified service name and trading partners.
`getServiceProfileWebServiceBinding`	Returns the web service binding of the service profile associated with the specified service name and trading partners.

Note: To learn about the syntax for using a particular method, view the method in Source View.

CHAPTER

15 ebXML Control Interface and 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 describes the ebXML control interface and annotations.

Topics Included in This Section

ebXML Control Interface
: Describes the ebXML control interface.

ebXML Control Annotations
: A reference for ebXML control annotations.

Related Topics

ebXML Control

Introducing Trading Partner Integration

ebXML Control Interface

The `EBXMLControl` interface extends `com.bea.control` and includes the following methods:

Method Name	Description
`onAck`	Invoked when an ebXML SOAP envelope containing an acknowledgement is received. Base control method.
`onError`	Invoked when an ebXML SOAP envelope containing an error is received.
`setProperties`	Sets dynamic properties at run-time, such as the business ID of the sender or recipient.
`getProperties`	Displays the current property settings.

After creating a JCX control instance, the following generated methods are available. You can use the default method names or rename them to be more descriptive of your implementation.

Method Name	Description
`request`	Sends the specified message. Specified in the **Control Send** node in the initiator business process. **Note:** The default return type for the `request` method is void. However, you can also specify the return type to be `XmlObject`. If you use `XmlObject` as the return type, the content the `XmlObject` is the ebXML envelope data.
`response`	Invoked when a business message is received. Specified in the Control Receive node in the initiator business process.

Note: To learn about the syntax for using a particular method, view the method in Source View.

Related Topics

ebXML Control Annotations

ebXML Control Annotations

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 the JCX class-level annotations for the ebXML control

@jc:ebxml-method Annotation
: Specifies the method-level annotations for the ebXML control.

@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

Tag Name	Description
`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).

Tag Name	Description
`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

Tag Name	Description
`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

16 RosettaNet Control Interface and Annotations

The RosettaNet control enables WebLogic Workshop business processes to exchange business messages and data with trading partners via RosettaNet. You use RosettaNet controls *only* in initiator business processes to manage the exchange of RosettaNet business messages with participants.

This section describes the RosettaNet control interface and annotations.

Topics Included in This Section

RosettaNet Control Interface
: Describes the RosettaNet control interface.

RosettaNet Control Annotations
: A reference for RosettaNet control annotations.

Related Topics

RosettaNet Control

Introducing Trading Partner Integration

RosettaNet Control Interface

The `RosettaNetControl` interface extends `com.bea.control` and includes the methods described in the following table:

Method Name	Description
`getProcessTimeout`	Returns the amount of time to wait for the entire business process to complete before timing out. Base method.
`getRetryCount`	Returns the maximum number of times to retry sending the message in case of failure. Base method.
`getRetryInterval`	Returns the amount of time to wait between retries. Base method.
`onAck`	Invoked when a RosettaNet acknowledgement is received.
`onError`	Invoked when a RosettaNet error notification is received.
`onMessage`	Invoked when a RosettaNet business message is received.
`sendAck`	Sends an acknowledgement of receipt of a business message to a trading partner and accepts the request. Base method.
`sendError`	Sends an RosettaNet error to a trading partner. Base method.
`sendMessage`	Sends the specified message to a trading partner.
`sendReject`	Sends an acknowledgement of receipt of the business message to a trading partner, but rejects the request. Base method.
`setProperties`	Sets dynamic properties at run-time, such as the business ID of the sender or recipient. Use this method to set the debug mode during testing. Base method.
`getProperties`	Lists current property settings.

Note: To learn about the syntax for using a particular method, view the method in Source View.

RosettaNetContext

`RosettaNetContext` is an XMLBean that can be used to obtain information from RosettaNet business messages. The following message elements can be retrieved and are returned as `java.lang.string`:

Element Name	Description
`from`	Sender's DUNS number.
`to`	Recipient's DUNS number.
`pip`	RosettaNet PIP code specified for the message.
`pip-version`	PIP version specified for the message.
`from-role`	RosettaNet role name for the sender as defined in the PIP specification. Examples include: Buyer, Initiator, Shipper, and so on..
`to-role`	RosettaNet role name for the recipient as defined in the PIP specification. Examples include: Seller, Participant, Receiver, and so on.
`failure-report-administrator`	Trading partner id of the trading partner which is specified to be the failure administrator. (In WebLogic Integration, this is specified in the sender trading partner's binding).
`global-usage-code`	Indicates whether the message was sent in test or production mode.
`debug-mode`	Returns `true` if the message was sent in debug mode.
`message-tracking-id`	Instance id of the action to which this message is in reply.
`protocol-name`	Name of the protocol used.
`protocol-version`	Version of the protocol used.
`conversation-id`	Id of the conversation.
`process-instance-id`	Instance id of the receiving process.

Element Name	Description
process-uri	URI of the receiving process.
business-action	The business action of the message, such as: Purchase Order Request, Purchase Order Confirmation, etc.
document-datetimestamp	The time and date the document was created.
proprietary-identifier	A unique number which tracks the document.

When you use the RosettaNetContext XMLBean, be sure to import the following classes:

```
com.bea.wli.control.rosettanetContext.RosettaNetContextDocument;
com.bea.wli.control.rosettanetContext.RosettaNetContextDocument.R
osettaNetContext;
```

The following are code examples of how to use RosettaNetContext:

- *Initiator business process receiving a message*:

```
public void rn_onMessage(RosettaNetContextDocument doc,
                         XmlObject msg)
   {
     System.out.println(">>>>> ContextInitiator.rn_onMessage()");
     RosettaNetContextDocument.RosettaNetContext context =
        doc.getRosettaNetContext();
     System.out.println("    from=" + context.getFrom());
     System.out.println("    to=" + context.getTo());
     System.out.println("    pip=" + context.getPip());
     System.out.println("    failure-report-admin=" +
        context.getFailureReportAdministrator());
   }
```

- *Initiator business process sending a message:*

```
public void rnSendMessage() throws Exception
   {
     String rnInfo = "Service Content";
     XmlObject xObj = XmlObject.Factory.parse(rnInfo);
     RosettaNetContextDocument doc = rn.sendMessage(xObj);
     System.out.println(doc.toString());
   }
```

Where *Service Content* is the service content of your RosettaNet message.

- *Participant business process receiving a message:*

```
public void onMessage(RosettaNetContextDocument doc, XmlObject msg)
   {
     System.out.println(">>>>> ContextParticipant.onMessage()");
     RosettaNetContext context = doc.getRosettaNetContext();
     System.out.println("    from=" + context.getFrom());
     System.out.println("    to=" + context.getTo());
     System.out.println("    pip=" + context.getPip());
     System.out.println("    failure-report-admin=" +
        context.getFailureReportAdministrator());
   }
```

- *Participant business process interface for callbacks:*

```
public interface Callback
   {
     /**
     * @common:message-buffer enable="false"
     */
     public RosettaNetContextDocument sendReply(XmlObject msg);
     /**
     * @common:message-buffer enable="false"
     */
     public void sendReceiptAcknowledgement();
     /**
     * @common:message-buffer enable="false"
     */
     public void sendError(String msg);
   }

public Callback callback;
```

- *Participant business process sending a reply:*

```
public void reply()
   {
     XmlObject xObj = null;
     try {
       xObj = XmlObject.Factory.parse("Service Content");
     } catch (Exception e) {
       e.printStackTrace();
     }

     RosettaNetContextDocument doc= callback.sendReply(xObj);
     System.out.println(doc.toString());
   }
```

Where *Service Content* is the service content of your RosettaNet message.

Related Topics

RosettaNet Control Annotations

RosettaNet Control Annotations

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

Topics Included in this Section:

@jc:rosettanet Annotation
Specifies the JCX class-level annotations for the RosettaNet control

@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:

Attribute Name	Description
`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.

Attribute Name	Description
pip	RosettaNet PIP code, such as 3B2. Must be a valid PIP code as defined in http://www.rosettanet.org/pipdirectory.
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.