# BEA WebLogic Integration™®

## Using the Worklist System

Version 8.1
July 2003

# Copyright



# Restricted Rights Legend



# Trademarks or Service Marks

# Contents

## About This Document



## 1. Introduction

# 2. Using Worklist Controls



# 3. Creating and Managing Worklist Tasks

# 4. Using the Worklist Control Methods

# 5. Advanced Topics

# About This Document

This document provides information about using the WebLogic Integration 8.1 Worklist System. It includes the following topics:

- Chapter 1, "Introduction."
- Chapter 2, "Using Worklist Controls."
- Chapter 3, "Creating and Managing Worklist Tasks."
- Chapter 4, "Using the Worklist Control Methods."
- Chapter 5, "Advanced Topics."

## What You Need to Know

This document is intended mainly for application developers who have an in-depth knowledge of Java or other object-oriented programming languages; business process management, especially business process design; B2B integration; data integration; and WebLogic Server security. Additionally, you should have a basic understanding of WebLogic Integration 8.1 and WebLogic Workshop.

## e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the "e-docs" Product Documentation page at http://e-docs.bea.com.

# How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File→Print option on your Web browser.

A PDF version of this document is available on the WebLogic Integration documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Integration documentation Home page, click the PDF files button and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at http://www.adobe.com/.

# Related Information

The following WebLogic Integration documents contain information that is relevant to building Worklist applications:

- Tutorial: Building a Worklist Application
- Worklist Administration in Managing WebLogic Integration Solutions
- Business Calendar Configuration in Managing WebLogic Integration Solutions

# Contact Us!

Your feedback on the BEA WebLogic Integration documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Integration documentation.

In your e-mail message, please indicate which version of the WebLogic Integration product and documentation you are using.

If you have any questions about this version of BEA WebLogic Integration, or if you have problems installing and running BEA WebLogic Integration, contact BEA Customer Support through BEA WebSupport at **www.bea.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

# Documentation Conventions

The following documentation conventions are used throughout this document.

| Convention | Item |
|---|---|
| **boldface text** | Indicates items that are displayed on the User Interface. |
| Ctrl+Tab | Indicates that you must press two or more keys simultaneously. |
| *italics* | Indicates emphasis or book titles. |
| `monospace text` | Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.<br>*Examples*:<br>`#include <iostream.h> void main ( ) the pointer psz`<br>`chmod u+w *`<br>`\tux\data\ap`<br>`.doc`<br>`tux.doc`<br>`BITMAP`<br>`float` |
| **`monospace boldface text`** | Identifies significant words in code.<br>*Example*:<br>`void` **`commit`** `( )` |

| Convention | Item |
|---|---|
| *monospace italic text* | Identifies variables in code.<br>*Example*:<br>`String expr` |
| UPPERCASE TEXT | Indicates device names, environment variables, and logical operators.<br>*Example*s:<br>LPT1<br>SIGNON<br>OR |
| `{ }` | Indicates a set of choices in a syntax line. The braces themselves should never be typed. |
| `[ ]` | Indicates optional items in a syntax line. The brackets themselves should never be typed.<br>*Example*:<br>`buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...` |
| \| | Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed. |
| `...` | Indicates one of the following in a command line:<br>• That an argument can be repeated several times in a command line<br>• That the statement omits additional optional arguments<br>• That you can enter additional parameters, values, or other information<br>The ellipsis itself should never be typed.<br>*Example*:<br>`buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...` |
| .<br>.<br>. | Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed. |

CHAPTER 1

# Introduction

This section provides an overview to the WebLogic Integration Worklist system. It introduces the components of the system, including the built-in WebLogic Integration Worklist controls, Worklist system Tasks, and task administration through Worklist user interfaces and through the WebLogic Integration Administration Console.

This section includes the following topics:

# What is the WebLogic Integration Worklist System?

Business User Integration enables people to use the Web to collaborate in business processes. Worklist Web interfaces enable task assignment and status tracking. The Worklist system provides capabilities for user, group, and role management, and routes business process tasks to the person with the right skills, at the right time, enhancing organizational efficiency and responsiveness.

People are at the center of the flow of work in an office environment. Decisions are made by people and work is assigned and managed by people. People do and manage the work that achieves the goals of enterprises. For people to succeed, the things they do must be managed carefully—who is doing the work, and how it should be done.

Documents usually accompany the flow of work in an office. Documents supply the background required to complete tasks. It is important to the flow of work that documents are routed appropriately within and between enterprises.

A typical example of a documents that provide background for tasks is a purchase order—it must be approved and forwarded to subsequent people in a defined flow before those people can act on it.

When work is flowing through people, management must have timely status updates. That is, they must receive notifications when deadlines pass and work goes overdue. Human activity is by nature error prone. When work is completed, detailed records are necessary. Management must study histories to understand where they can alter processes to fine tune them, increase productivity and lower response times.

Business Process Management (BPM) is commonly used for scenarios in which human activity combines with automated processing over time (business processes can be short- or long lived; work can continue over days or months). BPM systems must allow:

- People to concentrate on just those parts where they add value—where they are necessary to make a decision
- People to act in specific roles when they work within a process
- Business rules to dictate the routing of documents and work

BPM systems must coordinate people, documents, and events as the work in an enterprise proceeds. They must manage permissions for work items—for example, systems must manage who can see what documents and queues, and who has ownership of tasks.

Leveraging Worklist features in a BPM system ensures faster response times and higher productivity. The WebLogic Integration Worklist is such a task management system that allows enterprises to accomplish the integration of human activity and automated business processes.

# Worklist Tasks

The Worklist system allows for the creation, manipulation, and management of Tasks.

A Task Instance represents something that must happen, or some unit of work that must be done within a certain period of time. When work is done to address a given task, a Task Instance can be used to record that fact and the details of the work that was done.

Examples of tasks include:

- A Manager must read and approve an employee's vacation request.
- A customer must be phoned and their complaint recorded.

A Task Instance is an independent object in the Worklist system at run time. Tasks exist independently of any processes or controls. Multiple processes can interact with a given task throughout the lifecycle of that task. Different processes can interact with a given task simultaneously.

# Task Data Values

Task Instances have built-in data values for defining how work should be done, who should do it, by when it needs to be completed, and so on. These data values also are used to capture what actually was done when work is completed.

Examples of task data values include:

- An assignees list of users and group who can work on the task.
- Task Completion Due dates.
- The claimant. That is, the user who claims the Task as their own and attempts to complete the work.
- The Task owner. That is, the person responsible for managing the process of the work getting done.
- Request and Response documents that can describe the work to be done and the results of completing the work.
- A state that defines the point at which a Task instance is in its lifecycle.

- A priority that indicates the relative urgency of this task relative to other tasks.

## Due Dates

Due dates represent the date and time at which tasks should be claimed and completed. The Worklist system stores Due Dates as java.util.Date objects, tracks them, and can trigger callbacks to business processes that are listening for the Task due dates. Due dates can be set using a specific date and time. Or, they can be set using a business calendar and a business time duration (an example of a business duration specification is *3 business days from now on the customer support calendar*). Business calendars represent the operating hours of a business. A business calendar specifies a time zone and a set of time period rules. The time period rules determine the days, dates, and hours that are free (available for business activities) and busy (unavailable for business activities).

To learn about Task Due Dates and Business Calendars, see "Task Due Dates" on page 3-2.

## Task State

The *state* of a Task describes the point at which the task is in its life cycle. Each task instance is in one of the following states: ASSIGNED, CLAIMED, STARTED, COMPLETED, SUSPENDED, or ABORTED.

Operations on the controls or the API allow you to cause an instance to transition from its current state to another state.

To learn about Task States, see "Task Due Dates" on page 3-2.

## Task Owners

A Task Instance can optionally have a Task Owner, to signify a user or a group of users that play a managerial role with respect to this task. The owner may not actually complete the Task, but is responsible for the Task becoming completed, or manages the process of the Task getting completed. The permission to perform certain managerial operations on a Task instance can be restricted to only the Task Owner or an administrator.

For example, the manager of a café can be the owner of a task assigned to a chef to prepare a recently ordered dish. The manager takes responsibility if it does not get done.

To learn about Task Owners, see "Task Users and Groups" on page 3-8 and "Tasks and User Permissions" on page 3-10.

## Assignee Lists and Claimants

A Task Instance has an assignees list to specify which users can claim the task. The assignees list can contain both users and groups. When a user on the assignees list claims the task, that user become the *claimant*. The claimant takes ownership of the task, and performs the work needed to complete the task. The *State* of the Task is set to **Claimed** when a user claims the Task.

To learn about assignee lists and claimants, see "Task Users and Groups" on page 3-8 and "Tasks and User Permissions" on page 3-10.

## Request and Response Documents

The Task Request is generally used to specify how work should be done, or what should be done. This value can be read by the task worker who performs the work and completes the task. In addition, assignees can view this information to decide whether or not to claim the task. For example, the Task Request can contain a Purchase Order document that needs to be approved by a user.

The Task Response is generally used to specify what actually took place after a user has worked on the task. It can describe the results of the work that was done to complete a task. Callbacks can pass the Response value to business processes that are waiting for a particular task state.

For example, the Task Response can capture the agreement made between a Collections Agent and a delinquent customer after the two complete a phone conversation. The process that created the Task to call that customer can use those results to determine what to do next.

To learn about request and response documents, see "Request and Response Documents" on page 3-18.

## Operations on Tasks

Operations are used to create new tasks, alter task states or data values, delete tasks, or read information about an existing task. Some operations allow combinations of these actions in a single step. Examples of Task operations include:

**Operations to Create Tasks**

When a new Task Instance is created, the Worklist system assigns a unique ID (a taskID) to that instance. The Task state can be defined at creation time as Assigned or Claimed, depending on the operation. Some Task data values are specified at the time an instance of a Task is created and can not be changed after the instance is created.

**Operations to Modify Task Properties**

Some Task properties can be specified and modified after an instance of a Task is created.

**Operations to Get Task Properties**

You can use *get task* operations to access the properties of any Task Instance at any point during its life cycle.

**Operations to Modify Task State**

Task Instances can transition between states based on the operations defined for them. Different operations to modify a task's state are valid depending on the state in which the task resides before the operation is invoked.

To learn about the operations on Task and Task Worker controls, see "Using the Worklist Control Methods" on page 4-1.

# Archiving and Purging Task Information

WebLogic Integration supports archiving tracking data for business process instance history, trading partner message history, and task instance history.

As tasks go through their lifecycle in enterprise processes, their properties are modified, their states change, their due dates expire, and so on. Worklist task instances generate events that can be logged in Worklist history tables in the run-time repository. The records created in the archive tables are intended to be used by reporting applications. Those applications can query the tables to generate reports or statistical analyses of historical task processing. The following types of events can be tracked:

- Changes in task state and associated values
- Expiration of task claim or complete due date
- Changes in task owner or assignees
- Task requests and task responses
- The request and response XML

The tracking data is stored in a database at run time. We recommend that to optimize performance, the amount of tracking data stored is kept to a minimum. To this end, the archive and purge process can be configured to run at regular intervals set by an administrator. In addition to configuring the schedule, the administrator can enable or disable the archiver:

- When the archiver is enabled, the process copies the data to an offline database, then purges it from the runtime database.

- When the archiver is disabled, the process purges the data from the runtime database without copying it.

Archived information can be used for generating reports and compiling statistics about task processing in your WebLogic Integration application. To learn more about archiving and purging Task data, see "Additional Resources" on page 5-20.

### Task Queries

Task Queries allow an application to find all tasks in the Worklist system runtime that meet a general set of criteria. These queries are directly analogous to SQL and Databases Tables. The queries specified return results that contain information about all tasks that meet those criteria.

For example, you can create a custom user interface to show all tasks in the system that are assigned to the current user, have a priority equal to one, and are due in the next three days.

## Controls and Worklist APIs

The Worklist API provides operations to leverage all of the functionality available in the Worklist system for creating and operating on tasks. WebLogic Integration provides two controls to support the Worklist system: the Task control and the Task Worker control. The Task and Task Worker controls provide a subset of the functionality of the API. However, they provide the convenience of the WebLogic Workshop controls framework. Controls can be used easily in your business processes.

### Task Control

In an office environment, there are people that create and specify work to be done. These are usually managers, and they generally monitor the progress of these work items. The Task control is designed to provide the common operations required by the manager of a work item, such as creating the work, assigning the work, and receiving notifications when work completes or becomes overdue for completion. The most common usage of the Task Control is in business processes.

### Task Worker Control

There are also people that receive assignments and perform the work. The Task Worker control is designed to provide the common operations needed by those workers. For example, querying the task system for those that meet custom criteria (for example, a worker can query for the tasks that are assigned to themselves and due this week), claiming a task to mark ownership of it, or

completing a task to mark its completion. Because workers typically operate on work items through a user interface when a task management system is in place, the Task Worker control is most commonly used in the implementation of a custom Worklist user interface.

In addition, the Task Worker control has operations of an administrative nature, for example claiming a task on another user's behalf.

## Callbacks

Task Controls can notify a process when a Task's state changes in some way. Common callbacks include the expiration of a completion due date, the task being aborted, or getting completed. Callbacks allow processes to block somewhere in their logic, effectively waiting until that event takes place.

## Operations

Operations on controls are the general mechanism by which a business process or a user interface create new tasks, read or alter the task data values, read the current state, cause a transition to a new state, or delete a task instance.

## Controls are Extensible

Task and Task Worker controls have a built-in set of operations and callbacks. Controls are extensible through Java annotations. For example, you can define custom operations with custom signatures and custom callbacks for your controls. Through the annotations on these operations, you can configure a control’s data values, create new tasks, update existing tasks, and so on.

The Task and Task Worker controls provide operations that are commonly used, but they can be extended to provide operations tailored to specific use cases. Signatures on methods can be altered to take XML Bean types as arguments, when XML data associated with tasks conforms to custom defined schemas in your application.

New methods can be added to perform several updates at once. For example, you can write a new method to add an operation to create a task, set its priority and comment, and assign it to a user whose name is passed in, all in one step.

Callbacks can be added to detect other state change in a task. For example, you can add a callback to detect when a task is claimed.

# Administration and Management

You can administer and manage the Worklist system, specifically the tasks in the system, business calendars, task properties, and so on using the Worklist Administration module in the WebLogic Integration Administration Console and through a custom Worklist client you can create using the controls and the Worklist API (the Worklist API is available as EJBs and MBeans):

- WebLogic Integration Administration Console
- Worklist User Interfaces

## WebLogic Integration Administration Console

The Worklist Administration module in the WebLogic Integration Administration Console allows application administrators to administer and monitor the task instances in your WebLogic Integration application. The Worklist-specific administration and management functions you can perform include:

- View summary or detailed task status in order to monitor the progress of task completion against due dates
- Perform queries to show individual workload
- Reassign tasks in order to speed progress
- Change task properties, such as state or due date
- Control task routing by creating or changing substitute routing rules

To learn about using the administration console to manage your Worklist Tasks, see Worklist Administration in *Managing WebLogic Integraton Solutions*, which is available at the following URL:

`http://edocs.bea.com/wli/docs81/manage/index.html`

## Worklist User Interfaces

Worklist user interfaces enable end users—task creators, task workers, task administrators—to interact with running business processes for handling process exceptions, approvals, status tracking, and so on. A sample Worklist user interface is provided in WebLogic Integration, but you can also write your own custom user interfaces to support task assignment, approvals, and so on.

The majority of real-world applications involve a custom user interface that leverage the Worklist system functionality. People typically interact with Tasks in WebLogic Integration through these custom user interfaces. Pages that interact with the Worklist system are often added to a company's existing user interfaces. For example, a page can be added to an existing order fulfillment UI on a company's intranet, allowing a manager to approve very large orders. We recommend that you study the sample Worklist UI and use the strategies found there to design your custom UIs.

The Worklist Controls and API provide the operations needed to design custom UI pages that can:

- Query for tasks that meet certain criteria
- Access and present detailed information about particular tasks
- Collect data on screen and use it to perform operations on tasks that update them in some way.

You can use NETUI to leverage the Worklist controls in your custom user interfaces. You can also use the Worklist API with JavaServer Pages (JSPs).

You can create a custom Worklist user interface using the Worklist controls and the Worklist API. To learn about the Worklist API, see the `com.bea.wli.worklist.api` package in the BEA WebLogic Integration Javadoc, which is available at the following URL:

`http://edocs.bea.com/wli/docs81/javadoc/index.html`

WebLogic Workshop controls can be invoked from a Web page. You can create a custom Worklist user interface or portal using the WebLogic Workshop tools to manage Web applications using JSPs and Page Flows. For more information, see Developing Web Applications and Page Flows and JSPs in the WebLogic Workshop online help.

To access the sample Worklist user interface in your installation of WebLogic Platform, first start WebLogic Server, then invoke the Worklist:

1. Start WebLogic Workshop.
2. From the WebLogic Workshop menu, select **Tools→WebLogic Server→Start WebLogic Server**
3. When the server is running, start the sample Worklist UI by selecting the following options from the WebLogic Workshop menu:

   **Tools→WebLogic Integration→Worklist**

4. Login to the Worklist—the username and password for the Worklist in the sample integration domain are `weblogic`/`weblogic`.

   A JSP page is displayed. It allows you to view and manage the tasks in your application.

CHAPTER 2

# Using Worklist Controls

Java Controls are server-side components managed by the Workshop framework. Controls expose Java interfaces that can be invoked directly from your business process. In other words, controls represent the interfaces between your business processes and other resources.

This section describes the built-in controls provided by WebLogic Integration to support the integration of business users with business processes. It includes the following topics:



## Which WebLogic Integration Controls Support the Worklist System?

WebLogic Integration provides two build-in Java controls to support the integration of business users via the Worklist system: the **Task** control and the **Task Worker** control.

As is the case with other built-in controls in WebLogic Workshop, you use the controls by adding instances of the controls to your business process and then invoking operations on the controls at the point in the business process at which you want to integrate the business user logic.

The underlying control implementation takes care of most of the details of the interaction for you. Business processes invoke operations on the controls using Control Send and Control Send with

Return nodes. Business processes can block at Control Receive nodes waiting for events to be returned from controls. In other words, Control Receive nodes are triggered by control callbacks. You can extend Worklist controls through Java annotations. Common extensions include implementing callback functions and performing system queries.

The operations invoked on the controls allow the process to create tasks, get information about tasks, update tasks, and so on.

- The **Task control** enables a business process or UI to create a single Task instance, manage its state and data, and provide callback methods to report status, such as when the Task status changes or the Task is overdue. Each Task control operates on a single active Task instance.
- The **Task Worker control** enables a business process or UI to assume ownership of Tasks, work on them, complete them, and offers administrative operations, including operations to start, stop, delete, assign, and so on. Task Worker controls allow operations on several Task instances at the same time.

# Creating a Task Control

An instance of a Task control can be used to create a single task instance. If multiple tasks need to be created, use a factory type of Task control. (To learn about factories, see "Using Task Control Factories" on page 5-18.)

A Task control instance can also be used to interact with a task instance that already exists, by setting its *active task id.* After creating or setting the active task id, your control instance can be used to get information about that task or update that task in various ways.

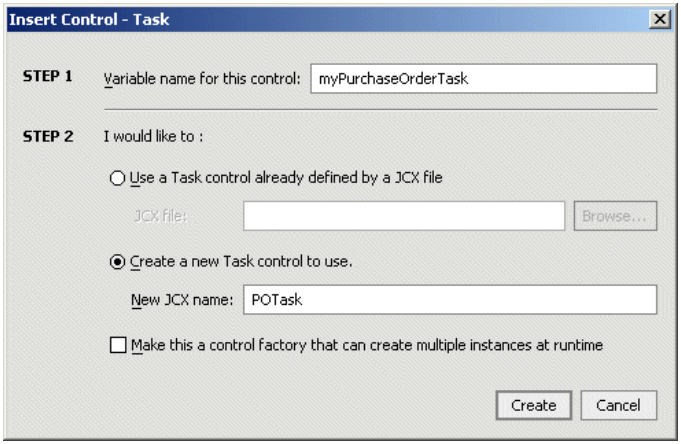
You can customize Task controls for different business purposes, by adding new operations or callbacks, or by altering the signatures of existing operations or callbacks.

To create a new Task control:

1. Open your WebLogic Integration application in WebLogic Workshop.
2. In the **Application** pane, double-click the business process (JPD file) to which you want to add the logic to integrate business users using the Worklist system. The business process is displayed in the **Design View**.
3. Click **Add→Integration Controls** on the **Controls** tab of the **Data Palette** to display a list of integration controls that represent the resources with which your business process can interact.

**Note:** If the Controls tab is not visible in WebLogic Workshop, click **View→Windows→Data Palette** from the menu bar.

4. Choose **Task**. The **Insert Control** dialog box is displayed.




5. In the **Insert Control** dialog box (**Step 1**), enter a name for the instance of this control. The name you enter must be a valid Java identifier.

6. In the **Insert Control** dialog box (**Step 2**), select one of the following options:

   – Use a Task control already defined by a JCX file

     Enter a filename for the Task control in the **JCX file** field, or click **Browse** to find the JCX file in your file system.

   – Create a new Task control to use

     Enter a filename in the **New JCX name** field.

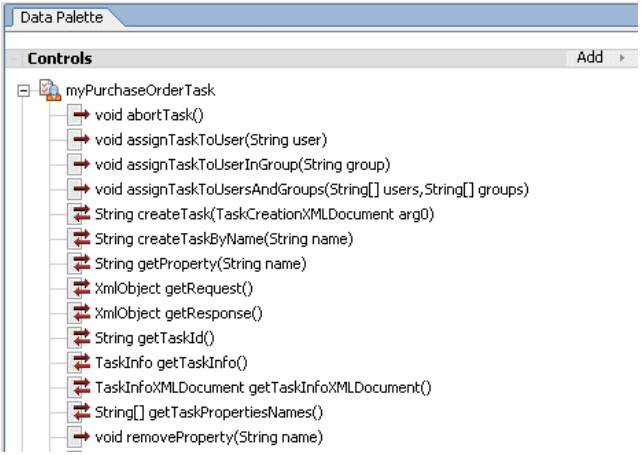
7. Choose whether you want to make this a control factory by selecting or clearing the **Make this a control factory that can create multiple instances at runtime** check box.

   To learn about control factories, see "Using Task Control Factories" on page 5-18.

8. Click **Create**. A new Task control and an instance of it are created and the **Insert Control** dialog box is closed.

   A new JCX file is created and displayed in the **Application** tab in the WebLogic Workshop IDE. (You can double-click on any JCX file to view or edit it in the **Design** or **Source** View.) The instance of the control is displayed in the **Controls** tab on the **Data**

**Palette**. Expand the control Instance by clicking the + beside its name on the **Data Palette** to display the base methods provided on a Task control.




9. After you create an instance of the Task control in your business process, you can design the interaction of the business process with the Task control by simply dragging and dropping the Task control methods from the **Data Palette** onto the **Design View** at the point in your business process at which you want to design the interaction.

   For examples of designing interactions between a business process and an instance of a Task control, see "Using Task and Task Worker Controls in Business Processes" on page 2-7.

10. After you create a Task control in your business process, you can view and edit the properties of the control type (represented as a JCX file in the **Application** pane in WebLogic Workshop) or of the instance of that control type (represented in the **Data Palette**) in the **Property Editor** in the WebLogic Workshop design environment.

    Task Instances have data values associated with them, many of which are set when the task is created. The properties on a Task control can be used to set the default values for some of these data values, which are used whenever that control instance is used to create a new task. Note that the properties set on a factory type Task control propagate to any Task control instances created from that factory.

    To learn about Task control properties, see "Using the Task Control Property Editor" on page 4-15.

**Note:** To learn how to use the **Property Editor** to specify properties for control types versus control instances, see Setting Control Properties in the WebLogic Workshop online help, which is available at the following URL:

`http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideControlsProperties.html`

# Creating a Task Worker Control

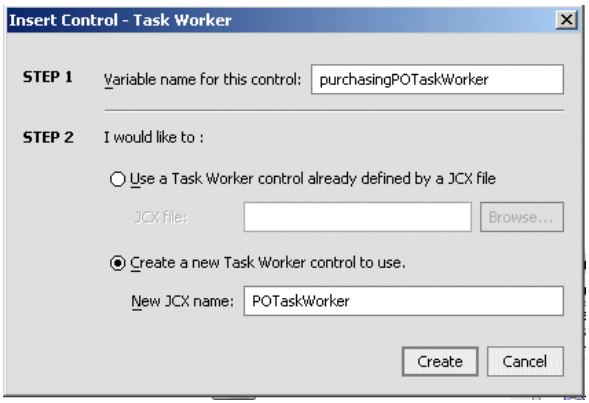
A Task Worker control provides operations for working with one or more Task instances. The control also provides administrative operations for managing one or more Task instances. You can customize each Task worker control for different business purposes.

This topic describes how to create a new Task Worker control. Task Worker controls do not have any properties to configure.

1. If you are not in Design View, click the **Design View** tab.
2. Click **Add→Integration Controls** on the **Controls** tab of the **Data Palette** to display a list of controls that represent the resources with which your business process can interact.

   **Note:** If the Controls tab is not visible in WebLogic Workshop, click **View→Windows→Data Palette** from the menu bar.

3. Choose **Task Worker**. The **Insert Task Worker** dialog box appears.




4. In the **Insert Control** dialog box (**Step 1**), enter a name for the instance of this control. The name you enter must be a valid Java identifier.
5. In the **Insert Control** dialog box (**Step 2**), select one of the following options:

– Use a Task Worker control already defined by a JCX file

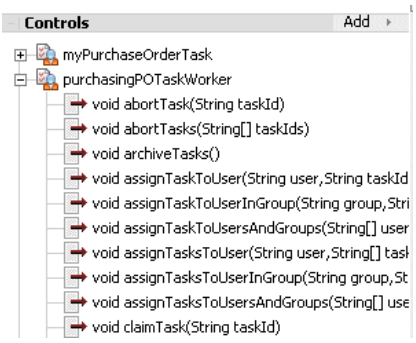
  Enter a filename for the Task Worker control in the **JCX file** field, or click **Browse** to find the JCX file in your file system.

– Create a new Task Worker control to use

  Enter a filename in the **New JCX name** field.

6. Click **Create** to close the **Insert Control** dialog box.

   When you click create, the control JCX file is displayed in the **Application** tab in the WebLogic Workshop IDE. You can double-click on any JCX file to view or edit it in the **Design** or **Source** View. The instance of the control is displayed in the **Controls** tab on the **Data Palette**. Expand the Control Instance by clicking the + beside its name on the **Data Palette** to display the base methods provided for this control. The following figure shows an example of a Task Worker control instance displayed in the **Controls** tab on the **Data Palette**.




7. After you create an instance of the Task control in your business process, you can design the interaction of the business process with the Task control by simply dragging and dropping the Task control methods from the **Data Palette** onto the **Design View** at the point in your business process at which you want to design the interaction.

   For examples of designing interactions between a business process and an instance of a Task control, see "Using Task and Task Worker Controls in Business Processes" on page 2-7.

# Using Task and Task Worker Controls in Business Processes

To design the interaction of a Task or Task Worker control with a business process, you must decide which methods on the control you want to call from the business process to support the business logic.

In the same way you design the interactions between business processes and other controls in the WebLogic Workshop graphical design interface, you can create the appropriate control node in your business process (**Control Send**, **Control Receive**, and **Control Send with Return**) and bind a Worklist control method to that node by simply dragging a control method from the **Data Palette** onto the business process in the **Design View** at the point in your business process at which you want to design the logic.

To learn how to use Worklist controls in a business process, we recommend you complete the steps in the following tutorial: Tutorial: Building a Worklist Application, which is available at the following URL:

`http://edocs.bea.com/wli/docs81/wltutorial/index.html`

Creating Task and Task Worker Controls is the first step in the process of creating a Worklist system. Before you begin working with the Task and Task Worker controls, you need to understand the features and components of the Worklist system available to you. You also need to work with the control methods and properties. To learn about Task and Task Worker properties, see "Using the Task Control Property Editor" on page 4-15.

CHAPTER 3

# Creating and Managing Worklist Tasks

Task Instances have properties that define what work needs to be done, who does the work, how the work is done, and so on. This section describes the details of working with task instances. It includes the following topics:

# Overview

Task instances are part of the WebLogic Integration server and exist independently of Worklist controls or specific business processes. Tasks remain in the run-time engine indefinitely, until they are either explicitly deleted or purged by the WebLogic Integration purging process. You create, delete, and manage Tasks through the following mechanisms:

- The Task and Task Worker controls in WebLogic Workshop.
- The Worklist module the WebLogic Integration Administration Console.
- The public Worklist API, using Enterprise Java Beans, and Message Beans.

There are no task *types*, all task instance lifecycles conform to the same state diagram and have the same types of data associated with them. In other words, task instances cannot be extended (in the Object Oriented sense).

# Task Due Dates

Business processes can take steps to handle work that becomes overdue by setting and tracking Task due dates. For example, a purchase order business process can email the manager assigned to approve the purchase order if that manager takes more than three business days to do so.

This section describes Task Dates and calendars. It contains the following topics:



## Claim and Completion Due Dates

You can optionally set one or both of the following due dates for a Task Instances:

- **Claim Due Date**—specifies by what time the task should be *Claimed* by a user on the assignees list. Setting **claimDueDate** to null indicates that there is no due date.
- **Completion Due Date**—specifies by what time the task should have reached the *Completed* state. Setting **completionDueDate** to null indicates that there is no due date.

Due Dates are stored as `java.util.Date` values. They mark a precise instant in time. You can specify Due Dates using Business Time or system time. (To learn about business time, see "To Set Task Due Dates Using Business Time" on page 3-3).

At run time, when the Worklist system detects that a due date has passed, it checks whether the associated task is claimed or completed. If the task is not claimed or complete, callbacks are invoked on any Task controls that are blocking on the task becoming overdue. Business processes can incorporate these callbacks that are invoked when due dates expire, allowing the processes to execute logic when the task becomes overdue.

## To Set Task Due Dates Using Absolute Time

You can set Task due dates by specifying a `java.util.Date`. The due date is a specific instant of time. You can unset Task due dates by passing `null` for the `java.util.Date`.

## To Set Task Due Dates Using Business Time

You can set Task due dates by specifying a duration of business time. Business Time durations are strings that define a period of time relative to a specified Business Calendar. Business calendars are required to convert business time durations to real time. In other words, business time durations have no meaning if they are not associated with a business calendar that converts the durations to real time. The Worklist system uses the `addBusinessTime` method to calculate the due dates.

For example, if a business calendar defines business hours as Mon, Tue, and Wed from 9AM to 5PM. If, on Saturday the 16th, you set a Task's completion due date with a duration of 4 business days on the specified calendar, the resulting due date is Monday the 30th.

A different due date is calculated if Tuesdays are removed as free time on the specified calendar.

To learn more about business calendars and the WebLogic Integration Administration Console, see Business Calendar Configuration in *Managing WebLogic Integration Solutions*

## To Specify a Calendar to Use When You Set Due Dates

If you use a business time duration, but do not specify a business calendar to use, the WebLogic Integration System Business Calendar is used. You specify a business calendar for the system to use when it calculates due dates by explicitly passing the name of the business calendar to a Task control. To learn how, see "Specify a Due Date for Completion of the Task" in Step 4. Create Task and Assign to User in *Tutorial: Building a Worklist Application*.

Alternatively, you can specify a user or group. In this case, the business calendar associated with that user or group is used to calculate the due date. To learn how to associate business calendars with users and groups in your system, see "Assigning Business Calendars to Users and Groups" in Business Calendar Configuration in *Managing WebLogic Integration Solutions*

## Formats for Business Time Duration

Business time durations are strings, in the following format: X d Y h Z min. Note that only days, hours, and seconds are supported for specifying business time durations. You can specify days, hours, or minutes only, or some combination of the three. For example, you can specify just days, or just hours and minutes:

- 3 business days = 3 d
- 2 business hours and 30 business minutes = 2 h 30 min.

To learn more about the business calendar options of the WebLogic Integration Administration Console, see Business Calendar Configuration in *Managing WebLogic Integration Solutions*.

# Task States

The Task and Task Worker controls allow a business process or Worklist UI to cause a Task Instance to transition from one state to another. Operations on the controls or the API guide the task through its lifecycle.

A Task can be in one of the states defined in Table 3-1. Many of the methods on Worklist controls make changes to states or properties of a Task instance. The transitional state operations for Worklist controls are defined in Table 3-2.

**Note:** The operations that can be invoked for a given Task are dependent on the state of the task and the user permissions. To learn about user permissions, see "Tasks and User Permissions" on page 3-10.

**Table 3-1 Task States**

| State | Description |
|---|---|
| ASSIGNED | New tasks begin in the ASSIGNED state. The assignees list is important in this state, as is specifies which users are allowed to become the claimant through the act of claiming the task.<br>Note that the assignees list may be empty, in which case the task is assigned to nobody, effectively unassigned. |
| CLAIMED | Claiming an ASSIGNED task causes the state to become CLAIMED. The claimed state specifies that a user on the assignees list has taken ownership of the task, and intends to complete the task. The *claimant* value will be set when a Task is CLAIMED.<br>Although the claimant has ownership of the work, the claimant may not yet have started working on it. |
| STARTED | The STARTED state indicates that the claimant started working on the task—is currently spending time on the work required to complete the task. The STARTED state exists for reporting purposes, allowing companies to track precisely how much time users spend working on individual tasks.<br>There can be significant time between declaring ownership of work (claiming the task) and starting doing that work. |
| COMPLETED | The COMPLETED state indicates that the work required to complete the task is finished, or as much of the work as is possible to do is finished.<br>The response document can be used to record the details of how work was done or the results of doing work. |
| SUSPENDED | A SUSPENDED task is frozen. In other words, it cannot be worked on. SUSPENDED tasks can be resumed at a later time, returning to the state they were in when they were suspended.<br>The SUSPENDED state can be used to temporarily mark that a task cannot progress for some reason. |
| ABORTED | An ABORTED task is effectively cancelled. The ABORTED state is generally used to indicate that something went wrong while work was being done on the task.<br>This state can also be used to mark work that should be permanently abandoned. |

**Table 3-2 Control Methods That Operate on Task States**

| Operation | Description |
|---|---|
| create | Creates a new task instance in the ASSIGNED state.<br>Some data values, such as the description, can only be set when this operation is invoked. Note that the currently executing principal must belong to a group that is specific to the Worklist system before they are granted the permissions to create a new task. |
| assign | Causes a task to move to the ASSIGNED state. The assignees list must be set when this operation is performed, specifying which users can claim the task.<br>This operation can be performed on tasks in a final state, such as COMPLETED or ABORTED. In this way, the task can be worked on again.<br>This operation can unassign or reassign a task. Assignment can be performed on a single task instances multiple times throughout its lifecycle. When assigning a task, an algorithm must be specified to determine how to set the assignees list. To learn about the algorithms, see "Assignment Algorithms" on page 3-7.<br>This operation is performed by the task owner, task creator, an assignee, or an administrator. |
| claim | Causes a task in the ASSIGNED state to become CLAIMED. A user that is on the assignees list is set as the claimant of the task. This signifies that a user on the assignees list has marked ownership of the task and intends to complete it. This operation is performed by a user who wishes to become the claimant for a task, or by an administrator or task owner on behalf of another user. |
| start | Causes a task in the CLAIMED state to become STARTED—signifies that the claimant is starting to work on the task. This operation is performed by the claimant, or by an administrator or task owner on behalf of a claimant. |
| stop | Causes a task in the STARTED state to return to the CLAIMED state—signifies that the claimant is stopping work on the task, possibly temporarily. They can start it again when they are ready to continue working. This operation is performed by the claimant, or by an administrator or task owner on behalf of the claimant. |
| complete | Causes a task in the STARTED state to become COMPLETED—signifies that the claimant is stopping work on the task, possibly temporarily. They can start it again when they are ready to continue working. This operation is performed by the claimant, or by an administrator or task owner on behalf of the claimant. |

| Operation | Description |
|---|---|
| suspend | Causes a task to become SUSPENDED—signifies that the task no longer progresses and should not be worked on, possibly temporarily. The task can be resumed (using the resume operation) when work should continue. This operation is performed by an administrator or task owner. |
| resume | Causes a SUSPENDED task to return to the state it was in when it was suspended. This operation is performed by an administrator or by the task owner. |
| abort | Causes a task to become ABORTED—signifies that the task should be cancelled and should not complete. In other words, work on the task is no longer necessary and should cease. This operation is performed by the claimant, an administrator, or the task owner. |

# Assignment Algorithms

Whenever a Task is assigned, one of the following assignment methods must be specified: `assignToUser`, `assignToUserInGroup`, `assignToUsersAndGroups`. These methods specify how the assignees list is set.

## assignToUser

This method sets the assignees list to a specific IntegrationUser. The name of the user must be specified. Because this user is the only one on the assignees list, this operation automatically causes the task to be claimed for the specified user. For an example of how to use this method, see "Assign the Task to a User" in Step 4. Create Task and Assign to User in *Tutorial: Building a Worklist Application*.

## assignToUserInGroup

This method behaves in the same way as the `assignToUser` method. However, when you use the `assignToUserInGroup` method, a load balancing algorithm is used to select the user in the specified group that has the fewest claimed tasks that are not completed, aborted, or suspended. A group name must be specified for this method.

### assignToUsersAndGroups

This method sets the assignees list to contain the users and groups specified. This operation requires a list of user names, or a list of group names, or both. Any of the users on the assignees list can then claim the task.

# Task Users and Groups

People and systems may be play various roles with respect to a task instance. They can be the Task Owner in the role of managing the task, a user on the assignees list who may claim the task, the claimant who has declared ownership and intent to complete the task, or an WLI Administrator.

The following list describes the roles in which a given user can be with respect a task instance. These roles determine what operations they are permitted to perform on a Task instance.

To learn more about the permissions that allow different groups and users to create and manage task instances, see "Tasks and User Permissions" on page 3-10.

- Task Owners
- Assignee Lists
- Claimants
- Integration Administrators
- Task Creators

## Task Owners

The Task Owner is the user or group with managerial responsibility for the work required to complete a Task Instance. For example, a dispatcher at a taxi company can be the task owner for tasks assigned to drivers to deliver a patron to his required destination.

Although a Task Owner usually does not complete the task, they can perform managerial operations on the task. For example, the manager in a collections office can reassign the task of calling a delinquent customer to a different collections officer when the officer originally assigned to the task (the claimant) is on vacation.

Task owners have administrative privileges for the tasks they own. They are effectively in the role of the WebLogic Integration Administrator when permissions are checked in the event an operation is invoked on that task. Note that the Task Owner is set automatically to the current user when a task is created, unless a different Task Owner is explicitly specified at creation time.

## Assignee Lists

The Assignees List specifies the users or groups that are permitted to take ownership of a task by claiming it. All instances of Tasks have an associated assignee list. When a task is assigned or reassigned, the assignee list is updated and the state of the Task is set to ASSIGNED.

The assignees list that is associated with a task can specify several users or groups (or both), but only one user can claim a Task to perform work. For a user to be on the assignees list, either the user name is explicitly on the list or a group to which the user belongs is explicitly on the list. The user in the assignee list that claims the task, becomes the *claimant*.

## Claimants

A Claimant is the users from the Assignees list who claims a Task and performs the work needed to complete the Task. Certain Task operations require a user to be the claimant.

Any user in the Assignee List can claim a task, thereby becoming the Claimant. The *State* of the Task is set to **Claimed** when a user claims the Task.

## Integration Administrators

WebLogic Integration server users with administrative privileges who can perform any operation on a Task, including the creation of new Tasks. To learn about the default roles and groups in WebLogic Integration, see User Management in *Managing WebLogic Integration Solutions*.

## Task Creators

Task Creators are users who, like the Integration Administrators, have permissions to create new Tasks.

WebLogic Integration provides a default group that defines which users can create new tasks. By default, the *anonymous* user is a member of this group in a new domain. To learn how you can enforce strict restraints on who can create new tasks, see Worklist Administration in *Managing WebLogic Integration Solutions*.

# Tasks and User Permissions

Only Integration administrators and users in the **TaskCreationRole** can create Tasks and simultaneously set new Task properties. The Worklist system verifies that the user attempting to create tasks and invoke operations on Tasks has the required permissions to do so. This section includes:



## Who Has Permission to Create Tasks?

Only Integration administrators and users in the **TaskCreationRole** can create Tasks and simultaneously set new Task properties.

**Default** WebLogic Integration **Users**—Any domain that supports WebLogic Integration includes a set of default WebLogic Integration roles and groups. Default security policies define the roles authorized to access specific WebLogic Integration resources. You must be logged in as a member of one of the following groups to make changes to task states: IntegrationAdministrators, IntegrationUsers, or IntegrationOperators. To learn about the default roles and groups in WebLogic Integration, see User Management in *Managing WebLogic Integration Solutions*.

You can configure the **TaskCreationRole** using the WebLogic Integration Administration Console. To do so:

1. Open the WebLogic Integration Administration Console.
2. From the home page, select the **System Configuration** module.
3. From the left panel, select **Worklist**.
4. On the **View Worklist Configuration** page, click **Configure**.
5. From the **Task Creation Role** drop-down list, select the role.
6. Click **Submit** to update the setting and return to the **View Worklist Configuration** page.

By default, the **TaskCreationRole** role contains the WebLogic Server Anonymous role. Thus, anonymous users have the permissions to create tasks. You can change this specification if you want more stringent control over the creation of tasks.

## Who Has Permission to Modify Task Data Values?

Whether a user can change Task properties after the Task is created depends on the following parameters:

- The *state* of the Task. You can not change the value of a Task property if the Task is in an aborted, suspended, or completed state.
- The role and group identification of the user executing the property change. Integration Administrators are never denied permissions to change Task properties in any circumstances where the operation is allowed; they have no restrictions.
- The status of the user executing the property change with respect to the Task:
  - For a Task that is in the assigned state, the user who modifies the Task must be an **assignee**, the **Task owner**, or an **Integration Administrator**.
  - For a Task that is in the claimed or started state, the user who modifies the Task must be the **claimant**, the **Task owner**, or an **Integration Administrator**.

## Who Has Permission to Invoke Task Operations?

Any operation on a Task requires the Worklist system to verify that the current principal executing the operation has the permissions to perform that operation.

Whenever an operation is invoked on a task instance, the Worklist system checks if the currently executing principal has the permission to do so. The decision to grant a permission is a function of the role of the current principal with respect to the task, the state of the task, the operation being invoked, and possibly some of the task data values (for example, the value of `canBeReassigned`).

A user can take on one of several roles when interacting with a task instance. These roles are described in "Task Users and Groups" on page 3-8. The following table presents the permissions that different users have to perform the operations on Tasks that result in a change to the state of a task. Each row presents the possibilities for a given starting Task state.

**Table 3-3 User Roles and Task Operations**

| Starting State | Ending State | Relevant Operations | Permitted Users |
|---|---|---|---|
| Assigned<br>Claimed<br>Completed<br>Started | Aborted | abortTask | Assignee list users and groups (if can-be-aborted property true)<br>claimant (if can-be-aborted property true)<br>Task Owner<br>Integration Administrator |
| Assigned, Claimed Started | Aborted | abortTask | Assignees List, if task is ASSIGNED and canBeAborted is TRUE. Claimant if canBeAborted is true. Task Owner and Integration Administrator |
| Assigned, Claimed, Started | Suspended | suspendTask | Task Owner and Integration Administrator |
| Suspended | Whatever the Task State was when suspended | resumeTask | Task Owner and Integration Administrator |
| Assigned | Assigned | assignToUsersAndGroups | Assignees List if canBeReassigned is TRUE. Task Owner and Integration Administrator |
| Completed or Aborted | Assigned | assignToUsersAndGroups | Task Owner and Integration Administrator |
| Assigned | Claimed | claimTask | Assignees List. Task Owner and Integration Administrator |
| Claimed | Assigned | returnTask | Claimant if canBeReturned is true. Task Owner and Integration Administrator |
| Assigned | Claimed | assignToUserInGroup, assignToUser | Assignees List if canBeReassigned is TRUE. Task Owner and Integration Administrator |

| Starting State | Ending State | Relevant Operations | Permitted Users |
|---|---|---|---|
| Completed or Aborted | Claimed | assignToUserInGroup, assignToUser | Claimant if canBeReassigned is TRUE. Task Owner and Integration Administrator |
| Claimed | Started | startTask | Claimant.Task Owner and Integration Administrator |
| Started | Claimed | stopTask | Claimant.Task Owner and Integration Administrator |
| Started | Completed | completeTask | Claimant. Task Owner and Integration Administrator |
| Any | Not Existing | deleteTask | Task Owner and Integration Administrator |

# Task Data Values

There are various data values associated with Task Instances. They provide a mechanism for describing the work that needs to be done to complete a task, describing work that is already done, who should do the work, by when, the results of work, and so on.

Some of these values are specified once at the time a new Task is created; some values can modified throughout the life cycle of the instance. Each data value has rules that specify the users that have permissions to modify the value.

The following table describes the Task Instance data values:

**Table 3-4 Task Instance Data Values**

| Name | Purpose | How to Set | Type | Unset Value | Default | Notes | Only Admin. Can Modify |
|---|---|---|---|---|---|---|---|
| TaskId | Identifies Task instance<br>Unique Value | System sets at creation time, user cannot set | String | always non-null | NA | | NA |
| Name | Any String | Set by user at creation time only. | String | always non-null | NA | | NA |
| Description | description of the task | Set by user at creation time only. | String | can be null | Null | | NA |
| ParentProcessURI | The business process type that created this task, if any. | System sets at creation time, user cannot set | String | can be null | NA | | NA |
| ParentProcessId | The instance id of the process that created this task, if any. | System sets at creation time, user cannot set | String | can be null | NA | | NA |
| Assignees | The assignees list for this task, those users that can claim it. | Set by user using an assignment operation. | String Array | Can be empty list, but non-null | empty | List of user and group names. All should be members of Integration Users group | NA |

| Name | Purpose | How to Set | Type | Unset Value | Default | Notes | Only Admin. Can Modify |
|---|---|---|---|---|---|---|---|
| Claimant | Tracks who claimed the task This user completes the task. | Set by user using a claim(…) operation. | String | can be null | Null | Must be a user name, a user that is on the assignees list, or a member of a group on the assignees list. | NA |
| Comment | Any | Set by user | String | can be null | Null | | False |
| Priority | Can be any integer > 0. System doesn't do anything with this, completely up to the application to interpret or ignore. | Set by user | Integer | always non-null | 1 | | True |
| CreationDate | When was the task created. | Set by Worklist system | java.util. Date | always non-null | NA | | NA |
| CanBeReassigned | Can a user on the assignees list assign the task again, effectively reassigning it. Otherwise requires administrative privileges. | Set by user | boolean | always non-null | True | | True |

| Name | Purpose | How to Set | Type | Unset Value | Default | Notes | Only Admin. Can Modify |
|---|---|---|---|---|---|---|---|
| CanBeReturned | Can the claimant return the task after claiming it, otherwise requires administrative privileges. | Set by user | boolean | always non-null | True | | True |
| CanBeAborted | Can the claimant abort the task. otherwise requires administrative privileges. | Set by user | boolean | always non-null | True | | True |
| Request | Can describe what work should be done, how to do it, or info needed to complete the work | Set by user | byte array can hold XML, String, RawData, and so on. | can be null | Null | | False |
| Response | Can describe what was done, or how the work was done, what problems were encountered, and so on. | Set by user | byte array can hold XML, String, RawData, and so on. | can be null | Null | | False |

| Name | Purpose | How to Set | Type | Unset Value | Default | Notes | Only Admin. Can Modify |
|---|---|---|---|---|---|---|---|
| State | Describes where in the Task life cycle the task is currently. ASSIGNED, CLAIMED, STARTED, COMPLETED, ABORTED, SUSPENDED. | System maintains this, operations cause it to change. | String | always non-null | NA | | NA |
| Request Type and Response Type | Can be used to describe the format of the Request or Response value of this task instance. | Set by user when Request or Response are set. | String | can be null | NA | System does nothing with this value; application can interpret and use. | False |
| Owner | Can be user or group name | Set by user | String | can be null | principal executing when task is created | User or Group must be member of Integration Users group | True |
| ClaimDueDate | By what time should some user have claimed the task? | Set by user | java.util.Date | can be null | Null | Null means no due date | True |

| Name | Purpose | How to Set | Type | Unset Value | Default | Notes | Only Admin. Can Modify |
|---|---|---|---|---|---|---|---|
| CompletionDueDate | By what time should the claimant have completed the task? | Set by user | java.util. Date | can be null | Null | Null means no due date. | True |
| Arbitrary, User Defined Properties | Any | Set by user | String | can be null | NA | | True |

## Request and Response Documents

Documents can be associated with task instances: the documents describe what work needs to be completed for the task, the status of what is being done, or the results of what was attempted or completed for the task. These documents populate the **request** and **response** data values, as described in the preceding table.

A Task Request is used to specify what work needs to be done to complete a task, or how that work is to be done. For example, a Task Request can contain a Purchase Order document that needs to be approved by a user. This value of a Task Request can be read by the person who performs the work and completes the task. In addition, assignees can view this information to decide whether or not to claim the task.

The Task Response is used to specify the work that took place after a user worked on the task, or the resulting data generated as the result of the work done to complete the task, or both. Callbacks can return the Response value to business processes that are waiting for a particular task state.

For example, the Task Response can capture the agreement made between a collections agent and a delinquent customer after the two complete a telephone conversation. The process that created the Task to call that customer can use those results to decide what to do next.

## Format and Type of Request and Response Documents

The Request and Response documents are stored as byte arrays in the Worklist system. Therefore they can hold any type of object. Methods and callbacks on the controls can set or get these values as XmlObjects, Strings, Raw Data Types, or XML Bean types.

For example, you can create a Task that matches purchase orders with receipts, and include an electronic version of a purchase order as request data. When the Task completes, it can include a matching receipt with the purchase order, along with a document that explains any differences, as response data.

The RequestType and ResponseType task properties can be used to specify the data assigned to the Request or Response type. The values of RequestType and ResponseType are provided for the interpretation of the application; they are not used by the Worklist system.

For example, the RequestType can be set to `xml`, `string`, `word document`, or `com.xyz.PurchaseOrder`. A custom Worklist UI can use these values to determine how to display a request or response value to the user.

# Task Operations

Operations are used to create new tasks, alter task states or properties, delete tasks, or read information about an existing task. Some operations allow combinations of these actions in a single step. Examples of Task operations include:

- Operations to Create Tasks
- Operations to Modify Task Properties
- Operations to Get Task Properties
- Operations to Modify Task State

**Operations to Create Tasks**

When a new Task Instance is created, the Worklist system assigns a unique ID (a taskID) to that instance. The Task *State* can be defined at creation time as Assigned or Claimed, depending on the operation. Some Task *Properties* are specified at the time an instance of a Task is created and can not be changed after the instance is created.

**Operations to Modify Task Properties**

Some Task Properties can be specified and modified after an instance of a Task is created.

**Operations to Get Task Properties**

You can use *Get Task* operations to access the properties of any Task Instance at any point during its lifecycle.

**Operations to Modify Task State**

Task Instances can transition between states based on the operations defined for them. Different operations to modify a Task's State are valid depending on the State in which the task is to start.

# Archiving and Purging Task Information

WebLogic Integration supports archiving tracking data for business process instance history, trading partner message history, and task instance history.

As tasks go through their lifecycle in enterprise processes, their properties are modified, their states change, their due dates expire, and so on. Worklist task instances generate events that can be logged in Worklist history tables in the runtime repository. The following types of events can be tracked:

- *Changes in task state and associated values*
  The type of transition and associated values. For example, a task is reassigned or claimed. In this case, the change in state and identity of the new assignee or claimant can be tracked.
- *Expiration of task claim or complete due date*
  The task is unclaimed or incomplete on the due date for claiming or completing.
- *Changes in task owner or assignees*
  The type of change and new values can be tracked.
- *Task requests and task responses*
  The request and response XML.

The Task Worker controls provide the following methods for archiving and deleting Tasks:

- `archiveTask`—for placing Task instance information into Task history tables for any Task in the completed or aborted state (see "Task History Tables" on page 3-22).
- `deleteTask`—for permanently removing Tasks from the WebLogic Integration server. The `deleteTask` method can be used for a Task that is in any state, including the suspended state.

  **Warning:** Use the `deleteTask` method with caution—there is no mechanism for rollback or retrieval of deleted Tasks.
- `purgeTask`—for deleting all archived Tasks from the runtime.

The `archiveTask` and `purgeTask` methods function according to the settings of the process archiver in the WebLogic Integration Administration Console:

- If the process archiver is off, the `archiveTasks` method does not function and the `purgeTasks` method removes all completed and aborted archived Tasks that have existed longer than the `purgeDelay` setting.

- If the process archive is on, the `purgeTasks` method removes all the aborted and completed Tasks that are in existence longer than the time specified by the `purgeDelay` setting, and were archived previously.

  The process archiver provides the following *tracking settings*: Basic, Full, None. The `achiveTask` method works with the process archiver according to the task tracking settings:

  - **Basic**—the process archiver archives all the Tasks in the completed or aborted states as well as all the Task operations and state transitions.
  - **Full**—the process archiver archives the tasks, operations and state transitions according to the **Basic** setting, and also archives all the requests and responses.
  - **None**—the process archiver does nothing if Task tracking is set to none.

To learn more about configuring your application for archiving and purging Task data, see System Configuration in *Managing WebLogic Integration Solutions*, which is available at the following URL:

`http://edocs.bea.com/wli/docs81/manage/index.html`

To learn about the Task states and the operations available for archival, see "Task Operations" on page 3-19.

To optimize performance, the amount of tracking data stored in the runtime database should be kept to a minimum. To this end, the archive and purge process can be configured to run at regular intervals set by an administrator. In addition to configuring the schedule, the administrator can enable or disable the archiver:

- When the archiver is enabled, the process copies the data to an offline database, then purges it from the runtime database.
- When the archiver is disabled, the process purges the data from the run-time database without copying it.

Archived information can be used for generating reports and compiling statistics about task processing in your WebLogic Integration application. To learn more about configuring your application for archiving and purging Task data, see System Configuration in *Managing WebLogic Integration Solutions*, which is available at the following URL:

`http://edocs.bea.com/wli/docs81/manage/index.html`

# Task History Tables

As described in the preceding section, worklist task instances generate events that can be logged in worklist history tables in the runtime repository. By default, the `archiveTask` method stores Task instance information in the following history tables:

- wli_task_archiving
- wli_task_data_archiving

The following tables show the task information that can be stored in the history tables.

**Table 3-5 Data in the Task History Tables**

| Name | Description |
|---|---|
| task_id | The unique Task ID. |
| action_type | An integer that represents the action recorded. For more information, see Table 3-6. |
| state_type | An integer that represents the state of the Task. For more information, see Table 3-6. |
| action_time | The time of the action. |
| action_user | The user for which the action happens. |
| details | Any other relevant information. |

The following table presents the action types and state types (see `action_type` and `state_type` in the preceding table) that can be archived, along with the integer that represents the action type or state type in the history tables.

**Table 3-6 Integers That Represent Action and State Types in the History Tables**

| Action Type | Integer | State Type [1] | Integer |
|---|---|---|---|
| create | 0 | assigned | 0 |
| assign | 1 | claimed | 1 |
| claim | 2 | started | 2 |

| Action Type | Integer | State Type [1] | Integer |
|---|---|---|---|
| suspend | 3 | suspended | 3 |
| resume | 4 | completed | 4 |
| complete | 5 | aborted | 5 |
| abort | 6 | | |
| return | 7 | | |
| start | 8 | | |
| stop | 9 | | |
| updateComment | 10 | | |
| updatePriority | 11 | | |
| updateExpirationDate | 12 | | |
| updateClaimDate | 13 | | |
| updateOwner | 14 | | |
| updateCanBeReassigned | 15 | | |
| updateCanBeReturned | 16 | | |
| updateCanBeAborted | 17 | | |
| updateRequest | 18 | | |
| updateResponse | 19 | | |
| addListener | 20 | | |
| removeListener | 21 | | |
| claimExpire | 22 | | |
| expire | 23 | | |

1. State types can be used in queries using the `@jc:selector` annotation tag or `TaskSelector` objects. To learn more about queries, see "Querying Tasks Using the Task Worker Control" on page 5-14.

# Task Queries

Task Queries allow a business process or UI to find tasks in the Worklist system that meet user defined criteria. This is analogous to SQL queries executed on Database Tables. The application defines the criteria, executes the query, and is returned results for each task that matches the criteria.

Business processes can use the task query mechanism to find tasks relevant to the business process, and then perform work on the tasks that are returned. For example, a manufacturing application can find all tasks related to a cancelled order and abort them.

You can create custom UI Pages that use the query mechanism to find tasks relevant to the user that is using the page, and display information about those tasks. For example, a bug tracking UI can allow users to query for tasks that are assigned to them, have a certain priority or higher, and are due within a certain number of days.

This section includes the following topics:



## To Specify the Criteria for a Query

The criteria you specify in when you define a query determine the tasks that are returned by the query. You can set the following criteria when you specify a query

**Table 3-7 To Specify Criteria for a Query**

| Query Criteria | Description |
|---|---|
| Task Ids | Returns only those tasks for which the instance id is matched. |
| Task Name | Returns only those tasks for which the name matches the value passed in. You can optionally specify that the value matches a pattern. For example: `OrderNumber%`[1]. |
| Comment | Returns only those tasks for which the comment matches the value passed in. You can optionally specify that the value matches a patterns. |

| Query Criteria | Description |
|---|---|
| Description | Returns only those tasks for which the description matches the value passed in. You can optionally specify that the value matches a pattern. |
| Owners | Returns only those tasks for which the task owner is on the specified list. You can specify a list of user and group names. |
| Claimant | Returns only those tasks for which the claimant is on the specified list. You can specify a list of user and group names. |
| Assignee | Returns only those tasks for which the associated assignees list contains the specified assignee. |
| State | Returns only those tasks for which the state is on the list of specified values. |
| ParentProcessId | Returns only those tasks that were created by one of the processes whose id is in the specified list. |
| ParentProcessURI | Returns only those tasks that were created by a processes whose URI matches the value passed in. You can optionally specify that the value matches a pattern. Example: `%PurchaseOrderProcess%` |
| Completion Due Date | You can specify whether the date should be before or after the specified date. |
| ClaimDueDate | Returns only those tasks whose date is before or after the specified java.util.Date. |
| Creation Date | Returns only those tasks that were created before or after the specified `java.util.Date`. You can specify whether the date should be before or after the specified date. |
| canBeReassigned | Returns only those tasks whose data value matches the boolean specified. |
| canBeAborted | |
| canBeReturned | |
| Priority | You can specify a maximum and minimum range. Returns only those tasks for which the priority falls within the specified range. |
| User Defined Property Name | Returns only those tasks with a named property defined—the name is specified in the query. |
| User Defined Property Value | Returns only those tasks for which a named property has a value equal to the String specified. |

1. See the Note About String Patterns.

### Note About String Patterns

Some query criteria can be patterns that match strings with wildcards. These strings can contain the following wildcards:

- % characters to match any sequence of characters
- _ characters to match any single character.

For example:

A query for task names like **%Process_** returns **PurchaseOrderProcess1**, but not **PurchaseOrderProcess23455**.

If you want to apply patterns checking, the query must specify that explicitly. Special characters can be escaped using a back slash: **\%**, or **_**.

## To Specify How the Results Are Sorted

A query can return results that are sorted according to the specified criteria. To define the sort order, you set an integer value for each sort criteria.

For example, if you specify the sort value for name to be 1, and the sort value for comment to be 2, the results of the query are sorted first by name, then by comment.

The values are returned sorted first by the lowest number, then the second lowest, and so on. By default, all the criteria are set to the same value. In this way, they are all weighted equally in the sort. Specifically, the default sort value is set as `java.lang.short.MAX_VALUE`.

## To Execute a Query

The Task Worker control can be used to execute a query and return results. The API also provides operations on the `com.bea.wli.worklist.api.WorklistManager` interface to execute queries.

## To Limit the Results Set

You can specify a *maximum results* value for a query. It limits the maximum number of results returned by the query.

Results can be returned as either an array of TaskInfo objects or an array of Task Ids. TaskInfo objects contain a summary of the state and data values of a task instance.

# The Relationship Between Processes and Tasks

For the case in which a business process creates a task, the Worklist system tracks not only the task, but also the business process that created it. For example, the Worklist system tracks business processes that are blocking, waiting for a callback from a given task instance.

You can view and manage this tracking information using the WebLogic Integration Administration Console. For a given task instance, the console reports the processes that are currently blocking on a given task instance. For a given process, the console reports the tasks that are blocking a given business process.

The Worklist Task system in WebLogic Integration keeps track of the following relationships between task instances and process instances:

- Which business processes are blocking on a given Task
- Which business processes, if any, created each Task instance

In your queries, you can specify the following IDs and URIs to access information about task instances for which the specified processes are listening, or to access information about tasks that were created by the specified processes:

- `listeningProcessIds`—identifies the process ID of a listening process such as a URI
- `listeningProcessUri`—identifies the type of the URI invoking a control
- `parentProcessIds`—identifies the process ID of the parent process that invokes a control
- `parentProcessUri`—identifies the URI of any parent processes that invoke a control

To learn how to extend to a Task Worker control to query WebLogic Integration Tasks instance properties, see "Querying Tasks Using the Task Worker Control" on page 5-14.

CHAPTER 4

# Using the Worklist Control Methods

After you create an instance of a Task or Task Worker control, you can invoke its methods from within your business processes to perform operations on Task Instances. Your business processes can also wait to receive callbacks from task instances. Note that the Task and Task Worker controls can be extended to add customized methods and additional callbacks.

This section includes the following topics:

# Task Control Active Task Model

Each instance of a Task Control operates on a single task instance only. Each task has a unique ID—the *Active Task ID* on a control uses this ID to identify the task instance on which a task control operates. All operations on a task control are performed on the active task.

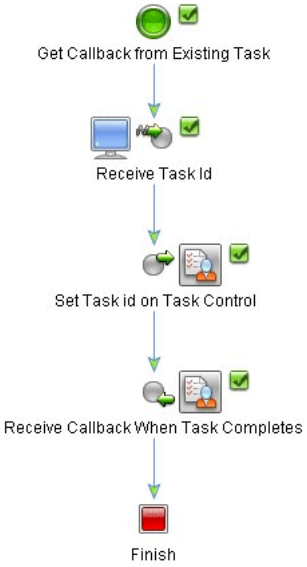
The Active Task Id is set either by creating a new task, or by invoking the `setActiveTaskId` method. When a new task is created, the Active Task Id is automatically set to the ID of the newly created task. In this way, subsequent operations are performed on the new task.

A consequence of the active task model is that a Task control instance can create a single task only. However, you can use a Task control factory to create new instances of Task controls dynamically. In other words, a Task control factory can be used to create a new Control instance every time a new Task is required. To learn more about control factories, see "Using Task Control Factories" on page 5-18.

Because it is possible to use the Task Id to define on which task a control operates or receives callbacks, multiple business processes can incorporate controls that all operate on the same task. New processes—that is, processes that are instantiated after a task is already in existence can interact with that task using the `setActiveTaskId` method.

Note that the Task Worker control does not use the active task model. Task ids are passed explicitly to methods on the Task Worker control to specify which tasks need to be updated.

For example, the business process shown in the following figure is designed to receive a Task Id from another business process that created a new task. The business process sets the Active Task Id for a Task control, and then waits for the `onCompletion` callback, which indicates that the task is completed.




# Creating New Tasks With a Task Control

A Task Control can be used to create a new Task with the following operations:

- `String createTaskByName(String name)`
- `String createTask(TaskCreationXMLDocument xml)`

Both operations create a new Task, return the instance id of that new task, and set the task control's Active Task Id to the id of the new task.

The `createTaskByName(String name)` method sets the task name to the value specified in the input parameter; other data values are set to their default values or values specified in the Property Editor.

The `createTask(TaskCreationXMLDocument xml)` first creates a new task and names it according to the name specified in the XML document. Then it alters the task in some way, using the elements in the XML document to do so.

# Assigning and Claiming Tasks

Worklist controls assign Tasks according the following methods.

`assignTaskToUser` methods and the `ToUser` property
: These methods and properties place Tasks in a claimed state and set a user as the claimant.

`assignTaskToUserInGroup` methods and the `ToUserInGroup` property
: These methods and properties accept the name of a group. The method and property use a load balancing mechanism to select a single user from that group to be the claimant, and then place the Task in a claimed state.

  The load balancing algorithm selects the user with the fewest claimed Tasks that are not in completed, aborted, or suspended states. If more than one user is identified (that is if two or more users in the group have the same number of claimed tasks, then the algorithm chooses one user randomly among them.

`assignTaskToUsersAndGroup` methods and the `ToUsersAndGroups` property
: These methods and properties accept values for the assignees list, and place a Task in an assigned state. Any user on the assignees list can claim the Task.

Although you can assign multiple users and groups to a Task, only one user on the assignee list can place a Task in a claimed state. If the assignee list contains only one user, the Task goes directly into a claimed state, and that user becomes the claimant.

If you create a Task with no users or groups on the assignees list, the Task behaves as if it were unassigned.

## Reassigning Tasks and Returning Them to Other States

Worklist controls allow a Task instance to be returned to the assigned state from the claimed state. The ability to assign a previously claimed Task depends on the privileges of the user initiating the assignment. To learn about the permissions required for changing task states, see "Permissions for Modifying Task Properties" on page 4-19.

You can reassign Tasks, placing them back into the assigned state while changing the assignee list, effectively giving them to different users. You can also return Tasks, placing them back into the assigned state with the same assignee list. You can return or reassign Tasks for work that repeats on a continuing basis.

The following methods, among others, allow a Task to be put back into an assigned state:

- `assignTaskToUsersAndGroups`

- `assignTaskToUserInGroup` (through load balancing)
- `returnTask`
- `resumeTask`

To learn about Task states and operations, see "Task Operations" on page 3-19. To learn more about the Worklist API, see the `com.bea.wli.worklist.api` package in the BEA WebLogic Integration Javadoc.

# Setting Task Data Values

You can set the data values for tasks depending on the permissions you are granted in the system. To learn about the Data values for task instances, the permissions you need to alter the task data, and the valid values for each type of data, see "Task Data Values" on page 3-13.

The Task control supports an operation that takes a `TaskUpdateXML` document. You can use this operation to set multiple data values in a single step. To learn how, see "Using XML With the Task Control" on page 4-9.

The following list describes the operations on the controls that can be used to set individual data values. describes the permissions required to perform these operations.

**Table 4-1 Setting Data Values With Operations**

| Operation | Description |
|---|---|
| `setClaimDueBusinessDate(String duration, String calendarID)` | Sets the claim due date using a business time duration and a calendar name.<br>The specified calendar is used to convert the business time duration to an absolute date. Conversions to absolute dates using the calendar are done relative to the current time.<br>To learn more about business dates and Tasks, see "Task Due Dates" on page 3-2. |
| `setClaimDueBusinessDateSystemCalendar (String duration)` | Sets the claim due date using a business time duration.<br>The system calendar is used to convert the business time duration to an absolute date.<br>To learn more about business dates and Tasks, see "Task Due Dates" on page 3-2. |

| Operation | Description |
|---|---|
| `setClaimDueDate(Date date)` | Sets the claim due date to an absolute date. You can specify `null` to unset the due date.<br>To learn more about dates and Tasks, see "Task Due Dates" on page 3-2. |
| `setCompletionDueBusinessDate(String duration, String calendarID)` | Sets the completion due date using a business time duration and a calendar name.<br>That calendar is used to convert the business time duration to an absolute date. Conversions to absolute dates using the calendar are done relative to the current time. |
| `setCompletionDueBusinessDateSystemCalendar(String duration)` | Sets the completion due date using a business time duration. The system calendar is used to convert the business time duration to an absolute date. |
| `setCompletionDueDate(Date date)` | Sets the completion due date to an absolute date. You can specify `null` to unset the due date. |
| `setComment(String comment)` | Sets the task comment. You can specify `null` to unset the comment. |
| `setOwner(String owner)` | The specified String sets the Task Owner as the user or group name. You can specify `null` to unset the owner. In other words, no owner is specified for the task. |
| `public void setPermissions(Boolean aborted, Boolean returned, Boolean reassigned)` | Sets the boolean values that pertain to permissions for a given task. |
| `setPriority(Integer priority)` | Sets the task priority. The value you specify for `priority` must be a positive integer. |
| `setRequest(XmlObject xml)` and `setResponse(XmlObject xml)` | Sets the request and response values to contain an XML document. You can specify `null` to unset the value. |
| `setProperty(String name, String value)` | Sets a user defined property with the given name to the given value. This method creates the specified property if it doesn't exist.<br>You cannot specify `null` for either the name or the value parameter in this case. |

# Altering State With a Task Control

Some methods on the Task control cause the state of a task instance to transition to a new state. Some of these methods also set related data values for the task instance in the process. The following list describes the operations that affect task state.

**Table 4-2 See updateStateOperations.xls.**

| Methods | Controls | Description |
|---|---|---|
| `abortTask` | Task<br>Task Worker | Invokes the abort operation on a task. To learn about Task states, see "Task States" on page 3-4. |
| `assignTaskToUser` | Task<br>Task Worker | Assigns tasks to the user whose name you provide as an argument to the method.<br>The user specified must belong to the WebLogic Integration Users Group.<br>The assignees list is set accordingly for the task instance.<br>The task is then automatically claimed for the specified user.<br>To learn about users and groups for Worklist controls, see "Task Users and Groups" on page 3-8. |
| `assignTaskToUserInGroup` | Task<br>Task Worker | Assigns the task to a user in the group whose name you provide as an argument to the method.<br>The Worklist system uses a load balancing algorithm to choose the least busy user in the group. |
| `assignTaskToUsersAndGroups` | Task<br>Task Worker | Assigns the task, and sets the assignees list to contain the users and groups you provide as an argument to the method.<br>The users and groups specified must belong to the WebLogic Integration Users Group. |
| `resumeTask` | Task<br>Task Worker | Invokes the resume operation on the task. To learn about Task states, see "Task States" on page 3-4. |

| Methods | Controls | Description |
|---|---|---|
| `suspendTask` | Task<br>Task Worker | Invokes the suspend operation on the task. To learn about Task states, see "Task States" on page 3-4. |
| `updateTask` | Task | Alters the state through assignment. To learn more about Task properties and XML, see "Using XML With the Task Control" on page 4-9. |
| `archiveTasks` | Task Worker | Causes completed and aborted tasks to be archived. To learn more about Task archival, see "Additional Resources" on page 5-20. |
| `purgeTasks` | Task Worker | This method purges all archived tasks from the archival tables. For more information about archiving, see "Additional Resources" on page 5-20. |
| `claimTask` | Task Worker | Causes the claim operation to be called. The currently executing principal claims the task. To learn about Task states, see "Task States" on page 3-4. |
| `claimTaskOnBehalfOf` | Task Worker | Causes the claim operation to be called. Claims the task on behalf of the user whose name is specified as an argument to the method. This is an administrative function. |
| `completeTask` | Task Worker | Invokes the complete operation on a task. To learn about Task states, see "Task States" on page 3-4. |
| `deleteTask` | Task Worker | Removes the task instance completely and permanently at run time. You can also use the WebLogic Integration to remove task instances.<br>This is an administrative function To learn about Task states, see "Task States" on page 3-4. |

| Methods | Controls | Description |
|---|---|---|
| `returnTask` | Task Worker | Invokes the return operation on a task. This method places Tasks back into the assigned state and makes no changes to the assignee list. To learn about Task states, see "Task States" on page 3-4. |
| `startTask` | Task Worker | Invokes the start operation on a task. To learn about Task states, see "Task States" on page 3-4. |
| `stopTask` | Task Worker | Invokes the stop operation on a task. To learn more about Task states, see "Task States" on page 3-4. |

# Using Controls to Get Task Status

The Task and Task Worker controls provide operations to access data values associated with a task instance. You can use these operations to access individual values, to receive a `TaskInfoXMLDocument`, and to return a `com.bea.wli.worklist.api.TaskInfo` object:

- A `TaskInfoXMLDocument` contains a summary of the task and data values in a single XML document. To learn about the `TaskInfoXMLDocument`, see "Using XML With the Task Control" on page 4-9.
- A `com.bea.wli.worklist.api.TaskInfo` object contains a summary of the task and data values in a Java object.

# Using XML With the Task Control

Several operations on the Worklist controls offer an XML interface for ease of use. WebLogic Integration provides a system XML Schema, `Worklist.xsd`, which defines the structure of XML documents. To use the `Worklist.xsd` schema, you must import it into a Schemas project in your application. To learn how, see "To Import the Worklist Schema into Your Application" on page 4-12.

These operations are concise and convenient. They allow you to configure and perform multiple operations on a Task instance in a single step; they allow you to access the summary for a task instance in a single document. The real power of these operations, however, is through their use with the XML mapper.

For example, if a business process contains several variables, all of which contain information relevant to the creation of a new task, the XML mapper can be used to extract the values from each of these variables and construct a single XML document that specifies aspects of a new task. You can review the XML document in the mapper to get an overview of the data values that will be set for a given task.

Similarly, you can use the mapper to extract several values from a Task Status XML document.to set several values for a business process at once.

# Creating New Tasks

A TaskCreationXML document can be used to create a new Task instance and configure that new instance in a single step. Operations on the Task Control take the document as an argument and use it to create a Task. Each element in the TaskCreationXML causes the new Task to be updated in a different way. This section describes the following methods:

- `public String createTask(TaskCreationXMLDocument doc)`
- `public TaskInfoXMLDocument getTaskInfoXMLDocument()`
- `public void updateTask(TaskUpdateXMLDocument doc)`

## public String createTask(TaskCreationXMLDocument doc)

The worklist system calls the following method using the value of the name element in the `TaskCreationXMLDocument`.

```
public String createTask(TaskCreationXMLDocument doc)
```

Then the method parses the document, element by element, invoking a state related operation or a data setting operation on the task instance for each.

For example, the following is an example of an XML document you can use to create a new task named **My Task**, assign it to a user named **Bill Smith**, set the priority to **5**, specify a task comment, specify the completion due date for **3 business days** and specify that the due date is calculated based on the **CustomerSupport** group's business calendar:

```
<TaskCreationXML xmlns="http://www.bea.com/wli/worklist/xml">
        <name>My Task</name>
        <comment>This work is important</comment>
        <priority>5</priority>
        <completionDueBusinessDate>
                <day>3</day>
                <calendar>
                        <userOrGroup>CustomerSupport</userOrGroup>
                </calendar>
```

```
        </completionDueBusinessDate>
        <assignee>
               <user>BillSmith</user>
               <algorithm>ToUser</algorithm>
        </assignee>
</TaskCreationXML>
```

In the preceding listing, note the following elements:

- **completionDueBusinessDate**—Allows optional specification of a Business Calendar, either specifying the Calendar's name, or the name of a user or group whose calendar should be used. This calendar is used to convert the business time duration to an absolute time to set the due date as a java.util.Date.
- **assignee**—Used to assign the task directly to a user, a user in a group, or to set the Assignees list to contain a list of users and/or groups. The example XML in the preceding listing shows how to specify the assignee as a single user (assignToUser). The following XML is also valid to assign a task to a user in a group, and to specify a list of users and groups for the Assignees list:
  - To assignToUserInGroup

    ```
    <assignee>
       <group>CollectionsGroup</ group >
       <algorithm>ToUsnGroup</algorithm>
    </assignee>
    ```

  - To assignToUsersAndGroups

    ```
    <assignee>
       <user>UserA</user>
       <user>UserB</user>
       <group>GroupA</ group >
       <group>GroupB</ group >
       <algorithm>ToUsersAndGroups</algorithm>
    </assignee>
    ```

## Setting the request Property for a Task Instance

An XML element can be used to set the **request** property for the Task Instance. The message value of this element can be any XML appropriate for the task instance. The mime-type element is for informational purposes only and can be interpreted by the application. To learn more about the mime-type elements, see "Request and Response Documents" on page 3-18.

```
<request>
   <message>
      <line:line-item xmlns:line="http://www.bea.com/line-item">
```

```
            <line:name>Widget</line:name>
            <line:quantity>100</line:quantity>
      </line:line-item>
   </message>
   <mime-type>LineItem</mime-type>
</request>
```

### public TaskInfoXMLDocument getTaskInfoXMLDocument()

The following method can be used to get a `TaskInfoXMLDocument` on the Worklist Controls. It contains the task properties, state, and so on in a single document:

```
public TaskInfoXMLDocument getTaskInfoXMLDocument()
```

### public void updateTask(TaskUpdateXMLDocument doc)

You can use a `TaskUpdateXML` document as an argument to the following method to update an existing Task instance in various ways in a single step:

```
public void updateTask(TaskUpdateXMLDocument doc)
```

Each element in the `TaskUpdateXML` document causes the Task to be updated in a different way. The XML document is similar to the `TaskCreationXML` document described for public String createTask(TaskCreationXMLDocument doc).

## To Import the Worklist Schema into Your Application

1. In the **Application** tab, right-click on the top-level application folder. (If the **Application** tab is not visible in WebLogic Workshop, choose **View →Application** from the menu bar.)
2. From the drop-down menu, select **New →Project...**.

   The **New Project** dialog box is displayed.
3. In the right-most pane of the **New Project** dialog box, select **WLI System Schemas**.

   The Schemas project you create contains WebLogic Integration System XSD files, including `Worklist.xsd`.
4. In the **Project name** field, enter a name (for example: Schemas).

   **Note:** You can name your schemas project anything you want, except for the case in which you plan to use the project for application view channels and schemas. In that case, you must name it **Schemas**.

# The Task Control Properties Sheet

The Task Control Properties Sheet allows the user to set defaults that will be used for properties of new task instances created with that control.

When a new task is created with the control, any value in the properties sheet will be used, unless the creation operation passes a parameter in to explicitly override the value. If the value is not set by the method parameters and doesn't exist in the properties sheet, the Worklist system defaults will be used for those values.

For example, say the properties sheet specifies values for the task name, task owner, and priority. Say the user creates a new task with a TaskCreationXML document that specifies just the task description and priority.

The description is not specified in the properties sheet; therefore the value from the TaskCreationXML is used. The priority is specified in the XML and the properties sheet; therefore the properties sheet value is used. The task comment is not specified in the XML or the properties sheet; therefore the system default is used.

The following table shows the task control properties:

**Table 4-3 Task Control Properties**

| Property Section | Property | Purpose | Valid Values |
|---|---|---|---|
| Task | Name | Sets the task name | String |
| Task | Description | Sets the task description | String |
| Task | Comment | Sets the task comment | String |
| Task | Priority | Sets the task priority | Integer |
| Task | Owner | Sets the task owner | String |
| Assignee | Algorithm | Specifies how to assign new task | Strings: ToUser ToUserInGroup |
| Assignee | User | If algorithm is ToUser, specifies the user name | String |
| Assignee | Group | If algorithm is ToUserInGroup, specifies the group name | |

| Property Section | Property | Purpose | Valid Values |
|---|---|---|---|
| Advanced | can-be-reassigned | Sets the task property called canBeReassigned | True or False |
| Advanced | can-be-returned | Sets the task property called canBeReturned. | True or False |
| Advanced | can-be-aborted | Sets the task property called canBeAborted. | True or False |
| Advanced | claim-due-business-date | Sets the due date for the task to be claimed, using a business time duration. | String, format must be valid business time duration |
| Advanced | completion-due-business-date | Sets the due date for the task to be completed using a business time duration. | String, format must be valid business time duration |
| Advanced | completion-user-calendar | Sets the name of the user whose Business Calendar should be used to convert the completion-due-business-date to a java.util.Date. | String user name |
| Advanced | claim-user-calendar | Sets the name of the user whose Business Calendar should be used to convert the claim-due-business-date to a java.util.Date. | String user name |
| Advanced | completion-calendar | Sets the name of the Business Calendar that should be used to convert the completion-due-business-date to a java.util.Date. | String calendar name |
| Advanced | claim-calendar | Sets the name of the Business Calendar that should be used to convert the claim-due-business-date to a java.util.Date. | String calendar name |

# Using the Task Control Property Editor

Controls you create in your application are represented as JCX files in the **Application** pane in WebLogic Workshop. *Instances* of controls that you create in your business process are represented in the **Data Palette**. You can view and edit the properties of control instances and their parent types in the **Property Editor** when you work in the WebLogic Workshop graphical design environment.

## Use the Property Editor to View and Edit Properties for Control Instances

1. On the **Application** pane, click the JPD file you are designing. The business process is displayed in **Design View**.
2. In the **Data Palette**, double-click an instance of a Task control. Its properties are displayed in the **Property Editor**.

   **Note:** If the **Data Palette** or the **Property Editor** is not visible in WebLogic Workshop, click **View→Windows→Data Palette** or **View→Property Editor** from the menu bar

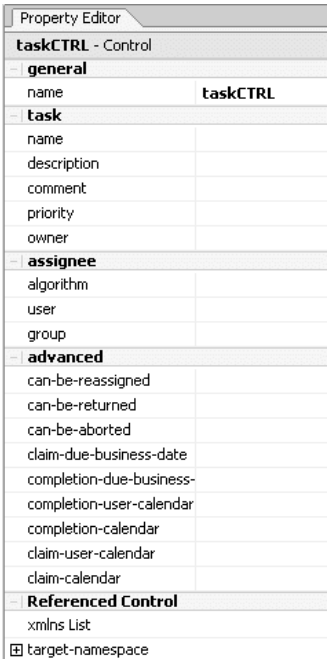
   Note that when you open the **Property Editor** for an instance of a control, the properties for that instance, are listed at the top of the **Property Editor** and the properties specified for the parent control (that is, the control on which the current instance is based) are listed at the bottom—in the **Referenced Control** section. The properties displayed in the **Referenced Control** section are read-only. You can edit the referenced control properties by opening the JCX file.

   To learn how to use the **Property Editor** to specify properties for control types versus control instances, see Setting Control Properties in the WebLogic Workshop online help, which is available at the following URL:

   `http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideControlsProperties.html`

For example, if you create an instance of a File control in your business process, and name it **taskCTRL**, the instance is displayed in the **Data Palette**. Click **taskCTRL** in the **Data Palette** to highlight it in the Data Palette and cause its properties to be displayed in the **Property Editor**.

The following figure shows the **Property Editor** for our example **taskCTRL** instance:




Default properties for Task control instances appear encoded in the Worklist control JCX file as attributes of the `@jc` (Java control) annotations. For detailed information on the `@jc` annotations and their attributes, see Worklist Control Annotations, which is available in the WebLogic Workshop online help.

# Using Callback Methods

Task controls provide callback methods. Other resources can use the callback interface to receive notification of events, such as changes in states or properties. The Task control includes the following callback methods.

**Table 4-4 Task Control Callback Methods**

| Method | Description |
|---|---|
| `onTaskAborted` | This is a callback method that another resource (for example, a business process) can implement to receive notification when a Task is in an aborted state. |
| `onTaskCompleted` | This is a callback method that another resource (for example, a business process) can implement to receive notification when a Task is in a completed state. |
| `onTaskOverdue` | This is a callback method that another resource (for example, a business process) can implement to receive notification when a Task completion due date is past. |

To learn more about Task states, see "Task States" on page 3-4.

You can also create custom callback methods. To learn more about building your own Worklist callback methods, see "Querying Tasks Using the Task Worker Control" on page 5-14.

# Permissions

When any operation is invoked on a Task, the Worklist system verifies that the current principal that is executing the operation, has the permission to do so. Permission is granted based on the role to which the user is assigned (see "Tasks and User Permissions" on page 3-10), the state of the task, the operation being invoked, and possibly data values associated with the task instance.

The following table describes which roles are allowed to perform operations that modify the state of a Task instance.

**Table 4-5 State Transitions for Tasks**

| Start States | End State | Relevant Operations | Permitted Users |
|---|---|---|---|
| ASSIGNED, CLAIMED, STARTED | ABORTED | `abortTask` | Assignees List, if task is ASSIGNED and `canBeAborted` is TRUE.<br>Claimant if `canBeAborted` is true. Task Owner and Integration Administrator |
| ASSIGNED, CLAIMED, STARTED | SUSPENDED | `suspendTask` | Task Owner and Integration Administrator |
| SUSPENDED | Whatever the Task State was when suspended | `resumeTask` | Task Owner and Integration Administrator |
| ASSIGNED | ASSIGNED | `assignToUsersAndGroups` | Assignees List if `canBeReassigned` is TRUE. Task Owner and Integration Administrator |
| COMPLETED or ABORTED | ASSIGNED | `assignToUsersAndGroups` | Task Owner and Integration Administrator |
| ASSIGNED | CLAIMED | `claimTask` | Assignees List. Task Owner and Integration Administrator |
| CLAIMED | ASSIGNED | `returnTask` | Claimant if `canBeReturned` is true. Task Owner and Integration Administrator |
| ASSIGNED | CLAIMED | `assignToUserInGroup`, `assignToUser` | Assignees List if `canBeReassigned` is TRUE. Task Owner and Integration Administrator |
| COMPLETED or ABORTED | CLAIMED | `assignToUserInGroup`, `assignToUser` | Task Owner and Integration Administrator |
| CLAIMED | STARTED | `startTask` | Claimant. Task Owner and Integration Administrator |

| Start States | End State | Relevant Operations | Permitted Users |
|---|---|---|---|
| STARTED | CLAIMED | `stopTask` | Claimant. Task Owner and Integration Administrator |
| STARTED | COMPLETED | `completeTask` | Claimant. Task Owner and Integration Administrator |
| ANY | DOES NOT EXIST | `deleteTask` | Task Owner and Integration Administrator |

## Permissions for Modifying Task Properties

Whether a user has permission to modify a Task property depends on factors including the state of the task, the current user executing, and the particular property to be modified.

Some properties can only be modified by the Task Owner or an Integration Administrator, no matter the state of the Task.

Properties cannot be modified on Tasks that are in any of the following states: suspended, completed, and aborted. If you need to modify the properties for a task that is any of these states, you must first transition it out of that state. To do so, you can resume the task, or reassign it and claim it.

Only the task owner, a member of the assignees list, or an Integration Administrator can modify Task properties for tasks that are in the assigned state.

Only the claimant, the task owner, or an Integration Administrator can modify Task properties for tasks that are in the claimed or started states.

To learn about Task states and moving tasks between states, see "Task States" on page 3-4 and "Task Operations" on page 3-19.

## Permissions for Reassigning Tasks and Returning Them to Other States

Task owners and Integration Administrators are always granted the permission to reassign and return Tasks. In addition, Worklist controls provide the following boolean properties to specify whether an assignee or claimant can reassign or return a Task:

- can-be-reassigned (valid values are **true** and **false**)

- can-be-returned (valid values are **true** and **false**)

To learn about the methods provided on the Worklist controls that allow a Task to be put back into an assigned state, see "Reassigning Tasks and Returning Them to Other States" on page 4-4.

# Modifying Task Data Values

The users and groups that are granted permissions to modify a task property depend on several factors: the state of the task, the current user executing the operation, and the property to be modified.

Some data values can only be modified by the Task Owner or an Integration Administrator, regardless of the state of the task. (To learn about the data values and the permissions required to modify them, see "Task Data Values" on page 3-13.)

Data values can not be modified for states in the SUSPENDED, COMPLETED, and ABORTED states. The task must be resumed or reassigned and claimed before you can modify data values on tasks in these states.

The current user must be on the assignees list, be the task owner, or be an Integration Administrator to modify Task Data values for tasks that are in the ASSIGNED state.

Only the claimant, task owner, and Integration Administrators can modify task data values for tasks that are in the CLAIMED and STARTED states.

# Creating Tasks

Only certain users can create tasks. For more information, see "Task Users and Groups" on page 3-8 and "Tasks and User Permissions" on page 3-10.

# Transactions

Task instances work with transaction contexts in the following ways:

**Method invocation made within a Transaction Context**
: The invocation of a method on the Worklist API or a control operation is done within the context of the current transaction. If the transaction rolls back, the effects of the operations on the task are undone.

**Method invocation made outside of a Transaction Context**
: The Worklist system starts a new transaction when a new method is called. The transaction is committed on successful completion of that method.

All operations on Task instances behave like an EJB operation with a *required* transaction attribute.

CHAPTER 5

# Advanced Topics

This section includes the following topics:

- Extending Worklist Controls
- Querying Tasks Using the Task Worker Control
- Using Task Control Factories
- Additional Resources

## Extending Worklist Controls

This section provides information about extending the base Worklist controls provided in WebLogic Integration to override the default method signatures, create custom methods and callbacks, query Tasks, and so on. It includes the following topics:

- Why Extend the Worklist Controls?
- Example Extended Task Control
- Altering Method Signatures—Request and Response
- Adding Custom Methods
- Creating Tasks With the Task Control
- Updating Tasks Using the Task and Task Worker Controls
- State Related Updates Using the Task and Task Worker Controls

# Why Extend the Worklist Controls?

When a new Task or Task Worker control is created from the Data Palette in WebLogic Workshop, it provides a standard interface of operations and callbacks. These basic signatures offer the most common operations used by the processes that create, configure and manage tasks, and by those who actually take ownership of tasks and perform their work.

You can extend the control instances and therefore customize them in the following ways:

- Signatures can be altered to accept specific data types as arguments. For example, you can design methods to accept XML Beans created from schemas specific to the application at hand.
- New methods can be added to wrap several different task modifications into a single method.
- Additional callbacks can be added to detect different types of state changes within a business process.

# Example Extended Task Control

The following listing shows an example of a Task control that is customized for managing tasks related to an automated taxi dispatching system.

**Listing 5-1 Customized Task Control**

```
/**
  * @jc:task
  */
public interface AutoTaxiDispatcher
extends TaskControl, com.bea.control.ControlExtension
{
   /**
     * @jc:task-create
     *   name="Pick up {passengerName}"
     *   description="Find customer at {pickupAddress}"
```

```
   *   claim-due-business-date="5 min"
   *   claim-calendar="24by7Calendar"
   *   completion-due-business-date="15 min"
   *   completion-calendar="24by7Calendar"
   *   request="<destination>{destinationAddress}</destination>"
   *   request-type="taxiRideDestination.xsd"
   * @jc:task-assign
   *        group="{locality}Group"
   *        algorithm="ToUsersAndGroups"
   */
  public String passengerReady(String passengerName,
     String destinationAddress, String pickupAddress, String locality );

/**
  * @jc:task-abort enabled="true"
  */
  public void cancelPickup();
/**
  * @jc:task-update
  *        request="<destination>{destinationAddress}</destination>"
  */
  public void changeDestination(String dest);
  public interface Callback extends TaskControl.Callback {
   /**
     * @jc:task-event event-type="claim"
     *   time="{time}" user="{driver}"
     */
     void passengerClaimed(Date time, String driver);
   /**
     * @jc:task-event event-type="complete"
     *   time="{time}" user="{driver}"
     */
     void passengerPickup(Date time, String driver);
   /**
     * @jc:task-event event-type="claimExpire" time="{time}"
```

```
response="{location}"
         */
         void nobodyClaimedPassenger(Date time, XmlObject location);

       /**
         * @jc:task-event event-type="expire" time="{time}"
response="{location}"
         */
         void nobodyPickedUpPassenger(Date time, XmlObject location);

    }

}
```

## Altering Method Signatures—Request and Response

Operations can have arguments that take, as input, the value of a Request or Response document. Callbacks can have return types that return those values.

Since the Request and Response can consist of various formats, operation and callback signatures can be modified to enforce specific types of Request and Response content. The enforcement of the data types can be done when these values are set, or when the values already set are returned.

Method parameters you use to set the Request or Response (and Return types in method signatures that are used to return the value of the Request or the Response) can be set to the following types:

- XmlObject
- XML Beans
- String
- RawData
- byte[]

It is the responsibility of the application to ensure that callbacks return Request or Response types that are compatible with the signatures in the relevant controls. The Worklist system must cast the value that is stored in the system—a mismatch causes an exception to be thrown.

The request type and the response type data values can be used by the application to determine the type of data that is stored in the Request or Response. An application can be designed to use this information in any way.

The following examples of code show how you can write a method to set the request and response using different data types, and how you can design a callback that returns the Response as a Purchase Order XML Bean.

**Listing 5-2 A Method That Sets the Request and the Response as Different Data Types…**

```
/**
 * @jc:task-update
 * request={req}
 * response={res}
 */
 public void setRequestAndResponse(XmlObject req, String res);
```

**Listing 5-3 A Callback That Returns the Response as a Purchase Order XML Bean**

```
/**
 * @jc:task-event event-type="complete" response="{response}"
 */
 void onTaskCompleted(com.bea.purchaseOrderDocument response);
```

# Adding Custom Methods

Methods on the controls can create or update tasks. Creating or updating Tasks can involve altering data values or affecting the task's state. Custom methods can be added to the controls to wrap several update steps into one operation.

First you define your method, passing the parameters needed to create or update the task. Then you ensure that the method annotations specify to the Worklist system what aspects of the task to alter using each parameter.

Constants can be used to alter the task in a fixed way. For example, you can set the task name to **CallDelinquentCustomer** for every new task created. In this way, you do not need to provide the same value as input to the method, repeatedly—the value you provide is a constant.

Note that the default methods on a new Task control or Task Worker control use the annotations mechanism. You can look at the default methods on the controls and use them as an example of how to create custom methods.

Note that, in addition to creating new methods for a control, you can customize a control by deleting some of its default methods.

# Creating Tasks With the Task Control

Methods that create new tasks can also configure the new tasks in several ways. Operations that allow you to create and configure new tasks are identified by the following annotation: `jc:task-create`. The annotation identified by `jc:task-create` contains a sequence of annotations, which specify the way or ways in which the new task is to be configured. Each annotation can specify a parameter to be used to set a particular aspect of the task instance.

Each annotation can specify which parameter should be used to set that particular aspect of the task instance

**Table 5-1 Annotations to Use When You Create and Configure Tasks**

| Aspect of Task to Modify | Annotation | Parameter Type | Notes |
|---|---|---|---|
| Name | name[1] | String | Required annotation |
| Description | description | String | |
| Comment | comment | String | |
| Priority | priority | int | |
| Claim Due Date | claim-due-date | java.util.Date | Use this annotation or the claim-due-business-date annotation. |

| Aspect of Task to Modify | Annotation | Parameter Type | Notes |
|---|---|---|---|
| | claim-due-business-date | String | Business time format.<br>Use this or the claim-due-date annotation. |
| | claim-calendar | String | Name of the business calendar to use if you want to specify a calendar in the method call.<br>This is only used with claim-due-business-date. |
| | claim-user-calendar | String | Name of the user whose calendar to use if you want to specify a calendar in the method call.<br>This is only used with claim-due-business-date. |
| Completion | completion-due-date | java.util.Date | Use this annotation or the claim-due-business-date annotation. |
| | completion-due-business-date | String | Business time format.<br>Use this or the claim-due-date annotation. |
| | completion-calendar | String | Name of the business calendar to use if you want to specify a calendar in the method call.<br>This is only used with claim-due-business-date. |
| | completion-user-calendar | String | Name of the user whose calendar to use if you want to specify a calendar in the method call.<br>This is only used with claim-due-business-date. |
| canBeReassigned | can-be-reassigned | boolean | |

| Aspect of Task to Modify | Annotation | Parameter Type | Notes |
|---|---|---|---|
| canBeAborted | can-be-aborted | boolean | |
| canBeReturned | can-be-returned | boolean | |
| Owner | owner | String | Name of user or group. |
| Request | request | See section on parameter types. | Request content. |
| Request Type | request-mime-type | String | Use only if you use the request annotation. |

1. Note the name annotation is required when you create new tasks; other annotations are optional. To specify a parameter to use as input to the annotations, enter the name of the parameter in curly braces. The following examples show you how.

The following code examples show how to associate annotations with the methods you write to create and configure tasks.

**Listing 5-4 Create a Task, Set the Name, Description, Comment, and Priority**

```
/**
 * @jc:task-create
 *   name="{name}"
 *   description="{desc}"
 *   comment="{comment}"
 *   priority="{priority}"
 */
 public String newCollectionsTask (String name, String desc, int priority,
String comment);
```

**Listing 5-5 Create a Task, Set the Due Dates (Using an Absolute Date and Referencing a Calendar)**

```
 /**
  * @jc:task-create
  *   name="{name}"
  *   claim-due-date="{claimDate}"
  *   completion-due-business-date="{completeDuration}"
  *   completion-calendar="{completeCal}"
  */
  public String createNewTask(
String name, Date claimDate, String completeCal, String completeDuration);
```

**Listing 5-6 Create a Task, Specify the request and request-mime-type**

```
 /**
  * @jc:task-create
  *   name="{name}"
  *   request="{req}"
  *   request-mime-type="{reqType}"
  */
  public String createWithRequest(String name, XmlObject req, String
reqType);
```

# Updating Tasks Using the Task and Task Worker Controls

In addition to the ability to create and configure new tasks, you can update existing tasks in multiple ways using a single custom method. The same annotations are supported for update operations as for create operations on the Task Control, with the exception of name and description. You use them in the same way. Annotations you associate with update methods are identified with the following annotation: `jc:task-update`.

Note that response and response-type annotations are supported for update operations. They are analogous to request and request-type annotations for create operations. the following example code shows the use of the response and response-mime-type annotations.

**Listing 5-7 Task Update Annotations**

```
/**
 * @jc:task-update
 *    comment="{comment}"
 *    response="{resp}"
 *    response-mime-type="{ respType }"
 */
 public String responseAndComment(String comment, XmlObject resp, String
respType);
```

# State Related Updates Using the Task and Task Worker Controls

You can use annotations to configure methods that alter task state. The following method annotations are supported for state transition:

- Annotations to Use With State Transition Operations for Task Controls
- Annotations to Use With State Transition Operations for Task Worker Controls

**Table 5-2 Annotations to Use With State Transition Operations for Task Controls**

| State Transition | Annotation | Parameter Type | Notes |
|---|---|---|---|
| Assignment | jc:task-assign | NA | Assign the task |
| | user | String | If algorithm is `ToUser`, use a String value<br>If algorithm is `ToUsersAndGroups`, can use a String[] value |
| | group | String | If algorithm is `ToUser`, use a String value<br>If algorithm is `ToUsersAndGroups`, can use a String[] value |
| | algorithm | String | Must specify: `ToUser`, `ToUserInGroup`, or `ToUsersAndGroups` |

| State Transition | Annotation | Parameter Type | Notes |
|---|---|---|---|
| Resume | jc:task-resume | | Resume the task |
| Suspend | jc:task-suspend | | Suspend the task |
| Abort | jc:task-abort | | Abort the task |

**Table 5-3 Annotations to Use With State Transition Operations for Task Worker Controls**

| State Transition | Annotation | Parameter Type | Notes |
|---|---|---|---|
| Assignment | jc:task-assign | NA | Assign the task |
| | user | String | If algorithm is `ToUser`, use a String value<br>If algorithm is `ToUsersAndGroups`, can use a String[] value |
| | group | String | If algorithm is `ToUser`, use a String value<br>If algorithm is `ToUsersAndGroups`, can use a String[] value |
| | algorithm | String | Must specify: `ToUser`, `ToUserInGroup`, or `ToUsersAndGroups` |
| Resume | jc:task-resume | | Resume the task |
| Suspend | jc:task-suspend | | Suspend the task |
| Abort | jc:task-abort | | Abort the task |
| Claim | jc:task-claim | | Claim the task |
| | claimant | | Use with task-claim, if claiming on behalf of another user. |
| Delete | jc:task-delete | | Delete the task |
| Return | jc:task-return | | Return the task |

| State Transition | Annotation | Parameter Type | Notes |
|---|---|---|---|
| Start | jc:task-start | | Start the task |
| Stop | jc:task-stop | | Stop the task |
| Complete | jc:task-complete | | Complete the task |

# Getting and Setting Task Data Values

The default methods on the Task and Task Worker controls can get, set, and remove control data values. For information about the data values you can set and get for the Task and Task Worker controls, see "Task Data Values" on page 3-13. The following examples show annotations you can use with get, set, and remove methods on the controls.

**Listing 5-8 Annotations to Use With Get, Set, and Remove Operations**

```
/**
 * @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);
```

# Adding Callback Methods

Callbacks provide a way for a control to asynchronously notify a client that an event has occurred. A callback is a method signature that is defined by a control and for which the method implementation is provided by the client. For example, a business process can implement a callback handler to enable reception of a callback from a control.

You can extend Task controls with callback methods to report state or property changes, or *events*. You can implement callback methods on Task controls only, not on Task Worker controls, because only Task controls identify with a single active instance of a Task.

Callbacks can optionally return up to three arguments:

- user—the user who executes the operation to make the state transition
- time—the time at which the due date expired or the operation was invoked
- response—the response document

The following example displays the user, time, and response annotations associated with a callback method

**Listing 5-9 Callback Method Using User, Time, and Response Annotations**

```
/**
 * @jc:task-event event-type="complete"
 *   response="{response}"
 *   time="{time}"
 *   user="{user}"
 */
 void onTaskCompleted(XmlObject response, Date time, String user);
```

Set the `event-type` annotation to specify the state transition that triggers the callback. Use the appropriate annotation for the type of event that triggers your method. The following table describes the types of events and the associated annotations.

| Type of Event . . . | Annotation . . . |
|---|---|
| CREATE | create |
| ASSIGN | assign |
| CLAIM | claim |
| SUSPEND | suspend |
| RESUME | resume |
| COMPLETE | complete |
| ABORT | abort |
| RETURN | return |
| START | start |
| STOP | stop |
| CLAIM_EXPIRE | claimExpire |
| EXPIRE | expire |

# Querying Tasks Using the Task Worker Control

This section explains how to extend a Task Worker control to query WebLogic Integration tasks. The `@jc:select` annotation accepts values to search for Tasks, including `TaskSelector` objects, and returns a set of Task IDs.

This section includes:

- Search Values and Selectors
- Querying Tasks With Annotations
- Querying Tasks With TaskSelectors

## Search Values and Selectors

The Java annotations for a Task Worker control provide a set of properties you can use to query on the `@jc:select` tag. The following table describes these properties.

**Table 5-4 Task Control Properties to Use With the jc:select Java Annotation**

| Search Property | Description |
|---|---|
| `assigned-group` | Search by groups on the assignee list for a Task. |
| `assigned-user` | Search by users on the assignee list for a Task. |
| `claimant` | Search by the claimant for a Task. |
| `claim-due-date-after` | Search by Tasks with a due date after the value you provide. |
| `claim-due-date-before` | Search by Tasks with a claim due date before the value you provide. |
| `comment` | Search by the Task comments. |
| `completion-due-date-after` | Search by Tasks with a completion due date after the value you provide. |
| `completion-due-date-before` | Search by Tasks with a completion due date before the value you provide. |
| `creation-date-after` | Search by Tasks with a creation date after the value you provide. |
| `creation-date-before` | Search by Tasks with a creation date before the value you provide. |
| `max-property` | Search by Tasks with no greater priority than the value you provide. |
| `min-priority` | Search by Tasks with no lesser priority of the value you provide. |
| `owner` | Search by The Task owner. |
| `property-name` | Search by Tasks with a given property. |
| `property-value` | Search by Tasks with a given value for the property defined by `property-name`. |

| Search Property | Description |
|---|---|
| `selector` | Search by the configuration of the `TaskSelector` object you provide for this value. To learn more about using this value to pass arguments to `TaskSelector` objects, see "Querying Tasks With TaskSelectors" on page 5-17. |
| `states` | Search by Tasks by state. Values can be as follows:<br>• A `String` or `String` array of valid state types such as completed or assigned.<br>• An `Integer` or `Integer` array representation of state types.<br>• A `com.bea.wli.worklist.api.StateType` or `StateType` array |
| `task-id` | Search by the unique Task ID. |
| `task-name` | Search by the groups on the assignee list for a Task. |

## Querying Tasks With Annotations

To create custom queries, you must extend the Task Worker control to provide a method for your search. You can start by taking a method that already uses the `@jc:select` annotation tag to perform a search and modify this method.

For example, the Task Worker control provides the following `@jc:select` tag and method:

```
/**
 * @jc:task-get-info enabled="true"
 * @jc:select task-id="{taskIds}"
 */
 public TaskInfoXMLDocument[] getTaskInfoXML(String[] taskIds);
```

The `TaskInfoXMLDocument` method allows you to search through Tasks by Task ID. You can extend this control to search through Tasks by some additional attribute, such as the claimant. To do so, you can take advantage of extensibility built into the `TaskInfoXMLDocument` method that allows you to pass additional arguments to the method, as shown in the following example. (Bold text is used in the following example code to indicate the additions made to the preceding example code.)

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

```
    * @jc:select claimant="James Gosling" task-id="{taskIds}"
    */
    public TaskInfoXMLDocument[] getTaskInfoXML(String user,String[]
taskIds);
```

In the previous code sample, the `TaskInfoXMLDocument` is extended to query for a claimant.

You can extend your control without setting a default value for the claimant. For example, you can provide a value within curly brackets to indicate the claimant must be a user. You must pass a value for the `user` parameter to the method at run time.

```
    /**
     * @jc:task-get-info enabled="true"
     * @jc:select claimant="{user}" task-id="{taskIds}"
     */
     public TaskInfoXMLDocument[] getTaskInfoXML(String user,String[]
taskIds);
```

To learn more about Worklist control annotations, see Worklist Control Annotations in the WebLogic Workshop online help.

# Querying Tasks With TaskSelectors

The Worklist API provides the functionality to create a more advanced search functionality than you can using the `@jc:select` Java annotation, described in the preceding section. You can query on all Task instance properties by making a call to a `TaskSelector` object. This allows you to order the results of queries, find parent process IDs, use regular expressions, and so on.

To use a `TaskSelector`, you must include the `selector` annotation and argument with a method. The following code shows a method in the Task Worker control that you can use as a starting point to extend your control.

```
   /**
    * @jc:task-get-info
    * @jc:select selector="{selector}"
    */
    public TaskInfoXMLDocument[] getTaskIdsWithSelector(TaskSelector
selector);
```

The method described in the preceding example expects, as input, a `TaskSelector` that is defined to query by Task ID. The method returns a `TaskInfoXMLDocument` array of resulting Task properties with Task IDs that match your query.

For more information about the constructor and methods of the `TaskSelector` class, see Class TaskSelector in the `com.bea.wli.worklist.api` package, in the BEA WebLogic Integration Javadoc, which is available at the following URL:

`http://edocs.bea.com/wli/docs81/javadoc/index.html`

# Using Task Control Factories

There are two circumstance in which you must use a Factory type of Task Control.

- You are creating multiple tasks in a loop, or you want the number of tasks created to be specified at run time.
- You are interacting with multiple existing tasks in a loop, or with a number of tasks that is not known until run time.

Task Control properties can be useful when using factory type Task Controls. Defining the control's properties in the Property Editor specifies default values for new tasks that are created by any control instance that was created from that factory.

For example, say you want your process to loop over a sequence of order elements, creating a new task to approve each order in the body of the loop. Say, the task name changes on each iteration, but the assignee is always the same manager. In that case you can set the assignee in the factory control's properties using the property editor. You need not specify it when creating a new task, that default assignee will be used.

The basic pattern is to create two Task controls in your business process: one is a factory type and the other is not. The following example describes a scenario for which you want to create a task for each iteration through a loop in a business process:

1. From the **Data Palette**, create a new Task control called **MyTaskCtrl** (name the Task control instance: `factoryCtrl`). As you do so, the **Insert Control** dialog box prompts you to specify if you want to make the control a factory that can create multiple instances at run time. Select this option.
2. Create another Task Control from the **Data Palette**. This time, in the **Insert Control** dialog box, select the option to **Use a task control already defined by a JCX file**, and specify the existing **MyTaskCtrl.jcx** file. (Name the Task control instance: `myTaskCtrl`.) Do *not* select the option to make this control a control factory.

   The controls variables in the business process are written as shown in the following code.

```
/**
 * @common:control
```

```
 */
 private processes.MyTaskCtrlFactory factoryCtrl;


/**
 * @common:control
 */
 private processes.MyTaskCtrl myTaskCtrl;
```

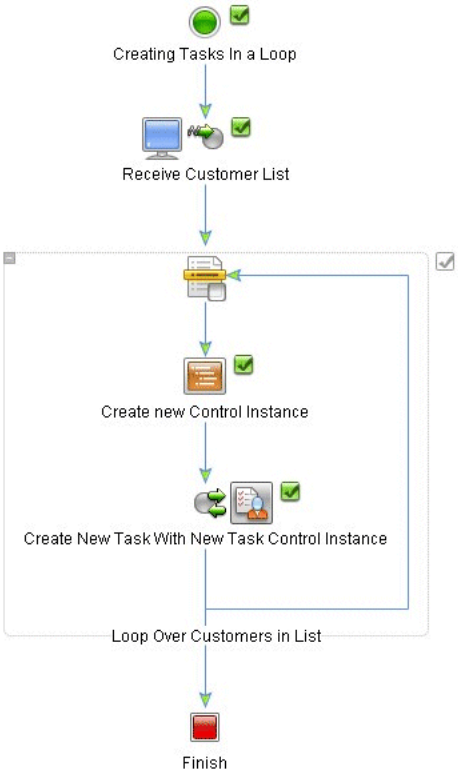
3. Open the business process you are designing in the **Design View**.
4. In the body of a loop, or any other place you want to create a new task or want to do work on an existing task, create a new control instance using **factoryCtrl** and store it in a variable named **myTaskCtrlVariable**. You design this logic in a Perform node. The code for the Perform node should resemble the following:

   ```
   public void factoryCreate() throws Exception
   {
       this.myTaskCtrl = this.factoryCtrl.create();
   }
   ```

   In this way, you create a new control that you can use to create a new task or set the Task Id (`setTaskId (String id)`) to the ID of an existing task

   Because you are designing the process to create a new control instance in a loop, you must create the instance using the factory control.

   The following figure shows the simple business process described in this example:




# Additional Resources

To learn more about working with the Worklist system, see the following resources:

- Tutorial: Building a Worklist Application

  `http://edocs.bea.com/wli/docs81/wltutorial/index.html`

- Worklist Control Annotations

  `http://edocs.bea.com/workshop/docs81/doc/en/integration/reference/refWorklistAnnotation.html`

- Worklist Control Interfaces

  `http://edocs.bea.com/workshop/docs81/doc/en/integration/reference/refWorklistInterfaces.html`

- Worklist Administration in *Managing WebLogic Integration Solutions*

  `http://edocs.bea.com/wli/docs81/manage/index.html`

- BEA WebLogic Integration Javadoc

  `http://edocs.bea.com/wli/docs81/javadoc/index.html`