BEA WebLogic Integration™

Using the Worklist

Version 8.1 Service Pack 4
Document Date: December 2004

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

1. Introduction

2. Using Worklist Controls

3. Creating and Managing Worklist Tasks

4. Worklist User Interface and Enterprise JavaBeans API

5. Advanced Topics

Index

CHAPTER 1

Introduction

The BEA WebLogic Integration Worklist addresses the flow of work in an enterprise. It directs the routing of business-process tasks to personnel and provides management capabilities that enhance organizational efficiency and responsiveness.

This section includes the following topics:

What is WebLogic Integration Worklist?

The WebLogic Integration Worklist is commonly used for scenarios in which human activity combines with automated business processing over time. This time can be short or can continue for days or months. The Worklist is designed to direct the flow of work and manage the routing of tasks to the people in an enterprise. Inherent in the progression of work are actions such as receiving, approving, modifying, and routing of documents. The documents that accompany work activities provide the information necessary for people to perform and complete tasks. The Worklist enables people to collaborate in the accomplishment of work, including creating tasks, assigning tasks, tracking the status of tasks, handling approvals, and other activities required to manage workflow.

The Worklist user interface enables end users—task creators, task workers, task administrators—to interact with running business processes, including handling process exceptions, approvals, and status tracking. For managing the Worklist components of your applications, WebLogic Integration provides a sample Worklist user interface that you can use to help develop your user interfaces.

Important capabilities of the Worklist include the following:

- Worklist allows people to act in specific roles when they work within a business process and to concentrate on just those parts of work where they add value.
- It allows business rules to dictate the routing of documents and work.
- It manages permissions for work items. For example, applications can manage who can see which documents and what work queues, and who has ownership of tasks.
- It provides notification to management when the work is completed or when deadlines are missed.
- Management can study the histories of workflows and use this information to fine tune business processes, increase productivity, and lower response times.

Worklist Tasks

The Worklist allows for the creation, manipulation, and management of Tasks. A Task instance represents a unit of work that requires completion within a certain time period. Examples of tasks include:

- A Manager reading and approving an employee's vacation request.
- Phoning a customer and recording that customer's complaints.

In the run-time Worklist system, a Task instance is a particular object that represents a work assignment in the real world. Task instances are part of the WebLogic Integration server and exist independently of any controls or business processes. Multiple business processes can interact with a Task throughout its lifecycle concurrently. Tasks remain in the run time indefinitely, either until they are explicitly deleted or purged by the WebLogic Integration purging process.

After the work is completed, you can use a Task instance to represent a detailed record of that unit of work.

To learn about controls, see Worklist Controls in the WebLogic Workshop Help.

Task Data Values

Task Instances have built-in data values for defining how work should be performed, who should do it, by when it needs to be completed, and more. You can also use these data values to capture what actually was done when work is completed.

Examples of task data values include:

- A list of users and groups who can work on the task (Assignees List).
- The date a task is due to be completed (Task Competition Due Date).
- The user who claims the Task and attempts to complete the work (Claimant).
- The person who is responsible for managing the process of the work being performed (Task owner).
- Documents that can describe the work to be done and the results of completing the work (Request and Response documents).
- The condition that defines the point at which a Task instance is in its lifecycle (Task State).
- A priority that indicates the relative urgency of this task relative to other tasks (Task Priority).

Due Dates

Due dates represent the date and time for which tasks should be claimed or completed. The Worklist 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.

Business calendars represent the operating hours of a business. A business calendar specifies a time zone and a set of time period rules. Time period rules determine the days, dates, and hours that are available (free) and unavailable (busy) for business activities.

To learn more 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 lifecycle. 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

To signify a user or a group of users that play a managerial role with respect to this task, a Task Instance *can* have a Task Owner. The owner manages the task and is the person responsible for getting the Task completed, but not necessarily the person who actually completes the Task. 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 the Task does not get done.

The permission to perform certain managerial operations on a Task instance can be restricted to only the Task Owner or to an administrator.

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

Assignees Lists and Claimants

A Task Instance has a list of assignees 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 more about the Assignees List and claimants, see "Task Users and Groups" on page 3-8 and "Tasks and User Permissions" on page 3-10.

Request and Response Documents

Generally, the Task Request is used to specify what work is done and how the work is 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.

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 is what specific actions were performed 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 they complete a phone conversation. The process that created the Task to call that customer can use those results to determine what should be done next.

To learn more 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. Certain Task data values are specified at the time an instance of a Task is created and cannot 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 in its lifecycle.

Operations to Modify Task State

Task Instances can transition between states based on the operations defined for them. Some kinds of operations that 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 Worklist Controls" on page 2-1.

Tracking and Purging Task Information

WebLogic Integration supports the recording of 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 forth. Worklist task instances generate events that can be logged in Worklist history tables in the run-time repository. The records created in the history tables are intended for use 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

To optimize performance, the amount of tracking data stored in the run-time database should be kept to a minimum. To help ensure this, the tracking and purge process can be configured to run at regular intervals by an administrator. Additionally, the reporting data policy for a process can be set to *on* or *off*. If on, the data is transmitted to the reporting database if the reporting data stream is enabled; if *off*, the data is not transmitted to the reporting database.

Stored 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 tracking and purging Task data, see System Configuration in *Managing WebLogic Integration Solutions*.

Stored information can be used for generating reports and compiling statistics about task processing in your WebLogic Integration application. To learn more about tracking and purging Task data, see System Configuration in *Managing WebLogic Integration Solutions*.

Task Queries

Task Queries allow an application to find all tasks in the run-time Worklist system that meet a specified set of criteria. These queries are directly analogous to SQL and Databases Tables. The results returned by the queries contain information about all tasks that meet the specified 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 for creating and operating tasks. WebLogic Integration provides two controls to support the Worklist system: the Task control and the Task Worker control. Although the Task and Task Worker controls provide a subset of the available functionality in the API, they provide the convenience of the WebLogic Workshop controls framework and you can easily use them in your business processes.

The Worklist controls are server-side components managed by the Workshop framework. They expose Java interfaces that can be invoked directly from your business processes. The Task control enables a business process to create a single Task instance, manage its state and data, and provide callback methods that report status. The Task Worker control allows specified users to acquire ownership of Tasks, work on the task, and complete the task. This control also provides administrative privileges, such as starting, stopping, deleting, and assigning of tasks. Access to the Task Worker control can be done through a business process or a user interface.

Task Control

In an office environment, managers usually create, specify, and monitor the work that is done. 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

The Task Worker control is designed to provide the most common operations needed by the people who receive assignments and perform the work. For example, workers can query for their assigned tasks that are due before the end of the week. The Task Worker control allows workers to mark ownership of a task (that is, claim the task) and mark its completion. In addition, the Task

Worker control has operations of an administrative nature, such as claiming a task on another user's behalf. The Task Worker control is most commonly used in the implementation of a custom Worklist user interface.

Callbacks

Task Controls can notify a process when a Task's state changes. Common callbacks include the expiration of a completion due date, aborting the task, or task completion. Callbacks allow processes to block within their logic, effectively waiting until that event takes place.

Control Methods

Controls methods are the mechanism by which a business process or a user interface creates new tasks, reads or alters the task data values, reads the current state, causes a transition to a new state, or deletes a task instance.

Controls are Extensible

Task and Task Worker controls have a built-in set of methods 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 methods, you can configure a control's data values, create new tasks, update existing tasks, and so forth. You can also extend controls in the following ways:

- You can alter signatures on methods to take XML Bean types as arguments when the XML associated with tasks conforms to custom-defined schemas in your application.
- You can add new methods 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.
- You can add callbacks to detect other state changes in a task. For example, you can add a callback to detect when a task is claimed.

Administration and Management

Using the Worklist Administration module in the WebLogic Integration Administration Console, you can administer and manage the tasks in the Worklist, business calendars, task properties, and other features. With Worklist controls and the Worklist API, you can create a custom Worklist client. The Worklist API is available as Enterprise JavaBeans (EJBs) and MBeans. These topics are discussed more fully in the following sections:

- WebLogic Integration Administration Console
- Worklist User Interfaces
- Enterprise JavaBeans API

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 for monitoring the progress of task completion with respect to due dates.
- Perform queries to show individual workload.
- Reassign tasks to speed progress.
- Change task properties, such as state or due date.
- Control task routing by creating or changing substitute routing rules.

To learn more about managing your Worklist Tasks, see Worklist Administration in *Managing WebLogic Integration Solutions*.

Worklist User Interfaces

Worklist user interfaces enable end users—task creators, task workers, task administrators—to interact with running business processes, including handling process exceptions, approvals, and status tracking. Typically, in real-world applications, people interact with tasks that require custom user-interfaces. For example, you could add a page to an existing order fulfillment user interface on a company's intranet that allows a manager to approve very large orders.

WebLogic Integration contains a sample Worklist user interface. This sample interface provides basic out-of-the-box functionality for inspecting and updating tasks.

The WorklistScrollableResultManager Interface allows you to query Worklist tasks. This interface allows client applications, especially user interface (UI) applications, to retrieve a specified number of records rather than returning all query results. This improves scalability and performance for querying operations. For an example of using the

`WorklistScrollableResultManager`, see "Use Case 2" in the *Worklist Sample* in the Solution Samples.

To learn more about Worklist user interfaces, see Chapter 4, "Worklist User Interface and Enterprise JavaBeans API."

Enterprise JavaBeans API

With the EJB API, you can use the Worklist EJB to create and manage tasks independent of the business processes (JPD) created in WebLogic Workshop. Code samples are included in Chapter 4, "Worklist User Interface and Enterprise JavaBeans API."

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 business processes. In other words, controls are the interfaces between your business processes and other resources.

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

About Worklist Controls

To support the integration of business users with the Worklist system, WebLogic Integration provides two Java controls: the Task control and the Task Worker control.

As with other built-in controls in WebLogic Workshop, you use the controls by adding instances of the controls to your business process and then invoke 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 forth.

- The *Task control* enables a business process or user interface (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, and complete them. It offers administrative operations, including operations to start, stop, delete, assign, and more. Task Worker controls allow operations on several Task instances simultaneously.

Creating a New Task Control

An instance of a Task control can create one or more Task instances. The single type of Task control creates a single Task instance. The factory type of Task control creates multiple Task instances. To learn about creating multiple Task instances, see "Using Task Control Factories" on page 5-17.

A Task control instance can also 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 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.

For instructions on creating a new Task control, see Creating a New Task Control in the WebLogic Workshop Help.

Creating a New Task Worker Control

The Task Worker control allows specified users to acquire ownership of Tasks, work on them, and complete them. It also provides administrative privileges, such as starting, stopping, deleting, and assigning. Access to the Task Worker control can be done with a business process or through a user interface. You can customize each Task worker control for different business purposes.

For instructions on creating a new Task Worker control, see Creating a New Task Worker Control in the WebLogic Workshop Help.

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 that you design the interactions between business processes and other controls in the WebLogic Workshop, you can bind the Worklist control method to the appropriate control node in your business process (**Control Send**, **Control Receive**, and **Control Send with Return**). You do this in the **Design View** by simply dragging a control method from the **Data Palette** onto the business process at the point in your business process at which you want to design the logic. 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 only operates on a single task instance. 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.

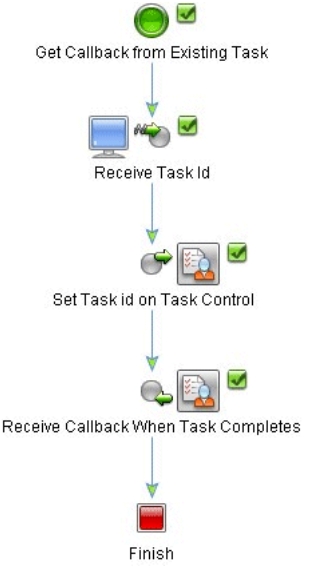
The Active Task ID is set either by creating a new task, or by invoking the `setTaskId` method. When a new task is created, the Active Task ID is automatically set to the ID of the newly created task. In this way, subsequent operations are performed on the new task.

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. This means that you can use a Task control factory to create a new Control instance every time a new Task is required. To learn more about Task control factories, see "Using Task Control Factories" on page 5-17.

Because it is possible to use the Task ID to determine which task a control operates or receives callbacks from, multiple business processes can incorporate controls that operate on the same task. New processes, that is, processes that are instantiated after a task that already exists can interact with that task using the `setActiveTaskId` method.

The Task Worker control does not use the active task model. Instead, to specify which tasks need to be updated, Task IDs are passed explicitly to methods on the Task Worker control. 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 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)` method 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 assigned and 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 assigned and claimed tasks), then the algorithm chooses one user randomly.

`assignTaskToUsersAndGroups` **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 Assignees List can place a Task in a claimed state. If the Assignees List contains only one user, such as in the case of `assignTaskToUser` or `assignTaskToUserInGroup` methods, 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 2-20.

You can reassign Tasks, placing them back into the assigned state while changing the Assignees List, effectively giving them to different users. You can also return Tasks, placing them back into the assigned state with the same Assignees 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

The data values for tasks that you can set depend on the permissions you have been 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 3-14.

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.

Control operations set values for business dates, such as due dates, completion dates, and claim dates, to make sure that work in done in a timely manner. To learn more, see "Task Due Dates" on page 3-2.

Table 2-1 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. The specified calendar converts the business time duration to an absolute date. Conversions to absolute dates using the calendar are done relative to the current time.
`setClaimDueBusinessDateSystemCalendar(String duration)`	Sets the claim due date using a business time duration. The system calendar is used to convert the business time duration to an absolute date.
`setClaimDueDate(Date date)`	Sets the claim 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.

Operation	Description
`setCompletionDueBusinessDate(String duration, String calendarID)`	Sets the completion due date using a business time duration and a calendar name. That calendar converts 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 converts 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.
`setOwner(String owner)`	The specified String sets the Task Owner as the user or group name. You can specify `null` to unset the owner, which means that no owner is specified for the task.
`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 zero or any integer greater than zero.
`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 does not exist. You cannot specify `null` for either the name or the value parameter.
`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.

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

Table 2-2 Operations That Affect Task State

Methods	Controls	Description
`abortTask`	Task Task Worker	Invokes the abort operation on a task.
`assignTaskToUser`	Task Task Worker	Assigns tasks to the user whose name you provide as an argument to the method. The specified user must belong to the WebLogic Integration Users Group. The Assignees List is set accordingly for the task instance. The task is then automatically claimed for the specified user. To learn about users and groups for Worklist controls, see "Task Users and Groups" on page 3-8.
`assignTaskToUserInGroup`	Task Task Worker	Assigns the task to a user in the group whose name you provide as an argument to the method. The Worklist uses a load balancing algorithm to choose the least busy user in the group.
`assignTaskToUsersAndGroups`	Task Task Worker	Assigns the task and sets the Assignees List to contain the users and groups that you provide as an argument to the method. The users and groups specified must belong to the WebLogic Integration Users Group.
`resumeTask`	Task Task Worker	Invokes the resume operation on the task.
`suspendTask`	Task Task Worker	Invokes the suspend operation on the task.
`updateTask`	Task	Alters the state through assignment. To learn more about Task properties and XML, see Using XML With the Task Control.

Methods	Controls	Description
`archiveTasks`	Task Worker	Not functional in SP3 and later.
`purgeTasks`	Task Worker	Purges all logged tasks from the worklist history tables. To learn more about purging Tasks, see System Configuration in *Managing WebLogic Integration Solutions*.
`claimTask`	Task Worker	Causes the claim operation to be called. The currently executing principal claims the task.
`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.
`deleteTask`	Task Worker	Removes the task instance completely and permanently at run time. You can also use the WebLogic Integration Administration Console to remove task instances. This is an administrative function.
`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.
`startTask`	Task Worker	Invokes the start operation on a task.
`stopTask`	Task Worker	Invokes the stop operation on a task.

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.
- 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

For ease of use, several operations on the Worklist controls offer an XML interface. 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 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 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 with a TaskCreationXML Document

You can use a TaskCreationXML document 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 document 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)
```

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

The following is an example of an XML document that 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 is to be used. To set the due date as a `java.util.Date`, this calendar converts the business-time duration to an absolute time.
- **`assignee`**—Assign the task directly to a user, a user in a group, or to set the Assignees List to contain a list of users 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>ToUserInGroup</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

You can use an XML element 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()

You can use public `TaskInfoXMLDocument getTaskInfoXMLDocument()` to get a `TaskInfoXMLDocument` on the Worklist Controls. It contains the task properties, state, and other attributes in a single document.

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).

Importing the Worklist Schema into Your Application

To import the Worklist schema into your application:

1. From WebLogic Workshop, in the **Application** tab, right-click the top-level application folder.

Note: 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 when you plan to use the project for application view channels and schemas. In that case, you must name it **Schemas**.

5. Click **Create**. The new project is created and displayed in the **Application** tab.

The Task Control Properties Sheet

You can use the Task Control Properties Sheet to set property defaults for new task instances created by a control.

When a new task is created with the control, the values in the properties sheet are used, unless the creation operation passes a parameter that explicitly overrides the value. If values are not set by the method parameters and do not exist in the properties sheet, the Worklist defaults are used for those values.

For example, if the properties sheet specifies values for the task name, task owner, and priority, and the user creates a new task with a TaskCreationXML document that specifies only the task description and priority, the following takes place:

- The value from the TaskCreationXML for the description is used.
- Because the priority is specified in both the XML and the properties sheet, the XML value is used.
- Because the task comment is not specified in the XML or the properties sheet, the system default is used.

Table 2-3 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 and ToUsersAndGroups
Assignee	User	If algorithm is ToUser, specifies the user name	String
Assignee	Group	If algorithm is ToUserInGroup, specifies the group name	String
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

Property Section	Property	Purpose	Valid Values
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

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

To view or edit properties for control instances:

1. In WebLogic Workshop, 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**.

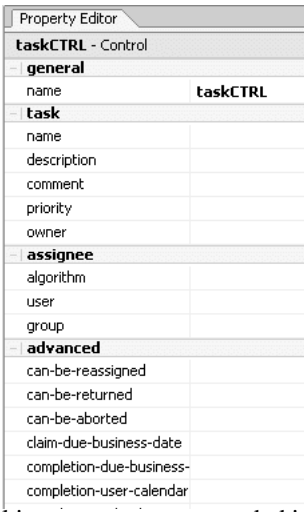
   Note: If the **Data Palette** or the **Property Editor** is not visible, from the menu bar, click **View→Windows→Data Palette** or **View→Property Editor**.

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 specify properties for control types versus control instances using the **Property Editor**, see Setting Control Properties in the WebLogic Workshop Help.

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**. To view the instance, in the **Data Palette**, click **taskCTRL**. The instance is highlighted and its properties are displayed in the **Property Editor**.

The following figure shows the **Property Editor** for the 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, see Worklist Control Annotations in the WebLogic Workshop 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.

Table 2-4 Task Control Callback Methods

Method	Description
`onTaskAborted`	This is a callback method that another resource, such as 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, such as 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, such as 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 5-13.

Permissions and Roles

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

Table 2-5 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. Claimant if `canBeAborted` is true. Task Owner and Integration Administrator
ASSIGNED, CLAIMED, STARTED	SUSPENDED	`suspendTask`	Task Owner and Integration Administrator
SUSPENDED	The Task State 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 that is executing the Task, and the particular property to be modified. The following list provides the details:

- Some properties can be modified only by the Task Owner or an Integration Administrator, regardless of 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 in 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 more about Task states and moving Tasks between states, see "Task States" on page 3-4.

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. Additionally, Worklist controls provide the following Boolean properties that 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 "Task Operations" on page 3-19.

Permissions for Creating Tasks

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

Modifying Task Data Values

Which users and groups 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, as described in the following list:

- Some data values can be modified only by the Task Owner or an Integration Administrator, regardless of the state of the task.
- Data values cannot 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.
- To modify Task Data values for tasks that are in the ASSIGNED state, the current user must be on the Assignees List, be the task owner, or be an Integration Administrator.
- Only the claimant, task owner, and Integration Administrators can modify task data values for tasks that are in the CLAIMED and STARTED states.

To learn about the data values and the permissions required to modify them, see "Task Data Values" on page 3-14.

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 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 performed, 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 actions to address overdue work 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

Worklist provides the option to set one or both of the following due dates for a Task Instance:

- **Claim Due Date**—specifies the deadline for a task to be *Claimed* by a user on the Assignees List. Setting **claimDueDate** to null indicates that there is no due date.
- **Completion Due Date**—specifies the deadline for a task to 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 Worklist detects that a due date has passed, it checks whether the associated task is claimed or completed. If the task is not claimed or completed, Worklist invokes callbacks 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, a business calendar defines business hours as Monday, Tuesday, and Wednesday from 9AM to 5PM. If on Saturday the 16th, you set a Task's duration for four business days, the resulting due date is Monday the 25th (Monday the 18th, Tuesday the 19th, Wednesday the 20th, Monday the 25th).

To learn more about business calendars and the WebLogic Integration Administration Console, see Business Calendar Configuration in *Managing WebLogic Integraton 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. To specify a business calendar for the system to use when it calculates due dates, do one of the following:

- Explicitly pass 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.*
- 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 Integraton Solutions*

Formats for Business Time Duration

Business-time durations are strings that use the following format: X d Y h Z min. You can specify days, hours, or minutes, or a subset of these values. 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 Integraton 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 depend on the state of the task and 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. 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. 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, that is, 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. 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. You can use the response document to record the details of how work was done or the results of doing work.
SUSPENDED	A SUSPENDED task is frozen—it cannot be worked on. SUSPENDED tasks can be resumed at a later time, returning to the state previous to their suspension. 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. 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. Some data values, such as the description, can be set only when this operation is invoked. Note that the currently executing principal must belong to a group that is specific to the Worklist system before permissions are granted 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 and specify which users can claim the task. This operation can be performed on tasks in a final state, such as COMPLETED or ABORTED. This allows you to work on the task again. 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. 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. It 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. It 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
complete	Causes a task in the STARTED state to become COMPLETED. It signifies that the claimant has finished the work required to complete the task, or as much of the work as is possible to do is finished. This operation is performed by the claimant, or by an administrator or task owner on behalf of the claimant.
suspend	Causes a task to become SUSPENDED. It 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 previous to its suspension. This operation is performed by an administrator or by the task owner.
abort	Causes a task to become ABORTED. It 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`, or `assignToUsersAndGroups`. The methods described in the following table specify how the Assignees List is set.

Table 3-3 Assignment Algorithms

Method	Description
assignToUser	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 examples of how to use this method, see Step 4. Create Task and Assign to User in the *Tutorial: Building a Worklist Application*.
assignToUserInGroup	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 assigned and claimed tasks that are not completed, aborted, or suspended. A group name must be specified for this method.
assignToUsersAndGroups	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 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 Assignees 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 the work. For a user to be on the Assignees List, either the user name is explicitly listed or a group to which the user belongs is explicitly on the list. The user in the Assignees List that claims the task, becomes the *claimant*.

Claimants

A Claimant is the user 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 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 Integraton 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 Integraton 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:

- Worklist Security
- Who Has Permission to Create Tasks?
- Who Has Permission to Modify Task Data Values?
- Who Has Permission to Invoke Task Operations?

Worklist Security

The Worklist relies on the WebLogic Server security framework and authentication provider to enforce security.

If more than one security provider is defined in the active realm of the domain, the Worklist gathers information from all of them. These authentication providers can or can not implement some of the MBeans that are used by Worklist.

If the following MBeans are not present, some features are deactivated, but the Worklist is still functional:

- `UserReaderMBean`—used to validate users when assigning a Task or setting the Task owner.
- `GroupReaderMBean`—used to validate a group when assigning a Task to a group or setting the Task Owner.

- `GroupMemberListerMBean`—used when assigning a Task using the algorithm `ToUserInGroup`.

If `UserReaderMBean` or `GroupReaderMBean` is not present, all the users and groups validations for the task owner and task assignee are deactivated. If `GroupMemberListerMBean` is not present you cannot use the algorithm `ToUserInGroup` when assigning a task. Additionally, you cannot select all the tasks whose assignee is in a specific group.

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 Integraton Solutions*.

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

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, such as 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 3-4 User Roles and Task Operations

Starting State	Ending State	Relevant Operations	Permitted Users
Assigned Claimed Completed Started	Aborted	abortTask	Assignee list users and groups (if can-be-aborted property true) Claimant (if can-be-aborted property true) Task Owner 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
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

Starting State	Ending State	Relevant Operations	Permitted Users
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. They also describe 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 only at the time a new Task is created; some values can be modified throughout the lifecycle 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 3-5 Task Instance Data Values

Name	Purpose	How to Set	Type	Unset Value	Default	Notes	Only Admin. Can Modify
TaskId	Identifies Task instance. 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

Name	Purpose	How to Set	Type	Unset Value	Default	Notes	Only Admin. Can Modify
ParentProcessURI	The business process type that created the 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 the task, if any.	System sets at creation time, user cannot set	String	Can be null	NA		NA
Assignees	The Assignees List for the task, that is, 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
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

Name	Purpose	How to Set	Type	Unset Value	Default	Notes	Only Admin. Can Modify
Priority	Can be zero or any positive integer. Worklist does not do anything with this value; the application can interpret or ignore.	Set by user	Integer	Always non-null	1		True
CreationDate	When the task was created.	Set by Worklist system	java.util. Date	Always non-null	NA		NA
CanBeReassigned	Allows a user on the Assignees List assign to reassign the task. Otherwise requires administrative privileges.	Set by user	Boolean	Always non-null	True		True
CanBeReturned	Allows the claimant to return the task after claiming 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
CanBeAborted	Allows the claimant to 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 what information is 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, 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
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

Name	Purpose	How to Set	Type	Unset Value	Default	Notes	Only Admin. Can Modify
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	Deadline for some user to have claimed the task	Set by user	java.util.Date	Can be null	Null	Null means no due date	True
CompletionDueDate	Deadline for claimant to 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 performed to complete the task, the progress of the work, 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.

You use a Task Request to specify what work needs to be done to complete a task, or how to do the work. 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 can specify the work that took place after a user worked on the task, or the resulting data generated as the result of the work performed 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 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. This means that 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 request and response types can be stored as `mimeType`, which can be any String. The value of the `mimeType` is provided for the interpretation of the application; it is 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

You use operations 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 are contained in the following table.

Table 3-6 Task Operations

Operation	Description
Create Tasks	When a new Task Instance is created, the Worklist system assigns a unique ID (a **taskId**) to that instance. Depending on the operation, the Task *State* can be defined at creation time either as **Assigned** or **Claimed**. **Note:** Some Task *Properties* are specified at the time an instance of a Task is created and cannot be changed after the instance is created.
Modify Task Properties	Some Task Properties can be specified and modified after an instance of a Task is created.
Get Task Properties	You can use *Get Task* operations to access the properties of any Task Instance at any point during its lifecycle.
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.

Worklist Tracking Data

Each worklist task instance generates events that can be logged into worklist history tables in the run-time 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 tracking level setting in the WebLogic Integration Administration Console determines the data that is logged as follows:

- **Full**—All transitions and changes, including task requests and responses, are logged.
- **Basic**—Transitions and changes are logged. Task requests and responses are not logged.

- **None**—No task history is tracked.

The Task Worker control provides the following methods for logging and deleting Tasks:

- `archiveTask`—not functional in SP3 or later.
- `deleteTask`—permanently removes 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. No mechanism exists for rollback or retrieval of deleted Tasks.

- `purgeTask`—deletes all Tasks in completed, aborted state, or final state that have existed longer than the `purgeDelay` setting.

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

To optimize performance, the amount of tracking data stored in the run-time database should be kept to a minimum. To help ensure this, the tracking and purge process can be configured to run at regular intervals by an administrator. Additionally, the reporting data policy for a process can be set to *on* or *off*. If on, the data is transmitted to the reporting database if the reporting data stream is enabled; if *off*, the data is not transmitted to the reporting database.

Stored 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 tracking and purging Task data, see System Configuration in *Managing WebLogic Integraton Solutions*.

Task History Tables

As described in the preceding section, worklist task instances generate events that can be logged in worklist history tables in the run-time repository. By default, Task instance information is stored 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 3-7 Data in the Task History Tables

Name	Description
task_id	The unique Task ID.
process_instance	The ID of the business process that created the Task. (Only for Tasks created by business processes.)
action_type	An integer that represents the action recorded. For more information, see Table 3-8.
state_type	An integer that represents the state of the Task. For more information, see Table 3-8.
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 stored, along with the integer that represents the action type or state type in the history tables.

Table 3-8 Integers That Represent Action and State Types in the History Tables

Action Type	Integer
create	0
assign	1
claim	2
suspend	3
resume	4
complete	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

State Type [1]	Integer
assigned	0
claimed	1
started	2
suspended	3
completed	4
aborted	5

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 5-13.

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. BEA recommends using the `WorklistScrollableResultManager Interface` for scalability and performance.

This section includes the following topics:

To Specify the Criteria for a Query

The criteria you specify when you define a query determine the tasks that are returned by the query. When you specify a query, you can set the following criteria:

Table 3-9 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. Optionally, you can specify that the value matches a pattern. For example: `OrderNumber%`. See the Note About String Patterns
Comment	Returns only those tasks for which the comment matches the value passed in. Optionally, you can specify that the value matches a patterns.
Description	Returns only those tasks for which the description matches the value passed in. Optionally, you can specify that the value matches a pattern.
Owners	Returns only those tasks for which the task owner is on the specified list. Optionally, 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. Optionally, you can 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.
Completed Date	Returns only those tasks completed before or after the specified `java.util.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.

Query Criteria	Description
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.

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. Values are sorted in an descending manner.

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 execute a query and return results. The API also provides operations on the com.bea.wli.worklist.api.WorklistManager interface to execute queries. For more information, see the BEA WebLogic Integration Javadoc.

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.

WorklistScrollableResultManager Interface

The `WorklistScrollableResultManager Interface` returns a specified number of records rather than returning all query results. Additionally, it allows you to scroll backward and forward to fetch different ranges of results. This ability improves scalability and performance for querying operations and is the recommended interface. For an example of using the `WorklistScrollableResultManager`, see "Use Case 2" in the *Worklist Sample* in the Solution Samples.

The Relationship Between Processes and Tasks

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

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

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

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

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

- `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 5-13.

CHAPTER 4

Worklist User Interface and Enterprise JavaBeans API

WebLogic Integration Worklist user interfaces (UI) enable end users to interact with running business processes. Generally, people interact with Tasks in WebLogic Integration through custom-user interfaces. A typical example is a mortgage loan processing interface for loan origination, underwriting, closing, funding, and delivery.

With the Enterprise JavaBeans (EJB) API, you can use the Worklist EJB to create and manage tasks independent of the business process (JPD) created in WebLogic Workshop.

This section includes the following information:

- Sample Worklist User Interface
- Samples to Access the Worklist EJB from a Client Application

Sample Worklist User Interface

WebLogic Integration contains a sample Worklist user interface. This sample interface provides basic out-of-the-box functionality for inspecting and updating tasks. The worklist user interface is built as a WLI project that you can open in WebLogic Workshop. The source code is located in the following directory:

BEA_HOME\weblogic81\integration\lib\worklistApp

In the previous line, BEA_HOME represents the directory in which you install the WebLogic Platform.

You can create a custom Worklist user interface using the Worklist controls and the Worklist API. The Worklist controls and API provide the following operations needed to design custom user interface pages:

- 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 to update them.

Additionally, you can use NETUI to leverage the Worklist controls in your custom-user interfaces and use the Worklist API with JavaServer Pages (JSPs).

To learn more about the Worklist API, see the `com.bea.wli.worklist.api` package in the BEA WebLogic Integration Javadoc.

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 Help.

To access the sample Worklist user interface in your installation of WebLogic Platform, take the following steps:

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` and `weblogic`, respectively.

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

   Note: Only the latest 250 tasks are shown on each page. They are displayed in descending order from creation time. If you have more than 250 tasks, the undisplayed tasks will be moved into the list as the displayed tasks are completed and removed from the list.

Samples to Access the Worklist EJB from a Client Application

The worklist public API is in the package `com.bea.wli.worklist.api`. To learn more about the Worklist API, see this package in the BEA WebLogic Integration Javadoc.

The following sample code shows you how to access the Worklist interfaces from a client application.

1. Define a standard method to create an initial context:

```
protected Context getInitialContext(String url, String user, String
password)
      throws Exception
   {
      Properties h = new Properties();
      h.put(Context.INITIAL_CONTEXT_FACTORY,
         "weblogic.jndi.WLInitialContextFactory");
      h.put(Context.PROVIDER_URL, url);
      h.put(Context.SECURITY_PRINCIPAL, user);
      h.put(Context.SECURITY_CREDENTIALS, password);
      h.put(Context.SECURITY_AUTHENTICATION, "simple");
      return new InitialContext(h);
   }
```

2. Create the Worklist EJB:

Alternative 1—using the `WorklistManager`

```
public WorklistManager getWorkListManager() throws Exception {
   RemoteWorklistManagerHome home;
   Object ref;
   Context ctx = getInitialContext("t3://localhost:7001",
      "installadministrator", "installadministrator");
   try {
      // Obtain a reference to the RemoteWorklistItemManagerHome
      ref = ctx.lookup("RemoteWorklistManagerBean");

   // Cast object returned by the JNDI lookup to the appropriate datatype
   home = (RemoteWorklistManagerHome) PortableRemoteObject.narrow(ref,
      RemoteWorklistManagerHome.class);
```

```
   // Use the home interface to create a new instance of the bean.
   return(WorklistManager) PortableRemoteObject.narrow( home.create(),
      RemoteWorklistManager.class);
   } finally {
     ctx.close();
   }
}
```

Alternative 2—using the `WorklistScrollableResultManager`

```
public WorklistScrollableResultManager
getWorklistScrollableResultManager()
throws Exception {
   RemoteWorklistManagerHome home;
   Object ref;
   Context ctx = getInitialContext("t3://localhost:7001",
      "installadministrator", "installadministrator");
   try {
      // Obtain a reference to the RemoteWorklistManagerHome
      ref = ctx.lookup("RemoteWorklistManagerBean");
     // Cast object returned by the JNDI lookup to the appropriate datatype
      home = (RemoteWorklistManagerHome) PortableRemoteObject.narrow(
         ref, RemoteWorklistManagerHome.class);
      // Use the home interface to create a new instance of the bean.
     return (WorklistScrollableResultManager) PortableRemoteObject.narrow(
         home.create(), RemoteWorklistManager.class);
   } finally {
      ctx.close();
   }
}
```

3. To get an instance of the `WorkSubstituteManager`, you do exactly the same thing as you did in step 2:

```
public WorkSubstituteManager getWorkSubstituteManager() throws Exception {
   RemoteWorklistManagerHome home;
   Object ref;
   Context ctx = getInitialContext("t3://localhost:7001",
      "installadministrator", "installadministrator");
   try {
```

```
      // Obtain a reference to the RemoteWorklistItemManagerHome
      ref  = ctx.lookup("RemoteWorklistManagerBean");

      // Cast object returned by the JNDI lookup to the appropriate datatype
      home = (RemoteWorklistManagerHome) PortableRemoteObject.narrow(ref,
         RemoteWorklistManagerHome.class);

      // Use the home interface to create a new instance of the bean.
      return(WorkSubstituteManager) PortableRemoteObject.narrow(
home.create(),
      RemoteWorklistManager.class);}
      } finally {
        ctx.close();
   }
}
```

Caution: You may be tempted to use the remote `WorklistManager` instance directly. This is generally not a good idea because the `RemoteWorklistManager Interface` extends both the `WorkSubstituteManager` and `WorklistManager` interfaces. This interface contains specific methods that are not intended to be public and therefore are not supported by BEA. If used incorrectly, these methods may hamper the Worklist system.

Now that you know how to access the Worklist EJB API, here are some short examples:

This following method returns all the task IDs of the tasks whose name starts with an “a”.

```
   public String[] getIds() throws Exception {
      try {
         WorklistManager wm = getWorklistManager();
         TaskSelector selector = new TaskSelector();
         selector.setTaskName("a%", true);
         return wm.getTaskIds(selector);
      } catch (Exception ex) {}
   }
```

Note: To learn more about selection patterns, see the TaskSelector in the BEA WebLogic Integration Javadoc.

Note: To specify a range of results that are returned from a query, use the `WorklistScrollableResultManager Interface`. For more information, see “WorklistScrollableResultManager Interface” on page 3-27.

The following example shows you how to create a task, set a request message XML, assign it to the user Joe, and return the `taskId`.

```
public String createTestTask() throws Exception {
   try {
      WorklistManager wm = getWorklistManager();
      String id = wm.createTask("test");
      XmlObject request = XmlObject.Factory.parse("<TaskRequest/>");
      wm.setTaskRequestAsXmlObject(request, "xml/test", id);
      wm.assignToUser("joe", id);
      return id;
      } catch (Exception ex) {}
   }
```

CHAPTER 5

Advanced Topics

This section provides information about the following topics:

- Extending Worklist Controls
- Querying Tasks Using the Task Worker Control
- Using Task Control Factories
- Using Multiple Authenticators

Extending Worklist Controls

This section contains information about extending the Task and Task Worker controls. It contains information on the following topics:

- About Extending Worklist Controls
- An Example of an 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 Control
- State Related Updates Using the Task Worker Control

About Extending Worklist Controls

In WebLogic Workshop, when a new Task or Task Worker control is created from the **Data Palette**, it provides a standard interface for 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 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 your application.
- New methods can be added to incorporate several different task modifications into a single method.
- Additional callbacks can be added to detect different types of state changes within a business process.

An Example of an Extended Task Control

The following listing shows a customized Task control 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 destinationAddress);

     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.

Because 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 either when these values are set or when the values already set are returned.

The method parameters that 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 must cast the value that is stored in the system, a mismatch causes an exception to be thrown.

The application can determine the type of data stored in the Request or Response using the request type and the response type data values. You can also design an application to use this information in other ways.

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

- This method sets the request and the response as different data types:

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

- This shows 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 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 incorporate several update steps into a single operation.

To do this, first define your method, passing the parameters needed to create or update the task. Then ensure that the method annotations and attributes specify to the Worklist system which aspects of the task to alter using each parameter.

You can use constants 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 `@jc:task-create` annotation. The `@jc:task-create` annotation contains attributes that specify the way or ways in which the new task is to be configured. Each attribute sets a particular aspect of the task instance, as shown in the following table.

Table 5-1 jc:task-create Attributes to Use For Creating and Configuring Tasks

Aspect of Task to Modify	Attribute	Parameter Type	Notes
Name	name	String	Required attribute. The `name` attribute is required when you create new tasks; other attributes are optional. To specify a parameter to use as input to an attribute, enter the name of the parameter in curly braces.
Description	description	String	
Comment	comment	String	
Request Type	request-mime-type	String	Use only if you use the request attribute.
Request	request	See "Altering Method Signatures—Request and Response" on page 5-4.	Request content.
Priority	priority	int	
Owner	owner	String	Name of user or group.
canBeReassigned	can-be-reassigned	Boolean	
canBeAborted	can-be-aborted	Boolean	

Aspect of Task to Modify	Attribute	Parameter Type	Notes
canBeReturned	can-be-returned	Boolean	
Claim	claim-due-business-date	String	Business time format. Use this or the claim-due-date attribute.
	claim-due-date	java.util.Date	Use this attribute or the claim-due-business-date attribute.
	claim-user-calendar	String	Name of the user whose calendar to use when specifying a calendar in the method call. Only used with claim-due-business-date.
	claim-calendar	String	Name of the business calendar to use when specifying a calendar in the method call. Only used with claim-due-business-date.

Aspect of Task to Modify	Attribute	Parameter Type	Notes
Completion	completion-user-calendar	String	Name of the user whose calendar to use when specifying a calendar in the method call. Only used with completion-due-business-date.
	completion-calendar	String	Name of the business calendar to use when specifying a calendar in the method call. Only used with completion-due-business-date.
	completion-due-business-date	String	Business time format. Use this attribute or the completion-due-date attribute.
	completion-due-date	java.util.Date	Use this attribute or the completion-due-business-date attribute.

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

- Creates a Task and sets 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);
```

- Create a Task and 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);
```

- Create a Task, and specifies 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. With the exception of name and description, the same attributes are supported for update operations as for create operations on the Task Control, and you use them in the same way. The annotation associated with update methods is the `jc:task-update` annotation.

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

```
/**
 * @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 Control

You can use annotations to configure methods that alter task state. Task controls and Task Worker controls use different annotations. The following table lists the annotations and their attributes to use with state transition operations for Task controls.

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

State Transition	Annotation	Attribute	Parameter Type	Notes
Assignment	jc:task-assign		NA	Assign the task
		user	String	If algorithm is `ToUser`, use a String value. If algorithm is `ToUsersAndGroups`, you can use a String[] value.
		group		If algorithm is `ToUserInGroup`, use a String value. If algorithm is `ToUsersAndGroups`, you 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.

State Related Updates Using the Task Worker Control

You can use annotations to configure methods that alter task state. Task controls and Task Worker controls use different annotations. The following table lists the annotations and their attributes to use with state transition operations for Task Worker controls.

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

State Transition	Annotation	Attribute	Parameter Type	Notes
Assignment	jc:task-assign		NA	Assign the task.
		user	String	If algorithm is `ToUser`, use a String value. If algorithm is `ToUsersAndGroups`, you can use a String[] value.
		group	String	If algorithm is `ToUserInGroup`, use a String value. If algorithm is `ToUsersAndGroups`, you 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, when claiming on behalf of another user.
Delete	jc:task-delete			Delete the task.
Return	jc:task-return			Return the task.
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. To learn about the data values you can set and get for these controls, see "Task Data

Values" on page 3-14. The following examples show annotations you can use with get, set, and remove methods on the controls:

- Get example:

```
/**
 * @jc:task-get-property name="{name}"
 */
 public String getProperty(String name);
```

- Set example:

```
/**
 * @jc:task-set-property name="{name}" value="{value}"
 */
 public void setProperty(String name, String value);
```

- Remove example:

```
/**
 * @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 changes, or *events*. You can implement callback methods only on Task controls—not Task Worker controls—because only Task controls identify with a single active instance of a Task.

Optionally, callbacks can return up to three arguments, as follows:

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

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

```
/**
 * @jc:task-event event-type="complete"
```

```
*    response="{response}"
*    time="{time}"
*    user="{user}"
*/
void onTaskCompleted(XmlObject response, Date time, String user);
```

To specify the state transition that triggers the callback, set the `event-type` attribute. Use the appropriate value for the type of event that triggers your method. The following table describes the types of events and the associated value.

Table 5-4 Event Types and event-type Attribute

Type of Event	Values for event-type Attribute
ABORT	abort
ASSIGN	assign
CLAIM	claim
CLAIM_EXPIRE	claimExpire
COMPLETE	complete
CREATE	create
EXPIRE	expire
RESUME	resume
RETURN	return
START	start
STOP	stop
SUSPEND	suspend

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. It contains the following topics:

- 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 attributes that you can use to query with the `@jc:select` annotation.

Table 5-5 Task Control Attributes to Use with @jc:select

Search Attribute	Description
`assigned-group`	Search by groups in the Assignee List for a Task.
`assigned-user`	Search by users in the Assignee List for a Task.
`claimant`	Search by the claimant for a Task.
`claim-due-date-after`	Search by Tasks with a claim 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.

Search Attribute	Description
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.
selector	Search by the configuration of the TaskSelector object that you provide for this value. To learn more about using this value to pass arguments to TaskSelector objects, see Querying Tasks With TaskSelectors.
states	Search by Tasks by state. Values can be as follows: • A String or String array of valid state types, such as completed or assigned. • An Integer or Integer array representation of state types. • 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 Assignees List for a Task.

Querying Tasks With Annotations

To provide a method for your custom query, you must extend the Task Worker control. 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[] getTasksInfoXML(String[] taskIds);
```

The `TasksInfoXML` method allows you to search through Tasks by Task ID. You can extend this control to search through Tasks by an additional attribute, such as the claimant. To do so, you can take advantage of the extensibility built into the `TasksInfoXML` method, which allows you to pass additional arguments to the method, as shown in the following example.

In the following example, the previous example code is extended to query for a claimant. The bold text indicates the additions made to the example.

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

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[] getTasksInfoXML(String user,String[]
taskIds);
```

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

Querying Tasks With TaskSelectors

The Worklist API provides the functionality for creating more advanced search functionality than described in the preceding section, which used the `@jc:select` Java annotation. 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 more.

To use a `TaskSelector`, you must include the `selector` attribute 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.

Using Task Control Factories

There are two circumstances in which you must use a Factory type of Task Control:

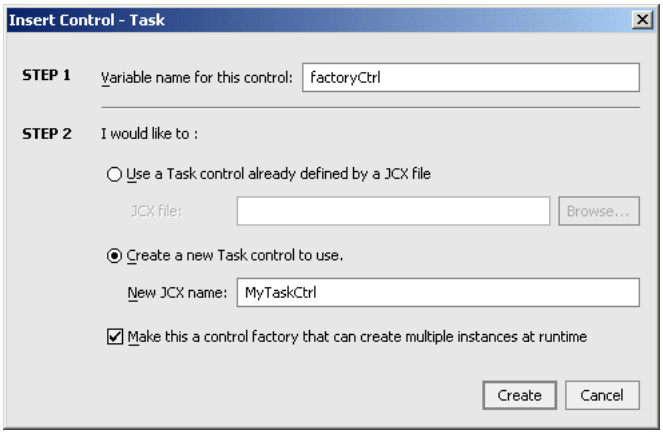
- You are creating multiple tasks in a loop, or you want the number of tasks created to be specified at run time.
- You are interacting with multiple existing tasks in a loop, or with a number of tasks that is not known until run time.

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 that your business process loops over a sequence of order elements that creates a new task to approve each order in the body of the loop. Additionally, the task name changes on each iteration, but the assignee is always the same manager. In this 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, the default assignee will be used.

The basic pattern is to create two Task controls in your business process: one that is a factory type and the other that is not a factory type. Callbacks for tasks created using factory-created task control instances will be received by the factory control callback handler. 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.
   a. In **Step 1**, name the Task control instance: `factoryCtrl`.
   b. In **Step 2**, select **Create a new Task control to use**, the enter **MyTaskCtrl** into the **New JCX name** field.
2. Make this control a factory that can create multiple instances at run time.



3. Create another Task Control from the **Data Palette**.

   c. Name the Task control instance: `myTaskCtrl`.

   d. 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.

   e. Do *not* select the option to make this control a control factory.

   The control's 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;
```

   Note: To view the code for the control's variables, click the **Source View** tab for the business process, then in **Document Structure**, double-click **factoryCtrl**.

4. Click the **Design View** tab to return to the business process.

5. 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**. You create this logic in a Perform node as follows:

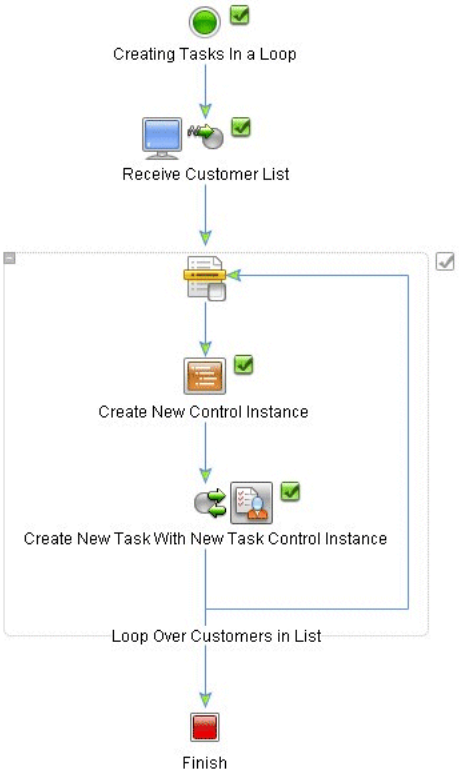
   a. Right-click the **Perform** node, then select **View Code**.

b. The code for the Perform node should resemble the following:

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

6. From the **Data Pallet**, select a method from **myTaskCtrl**, then drag it onto the business process placing it below the **Perform** node.

   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:



Using Multiple Authenticators

This section contains information about configuring your security realm to allow task assignment across users and groups when your WebLogic Integration domain is configured with multiple authenticators. An example of this is when the domain uses the default WebLogic Integration embedded LDAP and Netscape iPlanet.

When a security realm has multiple authentication providers configured, WebLogic Server uses the Java Authentication and Authorization Service (JAAS) classes to authenticate. Specifically, the control flag attribute on each authenticator determines the order of execution. The JAAS documentation (http://java.sun.com/j2se/1.4.2/docs/api/javax/security/auth/login/Configuration.html) provides the following information about this control flag:

1. Required—The LoginModule is required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

2. Requisite—The LoginModule is required to succeed. If it succeeds, authentication continues down the LoginModule list. If it fails, control immediately returns to the application (authentication does not proceed down the LoginModule list).

3. Sufficient—The LoginModule is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed down the LoginModule list). If it fails, authentication continues down the LoginModule list.

4. Optional—The LoginModule is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

The overall authentication succeeds only if all Required and Requisite LoginModules succeed. If a Sufficient LoginModule is configured and succeeds, then only the Required and Requisite LoginModules prior to that Sufficient LoginModule need to have succeeded for the overall authentication to succeed. If no Required or Requisite LoginModules are configured for an application, then at least one Sufficient or Optional LoginModule must succeed.

Note: LoginModules are the work-horses of authentication: all LoginModules are responsible for authenticating users within the security realm and for populating a subject with the necessary users and groups. LoginModules that are not used for perimeter authentication also verify the proof material submitted, such as a user's password.

If you have two user communities with exclusive user logins, avoid using *Required* and *Requisite* as values unless the authenticator for any of these settings can authenticate a superset of all user communities. Use the WebLogic Server Administration Console to set the JAAS control flags.

For information on how to set these flags, see the *WebLogic Server Administration Console Online Help* at http://e-docs.bea.com/wls/docs81/ConsoleHelp/index.html.

For more information about using multiple authenticators and external LDAP servers, see the following:

- "Security Fundamentals" in *Introduction to WebLogic Security*, which is available at the following URL:

  http://e-docs.bea.com/wls/docs81/secintro/concepts.html

- "Setting the JAAS Control Flag Attribute" in "Configuring Security Providers" in *Managing WebLogic Security*, which is available at the following URL:

  http://e-docs.bea.com/wls/docs81/secmanage/providers.html

- "Using an External LDAP Server" in *Security in WebLogic Platform 8.1*, which is available at the following URL:

  http://e-docs.bea.com/platform/docs81/secintro/user.html

Index

T

U

V

W

X