Oracle® Application Server Containers for J2EE

Job Scheduler Developer's Guide

10*g* Release 3 (10.1.3)

B15876-01

January 2006

ORACLE®

Oracle Application Server Containers for J2EE Job Scheduler Developer's Guide, 10*g* Release 3 (10.1.3)

B15876-01

Primary Author: Kevin Yu Hwang

Contributors: Gary Moyer, Tony D'Silva

Contents

10 Managing the Oracle Application Server Containers for J2EE

A RFC 2445 Excerpt: Recurrence

B Oracle Application Server Containers for J2EE Semantics

C JSP Tag Library Reference

D JMX MBean Reference

E Troubleshooting Oracle Application Server Containers for J2EE

Index

List of Examples

List of Figures

List of Tables

Preface

This guide describes how to use the Oracle Application Server Containers for J2EE, and how to configure Job Scheduler-enabled applications for deployment.

Audience

This guide is intended for anyone developing Enterprise JavaBeans for OC4J client applications. Written especially for programmers, it will also be of value to architects, systems analysts, project managers, and others interested in J2EE applications. To use this guide effectively, you must have a working knowledge of J2EE.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documentation

For more information, see the following guides in the Oracle Containers for J2EE 10*g* Release 10.1.3 documentation set:

- *Oracle Containers for J2EE Services Guide*
- *Oracle Containers for J2EE JSP Tag Libraries and Utilities Reference*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Overview of Oracle Application Server Containers for J2EE

Oracle Application Server Containers for J2EE enables J2EE clients to submit asynchronous, autonomous background jobs to be run in Oracle Application Server Containers for J2EE (OC4J). Some major features of this component are:

- API for submitting, controlling, and monitoring jobs
- API-level Java Transaction API (JTA) support for job submission and control
- Temporal- and trigger-based jobs
- Automatic retry of failed jobs
- Job blackout windows
- Java Management Extensions (JMX) MBeans for monitoring and administering Oracle Application Server Containers for J2EE
- Configurable logging of various system, error, and warning messages

This chapter provides an overview of Oracle Application Server Containers for J2EE. The following topics are covered:

1.1 Job Scheduler Concepts and Terminology

This section introduces basic concepts and terminology you should know before using Job Scheduler. The following topics are covered:

1.1.1 Notifications and Triggers

A notification is a message sent from the application to a trigger. The message contains information about the occurrence of a specific condition or conditions in the application.

The recipient of a notification is known as the trigger. Each trigger contains a description of a condition that is evaluated against any incoming notifications. When a

trigger (or multiple triggers) evaluates to true, a job associated with the condition is started.

Triggers are described by logical expressions, where the operands are the notifications. Notifications are generated either programatically by the application or as a result of a timer expiration (for example, if a job does not execute by a specific time). A notification may be sent to either a specific trigger or sets of triggers. Triggers, however, do not generate notifications when they receive notifications.

For more information, see Chapter 8.

1.1.2 Schedules

A schedule specifies the time and period or periods when a timeout notification is sent to the associated trigger. The specific point in time is expressed as the expiration, and the frequency is expressed as an interval. A schedule instance can be categorized as either single-action (non repeating) or repeating.

For more information about schedules, please refer to Chapter 3.

1.1.3 Jobs

For every job, there is an associated trigger. When the trigger expression evaluates to true, the job is executed. A job is implemented using a Java class and must comply with the job contract, which is a system-supplied Java interface implemented by all jobs. This contract specifies the interface used by Job Scheduler to run the job.

1.1.4 Blackout Windows

A blackout window specifies a period of time during which all jobs are suppressed. A blackout window contains a schedule and a duration (for example, Friday between 6:00 p.m. and 12:00 a.m.). A blackout window may also be repeating (for example, every Tuesday between 6:00 p.m. and 8:00 p.m.).

For more information, see Chapter 4.

1.2 Basic Job Scheduler Implementation Example

In this example, the application developer wants to create a report that can be run periodically according to the application's needs. The application will submit requests to run the report, supplying some query input parameters and stating how often to run the report.

1. The developer writes a class that implements the Job Scheduler `Executable` interface, using the `execute()` and `getContext()` methods. The `execute()` method is written by the developer and will be called by Job Scheduler. The application calls the system-supplied `getContext()` method to get the input parameters.

   For more information about adding jobs, see Chapter 2.

2. The developer writes a client program for the application through which a user can submit a report request, cancel a report request, or check to see what requests were made. To service requests, the client program gets a reference to a Scheduler Enterprise Java Bean (EJB) deployed with the application (the Job Scheduler is deployed as a stateless session EJB). Note that the client program could be implemented as a standalone GUI, a servlet, or indirectly through another EJB.

For more information about deploying Job Scheduler-enabled applications, see Chapter 9.

a. To submit a job, the program parameters are specified using the `java.util.Properties` class. Then, the `Scheduler.add()` method is used to submit the job to be run at a particular time.

b. To find out what jobs were submitted, use the `Scheduler.getJobs()` method. Both pending and completed jobs are displayed.

c. To remove a job, use the `Scheduler.remove()` method. This action terminates any future scheduling of a job.

d. To cancel jobs that are running, use the `Scheduler.cancel()` method.

For more information about canceling jobs, see Chapter 6.

3. The developer packages the application, including:

a. The classes described previously.

b. An EJB JAR file referencing Job Scheduler EJB. This is a pre-written, system-supplied EJB that has methods for submitting, querying, and controlling jobs.

c. The client portion of the application.

For more information, see Chapter 9.

2

Adding and Removing Jobs

This chapter describes how to add and remove jobs use Oracle Application Server Containers for J2EE. The following topics are covered:

2.1 Adding Jobs

Before a job can be run, it must first be submitted to Job Scheduler.

To add a job, you must implement the `oracle.ias.scheduler.Executable` interface, then submit the job to Job Scheduler using the `oracle.ias.scheduler.Scheduler.add()` API method.

For more information about `add()`, see *Oracle Containers for J2EE Job Scheduler API Reference*.

2.1.1 Implementing a Job with the Executable Interface

The `oracle.ias.scheduler.Executable` interface is defined as follows:

```
public interface Executable {
   public void execute (JobContext context) throws JobExecutionException,
JobCancellationException;
}
```

This interface specifies the contract by which a Java class is invoked by the Job Scheduler. All Java classes submitted to Job Scheduler must implement this interface.

> **Note:** Any class implementing this interface must provide an empty constructor. Each time a submitted job is run, a new instance of the object is created using this constructor. As such, a job implementation should not rely on instance or static member variables for maintaining state.

The `execute()` method is invoked by Job Scheduler when the associated job's trigger fires. Use the `oracle.ias.scheduler.JobContext` object as the input parameter

to enable a job to examine and evaluate all associated metadata related to the job definition and access to the logging subsystem.

The `oracle.ias.scheduler.JobContext` object provides the following methods

- `getLogger()`

  This method returns a JDK 1.4-compliant logger object `java.util.logging.Logger` that references the application's log.

- `getJob()`

  This method returns an `oracle.ias.scheduler.Job` object used to access the job's configuration information.

The `oracle.ias.scheduler.JobContext` object provides access to the job's data and associated subsystems. For more information, refer to *Oracle Containers for J2EE Job Scheduler API Reference*.

2.1.2 Submitting a Job

For a job to run, it must first be submitted to Job Scheduler. This is done using the `oracle.ias.scheduler.Scheduler.add()` method. As part of the submission, input parameters may be specified as name-value pairs using a `java.util.Properties` object. For maintenance and reusability purposes, job parameters should be used whenever possible (see Example 2–1).

With the `oracle.ias.scheduler.Scheduler.add()` method, you can add a job with a schedule, a trigger, or both.

Once the job is submitted, the specified class is executed by Job Scheduler according to the specified schedule or trigger. If the schedule does not repeat, the job becomes inactive after the timer expiration notification is sent to the associated trigger.

When a job is successfully submitted, an `oracle.ias.scheduler.JobHandle` object is returned. This object functions as a handle to the submitted job. This handle may be used to perform certain administration tasks on the job (for example, pausing the job). Additionally, this object may be stored by the application for later use.

The `add()` method provides transaction support. If the transaction is rolled back for any reason, the operation is canceled and the job is not created. In addition, a job will not run until the transaction is committed.

2.1.3 Examples of Adding Jobs

This section provides examples of how to implement and submit a job to Job Scheduler.

Example 2–1 Implementing a Job to Perform Backups

Scenario: A legacy application was migrated to the J2EE environment, part of which includes data stored in a file system. As part of the J2EE application, a job is required to back up the data on a regular basis. The job requires two input parameters:

1. Source directory: the directory from which files will be copied
2. Destination directory: the directory to which files will be copied

The source and destination directories could have been included in the job implementation. However, by specifying these as parameters to the job properties, the job can be used again without modification. See Example 2–2 for an example of how properties are specified.

```
import java.io.File;
import java.io.IOException;
import oracle.ias.scheduler.Job;
import oracle.ias.scheduler.Executable;
import oracle.ias.scheduler.JobContext;
import oracle.ias.scheduler.JobExecutionException;


public class BackupJob implements Executable {

    public void execute(JobContext context) throws JobExecutionException {

        // retrieve the source/destination directories
        Job job = context.getJob();
        String source = job.getProperties().getProperty("SourceDirectory");
        String destination =
            job.getProperties().getProperty("DestinationDirectory");

        // get the list of files to copy
        File directory = new File(source);
        File[] files = directory.listFiles();

        // copy the files
        Runtime runtime = Runtime.getRuntime();
        Process process;
        for (int x = 0; x < files.length; x++) {
            try {
                process = runtime.exec("/bin/cp " + files[x].toString() +
                                       " " + destination);
                process.waitFor();
            } catch(IOException e) {
                throw new RuntimeException("copy failed: "+files[x],e);
            } catch(InterruptedException e) {
                throw new RuntimeException("copy failed: "+files[x],e);
            }
        }
    }
}
```

Notice that the `getProperty()` object is used to retrieve the source and destination directories. Instead of specifying these directories directly in the job implementation, they are specified when the job is submitted to Job Scheduler (see Example 2–2).

Using the `execute()` method fulfills the contract for implementing the `oracle.ias.scheduler.Executable` interface. This method is invoked every time the job is executed.

Example 2–2 Specifying Job Properties and Submitting a Job

Once a job is created, it must be submitted to Job Scheduler. A job is submitted using the `add()` method provided by Job Scheduler. The following code example shows how the job is submitted with properties (in this case, the source and destination directories):

```
// set up the properties
java.util.Properties properties = new Properties();
properties.put("SourceDirectory","/mnt/data");
properties.put("DestinationDirectory","/mnt/backup");

// submit the job
```

```
jobHandle = scheduler.add("file backup job, runs every week",
                          new BackupJob().getClass().getName(),
                          new Schedule(),
                          properties);
```

Notice that the job properties and schedule are specified when the job is submitted to Job Scheduler. For more information about specifying scheduling options, see Chapter 3.

2.2 Accessing Job Scheduler Using JNDI Lookup

The Java Naming and Directory Interface (JNDI) is a native Java API that enables any Java-based application to store and retrieve any Java object. It provides naming and directory services for Java applications, enabling them to store and retrieve named Java objects of any type.

The following code example shows how to perform a JNDI lookup to access Job Scheduler:

```
InitialContext ic = new InitialContext();
Object objRef = ic.lookup("java:comp/env/ejb/scheduler");
SchedulerHome home = (SchedulerHome)
PortableRemoteObject.narrow(objRef, SchedulerHome.class);
Scheduler scheduler = home.create();
```

For more information about JNDI, go to:

`http://java.sun.com/products/jndi/index.jsp`

2.3 Removing Jobs

After a job is submitted, it can be removed with the `oracle.ias.scheduler.Scheduler.remove()` method. This method does not remove any job executions that are running, but it does remove the job definition, thus preventing any job executions from being run in the future.

For more information about the `remove()` method, see *Oracle Containers for J2EE Job Scheduler API Reference*.

The following code example shows how to remove the `BackupJob` job implemented in Example 2–1 and submitted in Example 2–2:

Example 2–3 Removing a Job

```
// remove a job
scheduler.remove(jobHandle);
```

If you want to stop a job execution that is currently running, you must cancel the job. For more information about canceling jobs, see Chapter 6.

If you want to stop scheduled jobs from running but do not want to have their definitions removed from the system, thereby preventing them from ever running again, you should pause the job. For more information about pausing jobs, see Chapter 5.

2.4 Best Practices for Adding and Removing Jobs

When designing and implementing a job, keep the following in mind:

- **All job metadata is available at run time.** Use the `oracle.ias.scheduler.JobContext` object to access it.
- **Input parameters can improve job reuse.** During implementation, identify input parameters and use properties as necessary.
- **A job that needs to be canceled must use the oracle.ias.scheduler.Cancellable interface.** Trying to cancel a job that does not use this interface causes an exception.
- **Implementation of the execute( ) method must eventually return control to the caller.** Avoid gating job completion based on an application condition or conditions that require a long time before the condition is met (for example, in excess of one minute). Instead, use a trigger to start the job execution when the application's condition or conditions have ben met. In doing so, you minimize the processing resources required by the application.

2.5 Frequently Asked Questions About Adding and Removing Jobs

What are the possible states for a job?

At any point in time, a job can be in one of three states: active, paused, or completed.

In the active state, a job can receive notifications and evaluate the trigger expression. If the trigger evaluates to true in the active state, the job is executed and the trigger is reset to false.

In the paused state, a job can receive notifications and evaluate trigger expressions. However, if a trigger evaluates to true in the paused state, the trigger is not reset to false and the job execution is suppressed.

In the completed state (which is a valid state for schedule-based jobs), scheduling of jobs is completed and no new jobs can be scheduled. If a job is not schedule-based, then it cannot be in the completed state. For more information about schedule-based jobs, see Chapter 3.

Does removing a job also remove the outstanding retry (of a job that failed to run) and replay (of a paused job) executions?

Yes, removing a job means no job executions will occur.

For more information about retry, see Chapter 3. For more information about replay, see Chapter 5.

3

Oracle Application Server Containers for J2EE Scheduling Options

This chapter describes how to create jobs based on schedules. The following topics are covered:

- Schedule-Based Jobs
- Retry Period and Execution Threshold
- Frequently Asked Questions About iCalendar and Execution Threshold

3.1 Schedule-Based Jobs

This section contains descriptions and some job implementation examples for schedule-based jobs.

A schedule-based (or schedule-driven) job is associated with a schedule, meaning that the job is time-based. In contrast, a job associated with a trigger is event-based and typically driven by events initiated by the application. When a schedule expires, a timeout is generated, which is used to trigger the execution of the job.

There are two primary types of schedule-based jobs:

- Single-action schedules.

  This type of schedule has a single expiration, and should be used when a job is run only once.

- Repeating Schedules.

  This type of schedule has multiple expirations, and should be used when a job is run repeatedly.

3.1.1 Single-Action Schedules

Single-action schedules are implemented with the `oracle.ias.scheduler.Schedule` class. This type of schedule has a single attribute called `expiration`, which is the initial expiration of the schedule.

Example 3–1 Submitting a Job at a Specific Time

In continuing the example started in Example 2–1, the developer and administrator need to run the backup jobs on an as-needed basis. To do this, a single-action schedule will be used. The following code example shows how the job is set up with a single-action schedule and submitted:

```
// set up the properties
```

```
java.util.Properties properties = new Properties();
properties.put("SourceDirectory","/mnt/data");
properties.put("DestinationDirectory","/mnt/backup");

// set up the schedule, single-action at midnight
Schedule schedule = new Schedule();
Calendar midnight = Calendar.getInstance();
midnight.set(Calendar.HOUR_OF_DAY,24);
midnight.set(Calendar.MINUTE,0);
midnight.set(Calendar.SECOND,0);
schedule.setExpiration(midnight);

// submit the job
scheduler.add("file backup job, runs at midnight",
              new BackupJob().getClass().getName(),
              schedule,
              properties);
```

3.1.2 Repeating Schedules

There are three types of repeating schedules:

- Fixed-interval schedule

  This type of schedule uses the `oracle.ias.scheduler.IntervalSchedule` class for repeating jobs with a fixed interval (for example, a job that runs once per week, every friday at midnight).

- Fixed-delay schedule

  This type of schedule uses the `oracle.ias.scheduler.IntervalSchedule` class for repeating jobs with a fixed interval between job executions (for example, a job where the end of one job execution and the start of the next job execution is one week).

- iCalendar recurrence schedule

  This type of schedule uses the `oracle.ias.scheduler.RecurSchedule` class for repeating jobs with a schedule that does not repeat at regular intervals (for example, the first day of every month, which is not a fixed interval because the number of days in each month varies).

3.1.2.1 Fixed-Interval Schedules

A fixed-interval schedule has the following attributes:

Attribute	Description
`expiration`	Initial expiration
`interval`	Interval (specified in milliseconds) between expirations
`end date`	Date and time at which the schedule ends

Example 3–2 Submitting a Repeating Job with a Fixed-Interval Schedule

To expand on Example 3–1, suppose the developer and administrator need to run the backup job on a weekly basis. To do this, a fixed-interval repeating schedule will be used, as shown in the following code example:

```
// set up the properties
java.util.Properties properties = new Properties();
properties.put("SourceDirectory","/mnt/data");
```

```
properties.put("DestinationDirectory","/mnt/backup");

// set up the schedule, repeats every week
IntervalSchedule schedule = new IntervalSchedule();
schedule.setInterval(IntervalSchedule.EVERY_WEEK);

// submit the job
scheduler.add("file backup job, runs at midnight",
              new BackupJob().getClass().getName(),
              schedule,
              properties);
```

3.1.2.2 Fixed-Delay Schedules

To expand on Example 3–1, suppose the developer and administrator need to run the backup job on a more regular basis (for example, one week between each backup). To do this, a fixed-delay repeating schedule will be used, as shown in the following code example:

Example 3–3 Submitting a Repeating Job with a Fixed-Delay Schedule

```
// set up the properties
java.util.Properties properties = new Properties();
properties.put("SourceDirectory","/mnt/data");
properties.put("DestinationDirectory","/mnt/backup");

// set up the schedule, repeats every week
IntervalSchedule schedule = new IntervalSchedule();
schedule.setInterval(IntervalSchedule.EVERY_WEEK);
schedule.setFixedDelay(true);

// submit the job
scheduler.add("file backup job, runs at midnight",
              new BackupJob().getClass().getName(),
              schedule,
              properties);
```

3.1.2.3 iCalendar Recurrence Schedules

The attributes for an iCalendar recurrence schedule are based on RFC 2445, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)." For more information, see Appendix A.

Example 3–4 Submitting a Repeating Job with an iCalendar Recurrence Schedule

To expand on Example 3–1, suppose the developer and administrator need to run the backup job on a monthly basis, on the first of each month. To do this, an iCalendar recurrence schedule will be used, as shown in the following code example:

```
// set up the properties
java.util.Properties properties = new Properties();
properties.put("SourceDirectory","/mnt/data");
properties.put("DestinationDirectory","/mnt/backup");

// set up the schedule, repeats on the first day of every month
RecurSchedule schedule = new RecurSchedule("freq=monthly;bymonthday=1;");

// submit the job
scheduler.add("file backup job, runs at midnight",
              new BackupJob().getClass().getName(),
              schedule,
```

```
                    properties);
```

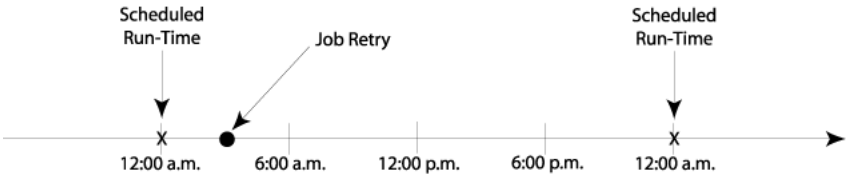
3.2 Retry Period and Execution Threshold

This section discusses retry period and execution threshold, and provides an example of each in relation to a scheduled job.

3.2.1 Retry Period

A job execution that fails may be retried after a time period. This time period is called the retry period and is specified in milliseconds. If this period is not set as part of the job definition, the job's executions will not be retried. For example, consider Figure 3–1:

Figure 3–1 Retry Period

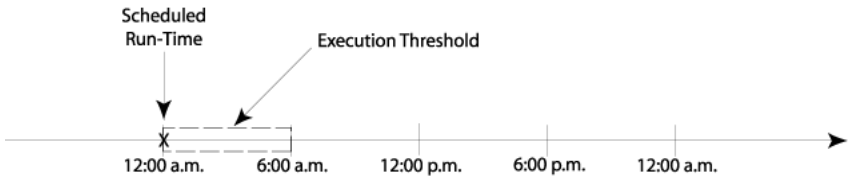


This illustration depicts a repeating schedule, where the job should run each night at midnight. The retry period is three hours, meaning that if the initial job execution fails, the job will be retried again three hours later, at 3:00a.m. If the job execution fails again at 3:00a.m., the job execution for this particular instance is discarded, and another attempt will not be made until the next scheduled run-time (in this case, midnight the following night).

3.2.2 Execution Threshold

If a job's scheduled execution is delayed beyond a specified time threshold, then the job execution will be discarded. This time threshold is called the execution threshold and is specified in milliseconds. If an execution threshold is not specified as part of job's definition, resultant job executions will not be constrained by an execution threshold. For example, consider Figure 3–2:

Figure 3–2 Execution Threshold



The job execution is scheduled to run at midnight on Monday, and the execution threshold is 6 hours. The job execution would be discarded if it did not run by 6:00a.m. Monday.

3.2.3 Submitting a Job with a Retry Period and Execution Threshold

To further expand on Example 3–1, a retry period and execution threshold were added to Example 3–5. In this example, if the job's executions do not occur within 30 seconds

(execution threshold) of the scheduled time, they will be discarded. If the executions do occur, but fail, they will be retried 3 seconds (retry period) later.

Example 3–5 Submitting a Job with a Retry Period and Execution Threshold

```
// set up the properties
java.util.Properties properties = new Properties();
properties.put("SourceDirectory","/mnt/data");
properties.put("DestinationDirectory","/mnt/backup");

// set up the schedule, repeats every week
IntervalSchedule schedule = new IntervalSchedule();
schedule.setInterval(IntervalSchedule.EVERY_WEEK);

// submit the job
scheduler.add("file backup job, runs at midnight",
              new BackupJob().getClass().getName(),
              schedule,
              properties,
              3000,
              30000);
```

3.3 Frequently Asked Questions About iCalendar and Execution Threshold

Does Job Scheduler check the execution threshold if the job trigger is something other than a timeout?

No. Because the execution threshold is based on time, notifications other than timeouts do not cause Job Scheduler to check the execution threshold.

Can I update the execution threshold or retry period for a job?

Currently, this is not possible, because a job is configured with these parameters at creation time.

Can a fixed-delay schedule be submitted in conjunction with a trigger in a job definition?

No. The period for a fixed-delay schedule is based on the completion of the previous job execution. When a trigger is used, this period cannot be determined because it is dependent on the receipt of one or more notifications as specified by the trigger.

4

Oracle Application Server Containers for J2EE Blackout Windows

This chapter describes how to create and remove blackout windows. The following topics are covered:

- Adding and Removing Blackout Windows
- Jobs Scheduled in Blackout Windows
- Frequently Asked Questions About Blackout Windows

4.1 Adding and Removing Blackout Windows

A blackout window is a period of time during which job executions are not permitted. A blackout window should be used when the system or dependent subsystems are unavailable for a pre-determined amount of time (for example, when the database is down for scheduled maintenance).

To create or add a blackout window, use the `oracle.ias.scheduler.Scheduler.addBlackoutWindow()` method, which is defined as follows:

```
public void addBlackoutWindow(java.lang.String windowName,
                              Schedule schedule,
                              long duration)
                       throws DuplicateWindowException,
                              InvalidWindowException,
                              java.rmi.RemoteException
```

This method provides the parameters described in Table 4–1.

Table 4–1 addBlackoutWindow Parameters

Parameter	Description
`windowName`	Name of the blackout window.
`schedule`	Start time of the blackout window. Note that the schedule may be a repeating schedule.
`duration`	Duration (in minutes) of the blackout window.

For more information about the `addBlackoutWindow()` method, see *Oracle Containers for J2EE Job Scheduler API Reference*.

Example 4–1 shows how to create a blackout window called "Not in prime time", which is in effect from 8 a.m. to midnight:

Example 4–1 Adding a Blackout Window

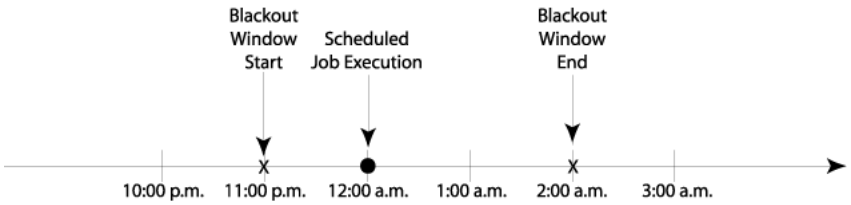
```
// set up the schedule, daily starting at 8 a.m.
Schedule schedule = new Schedule();
Calendar blackoutWindowStartTime = Calendar.getInstance();
blackoutWindowStartTime.set(Calendar.HOUR_OF_DAY,8);
blackoutWindowStartTime.set(Calendar.MINUTE,0);
blackoutWindowStartTime.set(Calendar.SECOND,0);

// create blackout window
scheduler.addBlackoutWindow("Not in prime time", schedule, 960);
```

4.2 Jobs Scheduled in Blackout Windows

If a job execution occurs when a blackout window is in effect, the job execution is suppressed. If the job was submitted with a retry period enabled, then the job execution will be retried at the period specified.

Figure 4–1 Jobs Scheduled in a Blackout Window with Retry Period Enabled



In Figure 4–1, a job execution is scheduled to occur at midnight, which falls in a blackout window starting at 11:00 p.m. and ending at 2:00 a.m. If this job had a retry period enabled, the job execution would be retried at 2:00 a.m., when the blackout window ends. Without the retry period enabled, the jobs execution would be suppressed and no attempt to run the job would be made.

Blackout windows takes the highest precedence among all Job Scheduler operations. For more information about job precedence, see Appendix B.

4.3 Frequently Asked Questions About Blackout Windows

What happens if a job is scheduled to be retried (in the event the job fails to run) or replayed (in the event the job is paused) in a blackout window?

The retry (or replay) of the job is suppressed. In other words, the job execution will not occur.

5

Pausing Jobs

This chapter describes what it means to pause a job and how to pause a job. The following topics are covered:

- What Does It Mean to Pause a Job?
- How to Pause a Job
- Frequently Asked Questions About Pausing Jobs

5.1 What Does It Mean to Pause a Job?

Pausing a job causes a scheduled job execution to be skipped. It does not stop a job execution that is currently running (to do this, you must cancel the job). Pausing a job does not remove the job's definition from the system, thus preventing the job from running in the future (to do this, you must remove the job). Use the `oracle.ias.scheduler.Scheduler.pause()` method to pause a job.

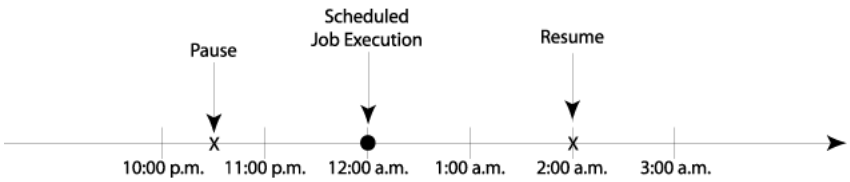
For more information about canceling jobs, see Chapter 6. For more information about removing jobs, see Chapter 2.

For more information about the `pause()` method, see *Oracle Containers for J2EE Job Scheduler API Reference*.

A job execution that was skipped because it was paused can be run again by resuming the job with the replay parameter set to true. Use the `oracle.ias.scheduler.Scheduler.resume()` method to resume a job.

To illustrate more clearly the effect of pausing and resuming a job, consider the following timeline in Figure 5–1.

Figure 5–1 Pausing and Resuming a Job with a Single-Action Schedule



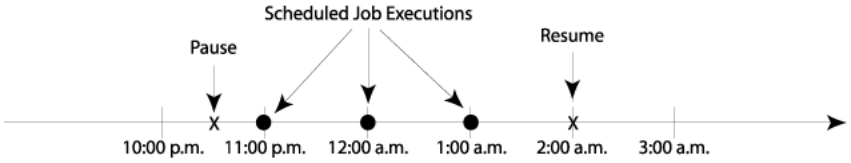
There is a pause implemented for a particular job at around 10:30 p.m., and it is scheduled to resume at 2:00 a.m. However, this job is scheduled to run at 12:00 a.m. Will this job run?

Due to the pause at 10:30 p.m., the job execution scheduled to run at midnight will be skipped. However, if the job is resumed at 2:00 a.m. with replay set to true, then the job

execution scheduled to run at midnight will run at 2:00 a.m. If replay is set to false, then the job execution scheduled to run at midnight will not run.

In contrast, consider Figure 5–2, which illustrates the effect of pausing and resuming on a job with a repeating schedule.

Figure 5–2 Pausing and Resuming a Job with a Repeating Schedule



In this scenario, there is a job with a repeating schedule (multiple job executions scheduled) that falls between pause and resume. If the job is resumed at 2:00 a.m. with replay set to true, then only the first job execution (the one at 11:00 p.m.) will run. The job executions at midnight and 1:00 a.m. will be skipped.

5.2 How to Pause a Job

This section shows some code examples of how to pause and resume a job.

Example 5–1 Pausing a Job

This example shows how to use the `oracle.ias.scheduler.Scheduler.pause()` method to pause the job called `BackupJob`.

```
//pause the "BackupJob" job
scheduler.pause(jobHandle)
```

Example 5–2 Resuming a Job Without Replay

This example shows how to set replay to false, so that a job execution that was skipped will not be run again.

```
//resume the "BackupJob" job without replay
scheduler.resume(jobHandle, FALSE)
```

Example 5–3 Resuming a Job with Replay

This example shows how to set replay to true, so that a job execution that was skipped will be run again.

```
//resume the "BackupJob" job with replay
scheduler.resume(jobHandle, TRUE)
```

5.3 Frequently Asked Questions About Pausing Jobs

What happens if you pause a job that is currently running?

Pausing a job that is currently running does not interrupt the job. However, pausing a job prevents the job from running in the future until it is resumed.

At execution time, what is the difference between a blackout window and a paused job?

A blackout window suppresses all job executions while a paused job suppresses only job executions that result from a trigger being fired. For example, pausing a job would not suppress replayed or retried job executions, but running a job in a blackout window would.

What happens if retry is attempted when a job is paused?

A paused job suppresses any job executions that occur due to a trigger expression evaluating to true. During a retry, the trigger expression is never evaluated and the job is allowed to run.

6

Canceling Jobs

This chapter describes the meaning of canceling a job and also describes how to cancel a job. The following topics are covered:

- What Does it Mean to Cancel a Job?
- Canceling a Job
- Frequently Asked Questions

6.1 What Does it Mean to Cancel a Job?

Canceling a job is the only way to stop a job execution while it is running. It is important to note that while canceling a job stops a job execution that is currently running, it does not prevent that job from being executed in the future. To eliminate all traces of a job from the system, you must remove the job. For more information about removing jobs, see Section 2.3.

Once a job is canceled, it is possible to run the job again by using retry. For more information about retry, see Section 3.2.

6.2 Canceling a Job

To cancel a job, the job must use the `oracle.ias.scheduler.Cancellable` interface, respond to the cancellation request, and then create the necessary exception, `JobCancelledException`, to designate the job as canceled.

Any Java job class submitted to Job Scheduler can provide an implementation of the `oracle.ias.scheduler.Cancellable` interface. Every time a job is canceled (by invoking the Job Scheduler `cancel()` method), it causes the implementing class's `cancel()` method to be invoked on all the job's instances.

For more information about the `oracle.ias.scheduler.Cancellable` interface or `cancel()` method, see *Oracle Containers for J2EE Job Scheduler API Reference*.

Example 6–1 shows how to cancel a job with the `oracle.ias.scheduler.Cancellable` interface.

Example 6–1 Backing Up Data on a Regular Basis with an Option to Cancel

During testing of the application outlined earlier in Example 2–1, it becomes apparent that the job execution may run for long periods of time. Therefore, there may be need to cancel the job execution when it is running.

The following code example shows the modified implementation that provides both `oracle.ias.scheduler.Executable` and `oracle.ias.scheduler.Cancellable` interfaces:

```
import java.io.File;
import java.io.IOException;
import oracle.ias.scheduler.Job;
import oracle.ias.scheduler.Executable;
import oracle.ias.scheduler.Cancellable;
import oracle.ias.scheduler.JobContext;
import oracle.ias.scheduler.JobCancelledException;
import oracle.ias.scheduler.JobExecutionException;


public class CancellableBackupJob implements Executable, Cancellable {

    boolean m_cancelled = false;


    public void cancel() {
        m_cancelled = true;
}


    public void execute(JobContext context) throws
        JobExecutionException, JobCancelledException {

        // retrieve the source and destination directories
        Job job = context.getJob();
        String source = job.getProperties().getProperty("SourceDirectory");
        String destination =
           job.getProperties().getProperty("DestinationDirectory");

        // get the list of files to copy
        File directory = new File(source);
        File[] files = directory.listFiles();

        // copy the files
        Runtime runtime = Runtime.getRuntime();
        Process process;
        for (int x = 0; x < files.length; x++) {

            // cancelled?
            if (m_cancelled) {
                throw new JobCancelledException();
            }

            try {
                process = runtime.exec("/bin/cp " + files[x].toString() +
                                       " " + destination);
                process.waitFor();
            } catch(IOException e) {
                throw new RuntimeException("copy failed: "+files[x],e);
            } catch(InterruptedException e) {
                throw new RuntimeException("copy failed: "+files[x],e);
            }
        }
    }
}
```

At a minimum, canceling a job means the following:

- The implementation must use the `oracle.ias.scheduler.Cancellable` interface.
- For the job execution to be canceled, you must invoke the `cancel()` method, which causes the `oracle.ias.scheduler.JobCancelledException` exception that will stop the job execution.
- It may not always be possible to immediately invoke the `cancel()` method; take this into account when you program.

6.3 Frequently Asked Questions

Is there a way to re-execute a job that has been canceled?

No. There is no mechanism to retry an execution that has been canceled. Only failed job executions can be retried.

For more information, see Section 3.2.

7

Oracle Application Server Containers for J2EE Events and Listeners

This chapter describes the Job Scheduler event listener framework. The following topics are covered:

- Events and Event Listeners
- Implementing and Binding a Event Listener
- Best Practices for Implementing and Binding Event Listeners
- Frequently Asked Questions About Job Listeners

7.1 Events and Event Listeners

An event represents a change in a job's state; each change in a job's state is represented by a corresponding event. An application can be programmed to react to these events using an event listener. An event listener can be bound to one or more jobs at any time during the life cycle of a job.

The Job Scheduler uses numerous events to represent job state changes. These events are listed in Table 7–1.

Table 7–1 Job Scheduler Events

Event	Description
`oracle.ias.scheduler.event.JobBlackoutEvent`	Job was suppressed due to a blackout window.
`oracle.ias.scheduler.event.JobCompletedEvent`	Job scheduled end date passed.
`oracle.ias.scheduler.event.JobCreatedEvent`	Job was created.
`oracle.ias.scheduler.event.JobExecutionCancelledEvent`	Job was canceled.
`oracle.ias.scheduler.event.JobExecutionFailedEvent`	Job failed.
`oracle.ias.scheduler.event.JobExecutionPausedEvent`	Job was suppressed because the job is currently paused.
`oracle.ias.scheduler.event.JobExecutionSucceededEvent`	Job successful.
`oracle.ias.scheduler.event.JobExecutionThresholdExceededEvent`	Job was suppressed because the execution threshold was exceeded.

Table 7–1 (Cont.) Job Scheduler Events

Event	Description
`oracle.ias.scheduler.event.JobPausedEvent`	Job was paused.
`oracle.ias.scheduler.event.JobRemovedEvent`	Job was removed.
`oracle.ias.scheduler.event.JobResumedEvent`	Previously paused job was resumed.

7.2 Implementing and Binding a Event Listener

To receive events, an event listener is required. An event listener must use the `oracle.ias.scheduler.event.EventListener` interface. This interface is defined as follows:

```
public interface EventListener extends java.util.EventListener {
    public void dispatch(SchedulerEvent event) throws Exception;
    public Class[] wants();
}
```

The `wants()` method is used to specify the events for which this listener is interested, and returns the associated class object for those specified events. After the listener is implemented, the `dispatch()` method is invoked every time one of the desired events occurs.

For more information about the `oracle.ias.scheduler.event.EventListener` interface and its methods, see *Oracle Containers for J2EE Job Scheduler API Reference*.

Example 7–1 shows how to implement an event listener that is interested in the `JobExecutionFailedEvent` and `JobExecutionSucceededEvent` events.

Example 7–1 Job Listener Implementation

```
import oracle.ias.scheduler.event.*;

public class TestListener implements EventListener {

    public void dispatch(SchedulerEvent event) {
        System.out.println("Got event, "+event.getClass().getName());
    }

    public Class[] wants() {
        return new Class[] {
            oracle.ias.scheduler.event.JobExecutionFailedEvent.class,
            oracle.ias.scheduler.event.JobExecutionSucceeded.class
        };
    }
}
```

Example 7–2 shows how to bind the `TestListener` listener created in Example 7–1.

Example 7–2 Binding a Listener to a Job

```
JobHandle handle = scheduler.add(...);

// bind the listener to the job
scheduler.addListener(handle,TestListener.class);
```

7.3 Best Practices for Implementing and Binding Event Listeners

When implementing and binding job listeners, keep the following in mind:

- **Keep job listener processing to a minimum.** Events are processed serially by job listeners and lengthy processing should be avoided. If lengthy processing cannot be avoided, consider serializing the event for later processing.

7.4 Frequently Asked Questions About Job Listeners

Can I use the same job listener for every job?

Yes. Use the event's `getHandle()` method to determine which event is associated with which job.

If I use the same job listener for every job, how many instances of the job listener will there be?

There will be one job listener instance per job.

Is the job listener dispatch( ) method reentrant (can this method be called while it is already in use)?

Yes. Use appropriate measures when modifying job listener member variables (for example, using locks to avoid resource conflicts).

Is the job listener instance state persistent across container restarts?

No. The job listener instance state is not persistent.

8

Oracle Application Server Containers for J2EE Triggers and Notifications

This chapter describes triggers and notifications. The following topics are covered:

8.1 Trigger-Driven Jobs

This section introduces two concepts: triggers and notifications. Notifications are messages sent from one object to another, in effect notifying the recipient that something happened. The recipient of a notification is called a trigger. A trigger contains certain conditions that are evaluated against one or more notifications that it receives. When a specified condition is met, an associated job is run.

A trigger's conditions are described by a logical expression where the operands are job notification assertions. Notifications can be generated in either of the following ways:

- Programatically by the application
- As the result of a timer expiration

Likewise, notifications can either be sent to a specific trigger or to a specified set of triggers. On receipt, however, triggers do not generate notifications. By employing the use of triggers, jobs can be enabled to respond to specific application conditions (for example, triggering a job based on revenue exceeding a certain threshold).

8.1.1 Triggers and Notifications

The system-supplied `oracle.ias.scheduler.Trigger` class is used to specify the conditions by which the associated job is run. A condition is expressed as a logical combination of operands. The following logical operators are allowed:

- AND (represented as '&&')
- OR (represented as '||')
- NOT (represented as '!')

Precedence can be specified using parentheses. The following are a few example expressions:

- `N`

Execute when the N notification is received.

- `N1 || N2`

  Execute when either notification is received.

- `N1 && N2`

  Execute when both notifications are received.

- `N1 || (N2 && N3)`

  Execute when either the N1 notification is received or when both N2 and N3 notifications are received.

The operand in a condition is the name associated with the notifications sent using the Job Scheduler's `notify()` method. For example, to send the `DataHasArrived` notification to all triggers, the application uses the following code:

```
Scheduler.notify(new Notification("DataHasArrived"));
```

Job Scheduler evaluates triggers when a notification is sent. The result of a trigger evaluation is boolean. If the trigger evaluates to true, then the associated job starts. After the trigger fires, it is immediately reset, before the job runs. When the trigger is reset, the record of all previously received notifications by the trigger is erased. A trigger is reset only when the job runs. If the trigger does not fire, the notification is recorded by the trigger for later use.

For example, suppose a trigger has the following condition:

```
N1 && N2
```

Assume the trigger receives only notification N1; the trigger evaluates to false, and the notification is recorded. Later, the trigger receives notification N2. Now that both conditions are met, the trigger evaluates to true; the job runs, and the trigger is reset.

Jobs can be associated with a schedule, trigger, or both a schedule and trigger. When a job is associated with a schedule only, an implicit trigger is associated with the job. A trigger of this type provides the following condition:

```
timeout
```

When the schedule expires, a timeout notification is sent to the associated trigger for processing. In this case, the trigger fires; the job runs, and the trigger is reset. The `timeout` notification may also be used in a trigger expression along with other notifications. For example:

- `timeout || N`

  Run when the either N notification is received or the schedule expires.

- `timeout && N`

  Run when the N notification is received and the schedule expires.

The `timeout` notification can only be used in cases where the job is associated with both a schedule and a trigger. The `timeout` notification name is likewise reserved and can not be used or sent by an application to the scheduler. This behavior is consistent with the Notification class. Additionally, the `timeout` notification must be referenced in the condition expression of the trigger.

8.1.2 Cautions For Using the NOT Operator

If you use the NOT operator in a trigger expression, then be aware of the following:

- **NOT expressions should include at least two operators.** Otherwise, the trigger fires when any other notification is received. For example, the expression `!N` would cause the trigger to fire whenever any notification except N was received.
- **NOT expressions should not be used with a schedule that repeats indefinitely.** This may result in a permanently hung trigger. Recall that a trigger retains all notifications that were received until the trigger fires. If the trigger receives a notification that satisfies the NOT condition, the trigger will never fire.

8.2 How Do I Submit a Job with a Trigger?

To submit a job with a trigger, use the `oracle.ias.scheduler.Trigger` class. For more information, see *Oracle Containers for J2EE Job Scheduler API Reference*.

Example 8–1 shows how to create a trigger to run a job when the `diskIsFull` notification is received.

Example 8–1 Submitting a Job with a Trigger

```
// set up the trigger, run when 'diskIsFull' notification is received
Trigger trigger = new Trigger("diskIsFull");

// submit the job
scheduler.add("disk is full job",
              diskFullJob.class.getName(),
              trigger,
              null);
```

To embellish Example 8–1, Example 8–2 shows how to create a trigger to run a job every night at midnight if either the `diskIsFull` or `timeout` notifications is received.

Example 8–2 Submitting a Job with a Trigger and a Schedule

```
// set up the schedule, repeats every night at midnight
RecurSchedule schedule = new RecurSchedule("freq=daily,byhour=0;");

// set up the trigger, run when either 'diskIsFull'
// notification is received or the schedule expires
Trigger trigger = new Trigger("diskIsFull || timeout");

// submit the job
scheduler.add("disk is full job",
              DiskFullJob.class.getName(),
              schedule,
              trigger,
              null,
              Level.WARN,
              0L,
              0L);
```

8.3 How Do I Send Notifications to a Job?

To send a notification to a job, use the `oracle.ias.scheduler.Scheduler.notify()` method. For more information, see *Oracle Containers for J2EE Job Scheduler API Reference*.

Example 8–3 shows how to send the `diskIsFull` notification to a job.

Example 8–3 Sending a Notification to a Job

```
// send the 'diskIsFull' notification
scheduler.notify(new Notification("diskIsFull");
```

8.4 Frequently Asked Questions About Triggers and Notifications

When are timeout notifications sent?

Timeout notifications are sent when a job expires.

Can a user send a timeout notification to a trigger?

No. Timeouts can only be sent by the Job Scheduler.

9

Deploying Job Scheduler-Enabled Applications

This chapter provides information on how to configure Job Scheduler-enabled applications for deployment. The following topics are covered:

- Bundling Job Scheduler with a J2EE Application
- Configuring Persistence for Job Scheduler
- Configuring Security for Job Scheduler
- Configuring Logging for Job Scheduler
- Configuring DMS for Job Scheduler
- Configuring JMX for Job Scheduler
- Configuring Execution Interval Threshold Recovery for Job Scheduler

9.1 Bundling Job Scheduler with a J2EE Application

Job Scheduler is deployed as a stateless session Enterprise Java Bean (EJB). Unlike a typical EJB, the actual class files do not need to be deployed with Job Scheduler. Instead, these files are included as part of Oracle Application Server Containers for J2EE (OC4J).

9.1.1 Generating the scheduler-ejb.jar File

As is the case with all EJBs, a deployment descriptor is required. For Job Scheduler, there are two:

1. `ejb-jar.xml` (Job Scheduler deployment descriptor)
2. `orion-ejb-jar.xml` (OC4J-specific Job Scheduler deployment descriptor)

Note that both of these files must be present in the `scheduler-ejb.jar` archive file for Job Scheduler to function correctly.

To deploy Job Scheduler with an application, an application deployer needs to include the `scheduler-ejb.jar` archive. In addition to the `ejb-jar.xml` and `orion-ejb-jar.xml` files, this archive must also include the following:

- All job implementations
- All event listener implementations

For Job Scheduler to access job and event listener implementations, the class files must be included in the `scheduler-ejb.jar` archive. It is the application deployer's responsibility to generate this file and bundle it with the application.

Example 9–1 shows a sample `scheduler-ejb.jar` file. A Job Scheduler-enabled application uses the `test.job` job and `test.watch` event listener implementations as part of the application.

Example 9–1 Sample scheduler-ejb.jar File

```
test/job.class
test/watch.class
META-INF/ejb-jar.xml
META-INF/orion-ejb-jar.xml
```

9.1.2 Bundling scheduler-ejb.jar in an Enterprise Archive (EAR) File

Once the `scheduler-ejb.jar` file is generated, it must be bundled in the application's EAR file. In addition, the archive's `application.xml` file must contain a module entry for Job Scheduler.

In Example 9–2, Job Scheduler is deployed with a J2EE application. The addition of Job Scheduler to the application is accomplished by adding a `<module>` element as shown in Example 9–2.

Example 9–2 Adding the Job Scheduler to the application.xml File

```
<module>
   <ejb>scheduler-ejb.jar</ejb>
</module>
```

9.2 Configuring Persistence for Job Scheduler

Job Scheduler provides three basic types of persistent job storage:

- In-memory (JMS persistence)
- File-based (JMS persistence)
- Database-backed (JDBC persistence)

To configure JMS persistence, set the `jobStoreProviderClassName` `<env-entry>` to `oracle.ias.scheduler.core.jobstore.jdbc.ProvderImpl`. In-memory or file-based persistence is achieved by configuring JMS queues to be in-memory or file-based, respectively.

To configure database-backed persistence, run the *J2EE_HOME*`/database/scheduler_jobstore.sql` script to create the database tables, set the `jobStoreProviderClassName` `<env-entry>` to `oracle.ias.scheduler.core.jobstore.jdbc.ProviderImpl`.

Examples of how to configure both JDBC and JMS persistence are provided in the following sub-sections.

9.2.1 Configuring JDBC Persistence

To configure JDBC persistence:

1. Run the `scheduler_jobstore.sql` SQL script to create the database tables.

2. In the `ejb-jar.xml` file, set the `jobStoreProviderClassName` `<env-entry>` value as follows:

```
<env-entry>
   <env-entry-name>jobStoreProviderClassName</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>
      oracle.ias.scheduler.core.jobstore.jdbc.ProviderImpl</env-entry-value>
</env-entry>
```

3. Create a new `<managed-data-source>` entry in `data-sources.xml`, making sure that the specified connection pool references an existing `<connection-pool>`:

```
<managed-data-source name="SchedulerJobstore"
                     connection-pool-name="Example Connection Pool"
                     jndi-name="scheduler/jobstore" />
```

For a complete example on configuring JDBC persistence, refer to the following How-To located on OTN:

http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/how-to-scheduler-db/doc/readme.html

9.2.2 Configuring JMS Persistence

To configure JMS persistence:

1. In the `ejb-jar.xml` file, set the `jobStoreProviderClassName` `<env-entry>` value as follows:

```
<env-entry>
   <env-entry-name>jobStoreProviderClassName</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>
      oracle.ias.scheduler.core.jobstore.jms.ProviderImpl
   </env-entry-value>
</env-entry>
```

2. In the `jms.xml` file, create a new `<queue>` entry as shown below:

```
<queue name="jms/scheduler_jobstore"
        persistence file="scheduler_jobstore">
   <description>scheduler job store queue</description>
</queue>
```

The queue destinations defined in `jms.xml` are persistent only if the `persistence-file` attribute on each queue destination is set. Please refer to the JMS documentation for more information about creating a persistent queue destination.

For a complete example on configuring JMS persistence, refer to the following How-To located on OTN:

http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/how-to-scheduler-jms/doc/readme.html

9.3 Configuring Security for Job Scheduler

Job Scheduler's `ejb-jar.xml` deployment descriptor file contains information about the security configuration for Job Scheduler. This information can be modified to limit

access to one or more of Job Scheduler APIs to a specific role. For example, removing a job can be limited to users with administrative privileges.

In Example 9–3, application users are divided into two general categories: users and administrators. Users can only submit jobs, while administrators can submit, pause, resume, cancel, and remove jobs.

Example 9–3 Two-Tier Security Model

```
<!-- role declarations -->
<security-role>
  <role-name>user</role-name>
</security-role>

<security-role>
  <role-name>administrator</role-name>
</security-role>


<!-- methods that can be invoked by the group 'user' -->
<method-permission>
  <role-name>user</role-name>

  <method>
    <ejb-name>scheduler</ejb-name>
    <method-name>add</method-name>
  </method>
</method-permission>


<!-- methods that can be invoked by the group 'administrator' -->
<method-permission>
  <role-name>administrator</role-name>
  <method>
    <ejb-name>scheduler</ejb-name>
    <method-name>remove</method-name>
  </method>

  <method>
    <ejb-name>scheduler</ejb-name>
    <method-name>pause</method-name>
  </method>

  <method>
    <ejb-name>scheduler</ejb-name>
    <method-name>resume</method-name>
  </method>

  <method>
    <ejb-name>scheduler</ejb-name>
    <method-name>cancel</method-name>
  </method>
</method-permission>
```

9.4 Configuring Logging for Job Scheduler

This section discusses the available log level settings for Job Scheduler. The root logger has a default log level set to `Level.WARNING`, since unexpected and fatal errors will be logged by the root logger. However, the log level can be changed by setting the

string value of the <env-entry> called globalLogLevel in the ejb-jar.xml file, as shown in Example 9–4.

Example 9–4 Changing the Log Level

```
<env-entry>
   <env-entry-name>globalLogLevel</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>FINEST</env-entry-value>
</env-entry>
```

Table 9–1 shows the mapping between the <env-entry> values and the corresponding log levels to which they match.

Table 9–1 <env-entry> Values and Log Levels

<env-entry> Value	Log Level
OFF	Level.OFF
FINEST	Level.FINEST
FINER	Level.FINER
FINE	Level.FINE
CONFIG	Level.CONFIG
INFO	Level.INFO
WARNING	Level.WARNING
SEVERE	Level.SEVERE
ALL	Level.ALL

9.5 Configuring DMS for Job Scheduler

To configure whether or not DMS statistics are published, set the <env-entry> value in the ejb-jar.xml file as shown in Example 9–5.

Example 9–5 Configuring DMS

```
<env-entry>
   <env-entry-name>oracle.ias.scheduler.dms</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>true</env-entry-value>
</env-entry>
```

The <env-entry-value> is set to true, meaning DMS statistics will be published. Set this value to false if you do not want DMS statistics published.

9.6 Configuring JMX for Job Scheduler

To configure whether or not JMX MBeans are published, set the <env-entry> value in the ejb-jar.xml file as shown in Example 9–6.

Example 9–6 Configuring JMX

```
<env-entry>
   <env-entry-name>oracle.ias.scheduler.jmx</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>true</env-entry-value>
```

```
</env-entry>
```

The <env-entry-value> is set to true, meaning JMX MBeans will be published. Set this value to false if you do not want JMX MBeans published.

9.7 Configuring Execution Interval Threshold Recovery for Job Scheduler

Some job executions might be scheduled during the time when the container has been shutdown. The Job Scheduler provides the ability to recover these missed executions. One or more executions scheduled during the time the container is down will result in one job execution when the container starts up.

In the case of fixed-interval and fixed-delay schedules, it is possible to enable execution recovery for jobs whose repetition interval is greater than a configurable execution recovery threshold duration.

To do so, set the <env-entry> value with the desired threshold value in minutes, as shown in Example 9–7. If this value is not set, the value defaults to 30 minutes. Job execution recovery is performed only for fixed-interval and fixed-delay jobs executing at intervals greater than 30 minutes.

Example 9–7 Configuring Execution Threshold Recovery

```
<env-entry>
   <env-entry-name>intervalThresholdMinutes</env-entry-name>
   <env-entry-type>java.lang.Long</env-entry-type>
   <env-entry-value>35</env-entry-value>
</env-entry>
```

10

Managing the Oracle Application Server Containers for J2EE

This chapter provides information on how to manage Job Scheduler using JMX MBeans.

The JMX MBeans are chosen to represent Job Scheduler and associated data types. Each MBean reveals attributes, operations, and relevant JSR77 statistics gathered by the Oracle Dynamic Monitoring Service (DMS).

Table 10–1 summarizes the MBeans that are provided.

Table 10–1 JMX MBean Summary

Management Bean	Description
`JobMBean`	Provides access to a job instance for management and monitoring.
`SchedulerMBean`	Provides access to a scheduler instance for management and monitoring.
`SchedulerAggregationMBean`	Provides access to all scheduler instances for management and monitoring purposes.

10.1 Job Management Bean

One Job MBean instance is registered for each job submitted to Job Scheduler, and persists until either the job is removed or the hosting application is undeployed. This MBean can be used to monitor and configure a job, including:

- Suppressing or resuming a previously suppressed job
- Canceling any outstanding scheduled jobs

For more information, see Appendix D.

10.2 Job Scheduler Management Bean

One instance of the Job Scheduler MBean is registered for each Job Scheduler application component deployed, and persists until the hosting application is undeployed. The management bean can be used to monitor and configure the Job Scheduler instance, including:

- Examining the run-time configuration.
- Retrieving all jobs associated with the instance.
- Creating, listing, or removing execution blackout windows.

For more information, see Appendix D.

10.3 Job Scheduler Aggregation Management Bean

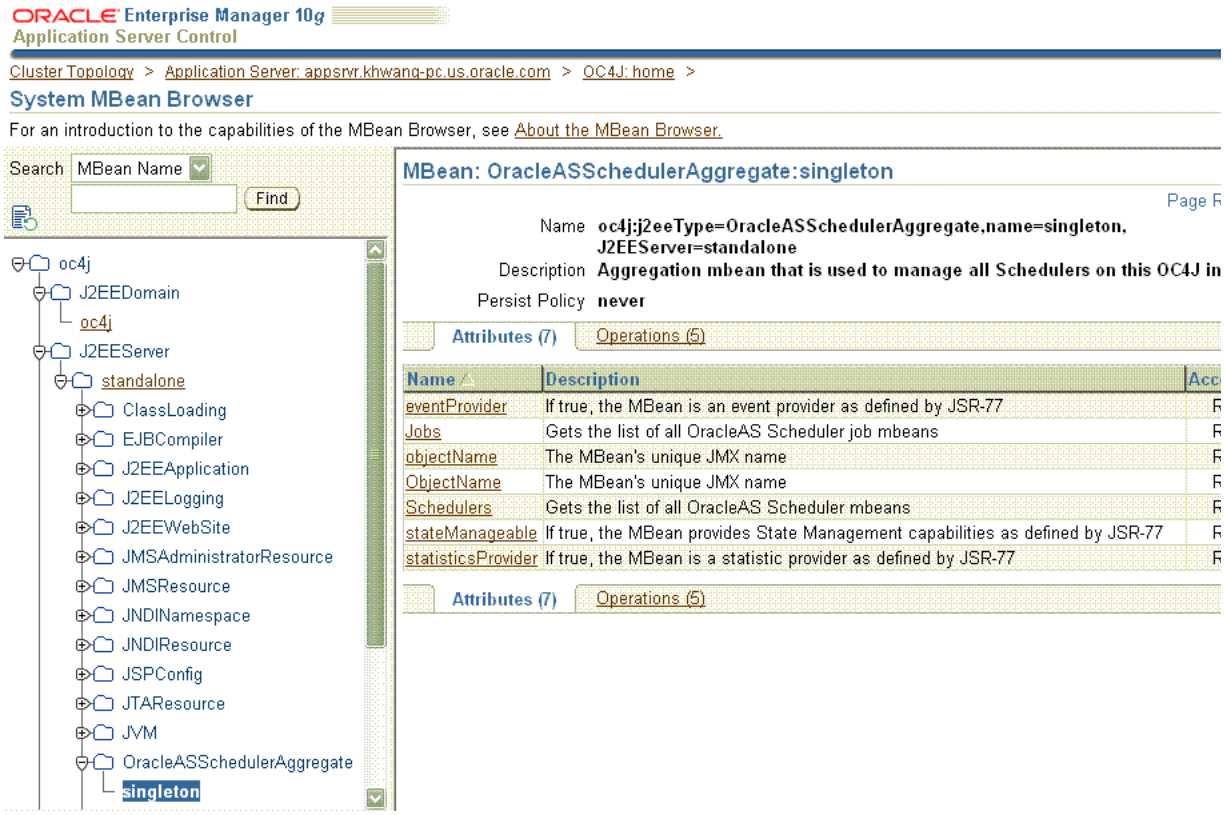
The Aggregation MBean provides an aggregated view of all Job Scheduler and job instances. This MBean can be used to monitor and configure all Job Scheduler and job instances, including:

- Retrieving all Job Scheduler instances.
- Retrieving all jobs on all Job Scheduler instances.
- Creating a blackout window across all Job Scheduler instances.
- Pausing or resuming jobs across all Job Scheduler instances.
- Canceling all jobs across all Job Scheduler instances.

For more information, see Appendix D.

The Job Scheduler Aggregation MBean can also be manage from the Application Server Control Console, as illustrated in Figure 10–1.

Figure 10–1 System MBean Browser for Job Scheduler Aggregation MBean



To access this screen:

1. Login to Application Server Control Console.
2. In the Members section, expand the entries in the "Name" column until you see the **home** link for OC4J. Click on **home**.
3. On the OC4J home page, click on **Administration**.

4. On the administration page, look for "System MBean Browser" under Administration Tasks > JMX. Click on the icon in the "System MBean Browser" row in the "Go to Task" column.
5. In the System MBean Browser page, scroll down in the left navigation pane until you see "OracleASSchedulerAggregate." Expand this entry and click on **singleton**. The Job Scheduler Aggregation MBean attributes are displayed.

   Click on the "Operations" tab to view the Job Scheduler Aggregation MBean operations.

A

RFC 2445 Excerpt: Recurrence

This appendix contains an excerpt of RFC 2445, "Internet Calendaring and Scheduling Core Object Specification (ICalendar)." Section 4.3.10 of this RFC is used as the basis for iCalendar recurrence schedules. This appendix provides a listing of Section 4.3.10, and highlights specific areas that are not supported. Some examples are also provided.

A.1 RFC 2445, Section 4.3.10. Recurrence Rule

This section provides a listing of RFC 2445, Section 4.3.10.

```
.
.
.
4.3.10 Recurrence Rule

   Value Name: RECUR

   Purpose: This value type is used to identify properties that contain
   a recurrence rule specification.

   Formal Definition: The value type is defined by the following
   notation:

     recur      = "FREQ"=freq *(

                ; either UNTIL or COUNT may appear in a 'recur',
                ; but UNTIL and COUNT MUST NOT occur in the same 'recur'

                ( ";" "UNTIL" "=" enddate ) /
                ( ";" "COUNT" "=" 1*DIGIT ) /

                ; the rest of these keywords are optional,
                ; but MUST NOT occur more than once

                ( ";" "INTERVAL" "=" 1*DIGIT )          /
                ( ";" "BYSECOND" "=" byseclist )        /
                ( ";" "BYMINUTE" "=" byminlist )        /
                ( ";" "BYHOUR" "=" byhrlist )           /
                ( ";" "BYDAY" "=" bywdaylist )          /
                ( ";" "BYMONTHDAY" "=" bymodaylist )    /
                ( ";" "BYYEARDAY" "=" byyrdaylist )     /
                ( ";" "BYWEEKNO" "=" bywknolist )       /
                ( ";" "BYMONTH" "=" bymolist )          /
                ( ";" "BYSETPOS" "=" bysplist )         /
                ( ";" "WKST" "=" weekday )              /
                ( ";" x-name "=" text )
```

```
           )

  freq       = "SECONDLY" / "MINUTELY" / "HOURLY" / "DAILY"
             / "WEEKLY" / "MONTHLY" / "YEARLY"

  enddate    = date
  enddate    =/ date-time            ;An UTC value

  byseclist  = seconds / ( seconds *("," seconds) )

  seconds    = 1DIGIT / 2DIGIT       ;0 to 59

  byminlist  = minutes / ( minutes *("," minutes) )

  minutes    = 1DIGIT / 2DIGIT       ;0 to 59

  byhrlist   = hour / ( hour *("," hour) )

  hour       = 1DIGIT / 2DIGIT       ;0 to 23

  bywdaylist = weekdaynum / ( weekdaynum *("," weekdaynum) )

  weekdaynum = [([plus] ordwk / minus ordwk)] weekday

  plus       = "+"

  minus      = "-"

  ordwk      = 1DIGIT / 2DIGIT       ;1 to 53

  weekday    = "SU" / "MO" / "TU" / "WE" / "TH" / "FR" / "SA"
  ;Corresponding to SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
  ;FRIDAY, SATURDAY and SUNDAY days of the week.

  bymodaylist = monthdaynum / ( monthdaynum *("," monthdaynum) )

  monthdaynum = ([plus] ordmoday) / (minus ordmoday)

  ordmoday   = 1DIGIT / 2DIGIT       ;1 to 31

  byyrdaylist = yeardaynum / ( yeardaynum *("," yeardaynum) )

  yeardaynum = ([plus] ordyrday) / (minus ordyrday)

  ordyrday   = 1DIGIT / 2DIGIT / 3DIGIT      ;1 to 366

  bywknolist = weeknum / ( weeknum *("," weeknum) )

  weeknum    = ([plus] ordwk) / (minus ordwk)

  bymolist   = monthnum / ( monthnum *("," monthnum) )

  monthnum   = 1DIGIT / 2DIGIT       ;1 to 12

  bysplist   = setposday / ( setposday *("," setposday) )

  setposday  = yeardaynum

Description: If the property permits, multiple "recur" values are
specified by a COMMA character (US-ASCII decimal 44) separated list
```

of values. The value type is a structured value consisting of a list of one or more recurrence grammar parts. Each rule part is defined by a NAME=VALUE pair. The rule parts are separated from each other by the SEMICOLON character (US-ASCII decimal 59). The rule parts are not ordered in any particular sequence. Individual rule parts MUST only be specified once.

The FREQ rule part identifies the type of recurrence rule. This rule part MUST be specified in the recurrence rule. Valid values include SECONDLY, to specify repeating events based on an interval of a second or more; MINUTELY, to specify repeating events based on an interval of a minute or more; HOURLY, to specify repeating events based on an interval of an hour or more; DAILY, to specify repeating events based on an interval of a day or more; WEEKLY, to specify repeating events based on an interval of a week or more; MONTHLY, to specify repeating events based on an interval of a month or more; and YEARLY, to specify repeating events based on an interval of a year or more.

The INTERVAL rule part contains a positive integer representing how often the recurrence rule repeats. The default value is "1", meaning every second for a SECONDLY rule, or every minute for a MINUTELY rule, every hour for an HOURLY rule, every day for a DAILY rule, every week for a WEEKLY rule, every month for a MONTHLY rule and every year for a YEARLY rule.

The UNTIL rule part defines a date-time value which bounds the recurrence rule in an inclusive manner. If the value specified by UNTIL is synchronized with the specified recurrence, this date or date-time becomes the last instance of the recurrence. If specified as a date-time value, then it MUST be specified in an UTC time format. If not present, and the COUNT rule part is also not present, the RRULE is considered to repeat forever.

The COUNT rule part defines the number of occurrences at which to range-bound the recurrence. The "DTSTART" property value, if specified, counts as the first occurrence.

The BYSECOND rule part specifies a COMMA character (US-ASCII decimal 44) separated list of seconds within a minute. Valid values are 0 to 59. The BYMINUTE rule part specifies a COMMA character (US-ASCII decimal 44) separated list of minutes within an hour. Valid values are 0 to 59. The BYHOUR rule part specifies a COMMA character (US-ASCII decimal 44) separated list of hours of the day. Valid values are 0 to 23.

The BYDAY rule part specifies a COMMA character (US-ASCII decimal 44) separated list of days of the week; MO indicates Monday; TU indicates Tuesday; WE indicates Wednesday; TH indicates Thursday; FR indicates Friday; SA indicates Saturday; SU indicates Sunday.

Each BYDAY value can also be preceded by a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of the specific day within the MONTHLY or YEARLY RRULE. For example, within a MONTHLY rule, +1MO (or simply 1MO) represents the first Monday within the month, whereas -1MO represents the last Monday of the month. If an integer modifier is not present, it means all days of this type within the specified frequency. For example, within a MONTHLY rule, MO represents all Mondays within the month.

The BYMONTHDAY rule part specifies a COMMA character (ASCII decimal 44) separated list of days of the month. Valid values are 1 to 31 or -31 to -1. For example, -10 represents the tenth to the last day of the month.

The BYYEARDAY rule part specifies a COMMA character (US-ASCII decimal 44) separated list of days of the year. Valid values are 1 to 366 or -366 to -1. For example, -1 represents the last day of the year (December 31st) and -306 represents the 306th to the last day of the year (March 1st).

The BYWEEKNO rule part specifies a COMMA character (US-ASCII decimal 44) separated list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1. This corresponds to weeks according to week numbering as defined in [ISO 8601]. A week is defined as a seven day period, starting on the day of the week defined to be the week start (see WKST). Week number one of the calendar year is the first week which contains at least four (4) days in that calendar year. This rule part is only valid for YEARLY rules. For example, 3 represents the third week of the year.

> Note: Assuming a Monday week start, week 53 can only occur when Thursday is January 1 or if it is a leap year and Wednesday is January 1.

The BYMONTH rule part specifies a COMMA character (US-ASCII decimal 44) separated list of months of the year. Valid values are 1 to 12.

The WKST rule part specifies the day on which the workweek starts. Valid values are MO, TU, WE, TH, FR, SA and SU. This is significant when a WEEKLY RRULE has an interval greater than 1, and a BYDAY rule part is specified. This is also significant when in a YEARLY RRULE when a BYWEEKNO rule part is specified. The default value is MO.

The BYSETPOS rule part specifies a COMMA character (US-ASCII decimal 44) separated list of values which corresponds to the nth occurrence within the set of events specified by the rule. Valid values are 1 to 366 or -366 to -1. It MUST only be used in conjunction with another BYxxx rule part. For example "the last work day of the month" could be represented as:

```
RRULE:FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BYSETPOS=-1
```

Each BYSETPOS value can include a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of the specific occurrence within the set of events specified by the rule.

If BYxxx rule part values are found which are beyond the available scope (ie, BYMONTHDAY=30 in February), they are simply ignored.

Information, not contained in the rule, necessary to determine the various recurrence instance start time and dates are derived from the Start Time (DTSTART) entry attribute. For example, "FREQ=YEARLY;BYMONTH=1" doesn't specify a specific day within the month or a time. This information would be the same as what is specified for DTSTART.

BYxxx rule parts modify the recurrence in some manner. BYxxx rule parts for a period of time which is the same or greater than the frequency generally reduce or limit the number of occurrences of the

recurrence generated. For example, "FREQ=DAILY;BYMONTH=1" reduces the number of recurrence instances from all days (if BYMONTH tag is not present) to all days in January. BYxxx rule parts for a period of time less than the frequency generally increase or expand the number of occurrences of the recurrence. For example, "FREQ=YEARLY;BYMONTH=1,2" increases the number of days within the yearly recurrence set from 1 (if BYMONTH tag is not present) to 2.

If multiple BYxxx rule parts are specified, then after evaluating the specified FREQ and INTERVAL rule parts, the BYxxx rule parts are applied to the current set of evaluated occurrences in the following order: BYMONTH, BYWEEKNO, BYYEARDAY, BYMONTHDAY, BYDAY, BYHOUR, BYMINUTE, BYSECOND and BYSETPOS; then COUNT and UNTIL are evaluated.

Here is an example of evaluating multiple BYxxx rule parts.

```
DTSTART;TZID=US-Eastern:19970105T083000
RRULE:FREQ=YEARLY;INTERVAL=2;BYMONTH=1;BYDAY=SU;BYHOUR=8,9;
 BYMINUTE=30
```

First, the "INTERVAL=2" would be applied to "FREQ=YEARLY" to arrive at "every other year". Then, "BYMONTH=1" would be applied to arrive at "every January, every other year". Then, "BYDAY=SU" would be applied to arrive at "every Sunday in January, every other year". Then, "BYHOUR=8,9" would be applied to arrive at "every Sunday in January at 8 AM and 9 AM, every other year". Then, "BYMINUTE=30" would be applied to arrive at "every Sunday in January at 8:30 AM and 9:30 AM, every other year". Then, lacking information from RRULE, the second is derived from DTSTART, to end up in "every Sunday in January at 8:30:00 AM and 9:30:00 AM, every other year". Similarly, if the BYMINUTE, BYHOUR, BYDAY, BYMONTHDAY or BYMONTH rule part were missing, the appropriate minute, hour, day or month would have been retrieved from the "DTSTART" property.

No additional content value encoding (i.e., BACKSLASH character encoding) is defined for this value type.

Example: The following is a rule which specifies 10 meetings which occur every other day:

```
FREQ=DAILY;COUNT=10;INTERVAL=2
```

There are other examples specified in the "RRULE" specification.

.
.
.

11. Full Copyright Statement

```
copyrights defined in the Internet Standards process must be
followed, or as required to translate it into languages other than
English.
```

A.2 Job Scheduler Implementation of the Recurrence Rule

Oracle Application Server Containers for J2EE implements the recurrence rule of RFC 2445 as follows:

- The UNTIL rule is not supported. Similar functionality can be achieved by using the `setEndDate()` method of the `recurSchedule` object.
- The COUNT rule is not supported. Similar functionality can be achieved by using the `setCount()` method of the `recurSchedule` object.
- The BYDAY clause supports both two- and three-letter abbreviations for days of the week (for example, either `MO` or `MON` may be used to represent Monday).
- The BYSETPOS rule is not supported.
- The WKST rule is not supported.

A.3 RFC 2445, Section 4.8.5.4. Recurrence Rule Examples

This section provides a listing of RFC 2445, Section 4.8.5.4. This section contains many examples which you might find helpful.

> **Note:** The examples that use the BYSETPOS and WKST rules are omitted, because these rules are not supported by Oracle Application Server Containers for J2EE.

```
4.8.5.4 Recurrence Rule

   Property Name: RRULE

   Purpose: This property defines a rule or repeating pattern for
   recurring events, to-dos, or time zone definitions.

   Value Type: RECUR

   Property Parameters: Non-standard property parameters can be
   specified on this property.

   Conformance: This property can be specified one or more times in
   recurring "VEVENT", "VTODO" and "VJOURNAL" calendar components. It
   can also be specified once in each STANDARD or DAYLIGHT sub-component
   of the "VTIMEZONE" calendar component.

   Description: The recurrence rule, if specified, is used in computing
   the recurrence set. The recurrence set is the complete set of
   recurrence instances for a calendar component. The recurrence set is
   generated by considering the initial "DTSTART" property along with
   the "RRULE", "RDATE", "EXDATE" and "EXRULE" properties contained
   within the iCalendar object. The "DTSTART" property defines the first
   instance in the recurrence set. Multiple instances of the "RRULE" and
   "EXRULE" properties can also be specified to define more
   sophisticated recurrence sets. The final recurrence set is generated
   by gathering all of the start date/times generated by any of the
```

```
specified "RRULE" and "RDATE" properties, and excluding any start
date/times which fall within the union of start date/times generated
by any specified "EXRULE" and "EXDATE" properties. This implies that
start date/times within exclusion related properties (i.e., "EXDATE"
and "EXRULE") take precedence over those specified by inclusion
properties (i.e., "RDATE" and "RRULE"). Where duplicate instances are
generated by the "RRULE" and "RDATE" properties, only one recurrence
is considered. Duplicate instances are ignored.

The "DTSTART" and "DTEND" property pair or "DTSTART" and "DURATION"
property pair, specified within the iCalendar object defines the
first instance of the recurrence. When used with a recurrence rule,
the "DTSTART" and "DTEND" properties MUST be specified in local time
and the appropriate set of "VTIMEZONE" calendar components MUST be
included. For detail on the usage of the "VTIMEZONE" calendar
component, see the "VTIMEZONE" calendar component definition.

Any duration associated with the iCalendar object applies to all
members of the generated recurrence set. Any modified duration for
specific recurrences MUST be explicitly specified using the "RDATE"
property.

Format Definition: This property is defined by the following
notation:

  rrule      = "RRULE" rrulparam ":" recur CRLF

  rrulparam  = *(";" xparam)

Example: All examples assume the Eastern United States time zone.

Daily for 10 occurrences:

  DTSTART;TZID=US-Eastern:19970902T090000
  RRULE:FREQ=DAILY;COUNT=10

  ==> (1997 9:00 AM EDT)September 2-11

Daily until December 24, 1997:

  DTSTART;TZID=US-Eastern:19970902T090000
  RRULE:FREQ=DAILY;UNTIL=19971224T000000Z

  ==> (1997 9:00 AM EDT)September 2-30;October 1-25
      (1997 9:00 AM EST)October 26-31;November 1-30;December 1-23

Every other day - forever:

  DTSTART;TZID=US-Eastern:19970902T090000
  RRULE:FREQ=DAILY;INTERVAL=2
  ==> (1997 9:00 AM EDT)September2,4,6,8...24,26,28,30;
       October 2,4,6...20,22,24
      (1997 9:00 AM EST)October 26,28,30;November 1,3,5,7...25,27,29;
       Dec 1,3,...

Every 10 days, 5 occurrences:

  DTSTART;TZID=US-Eastern:19970902T090000
  RRULE:FREQ=DAILY;INTERVAL=10;COUNT=5
```

```
  ==> (1997 9:00 AM EDT)September 2,12,22;October 2,12

Everyday in January, for 3 years:

  DTSTART;TZID=US-Eastern:19980101T090000
  RRULE:FREQ=YEARLY;UNTIL=20000131T090000Z;
   BYMONTH=1;BYDAY=SU,MO,TU,WE,TH,FR,SA
  or
  RRULE:FREQ=DAILY;UNTIL=20000131T090000Z;BYMONTH=1

  ==> (1998 9:00 AM EDT)January 1-31
      (1999 9:00 AM EDT)January 1-31
      (2000 9:00 AM EDT)January 1-31

Weekly for 10 occurrences

  DTSTART;TZID=US-Eastern:19970902T090000
  RRULE:FREQ=WEEKLY;COUNT=10

  ==> (1997 9:00 AM EDT)September 2,9,16,23,30;October 7,14,21
      (1997 9:00 AM EST)October 28;November 4

Weekly until December 24, 1997

  DTSTART;TZID=US-Eastern:19970902T090000
  RRULE:FREQ=WEEKLY;UNTIL=19971224T000000Z

  ==> (1997 9:00 AM EDT)September 2,9,16,23,30;October 7,14,21
      (1997 9:00 AM EST)October 28;November 4,11,18,25;
                        December 2,9,16,23
Monthly on the 1st Friday for ten occurrences:

  DTSTART;TZID=US-Eastern:19970905T090000
  RRULE:FREQ=MONTHLY;COUNT=10;BYDAY=1FR

  ==> (1997 9:00 AM EDT)September 5;October 3
      (1997 9:00 AM EST)November 7;Dec 5
      (1998 9:00 AM EST)January 2;February 6;March 6;April 3
      (1998 9:00 AM EDT)May 1;June 5

Monthly on the 1st Friday until December 24, 1997:

  DTSTART;TZID=US-Eastern:19970905T090000
  RRULE:FREQ=MONTHLY;UNTIL=19971224T000000Z;BYDAY=1FR

  ==> (1997 9:00 AM EDT)September 5;October 3
      (1997 9:00 AM EST)November 7;December 5

Every other month on the 1st and last Sunday of the month for 10
occurrences:

  DTSTART;TZID=US-Eastern:19970907T090000
  RRULE:FREQ=MONTHLY;INTERVAL=2;COUNT=10;BYDAY=1SU,-1SU

  ==> (1997 9:00 AM EDT)September 7,28
      (1997 9:00 AM EST)November 2,30
      (1998 9:00 AM EST)January 4,25;March 1,29
      (1998 9:00 AM EDT)May 3,31

Monthly on the second to last Monday of the month for 6 months:
```

```
  DTSTART;TZID=US-Eastern:19970922T090000
  RRULE:FREQ=MONTHLY;COUNT=6;BYDAY=-2MO

  ==> (1997 9:00 AM EDT)September 22;October 20
      (1997 9:00 AM EST)November 17;December 22
      (1998 9:00 AM EST)January 19;February 16

Monthly on the third to the last day of the month, forever:

  DTSTART;TZID=US-Eastern:19970928T090000
  RRULE:FREQ=MONTHLY;BYMONTHDAY=-3

  ==> (1997 9:00 AM EDT)September 28
      (1997 9:00 AM EST)October 29;November 28;December 29
      (1998 9:00 AM EST)January 29;February 26
  ...

Monthly on the 2nd and 15th of the month for 10 occurrences:

  DTSTART;TZID=US-Eastern:19970902T090000
  RRULE:FREQ=MONTHLY;COUNT=10;BYMONTHDAY=2,15

  ==> (1997 9:00 AM EDT)September 2,15;October 2,15
      (1997 9:00 AM EST)November 2,15;December 2,15
      (1998 9:00 AM EST)January 2,15

Monthly on the first and last day of the month for 10 occurrences:

  DTSTART;TZID=US-Eastern:19970930T090000
  RRULE:FREQ=MONTHLY;COUNT=10;BYMONTHDAY=1,-1

  ==> (1997 9:00 AM EDT)September 30;October 1
      (1997 9:00 AM EST)October 31;November 1,30;December 1,31
      (1998 9:00 AM EST)January 1,31;February 1

Every 18 months on the 10th thru 15th of the month for 10
occurrences:

  DTSTART;TZID=US-Eastern:19970910T090000
  RRULE:FREQ=MONTHLY;INTERVAL=18;COUNT=10;BYMONTHDAY=10,11,12,13,14,
   15

  ==> (1997 9:00 AM EDT)September 10,11,12,13,14,15
      (1999 9:00 AM EST)March 10,11,12,13

Every Tuesday, every other month:

  DTSTART;TZID=US-Eastern:19970902T090000
  RRULE:FREQ=MONTHLY;INTERVAL=2;BYDAY=TU

  ==> (1997 9:00 AM EDT)September 2,9,16,23,30
      (1997 9:00 AM EST)November 4,11,18,25
      (1998 9:00 AM EST)January 6,13,20,27;March 3,10,17,24,31
  ...

Yearly in June and July for 10 occurrences:

  DTSTART;TZID=US-Eastern:19970610T090000
  RRULE:FREQ=YEARLY;COUNT=10;BYMONTH=6,7
```

```
  ==> (1997 9:00 AM EDT)June 10;July 10
      (1998 9:00 AM EDT)June 10;July 10
      (1999 9:00 AM EDT)June 10;July 10
      (2000 9:00 AM EDT)June 10;July 10
      (2001 9:00 AM EDT)June 10;July 10
  Note: Since none of the BYDAY, BYMONTHDAY or BYYEARDAY components
  are specified, the day is gotten from DTSTART

Every other year on January, February, and March for 10 occurrences:

  DTSTART;TZID=US-Eastern:19970310T090000
  RRULE:FREQ=YEARLY;INTERVAL=2;COUNT=10;BYMONTH=1,2,3

  ==> (1997 9:00 AM EST)March 10
      (1999 9:00 AM EST)January 10;February 10;March 10
      (2001 9:00 AM EST)January 10;February 10;March 10
      (2003 9:00 AM EST)January 10;February 10;March 10

Every 3rd year on the 1st, 100th and 200th day for 10 occurrences:

  DTSTART;TZID=US-Eastern:19970101T090000
  RRULE:FREQ=YEARLY;INTERVAL=3;COUNT=10;BYYEARDAY=1,100,200

  ==> (1997 9:00 AM EST)January 1
      (1997 9:00 AM EDT)April 10;July 19
      (2000 9:00 AM EST)January 1
      (2000 9:00 AM EDT)April 9;July 18
      (2003 9:00 AM EST)January 1
      (2003 9:00 AM EDT)April 10;July 19
      (2006 9:00 AM EST)January 1

Every 20th Monday of the year, forever:

  DTSTART;TZID=US-Eastern:19970519T090000
  RRULE:FREQ=YEARLY;BYDAY=20MO

  ==> (1997 9:00 AM EDT)May 19
      (1998 9:00 AM EDT)May 18
      (1999 9:00 AM EDT)May 17
  ...

Monday of week number 20 (where the default start of the week is
Monday), forever:

  DTSTART;TZID=US-Eastern:19970512T090000
  RRULE:FREQ=YEARLY;BYWEEKNO=20;BYDAY=MO

  ==> (1997 9:00 AM EDT)May 12
      (1998 9:00 AM EDT)May 11
      (1999 9:00 AM EDT)May 17
  ...

Every Thursday in March, forever:

  DTSTART;TZID=US-Eastern:19970313T090000
  RRULE:FREQ=YEARLY;BYMONTH=3;BYDAY=TH

  ==> (1997 9:00 AM EST)March 13,20,27
      (1998 9:00 AM EST)March 5,12,19,26
      (1999 9:00 AM EST)March 4,11,18,25
```

```
  ...

Every Thursday, but only during June, July, and August, forever:

  DTSTART;TZID=US-Eastern:19970605T090000
  RRULE:FREQ=YEARLY;BYDAY=TH;BYMONTH=6,7,8

  ==> (1997 9:00 AM EDT)June 5,12,19,26;July 3,10,17,24,31;
                        August 7,14,21,28
      (1998 9:00 AM EDT)June 4,11,18,25;July 2,9,16,23,30;
                        August 6,13,20,27
      (1999 9:00 AM EDT)June 3,10,17,24;July 1,8,15,22,29;
                        August 5,12,19,26
  ...

Every Friday the 13th, forever:

  DTSTART;TZID=US-Eastern:19970902T090000
  EXDATE;TZID=US-Eastern:19970902T090000
  RRULE:FREQ=MONTHLY;BYDAY=FR;BYMONTHDAY=13

  ==> (1998 9:00 AM EST)February 13;March 13;November 13
      (1999 9:00 AM EDT)August 13
      (2000 9:00 AM EDT)October 13
  ...

The first Saturday that follows the first Sunday of the month,
 forever:

  DTSTART;TZID=US-Eastern:19970913T090000
  RRULE:FREQ=MONTHLY;BYDAY=SA;BYMONTHDAY=7,8,9,10,11,12,13

  ==> (1997 9:00 AM EDT)September 13;October 11
      (1997 9:00 AM EST)November 8;December 13
      (1998 9:00 AM EST)January 10;February 7;March 7
      (1998 9:00 AM EDT)April 11;May 9;June 13...
  ...

Every four years, the first Tuesday after a Monday in November,
forever (U.S. Presidential Election day):

  DTSTART;TZID=US-Eastern:19961105T090000
  RRULE:FREQ=YEARLY;INTERVAL=4;BYMONTH=11;BYDAY=TU;BYMONTHDAY=2,3,4,
   5,6,7,8

  ==> (1996 9:00 AM EST)November 5
      (2000 9:00 AM EST)November 7
      (2004 9:00 AM EST)November 2
  ...

Every 3 hours from 9:00 AM to 5:00 PM on a specific day:

  DTSTART;TZID=US-Eastern:19970902T090000
  RRULE:FREQ=HOURLY;INTERVAL=3;UNTIL=19970902T170000Z

  ==> (September 2, 1997 EDT)09:00,12:00,15:00

Every 15 minutes for 6 occurrences:

  DTSTART;TZID=US-Eastern:19970902T090000
```

```
  RRULE:FREQ=MINUTELY;INTERVAL=15;COUNT=6

  ==> (September 2, 1997 EDT)09:00,09:15,09:30,09:45,10:00,10:15

Every hour and a half for 4 occurrences:

  DTSTART;TZID=US-Eastern:19970902T090000
  RRULE:FREQ=MINUTELY;INTERVAL=90;COUNT=4

  ==> (September 2, 1997 EDT)09:00,10:30;12:00;13:30

Every 20 minutes from 9:00 AM to 4:40 PM every day:

  DTSTART;TZID=US-Eastern:19970902T090000
  RRULE:FREQ=DAILY;BYHOUR=9,10,11,12,13,14,15,16;BYMINUTE=0,20,40
  or
  RRULE:FREQ=MINUTELY;INTERVAL=20;BYHOUR=9,10,11,12,13,14,15,16

  ==> (September 2, 1997 EDT)9:00,9:20,9:40,10:00,10:20,
                             ... 16:00,16:20,16:40
      (September 3, 1997 EDT)9:00,9:20,9:40,10:00,10:20,
                            ...16:00,16:20,16:40
  ...
```

B

Oracle Application Server Containers for J2EE Semantics

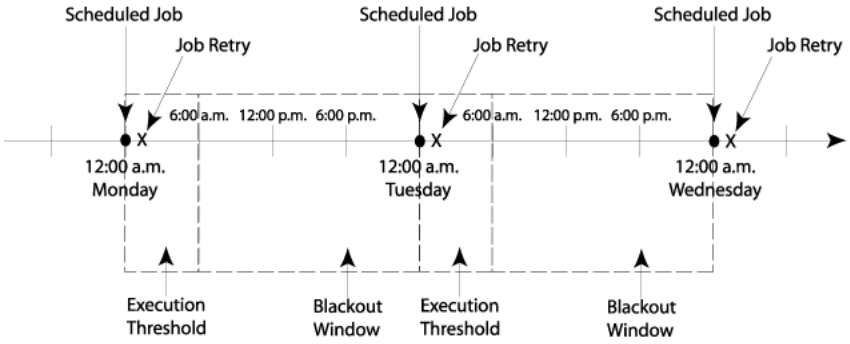
This appendix summarizes the differences in semantics among several closely related terms: remove, pause, cancel, retry, replay, and execution threshold. The following topics are covered:

- Semantics
- Job Precedence

B.1 Semantics

This section provides a summary of the differences in semantics of the various job operations. Figure B–1 illustrates the various job operations:

Figure B–1 Job Scheduler Semantics



Here is a job with a repeating schedule. The job is scheduled to run each night at midnight, with a retry period of 1 hour, an execution threshold of 6 hours, and a blackout window from 6:00 a.m. to midnight each day.

Removing a Job

Removing a job deletes the job definition, thus preventing the job from being run in the future. For example, if a job were removed at 2:00 a.m. on Monday, all future scheduled executions would be removed, and the job would not be run again on Tuesday, or Wednesday, or at any point in the future. If a job execution was running at the time the job was removed, the job execution would finish before the job was removed. To immediately stop a job execution that is running, you must cancel the job.

For more information about removing jobs, see Section 2.3.

Canceling a Job

Canceling a job stops the job execution while it is running. This is the only way to stop a job execution that is currently running. If you want to re-run the canceled job execution at some future point, you must specify a retry period.

For more information about canceling jobs, see Chapter 6.

Pausing a Job

Pausing a job causes future scheduled job executions to be skipped until the point at which the job is resumed. For example, if you pause a job at 2:00 a.m. on Monday, and resume the job at 2:00 a.m. on Tuesday, then the scheduled job execution at midnight on Tuesday would be skipped.

If you want the skipped job execution to run, set replay to true when you resume the job and the skipped job execution will be run when the job is resumed. If replay is set to false, then the skipped job execution is ignored and is not run.

It is important to note that replay will only attempt to run one skipped job execution. If you have a job with a repeating schedule and multiple job executions are skipped, replay will only attempt to run the first skipped job execution; all others are ignored.

For more information about pausing jobs, see Chapter 5.

Job Retry

If a job execution fails, job retry allows that job execution to be attempted again after a specified period of time. For example, if the job execution at midnight on Monday failed due to a power outage, that job execution would be attempted once at 1:00 a.m. If the job execution failed again, it would be ignored, and another attempt would not be made until the next scheduled run time (midnight on Tuesday).

For more information about job retry, see Section 3.2.

Execution Threshold

Execution threshold is essentially a time limit for a job to be run; if the job is not run within a specified period of time, then it is ignored and another attempt to run it is not made until the next scheduled run time. For example, a job scheduled to run nightly at midnight has a 6 hour execution threshold. If the job is not started by 6:00 a.m. on any day, then it is ignored and no attempt will be made to run the job again until midnight that night.

For more information about execution threshold, see Section 3.2.

Blackout Window

A blackout window is a period of time in which all job executions are suppressed. Any job execution that is scheduled to take place between 6:00 a.m. and midnight, for example (either directly scheduled or indirectly through a retry or replay), would be suppressed until the blackout window ends.

For more information about blackout windows, see Chapter 4.

B.2 Job Precedence

The combination of job semantics (for example, job retry or blackout windows) and associated operations (for example, pause and resume) requires precedence to ensure the overall correctness of Job Scheduler and resolve possible conflicts.

For example, consider the following scenarios:

- A job execution is resumed with replay during the time in which a blackout window is in effect. Does the job run?
- A job execution fails and is retried. Does execution threshold apply to the retry too?
- A job execution is paused and resumed with replay. Does execution threshold apply to the replay too?

Table B–1 specifies the precedence of these operations.

Table B–1 Precedence of Job Scheduler Operations

Precedence	Attribute/Operation	Discussion
1 (highest)	Blackout windows	All job executions are suppressed when a blackout window is active.
2	Job resume with replay	When a job execution is resumed with the replay parameter set to true, the job execution will run regardless of lower precedence attributes or operations.
3	Job retry	When a job execution fails, retry takes precedence over execution threshold or a (repeating) schedule end date. The same holds true if the job execution is paused after it is started, but before the retry period.
4	Job execution threshold	Execution threshold pertains to the initial running of the job execution only and not a retry (as a result of a failed attempt) or replay (as a result of resume).
5	Job pause	When a job execution is paused, execution may be postponed until the job is resumed. Both retry and replay take precedence.
6 (lowest)	Schedule end date	The job execution runs when not paused and falls within the execution threshold (if specified).

C

JSP Tag Library Reference

The Job Scheduler JSP Tag Library is used to interact with the Oracle Application Server Containers for J2EE. It is used to add, remove, pause, resume, and query jobs, as well as add, remove, and query blackout windows. In addition, a number of helper tags are provided for conditional operations on jobs and audit records (for example, distributing content based on the status of a job).

The following sections are covered:

C.1 Configuring an Application with the JSP Tag Library

Follow these steps to configure a web application with the JSP Tag Library:

1. Copy the JSP Tag Library descriptor file to the `/WEB-INF` subdirectory of your Web application.
2. Copy the JSP Tag library JAR file to the `/WEB-INF/lib` subdirectory of your Web application.
3. Add a `<taglib>` element to your Web application deployment descriptor in `/WEB-INF/web.xml`. For example:

```
<taglib>
<taglib-uri>scheduler-taglib</taglib-uri>
 <taglib-location>/WEB-INF/scheduler.tld</taglib-location>
</taglib>
```

To use the tags from this library in a JSP page, add the following directive at the top of each page:

```
<%@ taglib uri="scheduler-taglib" prefix="s" %>
```

The *s* is the tag name prefix for tags from this library, although any prefix can be specified.

C.2 JSP Tag Library Summary

Table C–1 provides a summary of the primary tags included in this library.

Table C–1 JSP Tag Library Summary

Tag	Description
scheduler	Top-level tag for all Job Scheduler tags.
addJob	Adds a new job.
removeJob	Removes an existing job.
pauseJob	Pauses a job.
resumeJob	Resumes a paused job.
cancelJob	Cancels a job.
addBlackoutWindow	Adds a new blackout window.
removeBlackoutWindow	Removes an existing blackout window.

C.3 JSP Tag Library Reference

This section provides reference information for each of the tags listed in Section C.2.

C.3.1 scheduler

The `scheduler` tag provides an implicit EJB context for interacting with Job Scheduler. Because context is implicit, this tag must be the parent of all other tags provided by this library.

Table C–2 describes the supported attributes for the `scheduler` tag.

Table C–2 scheduler Tag Attributes

Attribute	Required?	Description
`id`	Yes	Instance name of the Job Scheduler EJB by which the Job Scheduler may be accessed.
`name`	Yes	JNDI name of the Job Scheduler EJB responsible for processing all operations in the body of this tag.
`scope`	No	Scope[1] of the implicit EJB context. Valid values are `page`, `request`, `session`, or `application`. The default is `page`.

[1] This is equivalent to the JSP PAGECONTEXT scope.

Example:

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler"
scope="application">
.
.
.
</s:scheduler>
```

C.3.2 addJob

The `addJob` tag adds a new job to Job Scheduler. This tag must be enclosed within a `scheduler` tag.

Table C–3 describes the supported helper tags for the `addJob` tag.

Table C–3 Helper Tags for the addJob Tag

Helper Tag	Required?	Description
className	Yes	Class name of the job.
description	No	Description of the job.
schedule	No	Job schedule (specifies how often a timeout is sent to the trigger).
trigger	No	Job trigger (specifies a condition to be met before a job is run).
retry	No	Job retry period.
logLevel	No	Job log level.

C.3.2.1 className

Use this helper tag to specify the class name when adding a new job. This tag must be enclosed in an `addJob` tag.

Example:

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addJob>
      <s:className>TestJobImpl</s:className>
   </s:addJob>
</s:scheduler>
```

C.3.2.2 description

Use this helper tag to specify a job description when adding a new job. This tag must be enclosed in an `addJob` tag.

Example:

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addJob>
      <s:className>TestJobImpl</s:className>
      <s:description>Example job description</s:description>
   </s:addJob>
</s:scheduler>
```

C.3.2.3 schedule

Use this helper tag to specify a schedule for a job or blackout window. This tag must be enclosed in either an `addJob` or `addBlackoutWindow` tag.

Table C–4 describes the supported helper tags for the `schedule` helper tag.

Table C–4 Helper Tags for the schedule Helper Tag

Helper Tag	Required?	Description
duration	No	Initial expiration duration of the schedule.
interval	No	Repeat interval of the schedule.
threshold	No	Execution threshold for the job (applicable only when used in conjunction with the `addJob` tag).

C.3.2.3.1 duration

Use this helper tag to specify the initial expiration duration of the schedule. This tag must be enclosed in a `schedule` tag. The body of this tag is used to specify the

duration. The duration is specified as an arbitrary number of units and associated values or a specific date and time.

Some example durations are:

- 1 week
- 1 month, 5 days
- March 15, 2005
- January 5 2004 16:00:00 PST

Table C–5 describes the supported helper tags for the `duration` helper tag.

Table C–5 Helper Tags for the duration Helper Tag

Helper Tag	Required?	Description
date	No	Date of initial expiration. This tag can be combined with the `time` tag.
time	No	Time of initial expiration. This tag can be combined with the `date` tag.
years	No	Expiration in years relative to the time at which the job is submitted. This tag can be combined with any other helper tag.
months	No	Expiration in months relative to the time at which the job is submitted. This tag can be combined with any other helper tag.
weeks	No	Expiration in weeks relative to the time at which the job is submitted. This tag can be combined with any other helper tag.
days	No	Expiration in days relative to the time at which the job is submitted. This tag can be combined with any other helper tag.
hours	No	Expiration in hours relative to the time at which the job is submitted. This tag can be combined with any other helper tag.
minutes	No	Expiration in minutes relative to the time at which the job is submitted. This tag can be combined with any other helper tag.
seconds	No	Expiration in seconds relative to the time at which the job is submitted. This tag can be combined with any other helper tag.
milliseconds	No	Expiration in milliseconds relative to the time at which the job is submitted. This tag can be combined with any other helper tag.

Detailed descriptions and examples of the helper tags described in Table C–5 are provided in the following sections.

> **Note:** To avoid repetition, full code examples are provided for the first few tags. Partial code examples are provided for the remainder of the tags in this section.

date

Use this helper tag to specify an exact date for the initial expiration of a job. This tag must be enclosed in a `duration` tag. The format of the date expression must comply with the date parsing routines provided by the `java.text.DateFormat` class. If this tag is not used in conjunction with a `time` tag, the enclosing body of the `duration` tag uses the default time of 12:00:00 a.m.

Use the following code to set the schedule duration to October 27, 2003 12:00 a.m. PST.

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addJob>
      <s:className>TestJobImpl</s:className>
      <s:schedule>
         <s:duration>
            <s:date>October 27, 2003</s:date>
         </s:duration>
      </s:schedule>
   </s:addJob>
</s:scheduler>
```

time

Use this helper tag to specify an exact time for the initial expiration of a job. This tag must be enclosed in a `duration` tag. The format of the time expression specified must comply with the time parsing routines provided by the `java.text.TimeFormat` class. If this tag is not used in conjunction with a `date` tag, the enclosing body of the duration tag defaults to use the date on which the job was submitted.

Use the following code to set the schedule duration to October 27, 2003 4:30 p.m. PST.

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addJob>
      <s:className>TestJobImpl</s:className>
         <s:schedule>
            <s:duration>
               <s:date>October 27, 2003</s:date>
               <s:time>16:30:00 PST</s:date>
            </s:duration>
         </s:schedule>
   </s:addJob>
</s:scheduler>
```

years

Use this unit tag in conjunction with either the `duration` or `interval` tag to specify the number of years to expiration. This tag can be used in conjunction with any of the other unit tags (`months`, `weeks`, `days`, `hours`, or `minutes`). The body of the tag must be a positive non-zero integer. If the duration or interval occurs on a leap day, the expiration will be rounded to the last day of the month. For example, February 29 would be rounded to February 28 of the following year.

Use the following code to set the schedule duration to one year from the time of submission.

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addJob>
      <s:className>TestJobImpl</s:className>
      <s:schedule>
         <s:duration>
            <s:years>1</s:years>
         </s:duration>
      </s:schedule>
   </s:addJob>
</s:scheduler>
```

The following example shows how to set the schedule interval to 1 year:

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addJob>
```

```
      <s:className>TestJobImpl</s:className>
      <s:schedule>
         <s:interval>
            <s:years>1</s:years>
         </s:interval>
      </s:schedule>
   </s:addJob>
</s:scheduler>
```

months

Use this unit tag in conjunction with either the `duration` or `interval` tag to specify the number of months to expiration. This tag can be used in conjunction with any of the other unit tags (`years`, `weeks`, `days`, `hours`, or `minutes`). The body of the tag must be a positive non zero integer. If the duration or interval occurs at the end of the month, some rounding may occur so that the interval remains at the end of the month. For example, January 31 would be rounded to February 28 of the following month.

Use the following code to set the schedule to expire after 1 month.

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addJob>
      <s:className>TestJobImpl</s:className>
      <s:schedule>
         <s:duration>
            <s:months>1</s:months>
         </s:duration>
      </s:schedule>
   </s:addJob>
</s:scheduler>
```

The following example shows how to set the schedule to repeat every 3 months.

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addJob>
      <s:className>TestJobImpl</s:className>
      <s:schedule>
         <s:interval>
            <s:months>3</s:months>
         </s:interval>
      </s:schedule>
   </s:addJob>
</s:scheduler>
```

weeks

Use this unit tag in conjunction with either the `duration` or `interval` tag to specify the number of weeks to expiration. This tag can be used in conjunction with any of the other unit tags (`years`, `months`, `days`, `hours`, `minutes`, `seconds`, or `milliseconds`). The body of the tag must be a positive non zero integer.

Use the following code to set the schedule to expire after 1 week:

```
...
   <s:duration>
      <s:weeks>1</s:weeks>
   </s:duration>
...
```

days

Use this unit tag in conjunction with either the `duration` or `interval` tag to specify the number of days to expiration. This tag can be used in conjunction with any of the other unit tags (`years`, `months`, `weeks`, `hours`, `minutes`, `seconds`, or `milliseconds`). The body of the tag must be a positive non zero integer.

Use the following code to set the schedule to repeat every 14 days:

```
...
   <s:interval>
      <s:days>14</s:days>
   </s:interval>
...
```

hours

Use this unit tag in conjunction with either the `duration` or `interval` tag to specify the number of hours to expiration. This tag can be used in conjunction with any of the other unit tags (`years`, `months`, `weeks`, `hours`, `minutes`, `seconds`, or `milliseconds`). The body of the tag must be a positive non zero integer.

Use this code to set the schedule to expire after 48 hours:

```
...
   <s:duration>
      <s:hours>48</s:hours>
   </s:duration>
...
```

minutes

Use this unit tag in conjunction with either the `duration` or `interval` tag to specify the number of minutes to expiration. This tag can be used in conjunction with any of the other unit tags (`years`, `months`, `weeks`, `hours`, `days`, `seconds`, or `milliseconds`). The body of the tag must be a positive non zero integer.

Use the following code to set the schedule to repeat every 720 minutes:

```
...
   <s:interval>
      <s:minutes>720</s:minutes>
   </s:interval>
...
```

seconds

Use this unit tag in conjunction with either the `duration` or `interval` tag to specify the number of seconds to expiration. This tag can be used in conjunction with any of the other unit tags (`years`, `months`, `weeks`, `hours`, `days`, `minutes`, or `milliseconds`). The body of the tag must be a positive non zero integer.

Use the following code to set the schedule to expire after 86,400 seconds (24 hours):

```
...
   <s:duration>
      <s:seconds>86400</s:seconds>
   </s:duration>
...
```

milliseconds

Use this unit tag in conjunction with either the `duration` or `interval` tag to specify the number of milliseconds to expiration. This tag can be used in conjunction with any of the other unit tags (`years`, `months`, `weeks`, `hours`, `days`, `minutes`, or `seconds`). The body of the tag must be a positive non zero integer.

Use the following code to set the schedule to repeat every 43,200,000 milliseconds:

```
...
   <s:interval>
      <s:milliseconds>43200000</s:milliseconds>
   </s:interval>
...
```

C.3.2.3.2 interval

Use this helper tag to specify the interval of the schedule. This tag must be enclosed in a `schedule` tag. The body of this tag is used to specify the schedule interval. The interval is specified as an arbitrary number of units and associated values.

Some example intervals are:

- 1 week
- 1 month, 5 days
- 1 month, 6 days, 3 hours

Table C–6 describes the supported helper tags for the `interval` helper tag.

Table C–6 Helper Tags for the interval Helper Tag

Helper Tag	Required?	Description
`end`	No	End date of a repeating interval.
`years`	No	Repeating interval in years relative to the time at which the job is submitted. This tag may be combined with any other unit tag.
`months`	No	Repeating interval in months relative to the time at which the job is submitted. This tag may be combined with any other unit tag.
`weeks`	No	Repeating interval in weeks relative to the time at which the job is submitted. This tag may be combined with any other unit tag.
`days`	No	Repeating interval in days relative to the time at which the job is submitted. This tag may be combined with any other unit tag.
`hours`	No	Repeating interval in hours relative to the time at which the job is submitted. This tag may be combined with any other unit tag.
`minutes`	No	Repeating interval in minutes relative to the time at which the job is submitted. This tag may be combined with any other unit tag.

Use the `end` helper tag to specify an end date for a repeating interval. This tag must be enclosed in an `interval` tag. The interval is specified as an arbitrary number of units and associated values.

Use the following code to set an end date of 1 year for a monthly repeating interval:

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addJob>
      <s:className>TestJobImpl</s:className>
      <s:schedule>
         <s:interval>
```

```
            <s:months>1</s:months>
            <s:end>
               <s:years>1</s:years>
            </s:end>
         </s:interval>
      </s:schedule>
   </s:addJob>
</s:scheduler>
```

Table C–7 describes the unit tags supported be the `end` helper tag.

Table C–7 Helper Tags for the end Helper Tag

Helper Tag	Required?	Description
`date`	No	Date on which the schedule ends. This tag can be combined with the `time` tag.
`time`	No	Time at which the schedule ends. This tag can be combined with the `date` tag.
`years`	No	End date in years relative to the time at which the job is submitted. This tag can be combined with any other unit tag.
`months`	No	End date in months relative to the time at which the job is submitted. This tag can be combined with any other unit tag.
`weeks`	No	End date in weeks relative to the time at which the job is submitted. This tag can be combined with any other unit tag.
`days`	No	End date in days relative to the time at which the job is submitted. This tag can be combined with any other unit tag.
`hours`	No	End date in hours relative to the time at which the job is submitted. This tag can be combined with any other unit tag.
`minutes`	No	End date in minutes relative to the time at which the job is submitted. This tag can be combined with any other unit tag.

These unit tags are used in the same manner as with the `duration` helper tag. For more information, see Section C.3.2.3.1.

C.3.2.3.3 threshold

Use this helper tag to specify the execution threshold of the schedule; if the schedule is not run before the specified threshold, the job is suppressed and will be retried only if a `retry` tag is specified. This tag must be enclosed in a `schedule` tag. The body of this tag is used to specify the schedule threshold. The threshold is specified as an arbitrary number of units and associated values.

Some example thresholds are:

- 1 day
- 10 hours, 26 minutes

Table C–8 describes the helper tags available for the `threshold` tag.

Table C–8 Helper Tags for the threshold Helper Tag

Helper Tag	Required?	Description
`days`	No	Threshold in days. This tag can be combined with any of the other unit tags.
`hours`	No	Threshold in hours. This tag can be combined with any of the other unit tags.

Table C–8 (Cont.) Helper Tags for the threshold Helper Tag

Helper Tag	Required?	Description
`minutes`	No	Threshold in minutes. This tag can be combined with any of the other unit tags.

These unit tags are used in the same manner as with the `duration` helper tag. For more information, see Section C.3.2.3.1.

C.3.2.4 trigger

Use this helper tag to specify the trigger for the job. This tag must enclosed in an `addJob` tag. The body is used to specify the associated expression for the trigger. If no trigger is specified, a default trigger is provided for execution based on the associated schedule's expiration.

Example:

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addJob>
      <s:className>TestJobImpl</s:className>
      <s:trigger>do_it_now</s:trigger>
   </s:addJob>
</s:scheduler>
```

C.3.2.5 retry

Use this helper tag to specify the retry period for the job. This tag must be enclosed in an `addJob` tag. The body of this tag is used to specify the retry period and is specified as an arbitrary number of units and associated values.

Table C–9 describes the supported helper tags for the `retry` helper tag:

Table C–9 Helper Tags for the retry Helper Tag

Helper Tag	Required?	Description
`months`	No	Retry period in months. This tag can be combined with any of the other unit tags.
`weeks`	No	Retry period in weeks. This tag can be combined with any of the other unit tags.
`days`	No	Retry period in days. This tag can be combined with any of the other unit tags.
`hours`	No	Retry period in hours. This tag can be combined with any of the other unit tags.
`minutes`	No	Retry period in minutes. This tag can be combined with any of the other unit tags.

If a job fails and you want it to retry in 1 hour and 30 minutes, then use the following code:

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addJob>
      <s:className>TestJobImpl</s:className>
      <s:retry>
         <s:hours>1</s:hours>
         <s:minutes>30</s:minutes>
      </s:retry>
```

```
   </s:addJob>
</s:scheduler>
```

The unit tags described in Table C–9 are used in the same manner as with the `duration` helper tag. For more information, see Section C.3.2.3.1.

C.3.2.6 logLevel

Use this helper tag to set the log level for the job. This tag must be enclosed in an `addJob` tag. The body of this tag is used to specify the log level. The following values are supported:

- WARNING

  Logs a message each time a job results in a run time exception. This is the lowest logging level.

- FINE

  Logs a message each time a job begins and ends.

- FINER

  Logs a message each time the job's associated trigger is evaluated, and logs the result of the evaluation.

- FINEST

  Logs a message when the job completes, and logs the cumulative time it took to run This is the highest level of logging.

C.3.3 removeJob

Use this tag to remove an existing job from the Job Scheduler. This tag must be enclosed within a `scheduler` tag. Specify the `oracle.ias.scheduler.Job` bean instance name of the job you want to remove.

The `name` attribute is the only supported attribute for the `removeJob` tag. It is an optional attribute used to specify the bean instance name by which the associated job is accessed.

The following example shows how to remove a job named "job":

```
<%@ taglib uri="scheduler-taglib" prefix="s" %>
<jsp:useBean id="job" class="oracle.ias.scheduler.Job" scope="session"/>
<s:scheduler id="scheduler"  name="java:comp/env/ejb/scheduler">
   <s:removeJob name="job"/>
</s:scheduler>
```

C.3.4 pauseJob

Use this tag to pause an existing job in Job Scheduler. This tag must be enclosed within a `scheduler` tag. Specify the `oracle.ias.scheduler.Job` instance name of the job you want to pause.

The `name` attribute is the only supported attribute for the `pauseJob` tag. It is an optional attribute used to specify the bean instance name by which the associated job is accessed.

The following example shows how to pause a job named *job*:

```
<%@ taglib uri="scheduler-taglib" prefix="s" %>
<jsp:useBean id="job" class="oracle.ias.scheduler.Job" scope="session"/>
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
```

```
   <s:pauseJob name="job"/>
</s:scheduler>
```

C.3.5 resumeJob

Use this tag to resume a job in Job Scheduler. This tag must be enclosed within a `scheduler` tag. Specify the `oracle.ias.scheduler.Job` instance name of the job you want to resume.

The `name` attribute is the only supported attribute for the `resumeJob` tag. It is an optional attribute used to specify the bean instance name by which the associated job is accessed.

The following example shows how to resume a job named *job*:

```
<%@ taglib uri="scheduler-taglib" prefix="s" %>
<jsp:useBean id="job" class="oracle.ias.scheduler.Job" scope="session"/>
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:resumeJob name="job"/>
</s:scheduler>
```

C.3.6 cancelJob

Use this tag to cancel an existing job in Job Scheduler. This tag must be enclosed within a `scheduler` tag. Specify the `oracle.ias.scheduler.Job` instance name of the job you want to cancel.

The `name` attribute is the only supported attribute for the `cancelJob` tag. It is an optional attribute used to specify the bean instance name by which the associated job is accessed.

The following example shows how to cancel a job named *job*:

```
<%@ taglib uri="scheduler-taglib" prefix="s" %>
<jsp:useBean id="job" class="oracle.ias.scheduler.Job" scope="session"/>
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:cancelJob name="job"/>
</s:scheduler>
```

C.3.7 addBlackoutWindow

Use this tag to add a blackout window to Job Scheduler. This tag must be enclosed within a `scheduler` tag

Table C–10 describes the helper tags supported by the `addBlackoutWindow` tag.

Table C–10 Helper Tags for the addBlackoutWindow Tag

Helper Tag	Required?	Description
`description`	Yes	Description of the blackout window.
`duration`	Yes	Duration of the blackout window.
`schedule`	No	Schedule for the blackout window; specifies when, how often, and for how long the blackout window is in effect. If no schedule is specified, the blackout window is effective starting at the time of submission.

The following example adds a blackout window lasting 2 hours, effective immediately:

```
<%@ taglib uri="scheduler-taglib" prefix="s" %>
```

```
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addBlackoutWindow>
      <s:description>two hour downtime, effective immediately</s:description>
         <s:duration>
            <s:hours>2</s:hours>
      </s:duration>
   </s:addBlackoutWindow>
</s:scheduler>
```

C.3.8 removeBlackoutWindow

Use this tag to remove an existing blackout window from Job Scheduler. This tag must be enclosed within a `scheduler` tag.

The following example shows how to remove a an existing blackout window using the blackout window's description. (See Section C.3.7, where the blackout window was created).

```
<%@ taglib uri="scheduler-taglib" prefix="s" %>
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:removeBlackoutWindow>two hour downtime, effective
immediately</s:removeBlackoutWindow>
</s:scheduler>
```

C.4 JSP Tag Library Examples

This section contains more comprehensive examples illustrating the use of the various tags described in this chapter.

Example C–1 shows how to list all submitted jobs.

Example C–1 Listing All Submitted Jobs

```
<%@ taglib uri="scheduler-taglib" prefix="s" %>
<HTML>
<HEAD>
<TITLE>OracleAS Job Scheduler: all jobs</TITLE>
</HEAD>
<BODY>
<TABLE>
   <TR>
      <TH>Description</TH>
      <TH>Class Name</TH>
   </TR>
   <s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
      <s:jobIterator id="job">
         <TR>
            <TD><jsp:getProperty name="job" property="Description"/></TD>
            <TD><jsp:getProperty name="job" property="ClassName"/></TD>
         </TR>
      </s:jobIterator>
   </s:scheduler>
</TABLE>
</BODY>
</HTML>
```

Example C–2 shows how to submit a job to Job Scheduler. In this example, an HTTP request is sent to the JSP page. Once the request is processed, the JSP forwards the request to a status page. The parameters in the request are described in the following table:

Parameter	Description
`description`	Job description.
`class`	Job implementation class name.
`expirationDate`	Job schedule's expiration date.
`expirationTime`	Job schedule's time of expiration on the specified `expirationDate`.
`intervalDays`	Job schedule's repeat interval.

Example C–2 Submitting a Job to Job Scheduler

```
<%@ taglib uri="scheduler-taglib" prefix="s" %>
<jsp:useBean id="params" scope="request" class="RequestParametersBean" />
<jsp:setProperty name="params" property="*" />
<HTML>
<BODY>
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:addJob>
      <s:description><jsp:getProperty name="params"
property="description"/></s:description>
      <s:class><jsp:getProperty name="params" property="class"/></s:className>
      <s:schedule>
         <s:duration><jsp:getProperty name="params"
property="expirationDate"/></s:duration>
         <s:interval><jsp:getProperty name="params"
property="expirationTime"/></s:interval>
      </s:schedule>
   </s:addJob>
</s:scheduler>
<jsp:forward url="/JobScheduled.html"/>
</BODY>
</HTML>
```

Example C–3 shows how to remove a job from Job Scheduler. In this example, a job is removed based on its description and class. Once the job has been removed, the JSP forwards the request to a status page.

Example C–3 Removing a Job from Job Scheduler

```
<%@ taglib uri="scheduler-taglib" prefix="s" %>
<HTML>
<BODY>
<s:scheduler id="scheduler" name="java:comp/env/ejb/scheduler">
   <s:jobIterator id="job" desc="description of job to remove" class="test">
      <s:removeJob/>
   </s:jobIterator>
</s:scheduler>
<jsp:forward url="/JobRemoved.html"/>
</BODY>
</HTML>
```

D

JMX MBean Reference

This appendix contains detailed information about the JMX MBeans provided by Job Scheduler and their attributes and values. The following topics are covered:

- Job Management Bean Attributes
- Job Scheduler Management Bean Attributes
- Job Scheduler Aggregation Management Bean Attributes

D.1 Job Management Bean Attributes

Table D–1 summarizes the Job MBean attributes for monitoring the job.

Table D–1 Job Management Bean Attributes

Attribute	Access	Type	Description
`Description`	Read-only	`java.lang.string`	Job description.
`ClassName`	Read-only	`java.lang.string`	Job implementation class name. This class provides an implementation of the `oracle.ias.scheduler.Executable` interface.
`Schedule`	Read-only	`oracle.ias.scheduler.Schedule`	Job schedule.
`Triggers`	Read-only	`oracle.ias.scheduler.Trigger`	Job trigger.
`Properties`	Read-only	`java.util.Properties`	Job properties.
`State`	Read-only	`java.lang.string`	Job state (`ACTIVE`, `PAUSED`, or `COMPLETE`).
`LogLevel`	Read/ write	`java.lang.string`	String representation of log levels for the job. Possible values are `ALL`, `OFF`, `SEVERE`, `WARNING`, `CONFIG`, `INFO`, `FINE`, `FINER`, and `FINEST`.
`ExecutionThreshold`	Read-only	`long`	Job execution threshold (in milliseconds).
`RetryPeriod`	Read-only	`long`	Job retry period (in milliseconds).

Table D–2 summarizes the operations provided for configuring the job.

Table D–2 Job Management Bean Operations

Operation Name	Parameters	Return Type	Description
`pause`	None	None	Pauses the job.
`resume`	replay:boolean	None	Resumes the job. If the parameter is true, then replay the job if the trigger is set on resumption.
`cancel`	None	None	Cancels any currently running jobs.

Table D–3 summarizes the DMS metrics for a job.

Table D–3 Job Management Bean DMS Metrics

Metric Name	Metric Type
`Notifications`	count
`Execution`	average
`FailedExecutions`	value
`CancelledExecutions`	value
`SuccessfulExecutions`	value
`BlackoutExecutions`	value
`ExceededThresholdExecutions`	value

For more information about the metric types, see *Oracle Application Server Performance Guide*.

D.2 Job Scheduler Management Bean Attributes

Table D–4 summarizes Job Scheduler attributes for monitoring Job Scheduler.

Table D–4 Job Scheduler Management Bean Attributes

Attribute	Access	Type	Description
`jobs`	Read-only	`java.util.Collection`	All submitted jobs represented as a collection of `javax.management.ObjectName` objects, each of which references its associated job management bean instance.
`jobstoreProvider`	Read-only	`java.lang.string`	Class name of the configured job store provider implementation. The class specified implements the `oracle.ias.scheduler.jobstore.JobStoreProvider` interface.

Table D–5 summarizes the operations provided for configuring Job Scheduler.

Table D–5 Job Scheduler Management Bean Operations

Operation Name	Parameters	Return Type	Description
`addBlackoutWindow`	`java.lang.String Windowname, java.lang.String datetime, long durationMinutes`	None	Create a new execution blackout window with the specified window name. The date, time, and length of time the blackout window is in effect. The format of the date/time string must conform to the requirement of `java.text.DateFormat` using the `java.text.DateFormat.FULL` style for both date and time components. The duration is specified in minutes.
`removeBlackoutWindow`	`java.lang.String windowName`	None	Remove a previously defined execution blackout window identified by the specified window name.
`listBlackoutWindows`	None	`java.util.Collection`	List the names of all defined blackout windows.

Table D–6 summarizes the DMS metrics for a Job Scheduler.

Table D–6 Job Scheduler Management Bean DMS Metrics

Metric Name	Metric Type
`ExecJobInstances`	value
`ActiveJobs`	value
`PausedJobs`	value
`CompletedJobs`	value
`Notifications`	count

For more information about the metric types, see *Oracle Application Server Performance Guide*.

D.3 Job Scheduler Aggregation Management Bean Attributes

Table D–7 summarizes the Job Scheduler Aggregation MBean.

Table D–7 Job Scheduler Aggregation Management Bean Attributes

Attribute	Access	Type	Description
`schedulers`	Read-only	`java.util.Collection`	All Job Scheduler instances represented as a collection of `javax.management.ObjectName` objects, each of which references its associated Job Scheduler management bean instance.
`jobs`	Read-only	`java.util.Collection`	All job instances represented as a collection of `javax.management.ObjectName` objects, each of which references its associated job management bean instance.

Table D–8 summarizes the clusterwide operations provided to configure the Job Scheduler Aggregation MBean.

Table D–8 Job Scheduler Aggregation Management Bean Operations

Operation Name	Parameters	Return Type	Description
`pause`	none	None	Pause all jobs across all Job Scheduler instances.
`resume`	`replay; boolean`	None	Resume all paused jobs across all Job Scheduler instances. Replay jobs whose triggers are set on resumption.
`cancel`	None	None	Cancel any currently running jobs across all Job Scheduler instances.
`addBlackoutWindow`	`java.lang.String windowname java.lang.String datetime long durationMinutes`	None	Create a new blackout window across all Job Scheduler instances with the specified window name. The date, time, and duration determine when and for how long the blackout window is in effect. The format of the date/time string must conform to format required by `java.text.DateFormat`. The duration is specified in minutes.

D.4 Frequently Asked Questions About JMX MBeans

Does Job Scheduler expose a management interface?

Job Scheduler does expose a management interface that can be access through JMX MBeans. Two kinds of MBeans are published: `SchedulerMBean` and `JobMBean`. The former is used to manage a Job Scheduler instance and the later a specific job instance. These MBeans are accessible from the OC4J Administration Console.

Can I enable and disable MBean publication?

Yes. If the `<env-entry>` value of `oracle.ias.scheduler.jmx` is set to true, then MBean publication is enabled; otherwise, the beans are not published.

What privileges are required to access the Job, Job Scheduler, and Job Scheduler Aggregation MBeans?

Access to the Job and Job Scheduler MBeans requires the same privileges as those of the user application in which they are defined. Access to the Job Scheduler Aggregation MBean requires OC4J administrator privileges.

E

Troubleshooting Oracle Application Server Containers for J2EE

This appendix describes tools and methods that can be used to troubleshoot Oracle Application Server Containers for J2EE or any scheduler-based applications. The following topics are covered:

E.1 Oracle Diagnostic Logging (ODL)

To simplify integration with Oracle Application Server, the standard JDK1.4.1 `java.util.logging` APIs are used. These APIs make a clear separation of the logging APIs (`java.util.logging.Logger`) from the APIs that control writing logged messages to various destinations (`java.util.logging.Handler`) and also from APIs that control message formatting and localization (`java.util.logging.Formatter`). The message are logged in ODL format using Oracle's ODL handler.

E.1.1 Types of Logging

Job Scheduler provides the following types of logging:

- Run time logging, which is performed on behalf of the scheduler subsystem (for example, a warning message as a result of misconfiguration).
- Job logging, which is related to a job's execution. There are two types of job logging:
  - Implicit Job Logging. This type of logging is primarily performed by Job Scheduler and is specified as part of the job's definition.
  - Explicit Job Logging. This type of logging is performed by the actual job implementation, meaning that it is user-defined.

E.1.1.1 Implicit Job Logging

Implicit job logging is specified as part of the job's definition. Because Job Scheduler uses the Java logging APIs, log levels are specified using the log levels provided by the `java.util.logging.Level` class.

The default level of implicit logging is set at `Level.FINER`. Job Scheduler uses only a subset of these levels to log messages.

If the logging level is set to `Level.WARNING`, log entries are written under the following conditions:

- Running the job resulted in a `JobExecutionException` exception.
- Running the job resulted in a `RuntimeException` exception.

If the logging level is set to a value of `Level.FINE`, the following additional information is written to the log:

- Date and time at which the job started.
- Date and time at which the job ended.

If the logging level is set to a value of `Level.FINER`, the following additional information is written to the log:

- Date and time at which the associated trigger evaluated a notification, and the result of the evaluation.

If the logging level is set to a value of `Level.FINEST`, the following additional information is written to the log:

- Total elapsed time of the job.

Each log entry contains the following:

- Job description
- Job implementation class name
- Date and time
- Stack trace (if the job results in an exception)
- Associated message parameters

E.1.1.2 Explicit Job Logging

The same logging facilities used by implicit job logging are also available to the job implementation when the job runs. The logging context is available through the context that is passed to the job when it is run through the `JobContext` object:

```
public interface JobContext extends Serializable {
   public Job getJob();
   public java.util.logging.Logger getLogger();
   public java.util.logging.Logger getLogger(String resourceBundleName);
}
```

Either of the `getLogger()` methods can be used, but the latter method allows a resource bundle to be specified.

E.1.2 Configuring the Global Log Levels

You can configure the global log level of Job Scheduler. For more information, see Section 9.4.

E.1.3 Logging Example

Example E–1 shows how to add logging capabilities to a job. Specifically, an information log entry is written before every file is copied. This is done by retrieving

the logger from the job context and writing an informational log message before performing the copy command.

Example E–1 Job Implementation with Logging

```
import java.io.File;
import java.io.IOException;
import java.util.logging.Logger;
import oracle.ias.scheduler.Job;
import oracle.ias.scheduler.Executable;
import oracle.ias.scheduler.Cancellable;
import oracle.ias.scheduler.JobContext;
import oracle.ias.scheduler.JobCancellationException;
import oracle.ias.scheduler.JobExecutionException;


public class CancellableBackupJobLogged implements Executable, Cancellable {

    boolean m_cancelled = false;

    public void cancel() {
        m_cancelled = true;
    }

    public void execute(JobContext context) throws JobExecutionException,
JobCancellationException {

        // retrieve the source/destination directories
        Job job = context.getJob();
        Logger log = context.getLogger();
        String source = job.getProperties().getProperty("SourceDirectory");
        String destination =
job.getProperties().getProperty("DestinationDirectory");

        // get the list of files to copy
        File directory = new File(source);
        File[] files = directory.listFiles();

        // copy the files
        Runtime runtime = Runtime.getRuntime();
        Process process;
        for (int x = 0; x < files.length; x++) {

            // cancelled?
            if (m_cancelled) {
                throw new JobCancellationException();
            }

            log.info("copying file "+files[x]);
            try {
                process = runtime.exec("/bin/cp " + files[x].toString() +
                                       " " + destination);
                process.waitFor();
            } catch(IOException e) {
                throw new RuntimeException("copy failed: "+files[x],e);
            } catch(InterruptedException e) {
                throw new RuntimeException("copy failed: "+files[x],e);
            }
        }
    }
```

```
}
```

The log level can be set by invoking the appropriate API on the logger. For example:

```
<code>
      Logger logger = jobContext.getLogger();
      logger.setLoglevel(Level.FINEST);
</code>
```

> **Note:** Since all jobs share a single logger instance, setting the log level will affect the logging of all subsequent messages for all instances of the job.

E.2 DMS Metrics

Oracle Dynamic Monitoring Service (DMS) is used to measure application specific performance information. Two types of metrics are provided:

- Scheduler metrics. Provides statistics pertaining to a specific Job Scheduler instance (for example, total number of executing job instances, total number of active jobs, or total number of paused jobs).
- Job metrics. Provides aggregate job statistics as well as information pertaining to a specific job (for example, the job description, state of the job, or number of failed executions).

Table E–1 lists the information provided by scheduler metrics:

Table E–1 Statistic Types for Scheduler Metrics

Metric Name	Description
`schedulerStartTime`	`System.currentTimeMillis()` when Job Scheduler starts.
`executingJobs`	Total number of job instances that are currently running.
`activeJobs`	Total number of active jobs.
`pausedJobs`	Total number of paused jobs.
`completedJobs`	Total number of completed jobs.

Table E–2 lists the information provided by job metrics:

Table E–2 Statistic Types for JobStats

Metric Name	Description
`jobSchedule`	String version of the schedule.
`jobTrigger`	String version of the trigger.
`jobLogLevel`	Log level.
`jobClassName`	String version of the class name.
`jobDescription`	Job description.
`jobExecutionThreshold`	Execution threshold (in millliseconds).
`jobRetryPeriod`	Retry period (in milliseconds).
`jobState`	State of the job (active, paused, or complete)
`execution`	Duration for which an instance of this job runs.

Table E–2 (Cont.) Statistic Types for JobStats

Metric Name	Description
`failedExecutions`	Number of failed runs.
`cancelledExecutions`	Number of canceled runs since JVM startup.
`successfulExecutions`	Number of successful runs since JVM startup.
`blackoutExecutions`	Number of runs that were blacked out since JVM startup.
`exceededThresholdExecutions`	Number of executions that exceeded the execution threshold since JVM startup.

For more information about DMS, please refer to the *Oracle Application Server Performance Guide*.

E.3 Frequently Asked Questions About Job Scheduler Monitoring

How do I monitor Job Scheduler activities?

You can connect to the DMS Spy servlet to look at statistics for Job Scheduler and its various jobs (for example, the number of currently active, completed, and currently running jobs). For each job, information is provided about its current state, duration, and result (for example, whether or not the job failed). For detailed information, see the DMS Addendum.

Can I disable DMS statistics collection?

Yes. For more information, see Chapter 9.

E.4 Frequently Asked Questions About Job Scheduler Logging

How do I configure logging for Job Scheduler?

For more information about configuring logging for Job Scheduler, see Chapter 9.

How can I use logging to troubleshoot problems?

To troubleshoot a particular job, you can increase the granularity of the log messages by changing the log level of the particular job in question. This can be accomplished dynamically through the job MBean management interface in the Administration Console, or by directly invoking the remote scheduler interface. The global root logger's default level is set to `Level.WARNING`.

Index

J

L

M

N

O

P

R

S

T