



BEA WebLogic Server™

Programming WebLogic Time Services (Deprecated)

BEA WebLogic Server Version 6.1
Document Date: September 19, 2001

Copyright

Copyright © 2001 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, WebLogic, Tuxedo, and Jolt are registered trademarks of BEA Systems, Inc. How Business Becomes E-Business, BEA WebLogic E-Business Platform, BEA Builder, BEA Manager, BEA eLink, BEA WebLogic Commerce Server, BEA WebLogic Personalization Server, BEA WebLogic Process Integrator, BEA WebLogic Collaborate, BEA WebLogic Enterprise, and BEA WebLogic Server are trademarks of BEA Systems, Inc..

All other product names may be trademarks of the respective companies with which they are associated.

Programming WebLogic Server Time Services

Part Number	Document Date	Software Version
N/A	July 30, 2001	BEA WebLogic Server Version 6.1

Contents

About This Document

Audience.....	v
e-docs Web Site.....	v
How to Print the Document.....	vi
Contact Us!.....	vi
Documentation Conventions	vii

1. Programming WebLogic Time Services (Deprecated)

Time Services Deprecated.....	1-1
Overview	1-2
WebLogic Time Architecture.....	1-2
WebLogic Time API	1-3

2. Implementing with WebLogic Time

Scheduling a Recurring Trigger on a Client.....	2-1
Scheduling a Recurring Server-side Trigger from a WebLogic Client.....	2-3
Step 1. Implement the ScheduleDef and TriggerDef interfaces.....	2-3
Step 2. Create the ScheduledTrigger from a WebLogic Client.....	2-5
Setting up Complex Schedules	2-6
Rescheduling	2-7
Stopping a ScheduledTrigger	2-8



About This Document

This document describes the architecture of the WebLogic Time Services that runs on BEA WebLogic Server™.

The document is organized as follows:

- Chapter 1, “Programming WebLogic Time Services (Deprecated),” is an overview of the architecture of the WebLogic Time Services.
- Chapter 2, “Implementing with WebLogic Time,” describes how to implement the WebLogic Time Services.

Audience

This document is written for application developers who want to implement Time Services for their applications. It is assumed that readers know Web technologies, object-oriented programming techniques, and the Java programming language.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

How to Print the Document

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

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

Adobe Acrobat Reader is available at no charge from the Adobe Web site at <http://www.adobe.com>.

Contact Us!

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

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at <http://www.bea.com>. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

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

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using

-
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Usage
Ctrl+Tab	Keys you press simultaneously.
<i>italics</i>	Emphasis and book titles.
monospace text	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard. <i>Examples:</i> <pre>import java.util.Enumeration; chmod u+w * config/examples/applications .java config.xml float</pre>
<i>monospace italic text</i>	Variables in code. <i>Example:</i> <pre>String CustomerName;</pre>
UPPERCASE TEXT	Device names, environment variables, and logical operators. <i>Examples:</i> <pre>LPT1 BEA_HOME OR</pre>
{ }	A set of choices in a syntax line.

Convention	Usage
[]	Optional items in a syntax line. <i>Example:</i> <pre>java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]</pre>
	Separates mutually exclusive choices in a syntax line. <i>Example:</i> <pre>java weblogic.deploy [list deploy undeploy update] password {application} {source}</pre>
...	Indicates one of the following in a command line: <ul style="list-style-type: none"> ■ An argument can be repeated several times in the command line. ■ The statement omits additional optional arguments. ■ You can enter additional parameters, values, or other information
.	Indicates the omission of items from a code example or from a syntax line.

1 Programming WebLogic Time Services (Deprecated)

The WebLogic Time API provides a mechanism for scheduling actions (triggers) to take place at a future date and time, or on a regularly recurring schedule. The following sections provide an introduction to the Time Services features:

- Time Services Deprecated
- Overview
- WebLogic Time Architecture
- WebLogic Time API

Time Services Deprecated

WebLogic Time Services are deprecated in WebLogic Server Version 6.1. See J2EE Scheduling Tools in [WebLogic Server Tools](#) for third-party scheduling services.

Overview

The Time API allows any user-written trigger to be scheduled and then executed, either in the client JVM, or on WebLogic Server on behalf of a client. The Time API provides a dependable, distributable method of setting up actions that occur automatically.

Note: Although you can use the time service with individual WebLogic Server instances in a cluster, the service itself is non-clusterable. The WebLogic Time API does not make use of cluster features such as load balancing and failover.

WebLogic Time Architecture

WebLogic Time is a lightweight, efficient API that shares many characteristics of other WebLogic Server APIs. WebLogic Time is built around a *ScheduledTriggerDef* object, constructed from a *Schedulable* object. The *ScheduledTriggerDef* object is responsible for starting, stopping, or repeating the action schedule. A *Triggerable* object defines the action to be carried out on schedule. You use an object factory to create a *ScheduledTrigger*. Object factories provide a well-defined, easy-to-use methodology for managing scarce resources within WebLogic Server.

Accounting for scheduling is kept in a series of efficient linked lists that are sorted only at the most proximate chronological point as new triggers are scheduled and then acted upon. For example, a trigger for a week from Tuesday at 12:15:30 is initially inserted into the schedule for next Tuesday. Not until noon on Tuesday is the schedule for the noon hour sorted, and not until fifteen minutes past noon are the triggers for that minute sorted. This drastically reduces the overhead for scheduling in a heavily scheduled environment.

WebLogic Server also keeps accounting of the differences in time zone, clock accuracy, and latency between users of the Time service. Note that WebLogic triggers are not real time triggers that can be used to millisecond granularity. WebLogic triggers used properly will function reliably within an estimated 1 second of accuracy.

WebLogic Time API

A `ScheduledTrigger` takes two objects in its constructor:

- An object that implements either `weblogic.time.common.Schedulable` or `weblogic.time.common.ScheduleDef`
- An object that implements either `weblogic.time.common.Triggerable` or `weblogic.time.common.TriggerDef`

An object passed to the `ScheduledTrigger` object factory method may also be a client-side object, in which case the client creates, schedules, and executes a `ScheduledTrigger` within its own JVM. The client-side object must implement `Schedulable` (or `ScheduleDef`) and `Triggerable` (or `TriggerDef`).

The `TimeServicesDef` interface also provides methods for obtaining time-related information about client and server:

- `currentTimeMillis()` returns the current server time, in “local server time” format, which is the server’s time adjusted for propagation delay between the method invoker and the server (zero when the method invoker *is* the server, and some positive milliseconds when the invoker is the client or another WebLogic Server).
- `getRoundTripDelayMillis()` returns the number of milliseconds of round-trip delay between the client and server. This method depends on the algorithm described in the overview.
- `getLocalClockOffsetMillis()` returns the number of milliseconds of offset between the client and server clocks, based on the algorithm described in the overview.

The `weblogic.time.common.TimeRepeat` class implements `Schedulable`. This utility class is a prefabricated scheduler you can use to set up a repeating trigger. Just pass an `int` that is the interval (in milliseconds) at which the trigger should repeat. Then call its `schedule()` method with the starting time.

Warning: If your trigger throws an exception, it is not rescheduled. This is to ensure that a failing trigger is not re-executed indefinitely. If you want to reschedule a trigger after an exception, you must catch the exception and schedule the trigger again.

1 *Programming WebLogic Time Services (Deprecated)*

The package contains a single exception class, `TimeTriggerException`.

2 Implementing with WebLogic Time

The following sections describe how to implement with the deprecated WebLogic Time Service:

- Scheduling a Recurring Trigger on a Client
- Scheduling a Recurring Server-side Trigger from a WebLogic Client
- Setting up Complex Schedules
- Rescheduling
- Stopping a ScheduledTrigger

Scheduling a Recurring Trigger on a Client

The simplest case of scheduling a recurring trigger is to create a ScheduledTrigger that is scheduled and executed on a WebLogic client. In such a case, you write a class that implements both Schedulable and Triggerable, and implement the methods of those interfaces.

This example illustrates how to schedule and execute a trigger:

```
import weblogic.time.common.*;
import weblogic.common.*;
import java.util.*;
import weblogic.jndi.*;
import javax.naming.*;
```

2 Implementing with WebLogic Time

```
import java.util.*;

class myTrigger implements Schedulable, Triggerable {
    ...
}
```

First, obtain a `ScheduledTrigger` object from the `TimeServices` factory. Obtain the `TimeServices` factory from the `T3Services` remote factory stub on the WebLogic Server via the `getT3Services()` method.

Note: To obtain a handle to the `TimeServices` interface on a WebLogic Server instance (as opposed to a client application) use the static method, `weblogic.common.T3Services.getT3Services()`.

Next, call the `schedule()` and `cancel()` methods on the trigger, as shown in this example:

```
public myTrigger() throws TimeTriggerException {
    // Obtain a T3Services factory
    T3ServicesDef t3 = getT3Services("t3://localhost:7001");

    // Request a ScheduledTrigger from the factory. Use
    // this class for scheduling and execution
    ScheduledTriggerDef std =
        t3services.time().getScheduledTrigger(this, this);
    // Start the ball rolling
    std.schedule();
    // Your class may do other things after scheduling the trigger
    // When you are finished, cancel the trigger
    std.cancel();
}
```

Your class must implement the methods in the following interfaces.

`Schedulable`

The `Schedulable` interface has only one method, `schedule()`, which allows you to set the time at which the trigger should be executed.

```
public long schedule(long time) {
    // Schedule the trigger for every 5 seconds
    return time + 5000;
}
```

`Triggerable`

The `Triggerable` interface has only one method, `trigger()`, where the client performs an action in response to the timed triggered.

```
public void trigger() {  
    // The trigger method is where the work takes place  
    System.out.println("trigger called");  
}
```

This example is self-contained within a single class that implements both the scheduler and the trigger. This is convenient since both required methods share class variables necessary for scheduling or execution.

Scheduling a Recurring Server-side Trigger from a WebLogic Client

You can write more flexible schedulers and triggers, which may be executed anywhere within the WebLogic framework, by implementing `ScheduleDef` and `TriggerDef` instead of the simpler interfaces `Schedulable` and `Triggerable`. This example illustrates a flexible implementation that creates a recurring trigger that is rescheduled and executed on a WebLogic Server, or anywhere within the WebLogic framework.

Here are the steps to creating a scheduled trigger in this scenario. You will need to write a class that implements `ScheduleDef` and `TriggerDef`. We implement these interfaces in separate classes in this example.

Compile the classes and place them in the WebLogic Server `serverclasses` directory. Then create a `ScheduledTrigger` with those classes from a client application.

Step 1. Implement the `ScheduleDef` and `TriggerDef` interfaces

In this example, the scheduler implements `ScheduleDef` rather than `Schedulable` so that its `setServices()` and `scheduleInit()` methods are called. The trigger implements `TriggerDef` rather than `Triggerable` for the same reason. These objects differ from the

interfaces they implement in that they can be initialized with a `ParamSet`, and have access to WebLogic services through the `T3Services` stub. These two differences are important for the following reasons.

You do not need to write different versions for client-side and server-side deployment because the `T3ServicesDef` interface is a remote stub.

When you instantiate an object dynamically, you must call the default constructor. Consequently, all service-related interfaces, including the Time interfaces, require that you implement the `scheduleInit()` method which takes a `ParamSet`, thus allowing you to pass initialization parameters for the object.

Here is a simple implementation of `ScheduleDef`.

```
package examples.time;

import weblogic.common.*;
import weblogic.time.common.*;
import java.util.*;

class MyScheduler implements ScheduleDef {

    private int interval = 0;
    private T3ServicesDef services;

    public void setServices(T3ServicesDef services) {
        this.services = services;
    }

    public void scheduleInit (ParamSet ps) throws ParamSetException {
        interval = ps.getParam("interval").asInt();
    }

    public long schedule(long currentMillis) {
        return currentMillis + interval;
    }
}
```

Here is a simple class that implements `TriggerDef`. In this case, we do not need to set or get any parameters for the Trigger, so we implement the method to do nothing.

```
package examples.time;

import weblogic.common.*;
import weblogic.time.common.*;
import java.util.*;

public class MyTrigger implements TriggerDef {
```

```
private T3ServicesDef services;

public void setServices(T3ServicesDef services) {
    this.services = services;
}

public void triggerInit (ParamSet ps) throws ParamSetException {
    // Empty method definition
}

public void trigger(Schedulable sched) {
    System.out.println("trigger called");
}
}
```

Step 2. Create the ScheduledTrigger from a WebLogic Client

This method of setting up a scheduler and trigger require that you create a Scheduler and Trigger object to pass to the `get ScheduledTrigger()` factory method. We created those in “Step 1. Implement the ScheduleDef and TriggerDef interfaces.”

We have compiled those classes and placed them in the CLASSPATH of the WebLogic Server. Now we’ll write a client that uses those classes to schedule a trigger that runs in the server’s JVM.

We use a ParamSet to pass initialization parameters between the client and the objects that the WebLogic Server instantiates. The class that we wrote in Step 1 to implement ScheduleDef depends upon a Parameter “interval” to be set by the caller, so we’ll create a ParamSet with one Param. The class we wrote to implement TriggerDef doesn’t require any initialization parameters.

```
T3ServicesDef t3services = getT3Services("t3://localhost:7001");

// Create a ParamSet to pass initialization parameters for
// the ScheduleDef object. Set one parameter, "interval,"
// for 10 seconds
ParamSet schedParams = new ParamSet();
schedParams.setParam("interval", 10000);
```

Add the `getT3Services()` method to your client class and create the Scheduler and Trigger wrapper objects that instantiate a ScheduledTrigger on the server. The Scheduler and Trigger wrapper objects hold the name of the target class and a ParamSet to initialize it, if necessary.

```
Scheduler scheduler =
    new Scheduler("examples.time.MyScheduler", schedParams);
Trigger trigger =
    new Trigger("examples.time.MyTrigger");
```

Finally, use the time services object factory to manufacture a ScheduledTrigger. It takes two arguments, a Scheduler and a Trigger, which we have just created.

```
ScheduledTriggerDef std =
    t3.services.time().getScheduledTrigger(scheduler, trigger);
```

The `getScheduledTrigger()` method returns a ScheduledTriggerDef object. To initiate execute, the client calls the ScheduledTriggerDef's `schedule()` and `cancel()` methods.

If you are setting up a repeating schedule, you might also use the utility class [TimeRepeat](#), which is part of this package. Here is a simple example of how to use the TimeRepeat class to set up a regular schedule for a ScheduledTrigger that repeats every 10 seconds. Again, it uses the `getT3Services()` method to access the WebLogic server-side services.

```
T3ServicesDef t3services = getT3Services("t3://localhost:7001");

Scheduler scheduler = new Scheduler(new TimeRepeat(1000 * 10));
Trigger trigger = new Trigger("examples.time.MyTrigger");

ScheduledTriggerDef std =
    t3services.time().getScheduledTrigger(scheduler, trigger);

std.schedule();
```

Setting up Complex Schedules

You can design arbitrarily complex schedules with the `schedule()` method of a Schedulable object. Here are some examples and tips on scheduling.

There are several ways in which the argument to the `schedule()` method can describe the execution time:

- The current time, in milliseconds since the epoch.
- A specific future date and time, in a millisecond representation by performing date arithmetic using standard Java classes (such as `java.util.Date`).

The `schedule()` method returns a long value, which allows you to set up repeating triggers. Simply return the time at which the `schedule()` method was last called plus the interval (in milliseconds) at which the schedule should repeat.

Rescheduling

In this example, we write the `schedule()` method to delay for an incrementing interval between each call to the `trigger()` method. The `schedule()` and `trigger()` methods are implemented in the same class in this example.

In the `trigger()` method, we set an incrementing delay, using a private int *delay*, which we initialize to zero in the class constructor. Each time the trigger is called, it incrementally adjusts its own schedule.

```
public void trigger() {
    System.out.println("Trigger called");
    // Carry out some arbitrary tasks . . .
    System.out.println("Trigger completed");
    // Add a thousand milliseconds to the delay
    delay += 1000;
}
```

In the `schedule()` method, we return the next execution of the trigger as the time of the last scheduled execution, plus the delay incremented by the last scheduled execution (in milliseconds). We also include an upper bounds on the delay to end the scheduling.

```
public long schedule(long t) {
    System.out.println("-----");
    if (delay > 10000) {
        System.out.println("Cancelling Timer");
        return 0;
    }
}
```

```
    else {
        System.out.println("Scheduling next trigger for " +
            delay/1000 + " seconds");
        return t + delay;
    }
}
```

Stopping a ScheduledTrigger

There are two ways to stop a ScheduledTrigger:

- Call the ScheduledTrigger's `cancel()` method.
- Return zero (0) when the `schedule()` method is called ends the scheduling.

There is some slight difference in these two methods. If you return zero from the `schedule()` method, the schedule is immediately ended. If you call a ScheduledTrigger's `cancel()` method, the clock continues to run until the next scheduled instance of the `trigger()`, at which point it is cancelled.