Timer for Application Servers

International Business Machines Corp. and BEA Systems, Inc.
Version 1.0
November 2003

Authors

John Beatty, BEA Systems, Inc.
Chris D Johnson, IBM Corporation
Billy Newport, IBM Corporation
Stephan Zachwieja, BEA Systems, Inc.

Copyright Notice

© Copyright BEA Systems, Inc. and International Business Machines Corp 2003. All rights reserved.

License

The Timer for Application Servers Specification is being provided by the copyright holders under the following license. By using and/or copying this work, you agree that you have read, understood and will comply with the following terms and conditions:

Permission to copy and display the Timer for Application Servers Specification and/or portions thereof, without modification, in any medium without fee or royalty is hereby granted, provided that you include the following on ALL copies of the Timer for Application Servers Specification, or portions thereof, that you make:

1. A link or URL to the Timer for Application Servers Specification at this location:
http://dev2dev.bea.com/technologies/commonj/index.jsp
or at this location:
http://www.ibm.com/developerworks/library/j-commonj-sdowmt/

2. The full text of this copyright notice as shown in the Timer for Application Servers Specification.

IBM and BEA (collectively, the "Authors") agree to grant you a royalty-free license, under reasonable, non-discriminatory terms and conditions to patents that they deem necessary to implement the Timer for Application Servers Specification.

THE Timer for Application Servers SPECIFICATION IS PROVIDED "AS IS," AND THE AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, REGARDING THIS SPECIFICATION AND THE IMPLEMENTATION OF ITS CONTENTS, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT OR TITLE.

THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE Timer for Application Servers SPECIFICATION.

The name and trademarks of the Authors may NOT be used in any manner, including advertising or publicity pertaining to the Timer for Application Servers Specification or its contents without specific, written prior permission. Title to copyright in the Timer for Application Servers Specification will at all times remain with the Authors.

No other rights are granted by implication, estoppel or otherwise.

Status of this Document

This specification may change before final release and you are cautioned against relying on the content of this specification. IBM and BEA are currently soliciting your contributions and suggestions. Licenses are available for the purposes of feedback and (optionally) for implementation.

Introduction

The Timer for Application Servers specification provides a timer service API for use within managed environments on the Java^{TM} platform, such as Servlets, EJBs, and JCA Resource Adapters. The Timer API enables applications to schedule future timer notifications and receive timer notification callbacks to an application-specified listener.

When inside these managed environments, this API is a much better alternative to `java.util.Timer`: `java.util.Timer` should never be used within managed environments, as it creates threads outside the purview of the container. Further, there is no clean way of subclassing `java.util.Timer` to avoid thread creation, as all constructors create and start a thread. This API is also a better choice than using the JMX Timer Service because the JMX Timer Service API is tightly coupled with the JMX framework and thus does not provide a sufficiently user-friendly or independent API.

The Timer for Application Servers specification thus provides a clean, simple, and independent API that is appropriate for use within any J2EE container.

This specification is organized as follows:

- **Architecture** describes the design of the Timer API
- **Deployment** discusses how Timers are configured by deployment descriptors
- **Examples** provides a series of examples showing common usages of the Timer API
- The Java API is provided as Javadocs in a separate file

Architecture

The Timer for Application Servers specification is comprised of three primary interfaces: `TimerManager`, `Timer`, and `TimerListener`. Extensions to `TimerListener` with extra capabilities are also provided; these are discussed below. A `TimerManager` allows timers to be scheduled and manages the set of scheduled timers, each represented by a `Timer` instance. When a timer expires, the `timerExpired()` method on the provided `TimerListener` instance is executed. This execution is always in the same JVM as the thread that scheduled the timer with the `TimerManager`. `TimerManager` provides a set of `schedule()` and `scheduleAtFixedRate()` methods which take a `TimerListener` instance along with other parameters (including absolute first execution time, relative delays before first execution, and execution periods) and returns a `Timer` instance.

It is important to note the difference between *fixed-delay* execution, provided by the series of `schedule()` methods that take a `period` parameter, and *fixed-rate* execution, provided by the series of `scheduleAtFixedRate()` methods. Fixed-delay means that the `period` parameter specifies the time between actual execution time of the last `timerExpired()` method call. If the `timerExpired()` call was delayed for any reason (e.g., garbage collection or other background activity), this is taken into account. This is contrasted by fixed-rate execution, which tries to keep `timerExpired()` "caught up" and on schedule. Thus, under fixed-rate execution, the actually time interval between `timerExpired()` executions may be much smaller than the specified period.

The `Timer` instance returned by the `TimerManager` can be used to manipulate the timer (e.g., cancel, determine time to next execution, etc.).

A managed environment can support an arbitrary number of independent `TimerManager` instances. The common method for obtaining a `TimerManager` instance is through a JNDI lookup to the local Java environment (i.e., `java:comp/env/timer/[timername]`). Thus, Timer Managers are configured at deployment time through deployment descriptors, and may be further configured through implementation-specific management features. Each JNDI `lookup()` for a `TimerManager` returns a *new* logical instance of `TimerManager`. Thus, applications need to cache copies of `TimerManager` if they intend to reuse the same instance. `TimerManager` is thread-safe.

This specification places no requirements on persistence of timers: if the managed environment is shut down or fails, the timers will be irrevocably lost unless the implementation supports a higher quality of service.

`TimerManager` may also be suspended and resumed via the `suspend()` and `resume()` methods. When a TimerManager is suspended, none of the timers will expire.

`TimerManger` can also be destroyed via the `stop()` method. After `stop()` has been called, the `TimerManager` instance will never expire another timer.

Timer Interface

The `Timer` interface, instances of which are returned when timers are scheduled with the `TimerManager`, provides several capabilities:

- `cancel()`: Cancels the timer that is currently pending. If the listener associated with this timer implements the `CancelTimerListener` interface, the listener will be notified via the `timerCancel()` callback.
- `getPeriod()`: This returns the period that is used to compute the next time the timer will expire.
- `scheduledExecutionTime()`: This returns the absolute time that the timer will next expire.
- `getTimerListener()`: Returns the `TimerListener` associated with the timer.

Timer Listener Interfaces

The base `TimerListener` interface provides the `timerExpired()` callback. It is anticipated that this is sufficient for many applications. However, additional callbacks for timers being cancelled and TimerManagers being stopped are sometimes necessary.

Listener classes can implement `CancelTimerListener` if they want the `timerCancel()` callback in the case that the application cancels a `Timer`. Listener classes can implement the `StopTimerListener` if they want the `timerStop()` callback in the case that the `TimerManager` on which the `Timer` was scheduled is stopped. Listener classes can also implement both `CancelTimerListener` and `StopTimerListener` if desired.

Deployment

Applications signal their need for a timer manager through including a `resource-ref` in the appropriate deployment descriptor (e.g., web.xml, ejb-jar.xml, ra.xml, etc.). The absolute name for the JNDI namespace for `TimerManager` objects is `java:comp/env/timer`, and thus the relative name for use within the resource-ref is simply `timer`.

The following provides an example resource-ref fragment configuring a `TimerManager` named `MyTimer`:

```
<resource-ref>
  <res-ref-name>timer/MyTimer</res-ref-name>
  <res-type>commonj.timer.TimerManager</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Examples

The following example shows a `TimerManager` being looked up in JNDI and used to schedule a timer that fires in 60 seconds.

```
InitialContext ctx = new InitialContext();
TimerManager mgr = (TimerManager)
      ctx.lookup("java:comp/env/timer/MyTimer");
TimerListener listener =
      new StockQuoteTimerListener("QQQ", "johndoe@example.com");

// schedule timer to expire 60 seconds from now
mgr.schedule(listener, 1000*60);
```

The above code relies on the `StockQuoteTimerListener` class, which could be defined as follows:

```
import commonj.timers.Timer;
import commonj.timers.TimerListener;

public class StockQuoteTimerListener implements TimerListener {
      private String ticker;
      private String email;

      public StockQuoteTimerListener(String ticker, String email) {
            this.ticker = ticker;
            this.email = email;
      }
```

```
	public void timerExpired(Timer timer) {
		// retrieve stock quote for ticker and
		// email quote to recipient
		System.out.println("sent stock quote for " +
			ticker + " to " + email);

		System.out.println("timer will fire again: " +
			timer.scheduledExecutionTime());
	}
}
```

The TimerManager allows other fixed-delay schedule methods, as shown below:

```
// schedule timer to expire 60 seconds from now
mgr.schedule(listener, 1000*60);

// schedule timer to expire 60 seconds from now
// and repeat every 30 seconds
mgr.schedule(listener, 1000*60, 1000*30);

// schedule timer to expire at noon today
Calendar cal = Calendar.getInstance();
cal.set(Calendar.HOUR, 12);
mgr.schedule(listener, cal.getTime());

// schedule timer to expire at noon today
// and repeat every hour thereafter
cal = Calendar.getInstance();
cal.set(Calendar.HOUR, 12);
mgr.schedule(listener, cal.getTime(), 1000*60*60);
```

The `scheduleAtFixedRate()` method can also be used:

```
// schedule timer to expire 60 seconds from now
// and repeat every 30 seconds
mgr.scheduleAtFixedRate(listener, 1000*60, 1000*30);

// schedule timer to expire at noon today
// and repeat every hour thereafter
cal = Calendar.getInstance();
cal.set(Calendar.HOUR, 12);
mgr.scheduleAtFixedRate(listener, cal.getTime(), 1000*60*60);
```

The following shows an example listener class similar to the previous listener class, but it implements both `StopTimerListener` and `CancelTimerListener`:

```
import commonj.timers.CancelTimerListener;
import commonj.timers.StopTimerListener;
import commonj.timers.Timer;

public class StockQuoteTimerListener2
	implements StopTimerListener, CancelTimerListener {

	private String ticker;
	private String email;

	public StockQuoteTimerListener2(String ticker, String email) {
		this.ticker = ticker;
		this.email = email;
	}
	public void timerStop(Timer timer) {
		System.out.println("Timer stopped: " + timer);
	}

	public void timerCancel(Timer timer) {
		System.out.println("Timer cancelled: " + timer);
	}

	public void timerExpired(Timer timer) {
		// retrieve stock quote for ticker and
		// email quote to recipient
		System.out.println("sent stock quote for " +
			ticker + " to " + email);

		System.out.println("timer will fire again: " +
			timer.scheduledExecutionTime());
	}

}
```

Here is an example deployment descriptor that configures the `TimerManager` used above:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app …>
  <display-name>A Simple Application</display-name>
  <servlet>
    <servlet-name>OrderTracking</servlet-name>
    <servlet-class>com.mycorp.OrderTracking</servlet-class>
  </servlet>
  <resource-ref>
    <res-ref-name>timer/MyTimer</res-ref-name>
    <res-type>commonj.timer.TimerManager</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
</web-app>
```

Trademarks

IBM is a registered trademark of International Business Machines Corporation.

BEA is a registered trademark of BEA Systems, Inc.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.