

PersonalJava™ Application Environment Specification

Version 1.1.1 (Final)

January 7, 1999



Copyright © 1998-1999 Sun Microsystems, Inc.
901 San Antonio Road, Palo Alto, CA 94303 USA
All rights reserved.

Sun Microsystems, Inc. (SUN) hereby grants to you at no charge a nonexclusive, nontransferable, worldwide, limited license (without the right to sublicense) under SUN's intellectual property rights that are essential to practice this version of the PersonalJava Application Environment Specification ("Specification") to use the Specification for internal evaluation purposes only. Other than this limited license, you acquire no right, title, or interest in or to the Specification and you shall have no right to use the Specification for productive or commercial use.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-1(a).

This software and documentation is the confidential and proprietary information of Sun Microsystems, Inc. ("Confidential Information"). You shall not disclose such Confidential Information and shall use it only in accordance with the terms of the license agreement you entered into with Sun.

SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.

TRADEMARKS

Sun, the Sun logo, Sun Microsystems, JavaSoft, JavaBeans, JDK, Java, HotJava, HotJava Views, Visual Java, Solaris, NEO, Joe, Netra, NFS, ONC, ONC+, OpenWindows, PC-NFS, EmbeddedJava, PersonalJava, SNM, SunNet Manager, Solaris sunburst design, Solstice, SunCore, SolarNet, SunWeb, Sun Workstation, The Network Is The Computer, ToolTalk, Ultra, Ultracomputing, Ultraserver, Where The Network Is Going, Sun WorkShop, XView, Java WorkShop, the Java Coffee Cup logo, and Visual Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

Please submit any comments to the following email address:

personaljava-comments@java.sun.com

Due to the tremendous interest in the PersonalJava application environment, Sun may not be able to respond to each submission received on the personaljava-comments@java.sun.com alias. However, please be assured each comment is carefully reviewed.

For further information on Intellectual Property matters, contact Sun's Legal Department:

E-Mail: trademarks@sun.com

Phone: 650.960.1300

PersonalJava Application Environment Specification

Table of Contents

Introduction	1
Changes Since The 1.1 Specification	1
Changes Since The 1.0 Specification	2
Definitions	2
Requirements	3
JDK-based APIs	4
java.applet	4
java.awt	4
Scrolling Controls: Implementation and Behavior	7
java.awt.datatransfer	8
java.awt.event	8
java.awt.image	8
java.awt.peer	9
java.beans	9
java.io	9
java.lang	9
java.lang.reflect	9
java.math	10
java.net	10
java.rmi	10
java.rmi.dgc	10
java.rmi.registry	10
java.rmi.server	11
java.security	11
java.security.acl	11
java.security.interfaces	11
java.sql	11
java.text	11
java.text.resources	11
java.util	12
java.util.zip	12
PJAE-Specific APIs	13
Double Buffering	13
Specifying Component Input Behavior in Mouseless Environments	13
Unsupported Optional Features	14
Timer API	14

PersonalJava™ Application Environment Specification

Introduction

The PersonalJava™ application environment (PJAE) is a Java™ application environment that executes software written in the Java programming language. The PJAE addresses the software needs of networked applications running on personal consumer devices such as set-top boxes and smart phones rather than desktop computers. This document describes the facilities that the PJAE provides to Java applications.

Changes Since The 1.1 Specification

This specification has minor updates to the PersonalJava 1.1 API Specification which is posted at <http://java.sun.com/products/personaljava/spec-1-1/pJavaSpec.html>. This is a minor update for language, clarification, and errors in the 1.1 specification.

Here is a list of the main content changes in the 1.1.1 version of the PJAE specification:

- The title has been changed to more accurately reflect that this document is specifying the PJAE and not just the PersonalJava APIs.
- A Requirements section has been added which explicitly states that the PJAE must implement both the Java programming language and the Java virtual machine.
- Some informational sections not relating to specifying the PJAE have been removed.
- The implementation notes at the end of the previous document have been incorporated into the descriptions of the related APIs.
- Clarification has been added and errors corrected regarding the optional File I/O support. This has resulted in the `java.awt.peer` package changing to be more accurately defined as a *modified* package instead of fully supported.
- A definition of a *minimum* PJAE has been added to make more explicit what a minimum

implementation must support.

Changes Since The 1.0 Specification

The 1.1 specification has been substantially revised since the **PersonalJava 1.0** specification which is posted at <http://java.sun.com/products/personaljava/spec-1-0-1/personalJavaSpec1.0.1.html>. The organization has been streamlined and much of the explanatory material has been moved to more appropriate programming or API documentation. In addition, the definitions have been revised and condensed for clarity.

Here is a list of the major content changes in the PJAE 1.1 specification since version 1.0:

- Most JDK 1.1 packages and classes that were unsupported in the **PersonalJava 1.0 API** are now either optional or required. Only `java.security.acl` is unsupported.
- Some file I/O oriented classes in `java.io` are now optional.
- `java.net`, which was partially supported in the **PersonalJava 1.0 API**, is now fully supported.
- `java.util`, which was partially supported in the **PersonalJava 1.0 API**, is now fully supported. This expands the suitability of the PJAE to support the full range of date formats, including four-character and locale-based date formats. Note that most of the methods in the `java.util.Date` are deprecated.

Definitions

The PJAE API modifies the JDK 1.1.6 API at three levels: package, class and method. The following definitions describe the degrees of API support provided by the PJAE.

Required

The feature is completely supported by the PJAE and retains exactly the same API as its counterpart in the JDK 1.1.6.

For example,

- `java.applet` is a required package.
- `java.awt.Button` is a required class.
- `java.awt.Component.contains` is a required method.

Unsupported

The feature is not supported by the PJAE. Any reference to a class, method or field in an unsupported package will result in a `NoClassDefFoundError`.

For example,

- `java.security.acl` is an unsupported package.
- *There are no unsupported classes.*
- *There are no unsupported methods.*

Optional

The feature is not required to be supported in the PJAE. The choice of whether to support the feature is left to the PJAE implementor. However, if the implementor elects to support a given feature, then the implementation must support it completely and retain exactly the same API as its counterpart in the JDK. An optional feature that is left out of an implementation is called an *unsupported optional feature*.

For example,

- `java.math` is an optional package.
- `java.awt.CheckboxMenuItem` is an optional class.
- `java.awt.Toolkit.getPrintJob` is an optional method.

PJAE-specific

The feature has been added to the PJAE API and does not exist in JDK 1.1.6.

For example,

- `com.sun.lang` is a PJAE-specific package.
- `com.sun.util.PTimer` is a PJAE-specific class.
- `java.awt.Component.isDoubleBuffered` is a PJAE-specific method.

Modified

The feature is not completely supported by the PJAE. If a package is *modified*, then some of its classes may be optional, PJAE-specific or modified. If a class is *modified*, then some of its methods may be optional, PJAE-specific or modified. If a method is *modified*, then its semantics are changed from the JDK.

For example,

- `java.awt` is a *modified* package.
- `java.awt.Component` is a *modified* class.
- `java.awt.Component.setCursor` is a *modified* method.

Minimum PJAE

A minimum PersonalJava application environment is an implementation that meets this specification by omitting support for all optional packages, optional classes, and optional methods. A minimum PJAE does not implement the optional behavior described for each modified method. A minimum PJAE must support all of the PJAE-specific APIs.

Requirements

An implementation of the PJAE requires the following:

- Full support for the Java programming language as defined in **The Java Language Specification** which can be found at <http://java.sun.com/docs/books/jls/index.html>. The specification is published by Addison-Wesley, ISBN 0-201-63451-1.

- Full support for the Java virtual machine as defined in **The Java Virtual Machine Specification** which can be found at: <http://java.sun.com/docs/books/vmspec/index.html>. The specification is published by Addison-Wesley, ISBN 0-201-63452-X.
- The PJAE API is derived from the **JDK 1.1.6 API**, supplemented by a small number of PJAE-specific APIs designed to meet the needs of networked embedded applications. Java APIs introduced after JDK 1.1.6 will not automatically become a part of the PJAE API.
- The PJAE requires the **Java Native Interface 1.2** which is specified at: <http://java.sun.com/products/jdk/1.2/docs/guide/jni/index.html> for supporting native methods.
- The PJAE must implement at least the *minimum PJAE* APIs and features.

JDK-based APIs

java.applet

The PJAE requires full support of the JDK 1.1.6 API for the `java.applet` package.

An implementation of the PJAE which supports audio must support at least the 8-bit au sound format in the `getAudioClip` method of the `java.applet.AppletContext` interface.

java.awt

`java.awt` is a *modified* package. The PJAE has different levels of implementation support for the classes and methods in `java.awt`. These options allow a PJAE implementation to provide API functionality that matches the features of the underlying graphics and window system.

The following table describes the modified classes.

Table 1: *java.awt Modified Classes*

Class	Support Levels
Component	See Table 3 for a description of the modified methods in Component.
Dialog	<p>PJAE implementations can provide two levels of support for Dialog. At a minimum, a PJAE implementation must allow a single modal dialog to be visible at a time. In this case, if a Java program tries to display a dialog when one is already visible, the visible dialog may be hidden. When the new modal dialog disappears, the original dialog should become visible again. Minimal PJAE implementations do not support modeless dialogs. So the Dialog(Frame), Dialog(Frame, boolean) (where <i>boolean</i> is false), Dialog(Frame, String) and Dialog(Frame, String, boolean) (where <i>boolean</i> is false) constructors throw UnsupportedOperationException.</p> <p>Full featured implementations must provide the complete semantics of the Dialog class. Frame and Dialog are mutually dependent. If a PJAE implementation fully supports one, then it must fully support the other.</p> <p>See Table 3 for a description of the modified methods in Dialog.</p>
Frame	<p>PJAE implementations can provide two levels of support for Frame. At a minimum, a PJAE implementation must allow the Frame constructor to be called once to create a root for its component hierarchy. Subsequent calls to the Frame constructor will throw an UnsupportedOperationException.</p> <p>Full featured implementations must provide the complete semantics of the Frame class. Frame and Dialog are mutually dependent. If a PJAE implementation fully supports one, then it must fully support the other. Additionally, an implementation that fully supports Frame must implement the optional classes CheckboxMenuItem, Menu, MenuBar, and MenuItemShortcut.</p>
PopupMenu	See Table 3 for a description of the modified methods in PopupMenu.
Window	<p>PJAE implementations can provide different levels of support for Window. At a minimum, a PJAE implementation can prohibit direct creation of Window objects. In this case, application-level calls to the Window constructor will throw an UnsupportedOperationException.</p> <p>Full featured implementations must provide the complete semantics of the Window class.</p>

The following table describes the optional classes.

Table 2: *java.awt* Optional Classes

Class	Comments
CheckboxMenuItem	See Note below
FileDialog	If FileDialog is implemented, <ol style="list-style-type: none"> 1. the underlying RTOS must provide a file system that is user-visible, 2. all of the optional classes listed in the java.io section must be implemented, 3. Frame must be fully supported, which means 4. CheckboxMenuItem, Menu, MenuBar, and MenuShortcut must be implemented.
Menu	See Note below
MenuBar	See Note below
MenuShortcut	See Note below
Scrollbar	

Note: The CheckboxMenuItem, Menu, MenuBar, and MenuShortcut classes are *optional* as a group. Additionally, these classes must be implemented if an implementation fully supports the *modified* Frame class and vice-versa.

The following table describes the modified methods.

Table 3: *java.awt* Modified Methods

Class	Method	Comments
Component	public synchronized void setCursor(Cursor cursor)	The specified cursor may be ignored. Some implementations may not support cursors, while others may limit the types of cursors displayed for usability reasons.
Dialog	public synchronized void setResizable(boolean resizable)	The specified value may be ignored. This behavior may be allowed even if Dialog is otherwise fully supported.
PopupMenu	public MenuItem add(MenuItem mi)	A PJAЕ implementation can override the add method of the Menu superclass to throw UnsupportedOperationException when the class of its argument is Menu. Since Menu is a subclass of MenuItem, the class of the argument is explicitly checked.

The following table describes the optional methods.

Table 4: java.awt Optional Methods

Class	Method	Comments
Graphics	public abstract void setXORMode(Color c1)	Some displays, notably anti-aliased ones, are not capable of drawing in exclusive-or mode. Implementations in which this is the case will throw an <code>UnsupportedOperationException</code> when this method is called.
Toolkit	public abstract PrintJob Toolkit.getPrintJob(Frame frame, String jobtitle, Properties properties)	A program prints by first calling this method to obtain a <code>PrintJob</code> object, from which it obtains a series of objects implementing the <code>PrintGraphics</code> interface. If an implementation omits printer support, <code>getPrintJob</code> should throw <code>UnsupportedOperationException</code> .

Scrolling Controls: Implementation and Behavior

The scrollbar display policies for some of the classes in `java.awt` can have meanings that are different from the equivalent JDK 1.1.6 classes.

These changes affect the implementations of the following classes:

- `List` can use an alternate scrolling mechanism.
- `Scrollbar` can use an alternate scrolling mechanism.
- `ScrollPane`

Table 5: Scrollpane Scrolling Policy

Scrolling Policy	Visual Effect (look)	Functional Effect (feel)
SCROLLBARS_NEVER	Users see no indication that the item supports scrolling.	Scrolling this item can only be done programmatically.
SCROLLBARS_AS_NEEDED	If visual feedback for scrolling is supported, it should be given if and only if the size of the scrollable area requires it.	Scrolling can be done by the user using whatever means the toolkit provides.
SCROLLBARS_ALWAYS	If visual feedback for scrolling is supported, it must be given even if the size of the scrollable area does not require it.	Scrolling can be done by the user using whatever means the toolkit provides.

- `TextArea`

Table 6: `TextArea` Scrolling Policy

Scrolling Policy	Visual Effect (look)	Functional Effect (feel)
<code>SCROLLBARS_NONE</code>	Users see no indication that the <code>TextArea</code> supports scrolling.	Scrolling can only be done programmatically.
<code>SCROLLBARS_VERTICAL_ONLY</code>	If visual feedback for scrolling is supported, it should be given for vertical scrolling only.	Scrolling can be done by the user using whatever means the toolkit provides.
<code>SCROLLBARS_HORIZONTAL_ONLY</code>	If visual feedback for scrolling is supported, it should be given for horizontal scrolling only.	Scrolling can be done by the user using whatever means the toolkit provides.
<code>SCROLLBARS_BOTH</code>	If visual feedback for scrolling is supported, it should be given for the horizontal and vertical axes.	Scrolling can be done by the user using whatever means the toolkit provides.

java.awt.datatransfer

The PJAE requires full support of the JDK 1.1.6 API for the `java.awt.datatransfer` package.

java.awt.event

The PJAE requires full support of the JDK 1.1.6 API for the `java.awt.event` package.

java.awt.image

The PJAE requires full support of the JDK 1.1.6 API for the `java.awt.image` package.

The PJAE requires the ability to handle the image formats listed below. Where a version number is listed, it represents the *lowest* version level that the PJAE requires.

Format	Version	Required
CompuServe GIF	89a	x
JPEG (JFIF)		x
XBM (XBitmap)		x

java.awt.peer

`java.awt.peer` is a *modified* package. With the exception of the `java.awt.peer.FileDialog` interface, the PJAE requires full support of the JDK 1.1.6 API for the `java.awt.peer` package.

Implementations which do not support the optional `java.io` classes, must use an empty interface (an interface with no fields or methods) for `java.awt.peer.FileDialog`. Implementations which support the optional `java.io` classes, must support the full JDK 1.1.6 API for `java.awt.peer.FileDialog`.

java.beans

The PJAE requires full support of the JDK 1.1.6 API for the `java.beans` package.

java.io

`java.io` is a *modified* package. The following set of classes is *optional* as a group:

- `java.io.File`
- `java.io.FileInputStream`
- `java.io.FileNotFoundException`
- `java.io.FileOutputStream`
- `java.io.FileReader`
- `java.io.FileWriter`
- `java.io FilenameFilter`
- `java.io.RandomAccessFile`

The rest of the classes in `java.io` are required.

If this group of optional classes is implemented, the following requirements apply:

- There must be user-visible file system.
- The `java.awt.FileDialog` class must be fully implemented.
- The `java.awt.peer.FileDialog` interface must be identical to that of the JDK 1.1.6.

java.lang

The PJAE requires full support of the JDK 1.1.6 API for the `java.lang` package.

java.lang.reflect

The PJAE requires full support of the JDK 1.1.6 API for the `java.lang.reflect` package.

java.math

`java.math` is an *optional* package. When a PJAE implementation provides the `java.math` package it must support the full JDK 1.1.6 API of the `java.math` package.

java.net

The PJAE requires full support of the JDK 1.1.6 API for the `java.net` package.

The PJAE networking classes support the protocols listed below. Those marked *required* should be universally available. Implementors are encouraged to include those marked as *optional* if space and the specifics of a device or system allow it. Where a version number is listed, it represents the *lowest* version level that the PJAE requires.

Name	Version	Required	Optional	Dependencies
http	1.0	x		
Secure Sockets Layer (SSL)	3.0		x	requires the SSL Java standard extension
gopher	--		x	
ftp	--		x	
mailto (SMTP)	--		x	
file	--		x	file system

java.rmi

`java.rmi` is an *optional* package. When a PJAE implementation provides the `java.rmi` package it must support the full JDK 1.1.6 API of the `java.rmi` package and each of its sub-packages.

java.rmi.dgc

`java.rmi.dgc` is an *optional* package. See the requirements for `java.rmi`.

java.rmi.registry

`java.rmi.registry` is an *optional* package. See the requirements for `java.rmi`.

java.rmi.server

`java.rmi.server` is an *optional* package. See the requirements for `java.rmi`.

java.security

`java.security` is an *optional* package. When a PJAЕ implementation provides the `java.security` package it must support the full JDK 1.1.6 API of the `java.math` package and the full JDK 1.1.6 API of the `java.security` package and each of its sub-packages, except for `java.security.acl`.

java.security.acl

`java.security.acl` is an *unsupported* package.

java.security.interfaces

`java.security.interfaces` is an *optional* package. See the requirements for `java.security`.

java.sql

`java.sql` is an *optional* package. When a PJAЕ implementation provides the `java.sql` package it must support the full JDK 1.1.6 API of the `java.sql` and `java.math` packages.

java.text

The PJAЕ requires full support of the JDK 1.1.6 API for the `java.text` package.

The character encodings available in a PJAЕ implementation are platform-specific and may be quite limited. However, an implementation must guarantee the availability of converters for ISO 8859-1 ("Latin-1"), Unicode (big- and little-endian varieties, both marked and unmarked), and the native character encoding of the platform itself. These character encodings are used by `ByteArrayOutputStream`, `InputStreamReader`, `OutputStreamWriter` and `String`.

java.text.resources

`java.text.resources` is a *modified* package. The following three classes in this package are required:

- `java.text.resources.DateFormatZoneData`
- `java.text.resources.LocaleData`
- `java.text.resources.LocaleElements`

In addition, at least two matched locale classes are also required, one for `DateFormatZoneData` and one for `LocaleElements`, for example, `DateFormatZoneData_en_US` and `LocaleElements_en_US`.

The rest of the classes in `java.text.resources` are *optional*.

java.util

The PJAE requires full support of the JDK 1.1.6 API for the `java.util` package.

java.util.zip

`java.util.zip` is a *modified* package. The following table describes the degree of API support for each of the classes in the `java.util.zip` package. The *optional* classes are optional as a group except for `ZipFile`: see *Note 2* below for more information.

Table 7: *java.util.zip* Package

Class	Status
<code>Adler32</code>	optional
<code>CRC32</code>	required
<code>CheckedInputStream</code>	required
<code>CheckedOutputStream</code>	required
<code>Checksum</code>	required
<code>DataFormatException</code>	required
<code>Deflater</code>	optional
<code>DeflaterOutputStream</code>	optional
<code>GZIPInputStream</code>	required
<code>GZIPOutputStream</code>	optional
<code>Inflater</code>	modified (see <i>Note 1</i> below)
<code>InflaterInputStream</code>	required
<code>ZipConstants</code>	required
<code>ZipEntry</code>	required
<code>ZipException</code>	required
<code>ZipFile</code>	optional (see <i>Note 2</i> below)
<code>ZipInputStream</code>	required
<code>ZipOutputStream</code>	optional

Note 1: The semantics of `Inflater` are dependent on the implementation. If an implementation supports the ZLIB header and checksum fields, `Inflater` behaves exactly as in JDK 1.1.6. If an implementation does not support the ZLIB header and checksum fields, it will have the following limitations:

- The no argument `Inflater` constructor will throw an `UnsupportedOperationException` when called, since the `nowrap=false` option is implicit on this call and indicates that ZLIB header and checksum fields are supported.
- The `Inflater(boolean)` will throw an `UnsupportedOperationException` when called with the `nowrap=false` argument.
- `Inflater.getAdler` and `Inflater.setDictionary` will throw an `UnsupportedOperationException` when called.

Note 2: If the `ZipFile` class is implemented, then all of the optional classes in `java.io` must be implemented.

PJAE-Specific APIs

Double Buffering

The PJAE includes a PJAE-specific method for double buffering in `java.awt.Component`:

Method	Description
<pre>public boolean isDoubleBuffered();</pre>	Returns true if all the drawing done during the <code>update</code> and <code>paint</code> methods is automatically double buffered. The return value is not valid until after the peer has been created. The default value for the double buffering setting is platform-specific.

Specifying Component Input Behavior in Mouseless Environments

The PJAE API includes four PJAE-specific interfaces in `com.sun.awt` that help Java applications to adapt to mouseless environments like keyboard-only systems and systems operated by remote control. These input preference interfaces allow component developers to specify how users can navigate among and interact with their components.

```
package com.sun.awt
```

- `interface NoInputPreferred`
- `interface KeyboardInputPreferred`
- `interface ActionInputPreferred`
- `interface PositionalInputPreferred`

Unsupported Optional Features

The PJAE API includes a PJAE-specific class, `com.sun.lang.UnsupportedOperationException` for handling the case where a PJAE implementation does not support an optional feature of the PJAE API.

When a Java application attempts to access an unsupported optional feature, the PJAE handles the situation in one of the following ways:

optional packages

If a PJAE implementation does not support an optional package, then accessing any class in that package will cause the PJAE to throw `NoClassDefFoundError`.

optional classes

If a PJAE implementation does not support an optional class, then accessing any of its methods or fields will cause the PJAE to throw `NoClassDefFoundError`. However, for historical reasons, the methods of the following optional classes will throw `UnsupportedOperationException` if the class is not implemented:

- `java.awt.CheckboxMenuItem`
- `java.awt.FileDialog`
- `java.awt.Menu`
- `java.awt.MenuBar`
- `java.awt.MenuShortcut`
- `java.awt.Scrollbar`
- `java.util.zip.Adler32`
- `java.util.zip.Deflater`
- `java.util.zip.DeflaterOutputStream`
- `java.util.zip.GZIPOutputStream`
- `java.util.zip.ZipFile`

optional methods

When an application attempts to access an unsupported optional method, the PJAE will throw an `UnsupportedOperationException`.

Timer API

The PJAE includes a set of PJAE-specific classes in `com.sun.util` for creating and managing timer events.

```
package com.sun.util
```

- `PTimer`
- `PTimerSpec`
- `PTimerWentOffEvent`
- `interface PTimerWentOffListener`