JavaTest™ Harness Agent User's Guide

JavaTest Harness, 3.2.2_01

Please
Recycle

Adobe PostScript

Contents

Preface

This manual describes how to use the JavaTest agent in conjunction with the JavaTest harness to run tests of the test suite, write reports, and audit test results. This User's Guide is a PDF version of the JavaTest agent online help. It is provided in PDF format so that users can conveniently view and print the contents of the GUI online help without starting the JavaTest harness.

There are several differences between the online help and the PDF versions -- for example, the contents of the online help have been resequenced to be more useful in book format; in the PDF format page references embedded in the text are hypertext links in the online help; and, extensive online help navigation links have been removed from the PDF format.

The *JavaTest Agent Users' Guide* is one of three User's Guides that the JavaTest harness provides for users:

■ *JavaTest Users' Guide: Graphical User Interface*
■ *JavaTest Users' Guide: Command-Line Interface*
■ *JavaTest Agent Users' Guide*

Before You Read This Book

In order to fully use the information in this document, you must have thorough knowledge of the topics discussed in your TCK documentation as well as in these books:

■ *JavaTest Harness User's Guide: Graphical User Interface*

■ *JavaTest Harness User's Guide: Command-Line Interface*

How This Book Is Organized

Chapter 1 describes the requirements for installing a JavaTest agent on a test system.

Chapter 2 describes how a JavaTest agent is used to run tests on a test system.

Chapter 3 describes how a JavaTest agent is monitored during a test run.

Chapter 4 describes basic troubleshooting for problems in using the JavaTest agent.

Using UNIX® Commands

This document may not contain information on basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices.

See one or more of the following for this information:

- *Solaris Handbook for Sun Peripherals*
- AnswerBook2™ online documentation for the Solaris™ operating environment
- Other software documentation that you received with your system

Typographic Conventions

This User's Guide uses the following typographic conventions:

Typeface	Meaning	Examples
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
AaBbCc123	What you type, when contrasted with on-screen computer output	`% ` **`su`** `Password:`
AaBbCc123	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this.
	Command-line variable; replace with a real name or value	To delete a file, type `rm` *filename*.

Shell Prompts

Examples in this User's Guide might contain the following shell prompts:

Shell	Prompt
C shell	*machine_name*%
C shell superuser	*machine_name*#
Bourne shell and Korn shell	$
Bourne shell and Korn shell superuser	#

Related Documentation

The following documentation provides additional detailed information about the JavaTest harness:

Application	Title
JavaTest GUI	*JavaTest Harness User's Guide: Graphical User Interface*
JavaTest command-line interface	*JavaTest Harness User's Guide: Command-Line Interface*

Accessing Sun Documentation Online

The Java Developer Connection™ program web site enables you to access Java platform technical documentation at `http://java.sun.com/`.

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. Provide feedback to Sun at `http://java.sun.com/docs/forms/sendusmail.html`.

1

What is the JavaTest Harness Agent?

An agent is a separate program that works in conjunction with the JavaTest harness to run tests on a system other than the one that is running the JavaTest harness.

You can use custom agents or the JavaTest harness agent to run tests. The topics in this guide describe how to configure and run the agent provided with the JavaTest harness. If you are using a custom agent, refer to your test suite documentation for a description of how to configure and run it.

JavaTest Harness Agent Features

Depending on your test suite, agents are typically used to run tests on small devices that do not support online help.

The following table describes the features of the JavaTest harness agent.

TABLE 1   JavaTest Harness Agent Features

Feature	Description
Multiple User Interfaces	The JavaTest harness agent can be run from any one of the following supported user interfaces: • Agent GUI • Command-Line Interface • Applets
Configurable Modes	The JavaTest harness agent can be run in active, passive, or serial modes.
Monitoring	The JavaTest harness agent and the tests that it is running can be monitored in the Agent GUI.

2

Installing Agent Classes on a Test System

Before you can use the JavaTest harness agent to run tests, you must load the agent classes on your test system. You can load the agent classes by doing one of the following:

■ Copy the `javatest.jar` file directly to the test system if adequate space is available (approximately 5.7megs). The `javatest.jar` file contains all of the required JavaTest harness agent classes.
■ Extract the minimum set of classes from the `javatest.jar` file for the type of agent user interface and copy them to the test system.

The following table provides links to the required classes for each type of agent user interface.

TABLE 2   Agent User Interface Required Classes

Agent User Interface	Required Classes
Agent GUI	See Classes Required to Use the GUI for a list of the minimum set of classes required for using the GUI to run agents.
Command Line	See Classes Required to Use the Command Line for a list of the minimum set of classes required for using the command line to run agents.
Applets	See Classes Required to Use Applets for a list of the minimum set of classes required for using applets to run agents.

Classes Required to Use the GUI

The following list contains the minimum set of classes required to run an agent by using a GUI on your test system. You might require additional classes for some tests run in the same VM as the agent.

```
com.sun.javatest.Command
com.sun.javatest.JavaTestSecurityManager
```

```
com.sun.javatest.NewJavaTestSecurityManager
com.sun.javatest.ProductInfo
com.sun.javatest.Status
com.sun.javatest.Test
com.sun.javatest.agent.ActiveConnectionFactory
com.sun.javatest.agent.ActiveModeOptions
com.sun.javatest.agent.Agent
com.sun.javatest.agent.Agent$1
com.sun.javatest.agent.Agent$Notifier
com.sun.javatest.agent.Agent$Observer
com.sun.javatest.agent.Agent$Task
com.sun.javatest.agent.AgentFrame
com.sun.javatest.agent.AgentFrame$1
com.sun.javatest.agent.AgentFrame$2
com.sun.javatest.agent.AgentFrame$3
com.sun.javatest.agent.AgentFrame$Listener
com.sun.javatest.agent.AgentPanel
com.sun.javatest.agent.AgentPanel$1
com.sun.javatest.agent.AgentPanel$AgentObserver
com.sun.javatest.agent.AgentPanel$ButtonPanel
com.sun.javatest.agent.AgentPanel$ErrorPanel
com.sun.javatest.agent.AgentPanel$HelpPanel
com.sun.javatest.agent.AgentPanel$HistoryList
com.sun.javatest.agent.AgentPanel$MapReader
com.sun.javatest.agent.AgentPanel$ParamPanel
com.sun.javatest.agent.AgentPanel$StatsPanel
com.sun.javatest.agent.AgentPanel$TaskPanel
com.sun.javatest.agent.AgentPanel$TaskState
com.sun.javatest.agent.AgentWriter
com.sun.javatest.agent.BadValue
com.sun.javatest.agent.Connection
com.sun.javatest.agent.ConnectionFactory
com.sun.javatest.agent.ConnectionFactory$Fault
com.sun.javatest.agent.Deck
com.sun.javatest.agent.Deprecated
com.sun.javatest.agent.Folder
com.sun.javatest.agent.Folder$1
com.sun.javatest.agent.Folder$Entry
com.sun.javatest.agent.Folder$Layout
com.sun.javatest.agent.Icon
com.sun.javatest.agent.Map
com.sun.javatest.agent.ModeOptions
com.sun.javatest.agent.PassiveConnectionFactory
com.sun.javatest.agent.PassiveModeOptions
com.sun.javatest.agent.Proxy
com.sun.javatest.agent.SerialPortModeOptions
com.sun.javatest.agent.SocketConnection
com.sun.javatest.agent.SocketConnection$1
com.sun.javatest.util.DynamicArray
com.sun.javatest.util.ExitCount
com.sun.javatest.util.I18NResourceBundle
com.sun.javatest.util.MainFrame
```

```
com.sun.javatest.util.StringArray
com.sun.javatest.util.Timer
com.sun.javatest.util.Timer$1
com.sun.javatest.util.Timer$Entry
com.sun.javatest.util.Timer$Timeable
com.sun.javatest.util.WriterStream
```

Classes Required to Use the Command Line

The following list contains the minimum set of classes required to run an agent
from a command line on your test system. You might require additional classes for
some tests run in the same VM as the agent.

```
com.sun.javatest.Command
com.sun.javatest.JavaTestSecurityManager
com.sun.javatest.NewJavaTestSecurityManager
com.sun.javatest.Status
com.sun.javatest.Test
com.sun.javatest.agent.Agent
com.sun.javatest.agent.Agent$1
com.sun.javatest.agent.Agent$Notifier
com.sun.javatest.agent.Agent$Observer
com.sun.javatest.agent.Agent$Task
com.sun.javatest.agent.AgentMain
com.sun.javatest.agent.AgentMain$BadArgs
com.sun.javatest.agent.AgentMain$ErrorObserver
com.sun.javatest.agent.AgentMain$Fault
com.sun.javatest.agent.AgentWriter
com.sun.javatest.agent.Connection
com.sun.javatest.agent.ConnectionFactory
com.sun.javatest.agent.ConnectionFactory$Fault
com.sun.javatest.agent.Deprecated
com.sun.javatest.agent.Map
com.sun.javatest.util.DynamicArray
com.sun.javatest.util.StringArray
com.sun.javatest.util.Timer
com.sun.javatest.util.Timer$1
com.sun.javatest.util.Timer$Entry
com.sun.javatest.util.Timer$Timeable
```

```
com.sun.javatest.util.WriterStream
```

Classes Required to Use Applets

The following list contains the minimum set of classes required to run an agent as an applet on your test system. You might require additional classes for some tests run in the same VM as the agent.

```
com.sun.javatest.Command
com.sun.javatest.ProductInfo
com.sun.javatest.Status
com.sun.javatest.Test
com.sun.javatest.agent.ActiveConnectionFactory
com.sun.javatest.agent.ActiveModeOptions
com.sun.javatest.agent.Agent
com.sun.javatest.agent.Agent$1
com.sun.javatest.agent.Agent$Notifier
com.sun.javatest.agent.Agent$Observer
com.sun.javatest.agent.Agent$Task
com.sun.javatest.agent.AgentApplet
com.sun.javatest.agent.AgentApplet$1
com.sun.javatest.agent.AgentPanel
com.sun.javatest.agent.AgentPanel$1
com.sun.javatest.agent.AgentPanel$AgentObserver
com.sun.javatest.agent.AgentPanel$ButtonPanel
com.sun.javatest.agent.AgentPanel$ErrorPanel
com.sun.javatest.agent.AgentPanel$HelpPanel
com.sun.javatest.agent.AgentPanel$HistoryList
com.sun.javatest.agent.AgentPanel$MapReader
com.sun.javatest.agent.AgentPanel$ParamPanel
com.sun.javatest.agent.AgentPanel$StatsPanel
com.sun.javatest.agent.AgentPanel$TaskPanel
com.sun.javatest.agent.AgentPanel$TaskState
com.sun.javatest.agent.AgentWriter
com.sun.javatest.agent.BadValue
com.sun.javatest.agent.Connection
com.sun.javatest.agent.ConnectionFactory
com.sun.javatest.agent.ConnectionFactory$Fault
com.sun.javatest.agent.Deck
com.sun.javatest.agent.Deprecated
com.sun.javatest.agent.Folder
com.sun.javatest.agent.Folder$1
com.sun.javatest.agent.Folder$Entry
com.sun.javatest.agent.Folder$Layout
com.sun.javatest.agent.Icon
com.sun.javatest.agent.Map
com.sun.javatest.agent.ModeOptions
```

```
com.sun.javatest.agent.PassiveConnectionFactory
com.sun.javatest.agent.PassiveModeOptions
com.sun.javatest.agent.Proxy
com.sun.javatest.agent.SerialPortModeOptions
com.sun.javatest.agent.SocketConnection
com.sun.javatest.agent.SocketConnection$1
com.sun.javatest.util.DynamicArray
com.sun.javatest.util.I18NResourceBundle
com.sun.javatest.util.MainAppletContext
com.sun.javatest.util.MainFrame
com.sun.javatest.util.StringArray
com.sun.javatest.util.Timer
com.sun.javatest.util.Timer$1
com.sun.javatest.util.Timer$Entry
com.sun.javatest.util.Timer$Timeable
com.sun.javatest.util.WriterStream
```

Choosing the Type of Agent

The JavaTest harness agent is a lightweight program compatible with Java Development Kit, version 1.1, that uses a bi-directional serial connection supporting both TCP/IP and RS-232 protocols to communicate between the test system and the JavaTest harness.

You can use the agent provided by the JavaTest harness if your test system meets the following minimum requirements:

- The device supports a communication layer that can last the duration of a test (several minutes).
- The device must be able to have the agent classes loaded on it.

The type of agent that you use depends on the communication protocol used between your test system and the JavaTest harness and on the type of initial connection made between the agent and the JavaTest harness. The following table describes the types of agent and the communication protocol.

TABLE 3   Types of Agent Modes

Mode	Description
Active	Use active mode (active agent) when you want the agent to initiate the connection to the JavaTest harness via TCP/IP. Agents using active communication allow you perform the following actions: • Run tests in parallel using many agents at once • Specify the test machines at the time you run the tests Active agents are used for network connections and are recommended. If the security restrictions of your test system prevent incoming connections then you must use an active agent. The JavaTest harness must be running and agent pool listening must be enabled before starting an active agent. Use [Agent Monitor window](#) in the JavaTest harness GUI to enable listening. If listening is not enabled when the agent starts, it returns an error message and waits until its timeout period ends before re-contacting the JavaTest harness.

TABLE 3   Types of Agent Modes

Passive	Use passive mode (passive agent) when you want the agent to wait for the JavaTest harness to initiate the connection via TCP/IP. Because the JavaTest harness only initiates a connection to a passive agent when it runs tests, passive communication has the following characteristics: • Requires that you specify the test machine as part of the test configuration, not at the time you run the tests • Does not allow you to run tests in parallel Passive agents are used for network connections and must be started before the harness attempts to run tests. If the JavaTest harness issues a request before the passive agent is started, the harness waits for an available agent until its timeout period ends. If the timeout period ends before an agent is available, the JavaTest harness reports an error for the test.
Serial	Use serial mode (serial agent) when you want the agent to use an RS-232 serial connection. Serial agents wait for the JavaTest harness to initiate the connection. Infrared, parallel, USB, and firewire connections can also be added through the JavaTest harness API by modeling the existing serial system. Because the JavaTest harness only initiates a connection to serial agent when it runs tests, serial communication has the following characteristics: • Requires that you specify the test machine as part of the test configuration, not at the time you run the tests • Does not allow you to run tests in parallel
Other	If your system does not meet the minimum requirements or if you have unique performance requirements, you can use the JavaTest harness API to create a custom agent. Refer to your test suite documentation for a description of how to configure and run it.

Creating a Map File

Some tests require contextual information, such as the host name on which they are executed, before they can run. Because network file systems might be mounted differently on different systems, the path names used by the JavaTest harness might not be the same for the agent. The agent uses a map file to translate these strings into values it can use to run tests.

1. Use a text editor to open a simple ASCII file and enter the following types of lines:

   ■ **Comment line**   Begins with the # symbol and provides information that is not processed by the agent. Comment lines are optional.

Example:

```
 #Replace all /home/user1 with /user1
```

■ **Translation line**   Contains the target and substitution strings. Enter the string that is to be replaced followed by one or more spaces and the replacement string. The agent replaces all occurrences of the first string with the second.

Example:

```
 /home/user1  /user1
```

Because the agent uses the map file to perform global string substitution on *all* matching values received from the JavaTest harness, you must be as specific as possible when specifying strings in a translation line.

Refer to [Troubleshooting JavaTest harness agents](#) for additional information about determining the substitution strings required in a map file.

2. Save the map file in the test suite root directory.

You can use any name and extension. If you are unable to use the root directory, you can use any writable directory on the test system. When starting an agent you must specify which map file, if any, to use.

Example of a map file:

```
#This is a sample map file
#Replace all /home/user1 with /user1

/home/user1 /user1

#Replace all /home/user2/JT Harness with /myhome/JT Harness
/home/user2/JT Harness /myhome/JT Harness
```

Starting a JavaTest Harness Agent

You can start an agent either as an application or as an applet. While the application provides you with the option of using either a GUI or a command line to configure and run the agent, the applet requires that you use a GUI. The following table describes the agent interface support for application and applets.

TABLE 4   Supported Agent Interfaces

Interface	Application	Applet
GUI	Supported	Supported
Command Line	Supported	Not Supported

Agent Application

You can either use the application GUI or command line to configure and start an agent if the test system provides AWT support.

If a test platform is unable to or does not provide AWT support, you must use the command line to configure and start the agent. When using the command line to directly configure and run an agent, the following conditions apply:

- All agent options must be specified in the command line.
- Agent performance cannot be monitored during a test run.
- Agent properties cannot be modified without killing the agent and starting a new agent from the command line.

If you use the GUI to run the agent, the following conditions apply:

- Agent options can be included in the command line or the GUI can be started without specifying agent options.
- Agent performance is monitored during a test run.
- Agent can be configured or reconfigured after the GUI starts.

The GUI used by the application is the same as that used by the applet. Refer to Using the GUI for a description of the tabbed panes.

Agent Applet

You can use either an applet or an application to run the agent on any test system that supports a web browser. However, you must use the applet when testing Java virtual machines that run in web browsers.

The GUI used by the applet is the same as that used by the application. Refer to Using the GUI for a description of the tabbed panes.

When using the applet, you can perform the following actions:

- Include parameters in the applet tag or start the GUI without specifying any parameters
- Configure or reconfigure the agent after the GUI starts
- Monitor agent performance during a test run.

Using the GUI

The GUI contains four tabbed panes and three buttons used to configure, control, and monitor the agent.
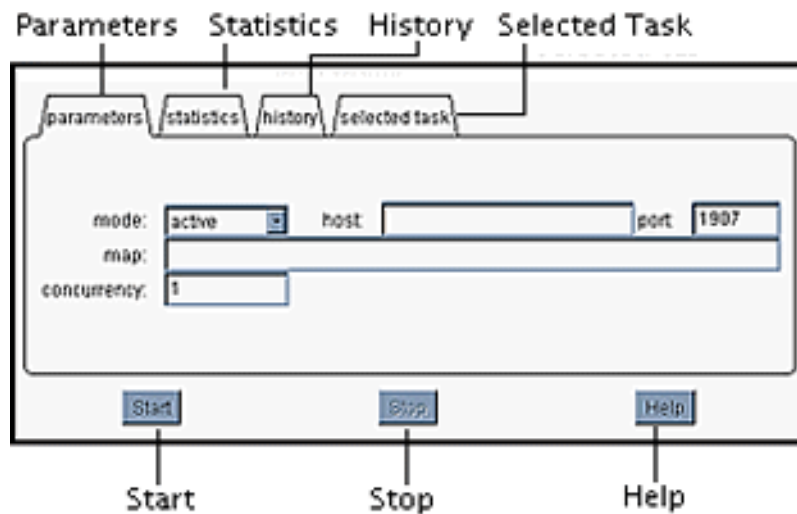
The parameters tabbed pane allows you to configure, start, and stop the agent.

The statistics tabbed pane displays detailed information about the tests that the agent is running.

The history and selected task tabbed panes allow you to monitor tasks performed by the agent.

The Start and Stop buttons control the agent.

The terms "Java Virtual Machine" and "JVM" mean a Virtual Machine for the Java" platform.

Starting an Agent Application

Before you can start an agent application, the required classes must be installed on your test system. Refer to Installing Agent Classes on Test Systems for the location and list of classes required to start the agent directly from the command line or using the application GUI. Complete the following actions to start an agent application:

1. Start the JavaTest harness GUI.

2. Open the Configuration Editor window and configure the JavaTest harness to use an agent. In most cases, the window displays detailed instructions about configuring the JavaTest harness to run tests using an agent.

3. If you are starting an active agent, open the Agent Monitor window and enable agent pool listening. Refer to Agent Monitor Window.

> **Note –** If the agent pool is not listening when an active agent starts, the agent cannot contact the harness. The agent returns an error message and then waits until its timeout period ends before recontacting the JavaTest harness.

4. Use the following example to enter the appropriate agent command at the command prompt:

   `java -cp` *classpath [application class] [options]*

   - The `-cp` option sets the classpath required to run the agent. Use the ; or : separator appropriate for your test system when more than one class path is included in the command string. Refer to [Class Paths](#) for detailed descriptions of the classes that your agent requires.
   - *[application class]* sets the class used to run the agent application. Refer to [Application Classes](#) for a list and description of the classes used to start an agent application.
   - *[options]* can be included in the command line to specify the agent parameters. Refer to [Agent Options](#) for a list and description of the parameters that you can use to configure and start an agent.

   Example:
   ```
   java -cp /lib/javatest.jar
   com.sun.javatest.agent.AgentFrame
   ```

> **Note –** You must include the path of the `javatest.jar` file (represented as `/lib/javatest.jar` in the example). The `javatest.jar` file is usually installed in the test suite `lib` directory when the JavaTest harness is bundled with a test suite.

5. If you are using the application GUI to run the agent, use the Parameters tabbed pane to verify the agent settings and start the agent.

The following topics provide detailed information about agent parameter settings:

- **Specifying Active Agent Options**   Parameter settings required to run an active agent.
- **Specifying Passive Agent Options**   P arameter settings required to run a passive agent.
- **Specifying Serial Agent Options**   Parameter settings required to run a serial agent.
-

Classpaths

The following table describes the classpaths that are required in the command line.

TABLE 5   Required Class Paths

Classes	Description
Agent	The location of the agent classes installed on your test system. The agent classes are either located in the `javatest.jar` file or in the directory containing the minimum set of classes required to run the agent from the GUI. Some test suites include additional .jar files containing classes needed for an agent to run tests. These `.jar` files must also be included in the command string. Refer to [Installing Agent Classes on a Test System](#) for a description of how agent classes can be installed.
Test	Test classes are located in the classes directory of the test suite.

The most common error made when setting up a test platform to use an agent is entering the wrong classpaths in the command string. Configuring your test platform to use the simplest classpaths increases the reliability of the test run.

Application Classes

An application class is required in the command line to run the agent. The following table describes the two application classes.

TABLE 6   Required Application Class

Mode	Application Class
No GUI	`com.sun.javatest.agent.AgentMain` *options* Used when the agent GUI is not wanted or not available. In this mode, all options must be fully specified on the command line. The agent automatically starts when the Return key is pressed. Refer to [Agent Options](#) for the *options* that are included on the command line.
With GUI	`com.sun.javatest.agent.AgentFrame` *options* Used to start the agent GUI. In this mode, options might either be given on the command line or in the agent GUI. The agent GUI is used to start and stop the agent. Refer to [Agent Options](#) for the *options* that are included on the command line.

Agent Options

The following table describes the two types of options used in the command line.

TABLE 7   Type of Agent Options

Type of Option	Description
Agent parameters	Set the parameters for the type of agent that you are using. See the following topics for additional information: • **Specifying Active Agent Options** The parameter settings required to run an active agent. • **Specifying Passive Agent Options** The parameter settings required to run a passive agent. • **Specifying Serial Agent Options** The parameter settings required to run a serial agent. If you are using the command-line application class (`com.sun.javatest.agent.AgentMain`) to directly configure and run the agent, you must include all options in the command line that are used to run the agent. If you are using the GUI application class (`com.sun.javatest.agent.AgentFrame`) you can either set the agent options in the command line or in the GUI before running the agent.
Additional parameters	Display help, run the agent, or configure other agent properties. Refer to Specifying Additional Agent Options for a description of the additional parameters that can be set.

Starting an Agent Applet

Before you can start an agent applet, the required classes must be installed on your test system. Refer to Installing Agent Classes on Test Systems for the location and list of classes required to start the agent applet.

1. If an HTML page containing the required applet is not available, create it in your test suite root directory.

Refer to Agent Applet Tag for a detailed description of an applet tag.

1. Use a web browser to open an HTML page containing the agent applet tag.

The applet tag must be compatible with your browser's VM.

1. Use the Parameters tabbed pane to configure and run the agent.

1. See the following topics for additional information:
   ■ **Specifying Active Agent Options**   Parameter settings required to run an active agent.

2. **Specifying Passive Agent Options**   Parameter settings required to run a passive agent.

3. **Specifying Serial Agent Options**   Parameter settings required to run a serial agent.

Agent Applet Tag

Because some browsers use built-in VMs to run applets, you must use a compatible applet or object tag. Refer to your VM documentation for a description of the tags required to run applets on your browser. The following example calls the agent applet and sets the parameters of the applet GUI. It might not be compatible with your browser VM.

Agent parameters and run options can also be set in the applet tag. Refer to Setting Parameters in the Applet Tag.

Example agent applet tag:

```
<APPLET/
code=applet-class-path/
archive=JavaTest harness-classes/
width=display-width/
height=display-height/
>
 Applets have not been enabled
on your browser. You must enable
applets on your browser to display
the applet GUI used to run the agent.
</APPLET>
```

The following table describes the tags used in the applet.

TABLE 8   Applet Tags and Descriptions

Tag	Description
code	The agent applet class installed on your test system. Example: `code=com.sun.javatest.agent.AgentApplet`

TABLE 8   Applet Tags and Descriptions

archive	The URL of the classes required to run the agent applet on your test system. The classes are either located in the `javatest.jar` file or in a directory containing the minimum set of classes required to run the agent applet. In the following example, the classes are contained in the `javatest.jar` file located in the same directory as the HTML page. Refer to [Installing Agent Classes on a Test System](#) for a description of how the agent applet classes can be installed. Example: `archive=javatest.jar`
width	Sets the width of the GUI. An initial value of 600 is suggested. However, you might need to adjust the value based on your screen size and resolution. Example: `width=600`
height	Sets the height of the applet. An initial value of 600 is suggested. However, you might need to adjust the value based on your screen size and resolution. Example: `height=600`

Setting Parameters in the Applet Tag

Parameters can also be set in the applet tag. Parameters in the applet tag are included as <param *name/value*> pair tags.

Example agent applet tag:

<APPLET
code=*applet-class-path*
archive=*JavaTest harness-classes*
width=*display-width*
height=*display-height*
>

...
 <param name=*parameter-name* value=*parameter-value*>
 Applets have not been enabled on your browser.
You must enable applets on your browser to display
the applet GUI used to run the agent.
</APPLET>

The following two types of parameters can be included in the applet tag:

- **Agent Parameters**   Specifies the agent type. Can be set either in the applet tag or in the GUI. Anytime the agent is not running, you can also use the Parameters tabbed pane to change the agent parameters. See the following topics for additional information:
- **Specifying Active Agent Options**   Parameter settings required to run an active agent.

- **Specifying Passive Agent Options**   Parameter settings required to run a passive agent.
- **Specifying Serial Agent Options**   Parameter settings required to run a serial agent.
- **Additional Parameters**   Specifies how an agent is run.

    See Specifying Additional Agent Options for additional information.

Specifying Active Agent Options

Active agents can be configured and run from the application command-line, the application or applet GUI, or the applet tag. Refer to Starting a JavaTest harness agent for a description of the different features and functions that each provides.

Depending on how you choose to start the agent, you must set the following minimum set of parameters either in the command line, the GUI Parameter pane, or the applet tag:

- Mode
- Host
- Port

Mode

The type of agent mode that you use determines how the agent communicates with the JavaTest harness and the protocol that is used. An active agent initiates the connection to the JavaTest harness using TCP/IP communications protocol.

To specify an active agent mode, use the appropriate setting or option from the following table.

TABLE 9   Specify Agent Mode Options and Settings

Interface	Option or Setting
Default	Active
Command line	`-active`
Applet tag	<param name=mode value=active>
GUI Parameter pane	mode: active host: port: 1907

Host

The host option identifies the system running the JavaTest harness. Because an active agent initiates the connection to the JavaTest harness, the location of the system running the JavaTest harness must be set before it can run.

To specify the system running the JavaTest harness, use the appropriate setting or option from the following table.
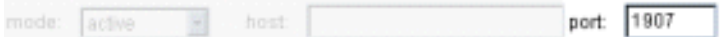
TABLE 10   Active Agent Host Option or Setting

Interface	Option or Setting
Default	None
Command line	-activeHost *host-name*
Applet tag	<param name=activeHost value=*host-name*>
GUI Parameter pane	mode: active host: [ ] port: 1907

Port

The port option specifies the port used by the active agent to communicate with the JavaTest harness. The agent and JavaTest harness must use the same port. If the ports are not the same, the agent cannot communicate with the JavaTest harness. The default value for active agents is 1907.

To specify a port other than 1907, use the appropriate setting or option from the following table.

TABLE 11   Active Agent Port Option or Setting

Interface	Option or Setting
Default	1907
Command line	-activePort *port-number*
Applet tag	<param name=activePort value=*port-number*>
GUI Parameter pane	mode: active host: [ ] port: 1907

Specifying Passive Agent Options

Passive agents can be configured and run from the application command line, the application or applet GUI, or the applet tag. Refer to [Starting a JavaTest harness agent](#) for a description of the different features and functions that each provides.

Depending on how you choose to start the agent, you must set the following minimum set of parameters either in the command line, the GUI Parameter pane, or the applet tag:

- [Mode](#)
- [Port](#)

Mode

The type of agent mode that you use determines how the agent communicates with the JavaTest harness and the protocol that is used. A passive agent waits for the JavaTest harness to initiate the connection using TCP/IP communications protocol.

To specify a passive agent, use the appropriate setting or option from the following table.

TABLE 12   Passive Agent Mode Option or Setting
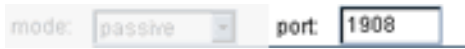
Interface	Option or Setting
Default	Active
Command line	`-passive`
Applet tag	<param name=mode value=passive>
GUI Parameter pane	

Port

The port option specifies the port that the passive agent uses to listen for the JavaTest harness. The JavaTest harness and agent must use the same port. If the ports are not the same, the JavaTest harness cannot communicate with the agent. The default value for passive agents is 1908.

To specify a port other than 1908, use the appropriate setting or option from the following table.

TABLE 13   Passive Agent Port Option or Setting

Interface	Option or Setting
Default	1908
Command line	`-passivePort` *port-number*
Applet tag	\<param name=activePort value=*port-number*\>
GUI Parameter Pane	mode: passive port: 1908

Specifying Serial Agent Options

Serial agents can be configured and run from a command-line, GUI, or applet tag. Refer to Starting a JavaTest harness agent for a description of the different features and functions that each provides.

Depending on how you choose to start the agent, you must set the following minimum set of parameters from the command line, GUI Parameter pane, or applet tag:

- Mode
- Port

Mode

The type of agent mode that you use determines how the agent communicates with the JavaTest harness and the protocol that is used. A serial agent waits for the JavaTest harness to initiate the connection via an RS-232 serial connection or a connection added through the JavaTest harness API that models the serial system.

To specify a serial agent, use the appropriate setting or option from the following table.

TABLE 14   Serial Agent Mode Option or Setting

Interface	Option or Setting
Default	Active
Command line	-serial
Applet tag	<param name=mode value=serial>
GUI Parameter pane	

Port

Specifies the com port that the serial agent uses to listen for the JavaTest harness. The JavaTest harness and agent must use the same port. If the ports are not the same, the JavaTest harness cannot communicate with the agent.

To specify a port, use the appropriate setting or option from the following table.

TABLE 15   Serial Agent com Port Option or Setting

Interface	Option or Setting
Command line	-serialPort *port-number*
Applet tag	<param name=serialPort value=*port-number*>
GUI Parameter pane	

Specifying Additional Agent Options

The following topics describe the additional options for using an agent:

- Options Used to Display Help
- Options Used to Run and Monitor the Agent

Options Used to Display Help

The help option only displays command-line help for the agent regardless of the application class used in the command line. To start an agent application or applet after displaying command-line help, perform the steps in [Starting a JavaTest Harness Agent](#).

The following table contains options that are only used on the command line to display help.

TABLE 16   Command-Line Options to Display Help

Option	Function

`-help` or `-usage`	Displays command-line help.
	Example:
	`java -cp /lib/javatest.jar`
	`com.sun.javatest.agent.AgentMain -help`

You must include the path of the `javatest.jar` file (represented as `/lib/javatest.jar` in the example). The `javatest.jar` file is usually installed in the test suite `lib` directory when the JavaTest harness is bundled with a test suite.

Options Used to Run and Monitor the Agent

The following options can be set in the application command line, the application or applet GUI, or the applet tag:

- [Specify a Map File](#)
- [Set Concurrency](#)
- [Set Number of Tasks in the History Tabbed Pane](#)
- [AutoStart the Agent](#)
- [Set Tracing](#)

Specify a Map File

The map option specifies that the agent use a map file to translate host specific values. Refer to [Create a Map File](#) for additional information about map files.

To specify a map file, use the appropriate setting or option from the following table.
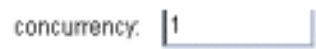
TABLE 17   Map File Options

Interface	Option or Setting
Default	None (empty)
Command line	–map *map-file*
Applet tag	<param name=map value=*map-file-url*>
GUI Parameter pane	map:

Set Concurrency

To run tests concurrently, set the maximum number of simultaneous requests handled by the agent. Each request requires a separate connection to the JavaTest harness and a separate thread inside the agent. The request might also require a separate process on the test system running the agent. The default setting is one.

To run concurrent tests, use the appropriate setting or option from the following table.

TABLE 18   Run Concurrent Tests Option

Interface	Option or Setting
Default	One
Command line	–concurrency *number-of-tests*
Applet tag	<param name=concurrency value=*number-of-tests*>
GUI Parameter pane	concurrency: 1

Set Number of Tasks in the History Tabbed Pane

The history option specifies the maximum number of tasks displayed in the history tabbed pane. Refer to History Tabbed Pane for a description of the history tabbed pane and how it is used to monitor an agent.

To set the tasks displayed in the history tabbed pane, use the appropriate setting or option from the following table.

TABLE 19   History Tabbed Pane Options

Interface	Option or Setting
Default	One
Command line	-history *number-of-items*
Applet tag	\<param name=history value=*number-of-items*>
GUI	Not supported

AutoStart the Agent

Th start option is only used with the application GUI class or as a parameter in the applet tag. When used, the start option automatically starts the agent after all command line options are validated and the GUI is displayed. The agent must be completely configured in the command line or applet tag. When the -start option is not used, click the Start button in the agent GUI to start testing.

To autostart the agent when the GUI is displayed, use the appropriate setting or option from the following table.

TABLE 20   AutoStart Agent Options

Interface	Option or Setting
Default	False
Command line	-start
Applet tag	\<param name=start value=*true*>
GUI	

Set Tracing

The trace option sends detailed information about agent activity to the system output stream.

To start tracing when the agent is run, use the appropriate setting or option from the following table.

TABLE 21 Set Tracing Options

| Interface	Option or Setting
Default	False
Command line	`-trace`
Applet tag	\<param name=trace value=*true*\>
GUI	Not Supported

3

Monitoring JavaTest Harness Agents

You can monitor JavaTest harness agents in one of the following ways:

- View all agents in a test system that are running tests. Refer to <u>Agent Monitor Window</u> for detailed information about opening and using the Agent Monitor window to view all agents in a test system that are running tests.
- Monitor specific information about an agent and the tests that it runs. To display information about the agent, you must use the tabbed panes in the application or applet GUI. See the following topics for detailed information about the GUI:
- **Statistics Pane**   Displays the current status of the tests that the agent is running.
- **History Pane**   Displays a list of tasks performed by the agent.
- **Selected Task Pane**   Displays details about a specific task or test chosen in the history tabbed pane.

Agent Monitor Window

Open the Agent Monitor window by using the JavaTest harness GUI Test Manager to choose Window **->** Open **->** Agent Monitor. See the *JavaTest Harness User's Guide: Graphical User Interface* for detailed description of the Test Manager window.

The Agent Monitor window contains two sections, Agent Pool and Agents Currently In Use.
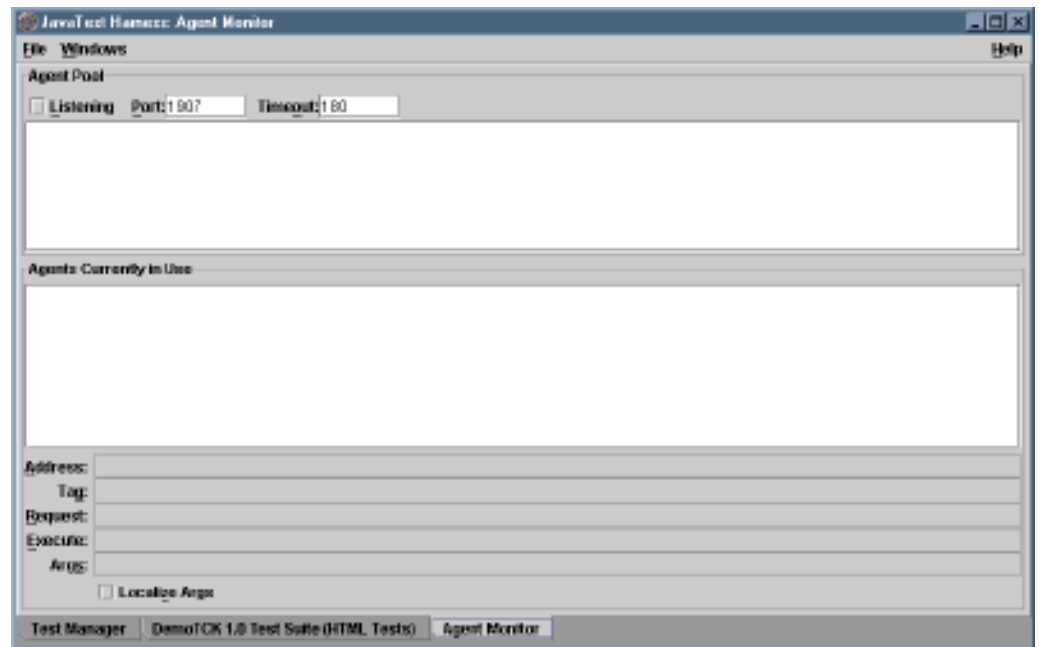
Agent Pool

Agent Pool lists the active agents that are available to run tests. When active agents connect to the JavaTest harness, they are added to the agent pool. When the JavaTest harness requires an active agent to run a test, it moves the agent from Agent Pool to Agents Currently In Use until the test is completed.

The following table lists and describes the contents of the Agent Pool GUI.

TABLE 22   Agent Pool GUI Contents

Field	Description

TABLE 22 Agent Pool GUI Contents

Listening	Click the check box to enable listening for active agents. If listening is not enabled when an agent starts, the agent issues a message that it cannot connect to the JavaTest harness and then waits for its timeout period to end before attempting to recontact the harness.
Port	Port 1907 is the default port used by active agents. If your agent uses a different port, you must either change the value used by the agent or change this value to match the agent.
Timeout	When the agent pool is empty, the timeout value sets the number of seconds that the JavaTest harness waits between tests for an available agent before reporting the test result as an error. If you run tests with one agent, a latent period might occur between the time when the agent completes the test and when it returns to the agent pool. The timeout value must be greater than the agent's latent period. The default value of 180 seconds is usually sufficient.

Agents Currently In Use

Agents Currently In Use lists all agents currently used by the JavaTest harness to run tests. When agents are not running tests they are removed from the list (active agents re-register with the agent pool). Click on an agent in the list to display detailed information about the agent and the test it is running. The detailed information is displayed in the text fields at the bottom and can be used to troubleshoot problems using an agent to run tests.
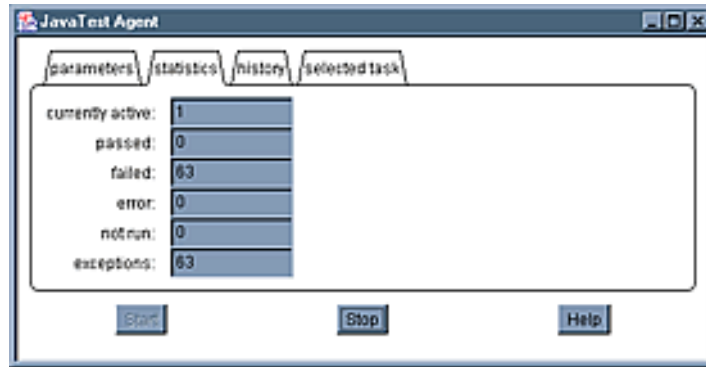
The following table lists and describes the contents of the Agents Currently In Use GUI.

TABLE 23 Agents Currently In Use GUI Contents

Field	Description
Address	Network address of the agent
Tag	Test executed by the agent
Request	Function executed by the agent
Execute	Class executed by the agent
Args	Arguments passed to the class executed by the agent
Localize Args	Checked if the agent uses a map file

Statistics Pane

The statistics tabbed pane in the agent GUI displays the cumulative statistics for the tests in the test suite.


The following table describes the contents of the statistics tabbed pane.

TABLE 24   Statistics Pane Contents

Field	Description
currently active	Number of tests being run by the agent
passed	Number of tests run by the agent that had passing results
failed	Number of tests run by the agent that had failing results
error	Number of tests run by the agent that had errors
not run	Number of tests not run by the agent and not filtered out by the JavaTest harness
exceptions	Number of tests filtered out of the test run by the JavaTest harness

History Pane

The agent GUI uses the history tabbed pane to enable monitoring and troubleshooting agent activity by displaying a dynamic list of tasks that an agent is currently executing and tasks that an agent recently completed. The number of tasks maintained in this list is not configurable.
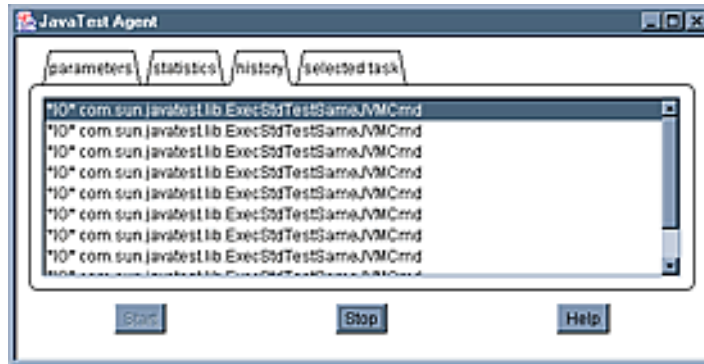
To view the details about a specific task, click on it in the list. The GUI displays the selected task tabbed pane contained details about the task.

Refer to Selected Task Pane for a detailed description of the task information that is displayed.

Each task in the list contains a code indicating its current state. The following table describes the state codes displayed in the GUI.

TABLE 25   Current State Codes

Current State	Description
CONN host:port	Shows that the JavaTest harness agent has an open connection to the JavaTest harness, at the specified network address, and that the JavaTest harness agent is waiting for a request to be sent over the connection. If the JavaTest harness agent is running in active mode, it waits until JavaTest harness sends the request. If the agent is running in passive mode, this state usually appears temporarily because JavaTest harness normally initiates a connection and then immediately sends the request. The host normally is identified by its host name. If JavaTest harness cannot determine the host name, the IP address of the host is shown instead.
EXEC tag	This state shows that the JavaTest harness agent is executing a task on behalf of JavaTest harness. The tag is an identification of the task supplied by JavaTest harness as part of the request.
IO tag	This state shows that the JavaTest harness agent was executing a task on behalf of JavaTest harness but that some exception occurred while trying to send the results to the JavaTest harness.

If a task in the list displays a state from the following table, this indicates that the JavaTest harness agent has completed a request for JavaTest harness. These states correspond to the various possible outcomes of the task and are the same as the

outcomes that the JavaTest harness gets when it runs tests directly (without the assistance of the JavaTest harness agent). The following table describes the states that a task might have.

TABLE 26 JavaTest Agent Completed Request States

State	Description
PASS:	Task completed successfully.
FAIL:	Task indicated that it failed.
ERR:	Task encountered some error before it could properly be executed.
!RUN:	Task has inappropriately indicated that it has not been run. This state must never occur.

Selected Task Pane

The selected task tabbed pane in the agent GUI displays detailed information about a task selected from the task list in the history pane.

Refer to [History Pane](#) for a description of the task list.

The following table describes the contents of the selected task tabbed pane.

TABLE 27   Selected Task Pane Contents

Field	Description
client	Displays the network address (host and port) of the source of the task request. The host is normally identified by its host name, but if JavaTest harness cannot determine the host name, the IP address of the host is displayed instead.
request	Displays the tag that was supplied with the request in order to identify itself.
class	Displays the name of the class that was specified in the request. This is the class that is loaded and run in fulfillment of the request.
args	Displays the arguments that were specified in the request. These arguments are passed to the class that is executed.
result	If and when the task is completed, this field contains the outcome of the task, as indicated by a JavaTest harness Status object.

4

Troubleshooting JavaTest Harness Agents

Because active agents initiate the connection with the JavaTest harness while passive agents wait for a request from the JavaTest harness, troubleshooting is different for each type of agent. The following topics provide guidelines for troubleshooting each type of agent:

- Active Agents
- Passive Agents

Troubleshooting Active Agents

Active agents initiate the connection with the JavaTest harness. You must set up the JavaTest harness agent pool so that the connection is made before running tests.

Errors in configuring, synchronizing, or implementing the connection between the agent and the JavaTest harness are the most probable causes of failure.

Use the Agent Monitor window, the JavaTest harness Test Manager window, the agent GUI, and the following list of actions as a guide when troubleshooting problems running active agents.

1. In the Agent Monitor window, verify that the agent is listed in the agent pool. If the agent is not listed in the agent pool perform the following actions:

    i. Verify that the Listening check box is selected.

    ii. Verify that the agent is configured to contact the correct active host and that the port value of the harness matches the port value used by the agent.

    iii. Check the physical connection between the JavaTest harness platform and the test platform.

2. Verify that the agent moves to Agents Currently in Use when tests are running. If the agent does not move to Agents Currently in Use when tests are running perform the following actions:

i. Use the Configuration Editor in the Test Manager window to verify that the harness is configured to use agents when running tests. See the *JavaTest Harness User's Guide: Graphical User Interface* for detailed description of the Configuration Editor and the Test Manager window.

ii. If you are running the tests using multiple Java Virtual Machines, use the Configuration Editor window to verify that the path you provided in the Java Launcher question is the path of the launcher for the *agent* running tests.

3. If tests are failing or have errors, check the error messages displayed in the Test Manager window. If the error indicates that tests are failing because of missing classes perform the following actions:

   i. Verify that the class paths used to start the agent are correct.

   ii. Use the Configuration Editor window to verify that the harness is correctly configured to use the agent on the test system.

   iii. Run the agent using the -trace option to verify that the paths in the stream messages for the test are correct. If the paths are not correct for the test system, create a map file for the agent to use in translating host specific values into values that the agent can use.

   iv. If a map file was used to run the test, use the Test Run Messages pane to verify that the -mapArgs command is present in the stream messages. If the -mapArgs command is not present, verify that both the agent *and* the harness are configured to use the map file. Use the Configuration Editor window to verify that the harness is configured to use the agent map file.

Troubleshooting Passive Agents

Because passive agents must wait for a request from the JavaTest harness before running tests, the port that the passive agent uses must be the same as that used by the JavaTest harness to send requests.

Errors in configuring, synchronizing, or implementing the connection between the agent and the JavaTest harness are the most probable causes of failure.

Use the Agent Monitor window, the JavaTest harness Test Manager window, the agent GUI, and the following list of actions as a guide when troubleshooting problems running passive agents:

1. Verify that the agent was started before the JavaTest harness started the test run. If not, repeat the test run.

2. Verify that the port value used when starting the agent matches the port value used by the JavaTest harness to send requests.

3. Check the physical connection between the JavaTest harness platform and the test platform.

4. Use the Configuration Editor in the Test Manager window to verify that the harness is configured to use agents when running tests. See the *JavaTest Harness User's Guide: Graphical User Interface* for detailed description of the Configuration Editor and the Test Manager window.

5. If you are running the tests using multiple Java Virtual Machines, use the Configuration Editor window to verify that the path you provided in the Java Launcher question is the path of the launcher for the *agent* running tests.

6. If tests are failing or have errors, check the error messages displayed in the Test Manager window. If the error indicates that tests are failing because of missing classes, perform the following actions:

    i. Verify that the class paths used to start the agent are correct.

    ii. Use the Configuration Editor window to verify that the harness is correctly configured to use the agent on the test system.

    iii. Run the agent using the `-trace` option to verify that the paths in the stream messages for the test are correct. If the paths are not correct for the test system, create a map file for the agent to use in translating host-specific values into values that the agent can use.

    iv. If a map file was used to run the test, use the Test Run Messages pane to verify that the `-mapArgs` command is present in the stream messages. If the the `-mapArgs` command is not present, verify that both the agent *and* the harness are configured to use the map file. Use the Configuration Editor window to verify that the harness is configured to use the agent map file.

Glossary

A

Active Agent

An agent that initiates a connection to the JavaTest harness.

Active agents enable you to run tests in parallel using many agents at once and to specify the test machines at the time you run the tests. Use the Agent Monitor window to view the list of registered active agents and synchronize active agents with the JavaTest harness before running tests.

Agent

A lightweight application that receives tests from the test harness, runs them on the implementation being tested, and reports the results back to the test harness. Normally, test agents are only used when the TCK and implementation being tested are running on different platforms. When running tests on a platform other than the one running the JavaTest harness, you must use an agent. The JavaTest harness uses the following three types of agents:

- Active agents
- Passive agents
- Serial agents

Agent Monitor

The JavaTest harness window used to synchronize active agents and to monitor agent activity. The Agent Monitor window displays the agent pool and the agents currently in use. To open the Agent Monitor window, choose Tasks **->** Monitor Agent Activity from the menu bar.

Agent Pool

A list in the Agent Monitor of the active agents that are connected with the JavaTest harness and available to run tests. Agents are removed from the agent pool when they are running tests.

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Parameters

Values used to configure an agent. The agent parameters can be set at the time the agent is started or, if the agent GUI is used, from the Parameters tab after the agent GUI has started.

Passive Agent

Agents that must wait for a request from the JavaTest harness before they can run tests.

The JavaTest harness initiates connections to passive agents as needed. Passive agents are simpler, but less flexible than active agents because you must specify the test machine as part of the test configuration, not at the time you run the tests. Passive agents do not allow you to run tests in parallel.

Q

R

S

Serial Agent

Use serial mode (serial agent) when you want the agent to use an RS-232 serial connection. Serial agents wait for the JavaTest harness to initiate the connection. Infrared, parallel, USB, and firewire connections can also be added through the JavaTest API by modeling the existing serial system.

T

U

V

W

X

Y

Z

Index

starting a JavaTest agent, 9, 10
starting an agent applet, 14, 15
statistics tabbed pane, 28
statistics tabbed pane, agent GUI, 28

T

task tabbed pane, 31
task tabbed pane, agent GUI, 31
tasks in history pane, 23
test classes, 13
troubleshooting, 32, 33, 34
troubleshooting active agents, 33
troubleshooting JavaTest agents, 32
troubleshooting passive agents, 34

W

window, Agent Monitor, 26