HP Diagnostics

for the Windows® and UNIX® operating systems

Software Version: 8.00

Installation and Configuration Guide

hp

i n v e n t

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Third-Party Web Sites

HP provides links to external third-party Web sites to help you find supplemental information. Site content and availability may change without notice. HP makes no representations or warranties whatsoever as to site content or availability.

Copyright Notices

Trademark Notices

Java™ and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Oracle® is a registered US trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of The Open Group.

Documentation Updates

This guide's title page contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates, or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign-in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

You can visit the HP Software Support web site at:

http://www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software Support Online provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the HP Software Support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Table of Contents

PART III: INSTALLATION AND SETUP OF THE JAVA AND .NET AGENT (PROBES)

PART IV: CUSTOM INSTRUMENTATION FOR MONITORING JAVA AND .NET APPLICATIONS

PART V: ADVANCED CONFIGURATION OF THE DIAGNOSTICS SERVER AND THE JAVA AND .NET AGENTS

**PART VI: CONFIGURING COMMUNICATIONS THROUGH
PROXIES AND FIREWALLS**

PART VII: CONFIGURING DIAGNOSTICS METRICS COLLECTORS

PART VIII: SETTING UP INTEGRATION WITH OTHER HP SOFTWARE PRODUCTS

PART IX: APPENDIXES

Welcome to This Guide

Welcome to the *HP Diagnostics Installation and Configuration Guide*. This guide describes how to install and configure the HP Diagnostics components. This guide also explains how to set up integration with other HP Software products.

How This Guide Is Organized

This guide contains the following parts:

Part I   Introduction

Provides the information and instructions to plan and prepare for the installation and configuration of the Diagnostics components.

Part II   Installation of the Diagnostics Server and Collectors

Describes how to install and configure the HP Diagnostics Server.

Part III   Installation and Setup of the Java and .NET Agent (Probes)

Describes the processes for installing and configuring the Diagnostics Agents.

Part IV   Custom Instrumentation for Monitoring Java and .NET applications

Describes how to control the instrumentation that HP Diagnostics applies to the classes and methods of monitored applications to enable it to gather the performance metrics.

Part V   Advanced Configuration of the Diagnostics Server and the Java and .NET Agents

Describes advanced configuration of the Diagnostics Server, and the Diagnostics .NET and Java Probes.

Part VI   Configuring Communications through Proxies and Firewalls

Describes how to set up your Diagnostics platform using different communication channels.

Part VII   Configuring Diagnostics Metrics Collectors

Describes metrics capture and how to configure the metric collectors.

Part VIII   Setting Up Integration with Other HP Software Products

Describes the necessary steps to set up LoadRunner, Performance Center, Business Availability Center and TransactionVision for integration with HP Diagnostics.

Part IX   Appendixes

Describes more detailed and advanced information about selected topics.

HP Diagnostics Online Documentation

Your HP Diagnostics application comes with the following documentation:

➤ **Diagnostics User's Guide and Online Help.** Explains how to use HP Diagnostics to analyze the performance of your enterprise applications. You access the online help for Using HP Diagnostics from the **Help** button in Diagnostics or from the help menu in the integrated HP Software product. You access the PDF version of this guide from the Windows Start menu (**Start** > **Programs** > **HP Diagnostics Server** > **User Guide)**, from the **Documentation** directory on the HP Diagnostics installation disk, or from the Diagnostics Server installation directory.

➤ **Diagnostics Installation and Configuration Guide.** Explains how to install and configure the Diagnostics components and how to configure Diagnostics for integration with other HP Software products. You access this guide from the Help button in Diagnostics or from the help menu in the integrated HP Software product. You can also access this guide from the Windows Start menu (**Start** > **Programs** > **HP Diagnostics Server** > **Install Guide)**, from the **Documentation** directory on the Diagnostics installation disk, or from the Diagnostics Server installation directory.

➤ **Readme.** Provides last-minute technical and troubleshooting information about HP Diagnostics. The file is located in the HP Diagnostics installation disk root directory.

➤ **Diagnostics Probe for Java Installation Quick Start.** Provides the basic instructions for installing the Diagnostics Probe for Java and is available in the **Documentation** directory on the HP Diagnostics installation disk or from the Windows Start menu (**Start** > **Programs** > **HP Java Agent** > **QuickStart**).

➤ **Diagnostics Profiler for Java Installation and User's Guide.** Describes how to install, configure and use the Diagnostics Profiler for Java. You access this guide online by clicking **Help** in the Diagnostics Profiler for Java.

➤ **Diagnostics Profiler for .NET Installation and User's Guide.** Describes how to install, configure and use the Diagnostics Profiler for .NET. You access this guide online by clicking **Help** in the Diagnostics Profiler for .NET.

Note: The information in the Diagnostics Profiler guides is also included in the **Diagnostics Installation and User's Guide**.

Additional Online Resources

Troubleshooting & Knowledge Base accesses the Troubleshooting page on the HP Software Support Web site where you can search the Self-solve knowledge base. Choose **Help** > **Troubleshooting & Knowledge Base**. The URL for this Web site is http://h20230.www2.hp.com/troubleshooting.jsp.

HP Software Web site accesses the HP Software Web site. This site provides you with the most up-to-date information on HP Software products. This includes new software releases, seminars and trade shows, customer support, and more. Choose **Help** > **HP Software Web site**. The URL for this Web site is www.hp.com/go/software.

HP Software Support accesses the HP Software Support Web site. This site enables you to browse the Self-solve knowledge base. You can also post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. Choose **Help** > **HP Software Support**. The URL for this Web site is www.hp.com/go/hpsoftwaresupport.

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:
http://h20230.www2.hp.com/new_access_levels.jsp

To register for an HP Passport user ID, go to:
http://h20229.www2.hp.com/passport-registration.html

Part I

Introduction

1

Preparing to Install HP Diagnostics

Before you install HP Diagnostics, please read the following information and instructions that will help you to plan and prepare for the installation and configuration of the Diagnostics components.

This chapter includes:

➤ HP Diagnostics Components and Data Flow on page 22

➤ Supported Application Servers and Environments on page 25

➤ System Requirements for the Diagnostics Components on page 26

➤ Information Required for Installation on page 33

➤ Pre-installation Considerations on page 39

➤ Recommended Order of Installation on page 40

➤ Licensing HP Diagnostics on page 42

➤ Upgrading from Earlier Versions of Diagnostics on page 42

HP Diagnostics Components and Data Flow

HP Diagnostics consists of the following components:

➤ **Diagnostics Probes.** Responsible for capturing events from your application, aggregating the metrics, and sending the performance metrics to a Diagnostics Server. The Probe captures events such as method invocations, the beginning and end of business transactions, and server transactions.

Note: The functionality of the Java probe is provided by the HP Diagnostics/ TransactionVision Java Agent (Java agent). The functionality of the .NET probe is provided by the HP Diagnostics/TransactionVision .NET Agent (.NET agent). These agents combine the capabilities of the Diagnostics probe and the TransactionVision sensors into a single component.

The agents are configured to serve as probes in a Diagnostics environment or as sensors in a TransactionVision environment and for combined environments, the agent simultaneously serves as both the probe and the sensor.

➤ **Diagnostics Collectors.** To gather data from external ERP/CRM environments including Oracle 10g Database, SQL Server, IBM WebSphere MQ or SAP NetWeaver - ABAP system, you install the Diagnostics Collector and define specific instances of these systems to be monitored. Each instance of a collector is represented as a Probe in the Diagnostics UI.

➤ **Diagnostics Servers.** Responsible for working with the Probes and with other HP Software products to capture, process, and present the performance metrics for your application.

The Diagnostics Server processes and further aggregates the data that it receives from each of the Probes that report to it, and formats the information so that it can be displayed in the views of the user interface.

The Diagnostics Server in Commander mode is responsible for the command and control functions between the various Diagnostics components and the components of the other products with which Diagnostics is working.

The Diagnostics Server in Commander mode keeps track of the location and status of the other Diagnostics components, and is the communication hub between the other components.

The Diagnostics Server in Commander mode is also responsible for displaying the performance information for the monitored applications in the charts and graphs of the Diagnostics views. If you are using Diagnostics with other HP Software products you can also access the Diagnostics views from the user interface of the other products.

A Diagnostics deployment may consist of one or many Diagnostics Servers. If there is only one Diagnostics server in your deployment, it is configured in Commander mode and must perform both the Commander and Mediator roles. If there is more than one Diagnostics Server in a deployment, one must be configured in Commander mode, and all the rest in Mediator mode.

Diagnostics Data Flow

The following diagram illustrates the data flow among Diagnostics components.
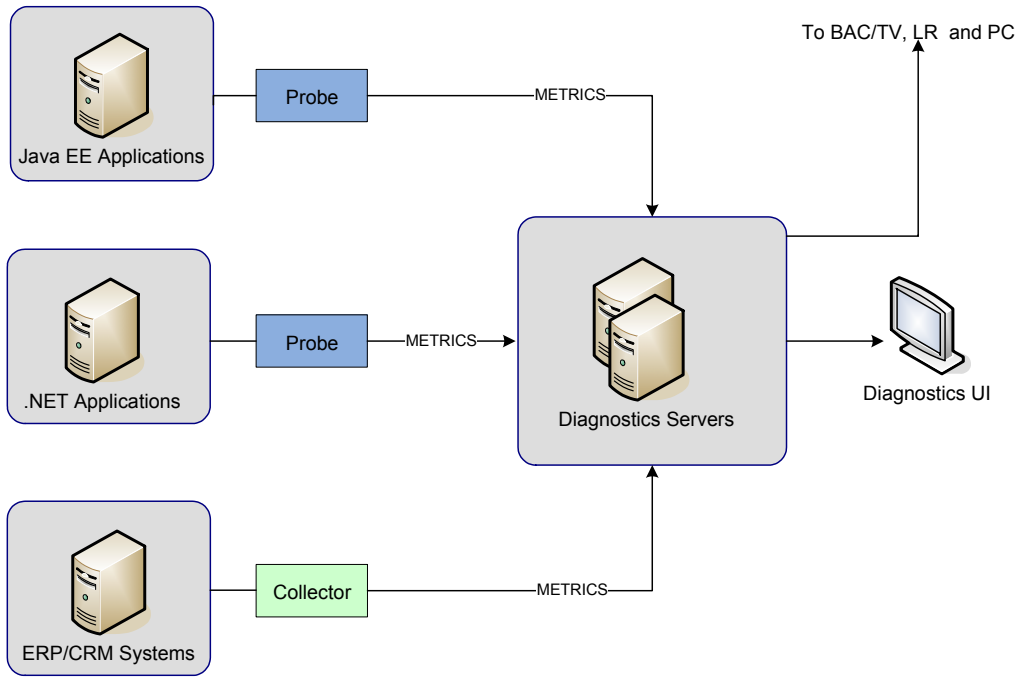
In the diagram displayed above, there is one Diagnostics Server in Commander mode connected to a one or more Diagnostics Servers in Mediator mode. Each Diagnostics Server in Mediator mode is connected to a number of probes or collectors. The Diagnostics probes and the Diagnostics collector capture events from the monitored applications.

The probes and collectors send these captured performance metrics to the Diagnostics Server in Mediator mode which filters and aggregates the events. This information is sent to the Diagnostics Server in Commander mode, which displays the processed metrics in customizable views.

When you are monitoring your application in real time, you access the Diagnostics UI from Business Availability Center or directly from the Diagnostics Server. During a load test, you access the Diagnostics UI from LoadRunner or Performance Center.

You can also access the Java Diagnostics Profiler and the .NET Diagnostics Profiler directly from the probe whose Profiler you want to view or through the Diagnostics UI.

Supported Application Servers and Environments

HP Diagnostics supports the monitoring of:

➤ **Java EE-based application servers.** Including: WebLogic, WebSphere, Oracle, Sun Java Enterprise Server, JBoss, and more.

➤ **.NET-based application servers.** HP Diagnostics supports the Microsoft IIS .NET Framework.

➤ **SAP NetWeaver - ABAP systems.**

➤ **Oracle 10g databases.**

➤ **SQL Server databases.**

➤ **IBM WebSphere MQ.**

For the most recent information on supported environments refer to the Diagnostics Product Availability Matrix at http://support.openview.hp.com/sc/support_matrices.jsp.

Note: HP Diagnostics monitors only those methods that are invoked by the application server, and are instrumented by the Probe. For information about instrumentation and how to instrument for additional methods, see Chapter 9, "Custom Instrumentation for Java Applications" and Chapter 10, "Custom Instrumentation for .NET Applications".

System Requirements for the Diagnostics Components

The following section describes the recommended system configurations for hosting the components of HP Diagnostics. Refer to the deployment diagram in the previous section to understand the component hosts that are described in this section.

When you select the machines that will host the Diagnostics components, you must make sure that the system configuration of the machines supports the processing load and the number of applications that you will be monitoring.

Requirements for the Diagnostics Server

Diagnostics uses the Java 1.6 JVM for the Server and the Collector.

Java 1.6 JVM is supported on the following operating systems and requires the patches listed below.

For the most recent information on supported environments refer to the Diagnostics Product Availability Matrix at http://support.openview.hp.com/sc/support_matrices.jsp.

Windows supported versions:

➤ 32-bit: Windows XP, Windows 2003, Windows 2003 R2, Windows 2008

➤ 64-bit: Windows 2003, Windows 2003 R2, Windows 2008

HP-UX supported versions:

➤ HP PA-RISC: HP-UX 11i v1 (11.11), HP-UX 11i v2 (11.23), HP-UX 11i v3 (11.31)

➤ HP Itanium: HP-UX 11i v2 (11.23), HP-UX v3 (11.31)

HP-UX required patches:

➤ To run Java on an HP-UX system (for either the Diagnostics Server or Java Probe), you must make sure that all Java required patches are installed. The Java required patches depend upon the Java version and the HP-UX version. For HP-UX 11.11 the PHCO_29903 patch is required. See the following website for details on additional patches that may be required for this and other HP-UX operating system versions: http://h18012.www.1.hp.com/java/patches/index.html

➤ Java 5.0 Quality Pack

➤ A linker patch is required. The patch ID is PHSS_35385 for HP-UX 11i v1 (11.11) systems, PHSS_37201 for HP-UX 11i v2 (11.23) systems, or PHSS_37202 for HP-UX 11i v3 (11.31) systems.

➤ For HP PA-RISC HP-UX 11i v2 (11.23) Diagnostics Server the following patches (and any patch dependencies) are required: PHKL_37121, PHSS_37947.

Linux supported versions:

➤ Red Hat 2.1, Red Hat Enterprise Linux 3.0, 4.0, 5.0

Solaris supported versions:

➤ Solaris 10, 9, 8

Solaris required patches:

➤ See http://sunsolve.sun.com/show.do?target=patches/JavaSE

Requirements for the Diagnostics Client Host

The user interface for Diagnostics is presented in a web browser using a JAVA applet that requires JRE 1.5 or above to be installed on the machine that displays the UI.

Requirements for the Diagnostics Server Host

The system configuration requirements for the host of the Diagnostics Server depends upon the number of probes and Diagnostics Servers in Mediator mode that are reporting to it. When a Diagnostics Server is designated as the Diagnostics Server in Commander mode, each Diagnostics Server in Mediator mode that reports to it counts as one half of a probe when determining the system configuration requirements.

Note: The requirements in the tables are guidelines, based on tests run with probes monitoring applications with an average number of server requests and server request depths. The actual system requirements that you will need and actual number of supported probes are affected by several characteristics of the monitored environment such as number of server requests, server request depth (methods in the call profile), number of trended methods, and number of out-bound calls. The type of server request also affects the requirements. For example, Web services require more resources and trimming doesn't apply to them.

The following table lists the desired system requirements for the host of a Diagnostics Server with Java Probes.

Platform	Item	Up to 50 Java Probes	Up to 100 Java Probes	Up to 200 Java Probes
Windows	CPU	2x 2.4 GHz	2x 2.8 GHz	2x 3.4 GHz
Windows	Memory	4 GB	4 GB	4 GB
Solaris	CPU	2x Ultra Sparc 3	2x Ultra Sparc 4	2x Ultra Sparc 4
Solaris	RAM	4 GB	4 GB	4 GB
Linux	CPU	2x 2.0 GHz	2x 2.4 GHz	2x 2.8 GHz
Linux	Memory	2 GB	4 GB	4 GB
HP-UX	CPU	PA-RISC 2x 650 MHz	PA-RISC 2x 699 MHz	PA-RISC 2x 750 MHz
HP-UX	Memory	2 GB	4 GB	4 GB
All	Heap Size	512 M	750 M	1280 M
All	Disk	4 GB per probe		
Notes regarding the test environment				
➤ Call profile (depth of method calls) for each Server Request: 5 ➤ Number of unique Server Requests per probe: 23				

The following table lists the desired system requirements for the host of a Diagnostics Server with .NET Probes.

Platform	Item	Up to 10 .NET Probes	Up to 20 .NET Probes	Up to 50 .NET Probes
Windows	CPU	1x 1.0 GHz	1x 2.0 GHz	2x 2.4 GHz
Windows	Memory	768 MB	1 GB	3 GB
Solaris	CPU	1x Ultra Sparc 2	2x Ultra Sparc 2	2x Ultra Sparc 3
Solaris	RAM	1 GB	1.5 GB	3 GB
Linux	CPU	1x 1.0 GHz	1x 2.0 GHz	2x 2.4 GHz
Linux	Memory	768 MB	1 GB	3 GB
HP-UX	CPU	1x 1.0 GHz	1x 2.0 GHz	2x 2.4 GHz
HP-UX	Memory	768 MB	1 GB	3 GB
All	Heap Size	350 M	700 M	1400 M
All	Disk	3 GB per probe		

Scalability numbers shown on the next page are derived based on the following reference hardware configuration:

Platform: Windows

Operating System: Windows Server 2008, 64bit

CPU: Intel Xeon 5160 @ 3.00Ghz (quad core)

Memory: 8GB

Disk I/O: Smart Array P400i, 2SCSI drives in RAID 0 (136GB) [130MB/S, sequential read and write]

Java Heap: 5.9GB (-Xmx6096m); 64bit JVM

Disk Space: 2-4 GB per probe (overall disk space can be adjusted by changing retention intervals)

Network: 1 GBps

Note: Note, 64bit OS and a 64bit JVM is highly recommended since it allows better scaling of memory (32bit JVMs on Windows have 1.2GB limit). Further, note that 64bit JVMs require more memory than their 32bit counterpart due to doubling of references.

Scalability numbers for the above reference hardware.

Up to 100 Java Probes:	100 Server Requests per Probe, 78 methods per Call Profile pulled every 45s (default)
Up to 400 Java Probes:	25 Server Requests per Probe, 78 methods per Call Profile pulled every 45s (default)
Up to 150 Java Probes:	150 Server Requests per Probe, 25 methods per Call Profile pulled every 240s
Up to 230 Java Probes:	100 Server Requests per Probe, 25 methods per Call Profile pulled every 240s
Up to 40 Java Probes:	75 Web Service Operations, 10 unique consumers per Web Service Operation, 25 methods per Call Profile pulled every 45s (default)
	Note, this load configuration requires 7 GB disk space per probe

See also "Configuring the Diagnostics Server for a Large Installation" on page 360.

Notes:

➤ For environments with many Probes, better performance can typically be achieved by having two or more instances of the Server on the same host and distributing the Probes among each Server instance.

➤ For configuration considerations related to the Diagnostics performance data that is stored on the host for the Diagnostics Server in Commander mode, see "Pre-Installation Data Management Considerations" on page 682.

➤ For information on how to optimize the Diagnostics server to handle more probes, see "Optimizing the Diagnostics Server in Production to Handle More Probes" on page 379.

Requirements for the Diagnostics Probe for Java Host

The overhead that the Diagnostics Probe for Java (Java Probe) imposes on the system being monitored is extremely low. The following are the recommendations for memory and disk space that will support the Probe's processing:

Platform:	All Platforms
Memory:	50 MB Additional RAM
Free Hard Disk Space:	200 MB Additional Space

Note: The additional memory must be allocated to the max heap for the JVM by adding **–Xmx???m** to the java settings in the application's start up script.

For information on setting the max heap for the Java Probe, see "Adjusting the Heap Size for the Java Probe in the Application Startup Script" on page 189.

Requirements for the Java Diagnostics Profiler User Interface Host

The user interface for the Diagnostics Profiler for Java is presented in a web browser using a JAVA applet that requires JRE 1.5 or above to be installed on the machine that displays UI. This machine must be able to access the Diagnostics Profiler URL: http://<probe_host>:<probeport>/profiler.By default, the probes are assigned to the first available port beginning at 35000.

Requirements for the Diagnostics Probe for .NET Host

The overhead that the .NET Agent imposes on the system being monitored is extremely low. The following are the recommendations for memory and disk space that will support the Agent's processing:

Platform	All Supported Platforms.
Memory	60 MB Additional RAM
Free Hard Disk Space	200 MB Additional Space
.NET Framework	1.1 or later

Requirements for the .NET Diagnostics Profiler User Interface Host

The user interface for the .NET Diagnostics Profiler is presented using DHTML/XML/XSLT/JScript technology that requires IE6 or later. The machine that is to be used to present the UI must be able to access the .NET Diagnostics Profiler URL: http://<probehost>:<probeport>/profiler. The probes are assigned to the first available port within the range defined during the Probe installation. The default port range is 35000 - 35100.

Requirements for the Diagnostics Collector Host

The Collector can be installed on any machine that can interact with the host machines of the SAP NetWeaver - ABAP, Oracle or SQL Server application from which it is collecting data.

Information Required for Installation

Before you start installing the Diagnostics components, you should carefully plan the configuration of the Diagnostics components and the machines that will host them. You should also consider the location of the component hosts within your network topography.

The tables in the following sections have been provided to help you gather the information that you must provide during the installation of the Diagnostics components.

Note: When you are installing Business Availability Center with Diagnostics, when entering the names for the hosts of the Diagnostics components it is strongly recommended that you use fully qualified host names, that is, the machine name and the domain name.

Diagnostics Server

Information Required	Where to find it	Value
For a Diagnostics Server in Commander mode, the location of the HP Diagnostics license that has been generated for the machine that will host the server.	Contact your HP Software support person to request a license and place it in a folder where it can be accessed from the Diagnostics Server installer.	
For a Diagnostics Server in Mediator mode, the URL for the Diagnostics Server in Commander mode.	After the Diagnostics Server in Commander mode has been installed, available from the System Health Monitor. (See Appendix D, "Using the System Health Monitor.")	
Will the Diagnostics Server be used in a SaaS environment or in the Business Availability Center environment?		

Java Probe

➤ **HP Software Product and Diagnostics Server Information**

Information Required	Where to find it	Value
Name of HP Software product(s) that will use the Probe	Choose according to product license. ➤ **Performance Center / LoadRunner** ➤ **Business Availability Center**	
Diagnostics Server in Commander mode URL	System Health Monitor (See Appendix D, "Using the System Health Monitor.")	
Diagnostics Server Event Host	System Health Monitor (See Appendix D, "Using the System Health Monitor.")	
Diagnostics Server Event Port	System Health Monitor (See Appendix D, "Using the System Health Monitor.")	**Default value**: 2006

➤ **Probe and Application Server Information**

Information Required	Where to find it	Value
probe name	A *unique* string; Created by user. **Note:** The name of the probe should indicate the probe type, to help you distinguish between the different types of probes	For example: JavaProbe1
probe group	Used to relate probes so that their metrics can be reported as a logical group. This is user defined at the time that the probe is installed.	**Default value:** DefaultProbeGroup
Type of application server that the Java Probe will be monitoring	The host system administrator.	
Application Server configuration properties	The host system administrator. The details will vary according to the application server that you are using.	
Location of the JRE executable	The host system administrator. This depends upon the type of application server you are configuring. See "Running the JRE Instrumenter" on page 141.	

.NET Probe

➤ **Diagnostics Server Information**

Information Required	Where to find it	Value
URL of Diagnostics Server in Commander mode	System Health Monitor URL that the Probe will use to register with the Diagnostics Server. This is not required for using the.NET Diagnostics Profiler in a standalone mode.	
Diagnostics Server Event Host	System Health Monitor This is not required for using the .NET Diagnostics Profiler in a standalone mode.	
Diagnostics Server Event Port	System Health Monitor This is not required for using the.NET Diagnostics Profiler in a standalone mode.	**Default value:** 2612

➤ **Probe and Port Information**

Information Required	Where to find it	Value
probe group	Used to relate probes so that their metrics can be reported as a group. This is user defined at the time that the probe is installed.	**Default value:** Default
Web Port Min	System Administrator. The lowest port number in a range of ports on the probe host that can be assigned to the probe.	**Default value:** 35000
Web Port Max	System Administrator The highest port number in a range of ports on the probe host that can be assigned to the probe.	**Default value:** 35100

Pre-installation Considerations

Note: Before you install any of the Diagnostics components on a Windows machine, make sure that the **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Services** window is not open.

LoadRunner and Performance Center Host Machines

➤ If LoadRunner is already installed, ensure that the Controller and main LoadRunner window are closed before you install the LoadRunner Diagnostics Add-in.

➤ The LoadRunner Diagnostics Add-in is not required for Performance Center.

➤ The time and time-zone settings of the host machines for the Diagnostics components must be consistent. You will encounter time-difference problems if the time is not properly set.

Diagnostics Server

➤ The performance metrics for HP Diagnostics cannot be displayed until the Diagnostics Server in Commander mode has been licensed with a valid license. For more information on obtaining a license and other licensing issues, see Chapter 3, "Licensing HP Diagnostics."

Note: For optimal display of the Diagnostics views, ensure that your screen resolution is at least 1024x768.

Diagnostics Probe for Java

➤ The Java Probe must be installed on the same system as the Java application under test.

➤ The Diagnostics Profiler for Java will operate in an unlicensed mode with load restrictions until it is able to connect to a Diagnostics Server in Commander mode that has been properly licensed. For more information on obtaining a license and other licensing issues, see Chapter 3, "Licensing HP Diagnostics."

Diagnostics Probe for .NET

➤ The .NET Probe must be installed on the same system as the .NET application under test.

➤ The Diagnostics Profiler for .NET will operate in an unlicensed mode with load restrictions until it is able to connect to a Diagnostics Server in Commander mode that has been properly licensed. For more information on obtaining a license and other licensing issues, see Chapter 3, "Licensing HP Diagnostics."

Recommended Order of Installation

Careful planning and preparation for installing the components of HP Diagnostics can help you to avoid complications and errors, and allow you to complete the installation and configuration steps quickly.

Before you start, review the following information to get an overview of the entire installation and configuration process.

Note: The order of the installation presented here is the recommended order of installation for the products and components. Deviation from this recommended order of installation could increase the complexity of the installation process and produce unpredictable results.

---

 1 Check the system requirements and installation considerations.

See "System Requirements for the Diagnostics Components" on page 26.

 2 Install the Diagnostics Server.

See Chapter 2, "Installing the Diagnostics Server."

 3 Install and configure the Diagnostics Probe(s) and/or Collector.

➤ For a Java EE environment, see Chapter 5, "Installing Java Agents."

➤ For a .NET environment, see Chapter 8, "Installing .NET Agents (Probes)."

➤ For Oracle, SAP NetWeaver-ABAP and SQL Server environments, see Chapter 4, "Installing Diagnostics Collectors."

 4 For Java Probes, configure the application server to work with the probes.

For more information, see Chapter 7, "Configuring Application Server Startup Scripts to Work with the Java Agent."

 5 If HP Diagnostics is integrated with LoadRunner, Performance Center or Business Availability Center, it must be set up to use HP Diagnostics.

➤ For Business Availability Center, see Chapter 23, "Setting Up Business Availability Center to Use Diagnostics."

➤ For LoadRunner integration, install the LoadRunner Diagnostics Add-in (see Chapter 24, "Installing the LoadRunner Diagnostics Add-in") and set up LoadRunner to use Diagnostics (see Chapter 25, "Setting Up LoadRunner to Use HP Diagnostics").

➤ For Performance Center, see Chapter 26, "Setting Up Performance Center to Use HP Diagnostics."

Licensing HP Diagnostics

To be able to see the metrics for applications in the Diagnostics views, you must obtain a valid license for the Diagnostics Server in Commander mode. The license is a node-locking license that unlocks the Diagnostics views that are displayed by the Diagnostics Server in Commander mode and removes the load restriction from the Profiler views that are accessed from the Diagnostics Probes. For more information on obtaining a license and other licensing issues, see Chapter 3, "Licensing HP Diagnostics."

Upgrading from Earlier Versions of Diagnostics

If you are installing HP Diagnostics into an environment where a previous version of the product was installed, or in an environment where other HP Software products need to be upgraded so that the features of Diagnostics can be accessed, follow the instructions in Appendix G, "Upgrading Diagnostics and Other HP Software Products." These instructions guide you to the appropriate instructions for upgrading your current HP Software products and the Diagnostics components.

Part II

Installation of the Diagnostics Server and Collectors

44

2

Installing the Diagnostics Server

Instructions are provided for installing the Diagnostics Server on Windows and UNIX machines.

This chapter includes:

➤ Installing the Diagnostics Server on Windows and UNIX on page 46

➤ Starting and Stopping the Diagnostics Server on page 54

➤ Verifying the Diagnostics Server Installation on page 56

➤ Licensing the Diagnostics Server in Commander Mode on page 57

➤ Configuring the Diagnostics Server on page 57

➤ Determining the Version of the Diagnostics Server on page 57

➤ Uninstalling the Diagnostics Server on page 58

Installing the Diagnostics Server on Windows and UNIX

The following sections provide detailed instructions for installing the Diagnostics Server. These instructions apply to:

➤ a Windows environment

➤ most UNIX environments using either a graphical installation or a console mode installation.

The UNIX example instructions are for component installation on a Solaris machine. These same instructions should apply for the other certified UNIX platforms.

Note: If an earlier version of the Diagnostics Server is installed on your machine, see Appendix G, "Upgrading Diagnostics and Other HP Software Products."

Important: The following instructions assume an understanding of UNIX console screens and commands. For more information about working with UNIX screens and commands, see Appendix I, "Using UNIX Commands."

This section includes:

➤ Launching the Diagnostics Server Installer

➤ Running the Installation

Launching the Diagnostics Server Installer

Depending on your environment, you launch the installer differently.

To launch the Windows installer:

1 **Run the setup.exe file in the root directory of the HP Diagnostics installation disk.** The Diagnostics setup program begins and displays the installation menu page.

The installer displays the HP Diagnostics main installation menu.

2 **Click Diagnostics Server to launch the installer.** This will install the 32-bit Windows version of the Diagnostics Server. If you want to install the 64-bit version of the Diagnostics Server you will need to **Browse the DVD** to locate the **Diagnostics_Servers** directory and double-click the **DiagnosticsServerSetupWinx64_<version>.exe** file.

Note: To launch the installer from a different location, copy the executable file **DiagnosticsServerSetupWin_<version>.exe** (32-bit) or **DiagnosticsServerSetupWinx64_<version>.exe** (64-bit that runs with a 64-bit JVM) from the **<HP Diagnostics Installation Disk>\Diagnostics_Servers** directory to the new location, and then double-click on it.

To launch the UNIX installer:

1 From the **<HP Diagnostics Installation Disk>/Diagnostics_Servers** directory, copy the installer **DiagnosticsServerSetup<platform>_<version>.bin** to the machine where the Diagnostics Server is to be installed.

2 Change the mode of the installer file to make it executable.

3 Run the installer.

➤ To run the installer in the graphical mode enter the installer **DiagnosticsServerSetup<platform>_<version>.bin** filename at the UNIX command prompt. An example is shown below:

```
./DiagnosticsServerSetupSolaris_8.00.bin
```

➤ To run the installer in console mode enter the installer **DiagnosticsServerSetup<platform>_<version>.bin** filename with the -**console** option, at the UNIX command prompt. An example is shown below:

```
./DiagnosticsServerSetupSolaris_8.00.bin -console
```

Running the Installation

After you have launched the installer, the software license agreement opens and you are ready to run the installation.

1 Accept the software license agreement.

The software license agreement is displayed.

Read the agreement and accept the terms of the agreement.

Select **Next** to continue with the installation.

Note: For the UNIX console mode installer, you can press ENTER as you read, to move to the next page of text, or type q to jump to the end of the license agreement.

---

2 Specify the location where the Diagnostics Server is to be installed.

Accept the default installation directory or type the path to a different location. In the Windows installer (or UNIX graphical mode installer) you can click **Browse** to navigate to another directory.

Select **Next** to continue with the installation.

Note: In the UNIX console mode installer you press 1 to select Next, 2 for Previous, 3 to Cancel or 4 to Re-display the screen.

---

3 Indicate the mode of the Diagnostics Server that you are installing.

The Diagnostics deployment that you are setting up may consist of one or many Diagnostics Servers. If there is only one Diagnostics Server in your deployment, it is installed in Commander mode and can perform both Commander and Mediator roles. When there is more than one Diagnostics Server in a deployment, one is configured in Commander mode, and all the rest in Mediator mode.

➤ If this is the only Diagnostics Server in your deployment, select **Commander Mode**.

➤ If there is more than one Diagnostics Server in your deployment, and the one that you are currently installing is to be configured in Commander mode, then select **Commander Mode**. Otherwise select **Mediator Mode**.

Select **Next** to continue with the installation.

Note: At this stage, the installation differs depending on whether you are installing the Diagnostics Server in Commander mode or in Mediator mode.

➤ To install the Diagnostics Server in Commander mode, continue with "Installing the Diagnostics Server in Commander Mode" on page 49.

➤ To install the Diagnostics Server in Mediator mode, continue with "Installing the Diagnostics Server in Mediator Mode" on page 52.

Installing the Diagnostics Server in Commander Mode

If you are installing the Diagnostics Server in Commander mode, continue as follows:

1 Select a time synchronization method.

In order for diagnostics data to be correlated properly, all the components in the Diagnostics deployment must be time-synchronized.

Select one of the following time synchronization methods:

➤ **Synchronize with an NTP server.** This option applies only if the Diagnostics Server can access an NTP Server outside the firewall. This is the default method.

➤ **Synchronize with the registered Business Availability Center Core Server.** If the Diagnostics Server is to work in a Business Availability Center environment, select this option to synchronize with the Business Availability Center core server.

➤ **Synchronize with system time.** Select this option if the Diagnostics Server is to work in an environment other than Business Availability Center and there is no access to an NTP server.

Select **Next** to continue with the installation.

2 **Indicate whether the Diagnostics Server will be used in an HP Software-as-a-Service (SaaS) environment or with Business Availability Center.**

Select the options that apply to this Diagnostics Server, and select **Next** to continue with the installation.

Note: If you selected only the **HP Software-as-a-Service (SaaS)** option, skip to step 4.

---

3 **Business Availability Center environment only: Provide the path to the Agent and Collector installers.**

Note: You need to have the Diagnostics installation disk available for this step.

To be able to download the Diagnostics Agent and Collector installers from the Diagnostics Configuration page in Business Availability Center, you need to specify the path to the directory on the Diagnostics installation disk where these installers are located (**Diagnostics_Installers)**.

Enter the path to the Diagnostics installers on the Diagnostics installation disk, and select **Next** to continue with the installation.

The installers are automatically copied to the Diagnostics Server installation directory, which Business Availability Center can access. The **Diagnostics_Installers** directory is approximately 1.85 GB so the copy operation can take several minutes to complete.

Note: You can choose to skip this step and always access the Agent and Collector installers directly from the Diagnostics installation disk. Alternatively, you can perform this step manually at a later stage, by copying the Agent and Collector installers from the Diagnostics installation disk (**/Diagnostics_Installers**) to the Diagnostics Server installation directory (**<diagnostics_server_install_dir>/html/opal/downloads**) for Business Availability Center to access.

 4 **Review the pre-installation summary.**

The installation settings that you selected are displayed. Review the information to make sure that you are satisfied.

Note: The estimated total size of the Diagnostics Server in Commander mode installation does not include the size of the Probe installers, if they were made available for Business Availability Center.

You can change your settings by going back to previous installation steps. (For Windows, click **Back**. For UNIX, select **Previous.**)

To start the installation of the Diagnostics Server, select **Next**.

 5 **Close the installation wizard.**

When the installation has completed, review the post-installation summary information to make sure that the installation completed successfully.

Select **Finish** to exit the installation.

Note: On Windows machines, the Diagnostics Server attempts to start automatically. The Diagnostics Server will not start if any other applications are using the default Diagnostics Server ports. For instructions on starting the Diagnostics Server manually, see "Starting and Stopping the Diagnostics Server" on page 54.

6 Upload the Diagnostics license file.

In order to view performance metrics, you must upload a valid HP Diagnostics license file after you have installed the Diagnostics Server in Commander mode.

For instructions on requesting and uploading a valid license file, see Chapter 3, "Licensing HP Diagnostics."

Installing the Diagnostics Server in Mediator Mode

If you are installing the Diagnostics Server in Mediator mode, continue the installation as follows:

1 Provide the location of the Diagnostics Server in Commander mode.

Provide the information that enables the Diagnostics Server in Mediator mode to connect to the Diagnostics Server in Commander mode.

a Enter the host name for the Diagnostics Server in Commander mode.

b Enter the port for the Diagnostics Server in Commander mode.

The default port for the Diagnostics Server in Commander mode is **2006**. If you have changed the port since the Diagnostics Server was installed, you should specify that port number here instead of the default. For information on changing the Diagnostics Server port, see "Changing the Default Diagnostics Server Port" on page 365.

c To allow the installer to check the connectivity to the host and port that you specified, select **Check the Connectivity to the Diagnostics Server**.

If you do not want to check for connectivity problems at this stage, clear the **Check the connectivity to the Diagnostics Server** option so that the installation can proceed.

Select **Next** to continue with the installation.

If you instructed the installer to perform the test for connectivity, it tests the connectivity at this point. If there are negative results, it reports these before proceeding with the next installation step.

2 Indicate whether the Diagnostics Server will be used in an HP Software-as-a-Service (SaaS) environment or with Business Availability Center.

Select the options that apply to this Diagnostics Server, and select **Next** to continue with the installation.

Note: If you selected only the **Business Availability Center** option, skip to step 4.

---

3 If you indicated that the Diagnostics Server is to be used in an HP Software-as-a-Service (SaaS) environment, provide the customer name for the HP Software-as-a-Service (SaaS) environment.

Enter a unique name for the customer in the HP Software-as-a-Service (SaaS) environment.

Select **Next** to continue with the installation.

4 Review the pre-installation summary.

The installation settings that you selected are displayed. Review the information to make sure that you are satisfied.

You can change your settings by going back to previous installation steps. (For Windows, click **Back**. For UNIX, select **Previous.**)

To start the installation of the Diagnostics Server, select **Next**.

5 Close the installation wizard.

When the installation has completed, review the post-installation summary information to make sure that the installation completed successfully.

Select **Finish** to exit the installation.

Note: On Windows machines, the Diagnostics Server attempts to start automatically. The Diagnostics Server will not start if any other applications are using the default Diagnostics Server ports. For instructions on starting the Diagnostics Server manually, see "Starting and Stopping the Diagnostics Server" on page 54.

Starting and Stopping the Diagnostics Server

Instructions for a Windows Machine

To start the Diagnostics Server on a Windows machine:

Select **Start** > **Programs** > **HP Diagnostics Server** > **Start HP Diagnostics Server**.

To stop the Diagnostics Server on a Windows machine:

Select **Start** > **Programs** > **HP Diagnostics Server** > **Stop HP Diagnostics Server**.

Instructions for UNIX or Linux Machines (Using the Nanny)

The *nanny* is a process that runs as a daemon to ensure that the Diagnostics Server is always running. The nanny also starts a LoadRunner agent to allow offline data collation for LoadRunner or Performance Center.

The following procedures start and stop the Diagnostics Server using the nanny.

To start the Diagnostics Server on a UNIX or Linux machine:

1 Ensure that the **M_LROOT** environment variable is defined as the root directory of the Diagnostics Server nanny.

For example, in *ksh*, you could enter the following:

```
export M_LROOT=<diagnostics_server_install_dir>/nanny/<platform>
```

Where <platform> is solaris, linux or hpux. If the **M_LROOT** environment variable is not defined as the root directory, you will encounter the following error:

```
Warning : MDRV: cannot find lrun root directory . Please check your M_LROOT
Unable to format message id [-10791]
m_agent_daemon ( is down )
```

2 Change directories to **$M_LROOT/bin**.

3 Run **m_daemon_setup** with the **-install** option, as in the following example:

```
cd $M_LROOT/bin
$./m_daemon_setup -install
```

On Linux, if you encounter an error you may need to install the libstdc++.so.5 shared library.

To stop the Diagnostics Server on a UNIX or Linux machine:

1 Change directories to **$M_LROOT/bin**.

2 Run **m_daemon_setup** with the **-remove** option, as in the following example:

```
cd $M_LROOT/bin
$./m_daemon_setup -remove
```

Instructions for UNIX or Linux Machines (without using the Nanny)

The following procedures start and stop the Diagnostics Server without using the nanny.

To start the Diagnostics Server on a UNIX or Linux machine:

Run **<diagnostics_server_install_dir>/bin/server.sh**.

To stop the Diagnostics Server on a UNIX or Linux machine:

Terminate the process using a utility such as **kill**.

Verifying the Diagnostics Server Installation

To verify that the Diagnostics Server has been installed correctly, and has started properly, use the System Health Monitor. (For instructions on starting the System Health Monitor, see Appendix D, "Using the System Health Monitor").

According to the recommended installation sequence, after you install the Diagnostics Server you can use the System Health Monitor to verify that the Diagnostics Server was installed and started.

The Diagnostics Server in Commander mode should be displayed on the System Health Monitor as shown in the following example:
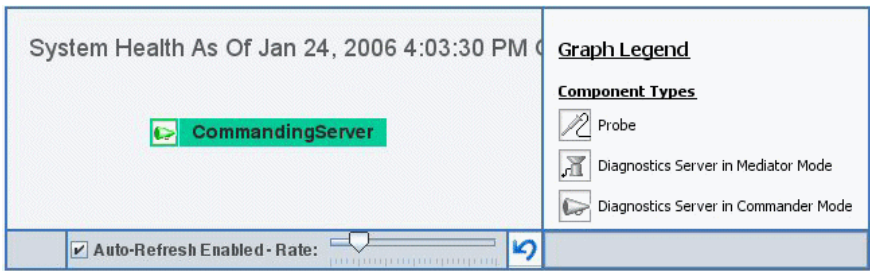
When a Diagnostics Server is deployed in Commander mode, it has both commanding and mediating responsibilities. The Diagnostics Server in Commander mode is represented in the System Health Monitor component map by an icon labelled **Commanding Server**.

When a Diagnostics Server is deployed in Mediator mode, it is represented by a single icon labelled **server-<name of host>**.

The System Health Monitor is part of the Diagnostics Server component. If you are unable to access the System Health Monitor after the Diagnostics Server has been installed, then either you are entering an incorrect URL or the Diagnostics Server did not start. For instructions on starting the Diagnostics Server, see "Starting and Stopping the Diagnostics Server" on page 54.

You can leave the System Health Monitor displayed in your browser to verify the progress of the component installations, and to identify and troubleshoot any problems that you encounter as you proceed through the rest of the component installations.

Licensing the Diagnostics Server in Commander Mode

After you have installed the Diagnostics Server you will need to provide a valid license file. For instructions on requesting a license file and uploading the file, see Chapter 3, "Licensing HP Diagnostics."

Configuring the Diagnostics Server

The Diagnostics Server is installed with a default configuration that enables it to perform effectively in most situations. You may encounter situations where changing the configuration enables better Diagnostics Server performance, or allows it to work in unusual situations.

For information about configuring the Diagnostics Server, see Chapter 14, "Advanced Diagnostics Server Configuration."

Determining the Version of the Diagnostics Server

When you request support, it is useful to know the version of the Diagnostics Server.

To determine the version of the Diagnostics Server:

Click the **Show Help** button from the Diagnostics tool bar and choose **About HP Diagnostics** from the menu.

The About HP Diagnostics dialog box opens, displaying the version of the Diagnostics Server.

Uninstalling the Diagnostics Server

The following section contains instructions for uninstalling the Diagnostics Server.

Uninstalling the Diagnostics Server From a Windows Machine

➤ Uninstall the Diagnostics Server by selecting **Start** > **All Programs** > **HP Diagnostics Server** > **Uninstall HP Diagnostics Server**.

➤ Alternatively, you can execute **uninstaller.exe** which is located in the **<diagnostics_server_install_dir>_uninst directory**.

During the uninstallation process, a message asks if you want to remove specific files. Do the following:

➤ If you want to completely uninstall the Diagnostics Server, as well as any property settings, click **Yes** or **Yes to All**.

➤ If you plan on reinstalling the Diagnostics Server, and want to keep the custom property settings of the Diagnostics Server you are uninstalling, then you need to back up the property files located in the **etc** directory to a new location.

   If you have backed up these files, click **Yes** or **Yes to All**.

   If you have not yet backed up these files, select **No** or **No to All**.

Uninstalling the Diagnostics Server From a UNIX Machine

You can uninstall the Diagnostics Server in console mode or graphical mode.

To uninstall the Diagnostics Server:

1 Stop the Diagnostics Server. For instructions, see "Starting and Stopping the Diagnostics Server" on page 54.

2 Change the directory to the root directory.

3 Enter the following at the UNIX command prompt:

➤ **In console mode:**

```
<diagnostics_server_install_dir>/Server/_uninst/uninstaller.bin -console
```

➤ **In graphical mode:**

Export your display before running in graphical mode.

export DISPLAY=<hostname>.0.0

```
<diagnostics_server_install_dir>/Server/_uninst/uninstaller.bin
```

3

Licensing HP Diagnostics

HP Diagnostics requires you to upload valid licenses onto the Diagnostics Server in Commander mode.

This chapter includes:

➤ About HP Diagnostics Licensing on page 62

➤ Types of Licenses on page 62

➤ Licensing the Diagnostics Server in Commander Mode on page 63

➤ Licensing the Other Diagnostics Components on page 66

About HP Diagnostics Licensing

Diagnostics is licensed using a file that you upload to the Diagnostics Server in Commander mode. The license uses node-locking based on the MAC address for the host of the Diagnostics Server. You request this license file from your HP Software Customer Support representative.

When the probes and Diagnostics Server in Mediator mode first connect with the Diagnostics Server in Commander mode they are licensed based upon the license installed on the Diagnostics Server in Commander mode.

Types of Licenses

At installation you are given a 30-day trial license. With the 30-day trial (Instant-On license) you can install Diagnostics components and begin to monitor applications and process the performance metrics. Within this 30 day period you must obtain either a perpetual or custom license.

➤ A *perpetual* license is generated without an expiration date.

➤ A *custom* license is generated with a built in expiration date. The expiration date is set so that the license will be valid for a specific length of time. This type of license allows for a trial period before you purchase a perpetual license.

If the 30-day trial expires before you obtain a perpetual or custom license, the Diagnostics Server won't accept any new data.

Note: The full Enterprise Diagnostics product comes with the Instant-On license. The standalone Diagnostics profilers are load-limited until you have provided a valid license file.

Licensing the Diagnostics Server in Commander Mode

You obtain your Diagnostics license from you HP Software Support contact, and then upload it to the Diagnostics Server in Commander Mode.

To license your Diagnostics deployment:

1 Access the License Management page for the Diagnostics Server in Commander mode by selecting **Configure Diagnostics** > **Licensing** in the Diagnostics UI Main window.

The License Management page opens providing the following:

➤ MAC license for the Diagnostics Server (needed to obtain a license)

➤ Information about current licenses

➤ A utility to upload a license received from HP Software Support

➤ Information on logical processor/core totals in your monitored environment

Diagnostics

License Management

System Information

Attribute	Value
MAC Address:	00:16:35:3B:6D:D1, 00:16:35:3B:6D:D2

[Refresh]

License Information

Attribute	Value
License:	Diagnostics Server License
Type:	Custom
Registered To:	Norbert Vicente - 916.748.4090 at Hewlett Packard
Issue Date:	Friday, June 20, 2008
Expiration:	Tuesday, September 30, 2008
Maximum Java Probes:	Unlimited
Maximum .NET Probes:	Unlimited
Maximum SAP, ABAP Probes:	Unlimited
Maximum Oracle Probes:	Unlimited
Maximum MQ Probes:	Unlimited
Maximum SqlServer Probes:	Unlimited
MAC Address:	Any

License Upload

Note:	You may only upload files ending in ".lic". The uploaded file will be renamed to "DiagnosticsServer.lic".
License File:	[ ] [Browse...]

[Upload]

Logical Processor/Core Totals of Currently Connected Probes:

Attribute	Value
Total Number of Hosts:	27
Total Number of Logical Processors/Cores (across all Hosts):	68
Total Number of Hosts with Unknown Number of Logical Processors/Cores:	3
Total Number of Probes:	46
Total Number of Probes with Unknown Number of Logical Processors/Cores:	3
Total Number of VMware Hosts:	4

[Refresh]

Show Logical Processors/Cores by Host

2 Determine the MAC address for the host of the Diagnostics Server in Commander mode by looking at the value in the System Information section of the License Management page. Click **Refresh** to make sure that the Diagnostics Server has discovered the current MAC Address information.

Be sure to let your HP Software Support contact know if the MAC Address value is **Unknown** when you request the license file. Do not look up the MAC Address using another method.

Note: You must use the MAC Address that is displayed in the System Information section of the License Management page when requesting a license. If the Diagnostics Server cannot determine a MAC address to display on the License Management page, it will not be able to find the MAC address when it attempts validate a MAC address based license.

---

3 Request a license from your HP Software Customer Support representative.

Note: Store the license file in a directory that can be accessed from the License Management page for the Diagnostics Server in Commander mode. The name of the file must end with the extension **.lic**.

In the License Management page you can see information on the **Logical Processor/Core Totals of Currently Connected Probes** or **Logical Processors/ Cores by Hosts**. This is useful in determining the number of licenses required without having to manually retrieve the information from each system. This information is only available for Diagnostics 8.00 probes.

4 When you receive the license file for your Diagnostics deployment, upload the file using the License Upload section of the License Management page.

Type the path to the location where the license file was stored or click **Browse** to navigate to the license file location. Click **Upload** to apply the license file to the Diagnostics Server.

The file is renamed by the upload process and stored in the proper location in the install directory of the Diagnostics Server in Commander mode.

Note: Do not attempt to copy the license file directly to the Diagnostics Server installation directory. Always upload the file using the License Upload section of the License Management page.

---

5 View license information. Information is reported in the License Management page on your current licenses. You can see the type of license, the expiration date, if any, and the maximum number of probes covered in the license.

Licensing the Other Diagnostics Components

The Diagnostics Server in Mediator mode and the Diagnostics Probes do not have independent licenses. Their license is based upon the license of the Diagnostics Server in Commander mode. The first time they connect to a licensed Diagnostics Server in Commander mode, the Diagnostics Probes and Diagnostics Server in Mediator mode are automatically licensed.

When you install the Java or .NET probe, the Diagnostics Profiler is automatically installed as well. The Profiler is an independent diagnostics application that can be accessed either directly through the built in Profiler UI or through the HP Diagnostics UI.

The Diagnostics Profiler operates in an unlicensed mode with load restrictions until the probe is able to connect to a Diagnostics Server in Commander mode that has been properly licensed. In unlicensed mode, the Profiler is limited to capturing data from 5 concurrent threads.

4

Installing Diagnostics Collectors

Instructions are provided for installing a Diagnostics Collector on Windows and UNIX machines.

This chapter includes:

About Installing the Diagnostics Collector

The Diagnostics Collector gathers data from external active systems. The Collector can be configured to collect performance data from these types of active systems:

➤ SAP NetWeaver - ABAP system

➤ Oracle 10g Database

➤ IBM WebSphere MQ

➤ SQL Server

During the installation of the Collector, you can choose to monitor any of these active systems. After the installation, you define instances of Oracle 10g, SAP NetWeaver - ABAP, SQL Server systems and IBM WebSphere MQ environments to be monitored.

Installing the Diagnostics Collector on a Windows Machine

The following steps provide detailed instructions for installing the Collector on a Windows machine. These instructions also apply when you are installing the Collector on a UNIX machine using the graphical installer.

Note: The Collector can be installed on any machine. It does not necessarily have to be installed on the host machine of the SAP, Oracle, MQ, or SQL Server application. For Collector host requirements, see "Requirements for the Diagnostics Collector Host" on page 32.

Launching the Installation

The installation can be launched directly from the Diagnostics installation disk, or from the Diagnostics Downloads page in Business Availability Center.

Note: If there is a pre-existing installation of the Collector on the host machine, see Appendix G, "Upgrading Diagnostics and Other HP Software Products."

To launch the installer from the product installation disk:

1 Run the **setup.exe** file in the root directory of the installation disk. The Diagnostics setup program begins and displays the installation menu page.

Note: To launch the installer from a different location, copy the executable file **CollectorSetupWin_<version>.exe** from the **<HP Diagnostics Installation Disk>\Diagnostics_Installers** directory to the new location, and then run it.

---

2 From the installation menu page, select **Diagnostics Collector** to launch the installer.

3 Continue with "Running the Installation" on page 70.

To launch the Installer from the Business Availability Center Diagnostics downloads page:

Note: The Collector installer is available in Business Availability Center if put into the required directory for Business Availability Center to access.

You can enable the Collector installer download from Business Availability Center during the installation of the Diagnostic Server, by providing the path to the Diagnostics Agent and Collector installers, or the **\Diagnostics_Installers\CollectorSetupWin_<version>.exe** file can be copied manually from the installation disk to the following folder of the Diagnostics Server installation directory: **<diag_server_install_dir>/html/ opal/downloads.** See Step 3 on page 50 of Chapter 2, "Installing the Diagnostics Server."

---

 1 In **Business Availability Center**, select **Admin > Diagnostics** from the top menu in and click the **Downloads** tab.

 2 On the Downloads page, click the appropriate link to launch the Collector installation for Windows.

Continue with "Running the Installation" that follows.

Running the Installation

After you have launched the installation, the software license agreement opens.

To install the Collector:

 1 **Accept the software license agreement.**

Read the agreement and select **I accept the terms of the license agreement**.

Click **Next** to continue.

2 Specify the location where you want to install the Collector.

In the **Installation Directory Name** box, type the name of the directory where you want to install the Collector. Alternatively, accept the default directory, **C:\MercuryDiagnostics\Collector**, or click **Browse** to navigate to another directory.

If the directory contains an existing installation of the Collector that you want to upgrade, cancel this installation and follow the upgrade procedure for Collectors as described in Appendix G, "Upgrading Diagnostics and Other HP Software Products."

Click **Next** to continue.

3 Assign a unique name to the Collector.
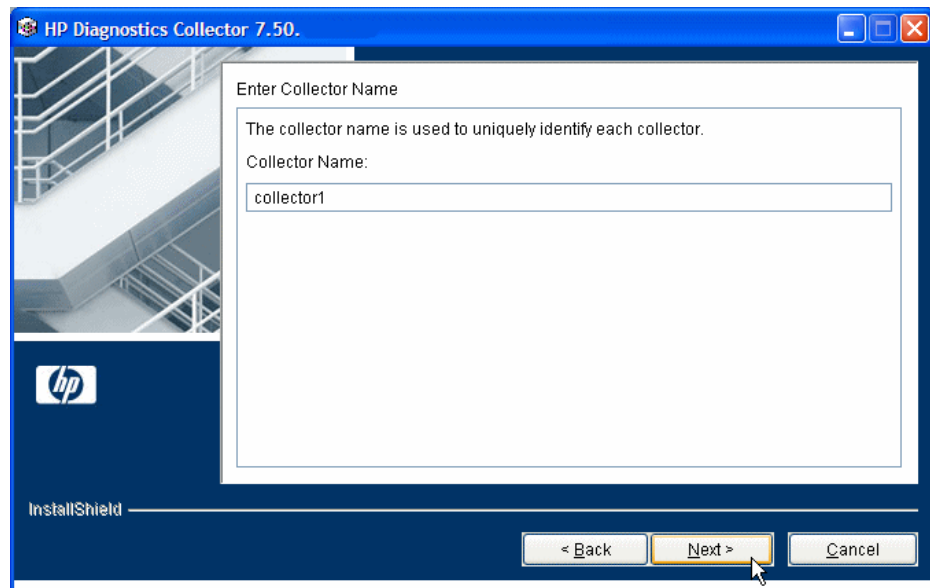
Assign a name to the Collector that will uniquely identify this specific Collector in the System Health Monitor. You use the System Health Monitor to verify the Collector installation and configuration. For more information, see "Verifying the Diagnostics Collector Installation" on page 96.

The following characters can be used in the name: **-** , **_** , and all alphanumeric characters.

Click **Next** to continue.

4 Select the environment to monitor.
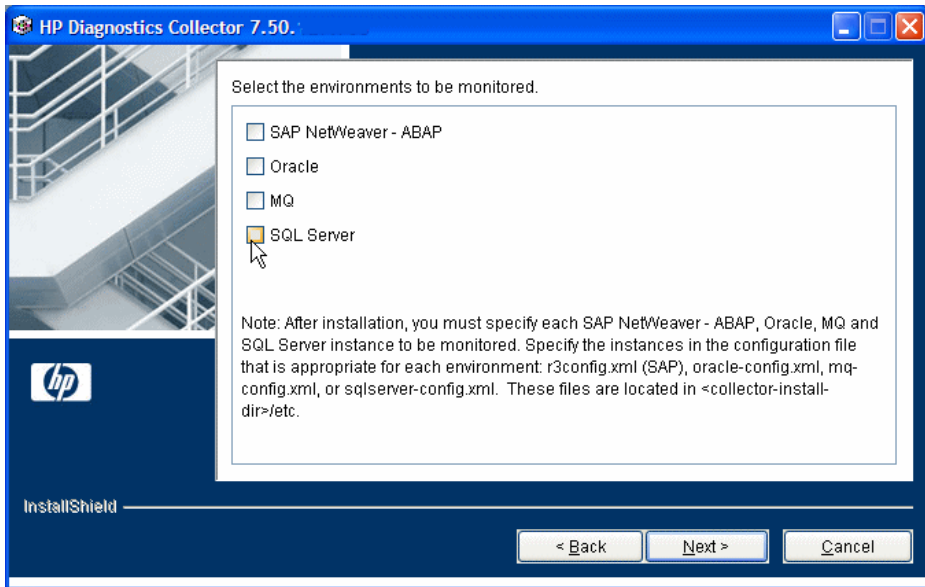
Select the options that apply to this Collector. You can select either one or both options.

➤ To collect data in an SAP NetWeaver - ABAP environment, select **SAP NetWeaver - ABAP**.

➤ To collect data on an Oracle 10g database server, select **Oracle**.

➤ To collect data in an MQ series environment, select **MQ**.

➤ To collect data on an SQL Server database, select **SQL Server.**

Important: After installation, you specify each of the SAP NetWeaver - ABAP, Oracle, MQ and SQL Server instances to be monitored. These instances are manually defined in the XML files provided with the installation. For more information, see "Configuring the Active System Property Files" on page 85.

Click **Next** to continue.

5 Provide information about the Diagnostics Server in Mediator mode.

Provide the details that enables communication with the Diagnostics Server in Mediator mode.
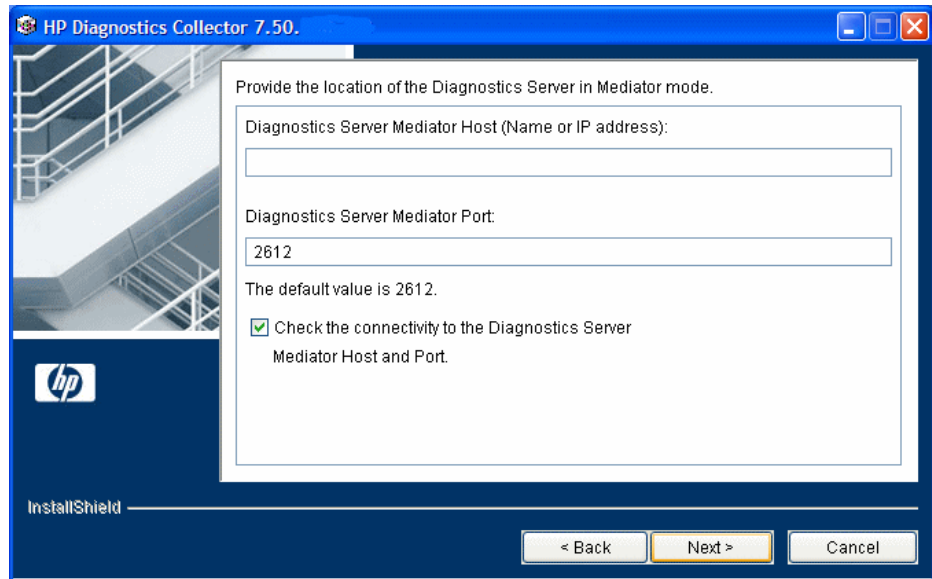
If there is only one Diagnostics Server in the Diagnostics deployment where the Collector will run, enter the host name of the Diagnostics Server and its event port information.

If there is more than one Diagnostics Server in the deployment, enter the information for the Diagnostics Server in Mediator mode that is to receive the events from the Collector.

a In the **Diagnostics Server Mediator Host** box, type the host name or IP address of the host for the Diagnostics Server in Mediator mode.

Note: You should specify the fully qualified host name. In a mixed OS environment, where UNIX is one of the systems, this is essential for proper network routing.

---

  b In the **Diagnostics Server Mediator Port** box, type the port number where the Diagnostics Server is listening for Collector communication. The default port number is **2612**. If you have changed the port since the Diagnostics Server was installed, specify that port number instead of the default.

  c If you want to make sure that the Diagnostics Server is running and accessible from the installation host, select **Check the connectivity to the Diagnostics Server Mediator Host and Port**.

  Click **Next** to continue.

  Note: If you selected **Check the connectivity to the Diagnostics Server Mediator Host and Port** and connectivity problems were encountered, the installer provides you with the results of the connectivity check. If you do not want to address these problems at this stage, clear the **Check the connectivity to the Diagnostics Server Mediator Host and Port** check box, proceed with the installation, and address the problem later.

---

6 **If you selected SAP NetWeaver - ABAP in step 4, provide the location of the SAP Java Connector.**
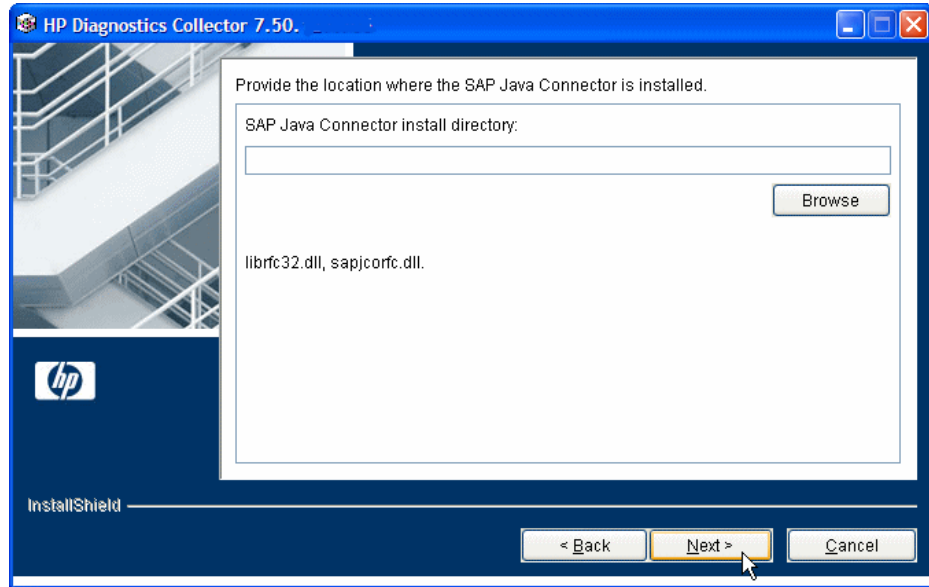
In the **SAP Java Connector install directory** box, enter the name of the directory where the SAP Java Connector is installed.

This directory must contain the following files:

➤ sapjco.jar

➤ librfc.dll

➤ sapjcorfc.dll

Note:

➤ If you do not know the SAP Java Connector directory name, contact your SAP representative.

➤ If any of the files is missing from this directory, contact your SAP representative.

---

7 Review the pre-installation summary.

The installation settings that you selected are displayed in a read-only window. Review the information to make sure that you are satisfied.

To select different installation settings, click **Back**.

To begin installation, click **Next**.

8 Close the installation wizard.

When the installation has completed, a message is displayed, confirming that the Collector was successfully installed. Click **Finish** to exit the installation.

9 Configure the XML files for your active systems.

In step 4 you selected the active systems that will be monitored. For each of these active systems, you need to configure properties that enable the Collector host and the active system host to communicate.

For instructions on configuring the relative active system properties, see "Configuring the Active System Property Files" on page 85.

10 Verify that Collector was installed properly, and is running.

Using the System Health Monitor, you can verify that the Collector is running as it should be. For details see, "Verifying the Diagnostics Collector Installation" on page 96.

In the Diagnostics UI, each collector instance is represented as a probe of the system type: Oracle Probe, SAP Probe, MQ Probe or SQL Server Probe.

Installing the Diagnostics Collector on a UNIX Machine

The following instructions provide you with the steps necessary to install the Collector in UNIX environments, using either a graphical installation or a console mode installation.

The installation screens that are displayed in a graphical installation are the same as those documented for the Windows installation in "Installing the Diagnostics Collector on a Windows Machine" on page 68.

Launching the Installation

The installer can be launched directly from the Diagnostics installation disk, or from the Diagnostics Downloads page in Business Availability Center.

To launch the installation from the installation disk:

1 From the **<HP Diagnostics Installation Disk>/Diagnostics_Installers** directory, copy the installer **CollectorSetup<platform>_<version>.bin** to the machine where the Collector is to be installed. For example **CollectorSetupLinux_8_00.bin**.

2 Continue with "Running the Installation" on page 78.

To launch the Installer from the Business Availability Center Diagnostics downloads page:

1 From the top menu in Business Availability Center, select **Admin > Diagnostics**, and click the **Downloads** tab.

2 On the Downloads page, click the link to the installer that is appropriate for your environment.

3 Save the installer on the machine where the Collector is to be installed.

Note: The Collector installer is available in Business Availability Center only if you provided the path to the Probe installers directory when you installed the Diagnostics Server in Commander mode. See "Installing the Diagnostics Server in Commander Mode" on page 49.

Continue with "Running the Installation" on page 78.

Running the Installation

After you have copied the installer to the machine where the Collector is to be installed, you are ready to run the installation.

Notes:

➤ If there is a pre-existing installation of the **Collector** on the host machine, see Appendix G, "Upgrading Diagnostics and Other HP Software Products."

➤ The following instructions assume an understanding of UNIX console screens and commands. For more information about UNIX screens and commands, see Appendix I, "Using UNIX Commands."

To install the Collector:

1 Run the installer.

Where necessary, change the mode of the installer file to make it executable.

➤ To run the installer in the graphical mode, enter the <installer> executable at the UNIX command prompt, where <installer> is, for example **CollectorSetupSolaris_<release number>.bin**, **CollectorSetupHP11x_<release number>.bin** or **CollectorSetupLinux_<release number>.bin**:

The installer displays the same screens that are displayed for the Windows installer. Continue with "Installing the Diagnostics Collector on a Windows Machine" on page 68.

➤ To run the installer in console mode, enter <installer> -console at the UNIX command prompt, where <installer> is, for example **CollectorSetupSolaris_<release number>.bin**, **CollectorSetupHP11x_<release number>.bin** or **CollectorSetupLinux_<release number>.bin**:

The installer continues in console mode, and displays the software license agreement.

2 Accept the software license agreement.

Read the license agreement and accept the terms.

Press **Enter** to continue through the license agreement.

When prompted, enter **1** to accept the agreement.

Select **Next** to continue with the installation.

3 Specify the location where you want to install the Collector.

At the **Installation Directory Name** prompt, accept the default installation location shown in brackets, or type the path to a different installation location.

Select **Next** to continue with the installation.

4 Assign a unique name to the Collector.
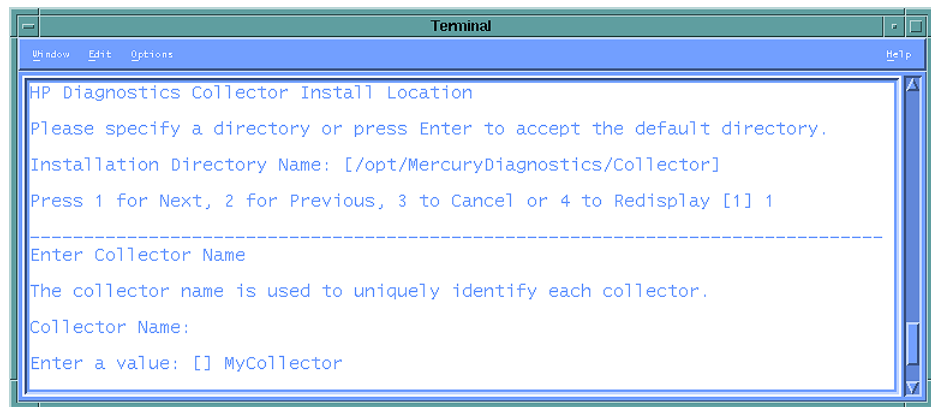
```
┌─────────────────────────────────── Terminal ───────────────────────────────────┐
│ Window   Edit   Options                                                    Help │
├─────────────────────────────────────────────────────────────────────────────────┤
│HP Diagnostics Collector Install Location                                        │
│                                                                                  │
│Please specify a directory or press Enter to accept the default directory.       │
│                                                                                  │
│Installation Directory Name: [/opt/MercuryDiagnostics/Collector]                 │
│                                                                                  │
│Press 1 for Next, 2 for Previous, 3 to Cancel or 4 to Redisplay [1] 1            │
│------------------------------------------------------------------------------   │
│Enter Collector Name                                                             │
│                                                                                  │
│The collector name is used to uniquely identify each collector.                  │
│                                                                                  │
│Collector Name:                                                                  │
│                                                                                  │
│Enter a value: [] MyCollector                                                    │
└─────────────────────────────────────────────────────────────────────────────────┘
```

Assign a name to the Collector that will uniquely identify this specific Collector in the System Health Monitor. You use the System Health Monitor to verify the Collector installation and configuration. For more information, see "Verifying the Diagnostics Collector Installation" on page 96.

The following characters can be used in the name: **-** , **_** , and all alphanumeric characters.

Select **Next** to continue.

5 Specify the active systems that the Collector will monitor.

```
┌─                              Terminal                              □ □
 Window  Edit  Options                                               Help

 Collector Name:

 Enter a value: [] MyCollector


 Press 1 for Next, 2 for Previous, 3 to Cancel or 4 to Redisplay [1] 1

 ----------------------------------------------------------------------
 Select the environments to be monitored.

 [ ] 1 - SAP NetWeaver - ABAP
 [ ] 2 - Oracle
 [ ] 3 - MQ
 [ ] 4 - SQL Server

 To select an item enter its number, or 0 when you are finished: [0] ▌
```
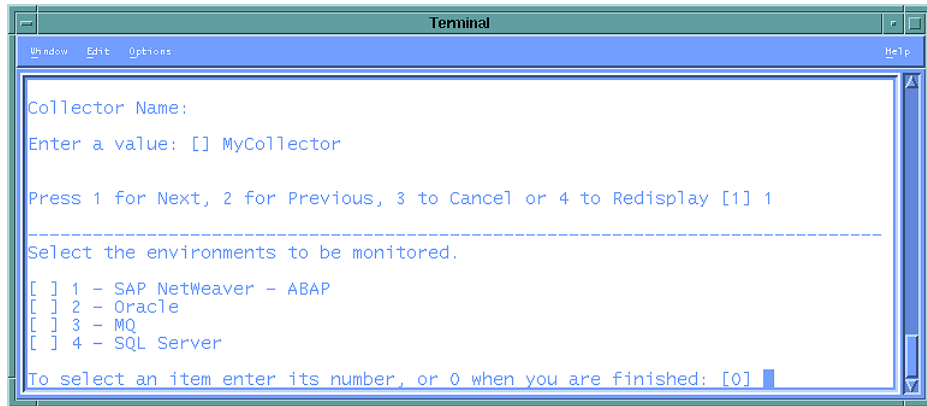
Select the active systems that apply to this Collector.

➤ To collect data in an SAP NetWeaver - ABAP environment, select **SAP NetWeaver - ABAP.**

➤ To collect data on an Oracle 10g database server, select **Oracle**.

➤ To collect data in an MQ series environment, select **MQ**.

➤ To collect data on an SQL Server database, select **SQL Server.**

Important: After installation, you specify each of the SAP NetWeaver - ABAP, Oracle, MQ and SQL Server instances to be monitored. These instances are manually defined in the XML files provided with the installation. For more information, see "Configuring the Active System Property Files" on page 85


Select **Next** to continue with the installation.

6 **Provide the details for the Diagnostics Server in Mediator mode.**

Provide the information that enables communication with the Diagnostics Server in Mediator mode.
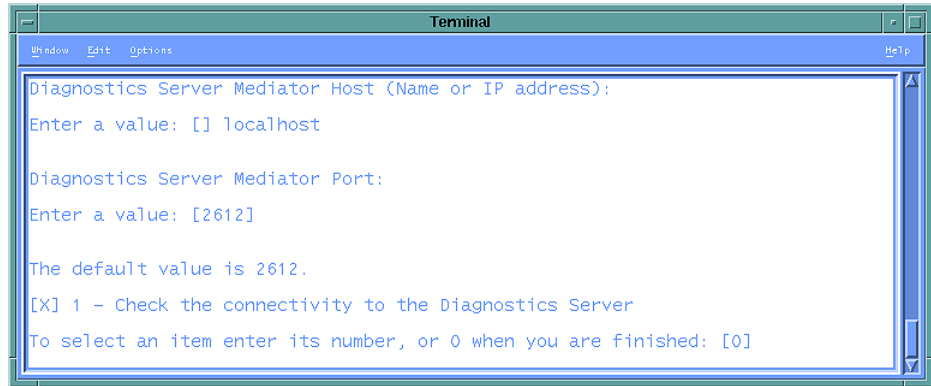
```
Terminal
Window   Edit   Options                                                    Help

Diagnostics Server Mediator Host (Name or IP address):
Enter a value: [] localhost

Diagnostics Server Mediator Port:
Enter a value: [2612]

The default value is 2612.
[X] 1 - Check the connectivity to the Diagnostics Server
To select an item enter its number, or 0 when you are finished: [0]
```

If there is only one Diagnostics Server in the Diagnostics deployment where the Collector will run, enter the host name of the Diagnostics Server and its event port information.

If there is more than one Diagnostics Server in the deployment, enter the information for the Diagnostics Server in Mediator mode that is to receive the events from the Collector.

a Enter the host name or IP address of the host of the Diagnostics Server in Mediator mode.

Note: You should specify the fully qualified host name. In a mixed OS environment, where UNIX is one of the systems, this is essential for proper network routing.

---

b Enter the number of the port where the Diagnostics Server is listening for Collector communication. The default port is **2612**. If you have changed the port since the Diagnostics Server was installed, specify that port number instead of the default.

c If you want to make sure that the Diagnostics Server is running and accessible from the installation host, select **Check the connectivity to the Diagnostics Server**.

Select **Next** to continue with the installation.

If you instructed the installer to perform the test for connectivity, it tests the connectivity at this point. If there are negative results, it reports these before proceeding with the next installation step.

If you do not want to address these problems at this stage, clear the **Check the connectivity to the Diagnostics Server** option so that the installation can proceed.

7 Provide the location of the SAP Java Connector.

At the prompt, type the name of the directory where the SAP Java Connector is installed.

This directory must contain the following files:

➤ sapjco.jar

➤ librfc.dll

➤ sapjcorfc.dll

Notes:

➤ If you do not know the SAP Java Connector directory name, contact your SAP representative.

➤ If any of the files is missing from this directory, contact your SAP representative.

Select **Next** to continue with the installation.

 8 **Review the pre-installation summary.**

The installation settings that you selected are displayed. Review the information to make sure that you are satisfied.
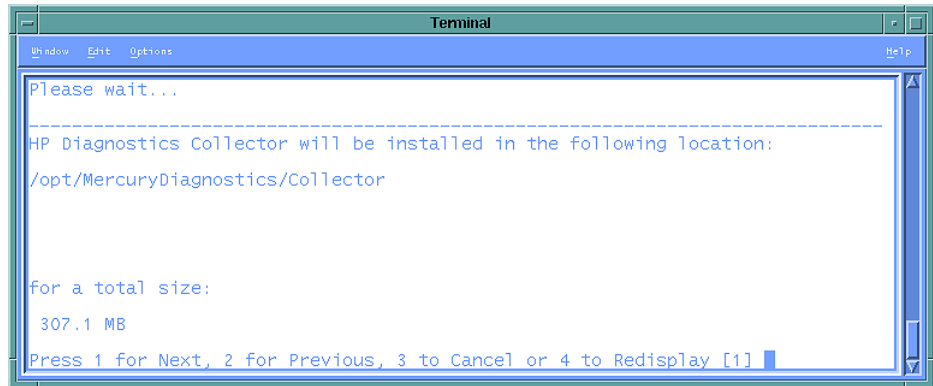
To change your settings, select **Previous** to return to the previous prompts.

To start the installation of the Collector, select **Next**.

 9 **Close the installation wizard.**

When the installation has completed, review the post-installation summary information to make sure that the installation completed successfully.

Select **Finish** to exit the installation.

 10 **Configure the XML files for your active systems.**

In step 5 you selected the active systems that will be monitored. For each of these active systems, you need to configure properties that enable the Collector host and the active system host to communicate.

For instructions on configuring the active system properties, see "Configuring the Active System Property Files" on page 85.

 11 **Verify the Collector installation.**

Using the System Health Monitor, you can verify that the Collector is running as it should be. For details, see "Verifying the Diagnostics Collector Installation" on page 96.

Silent Installation of the Diagnostics Collector

A *silent installation* is an installation that is performed automatically, without the need for user interaction. In place of user input, the silent installation accepts input from a response file for each step of the installation.

For example, a system administrator who needs to deploy a component on multiple machines can create a response file that contains all the prerequisite configuration information, and then perform a silent installation on multiple machines. This eliminates the need to provide any manual input during the installation procedure.

Before you perform silent installations on multiple machines, you need to generate a response file that will provide input during the installation procedure. This response file can be used in all silent installations that require the same input during installation.

The response file has the suffix **.rsp**. You can edit the response file with a standard text editor if desired.

To generate a response file:

➤ Perform a regular installation with the following command-line option:

```
<installer> -options-record <responseFileName>
```

This creates a response file that includes all the information submitted during the installation.

To perform a silent installation:

➤ Perform a silent installation using the relevant response file.

You perform the silent installation with the **-silent** command-line option as follows:

```
<installer> -silent -options <responseFileName>
```

Configuring the Active System Property Files

When you install the Collector, you are asked to indicate the active systems that the Collector will monitor. After installation, you define instances of the active systems to be monitored. These instances are manually defined in the XML files provided with the installation. An instance definition in the XML file behaves like a probe on the instance of the active system.

This section includes:

Configuring SAP NetWeaver - ABAP Probes

A SAP NetWeaver - ABAP system deployment can include one or more SAP NetWeaver - ABAP application instances. These instances together form an SAP NetWeaver - ABAP system.

Depending on user permissions, access to the system or application instances on the system may be direct or may require connection through the SAP Message Server. For each SAP NetWeaver - ABAP probe that you define, you need to know what connection option is used.

You define and configure SAP NetWeaver - ABAP probes in the **<collector_install_dir>\etc\r3config.xml file**. The layout, elements, and attributes of the xml file are described in the **<collector_install_dir>\etc\r3config.xsd**.

To configure an SAP NetWeaver - ABAP Probe:

1 Open **<collector_install_dir>\etc\r3config.xml**.

2 If you are defining an SAP NetWeaver - ABAP probe where access to the SAP NetWeaver - ABAP instance is through the SAP Message Server, locate the section of code preceded by the following comment:

```
<!--
Template to be used with the message server connection option.
-->
```

If you are defining an SAP NetWeaver - ABAP probe where access to the SAP NetWeaver - ABAP instance is direct, locate the section of code preceded by the following comment:

```
<!--
Template to be used with the direct connection option.
-->
```

3 Make a copy of the comment together with the template code below the comment, and paste it at the end of the file.

4 Comment out the original template code by typing **<!--** in an empty line above the template code, and **-->** in an empty line thereafter.

5 In the copied code at the end of the file, alter the value of each property, as described in the following table and then save the file:

Property	Description	Value
r3system name	A logical name for the probe group under which this SAP NetWeaver - ABAP Probe appears in the Diagnostics UI.	User-defined.
systemId	The ID of the SAP NetWeaver-ABAP system. Comprises 3 characters only.	Format: [XXX] Obtainable from the SAP system administrator.
client	The client name for the SAP NetWeaver - ABAP system.	Obtainable from the SAP system administrator.

Property	Description	Value
user	The name of the user connecting to the SAP NetWeaver - ABAP system. This user needs to have at least the S_RFC authorization object. However, for systems R/3 4.7 and earlier this is not sufficient. The workaround for those systems is to install the Collector on a machine that is time-synched with the R/3 host and then disable time-synching in the collector by setting property timesynch.interval.secs = 0 (in <collector-install-dir\etc\r3.properties).	Obtainable from the SAP system administrator.
password	The password (plaintext) of the user connecting to the SAP NetWeaver - ABAP system.	Obtainable from the SAP system administrator.
encrypted-password	The password (encrypted) of the user connecting to the SAP NetWeaver - ABAP system.	Use the EncryptPassword.jsp utility (see "Password Obfuscation" on page 94) to encrypt the password.
messageServerHost (Message Server connection only)	The name of the SAP Message Server host machine.	Obtainable from the SAP system administrator.

Property	Description	Value
r3Name (Message Server connection only)	Comprises 3 characters only.	Format: [XXX] Obtainable from the SAP system administrator.
group (Message Server connection only)	The group of the SAP application servers.	Obtainable from the SAP system administrator.

Configuring Oracle Probes

You define and configure Oracle probes in **<collector_install_dir>\etc\oracle-config.xml**. The layout, elements and attributes of the xml file are described in the **<collector_install_dir>\etc\oracle-config.xsd.**

To configure an Oracle Probe:

1 Open **<collector_install_dir>\etc\oracle-config.xml**.

2 Copy the template code, and paste it at the end of the file.

3 Comment out the template code by typing **<!--** in an empty line above the template code, and **-->** in an empty line thereafter.

4 In the copied code, alter the value of each property, as described in the following table and then save the file:

Properties	Description	Value
hostName	The name of the Oracle database server host machine.	Obtainable from the Oracle administrator.
portNumber	Port where the Oracle database server listens for requests.	Default value: **1521**
instanceName	The name given to the Oracle instance during installation of the Oracle database server.	Default value: **Orcl** Obtainable from the Oracle administrator.
userId	The ID of the user connecting to the Oracle database server. **Note:** The user needs at least **CREATE SESSION** and **SELECT ANY DICTIONARY** to collect performance metrics.	Obtainable from the Oracle administrator.
password	The password (plaintext) of the user connecting to the Oracle database server.	Obtainable from the Oracle administrator.
encrypted-password	The password (encrypted) of the user connecting to the Oracle database server.	Use the EncryptPassword.jsp utility (see "Password Obfuscation" on page 94) to encrypt the password.
probeName	The logical name to represent this Oracle instance in the Diagnostics UI. This name must be unique.	User-defined. If this value is not defined, the same value given for **instanceName** will be used here.
probeGroupName	The logical name of the Probe group under which this Probe appears in the Diagnostics UI. It can be an existing Probe group, or you can define a new one.	User-defined. For example: Existing: **Default** New: **Oracle**

Configuring MQ Probes

You define and configure MQ probes in **<collector_install_dir>\etc\
mq-config.xml**. The layout, elements and attributes of the xml file are
described in the **<collector_install_dir>\etc\mq-config.xsd.**

The MQ probe (collector) requires the following permissions:

```
setmqaut -m QM_ovrbat5 -n ** -t queue -g testMQGroup +dsp +get
setmqaut -m QM_ovrbat5 -n SYSTEM.ADMIN.COMMAND.QUEUE -t queue -g
testMQGroup +dsp +get +put
setmqaut -m QM_ovrbat5 -n ** -t channel -g testMQGroup +dsp
setmqaut -m QM_ovrbat5 -t qmgr -g testMQGroup +connect +dsp +inq
```

You can limit the types of queues from which the MQ probe collects metrics
to isolate the most interesting metrics for your application. By default, the
MQ probe collects metrics from all queue types. You specify the queue types
to ignore by setting properties in the
<collector_install_dir>\etc\mq.properties file.

To define and configure an MQ Probe:

1 Open **<collector_install_dir>\etc\mq-config.xml**.

2 Copy the template code, and paste it at the end of the file.

3 Comment out the template code by typing **<!--** in an empty line above the
template code, and **-->** in an empty line thereafter.

4 In the copied code, alter the value of each property, as described in the following table and then save the file:

Properties	Description	Value
hostName	The hostName.	Obtainable from the MQ administrator.
portNumber	The number of the port.	Default value: **1414**
queueManagerName	The MQ Manager to connect to.	Obtainable from the MQ administrator.
channelName	The channel through which to connect to the Queue Manager.	Obtainable from the MQ administrator.
securityExit	A Security Exit is an IBM term for a pluggable security provider (a piece of code that provides a secure interface to MQ. If you are using one as a gateway to MQ, specify the complete class name as a parameter and make sure your security Exit class is available on the classpath.	
probeName	The name to be used to represent this instance as a probe in the HP Diagnostics UI. This name must be unique.	User-defined. If this value is not defined, it defaults to the Queue Manager name.
probeGroupName	The logical name of the Probe group under which this Probe appears in the Diagnostics UI. This can be an existing Probe group, or you can define a new one.	User-defined. For example: Existing: **Default** New: **MQ**

To limit the queues for which metrics are collected:

1 Open **<collector_install_dir>\etc\mq.properties** file.

2 Locate the property name that corresponds to the MQ Queue definition type from which you do not want the collector to collect metrics. The following table lists the property names and their corresponding MQ Queue definition types:

Properties	MQ Queue Definition Types
collect.predefined.queues	MQQDT_PREDEFINED
collect.permanent.dynamic.queue	MQQDT_PERMANENT_DYNAMIC
collect.temporary.dynamic.queues	MQQDT_TEMPORARY_DYNAMIC
collect.shared.dynamic.queues	MQQDT_SHARED_DYNAMIC

3 Specify 'false' in place of 'true' for any type for which you do not want the collector to gather metrics and then save the file.

Note: These properties are supported for MQ 6.x and later versions only.

Configuring SQL Server Probes

You define and configure the SQL Server probes in **<collector_install_dir>\etc\sqlserver-config.xml**. The layout, elements and attributes of the xml file are described in the **<collector_install_dir>\etc\sqlserver-config.xsd.**

To configure an SQL Server Probe:

1 Open **<collector_install_dir>\etc\sqlserver-config.xml**.

2 Copy the template code, and paste it at the end of the file.

3 Comment out the template code by typing **<!--** in an empty line above the template code, and **-->** in an empty line thereafter.

4 In the copied code, alter the value of each property, as described in the following table and then save the file:
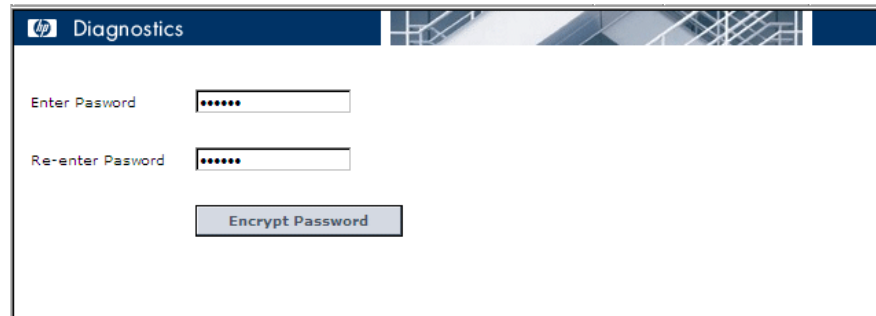
Properties	Description	Value
hostName	The name of the SQL Server database host machine.	Obtainable from the SQL Server administrator.
portNumber	The number of the port where the SQL Server database listens for requests.	Default value: **1433**
instanceName	The name given to the SQL Server instance during installation of the SQL Server database. When you specify an instance name, Diagnostics **automatically discovers all SQL Server databases in the instance**. If you want to exclude some of these databases from collection (for example system databases), you can specify a comma-separated list in the **exclude.db.list** property in the **<collector_install_dir>\etc\sqlserver.properties** file.	Default value: **Default** Obtainable from the SQL Server administrator.
userId	The ID of the user connecting to the SQL Server database. **Note:** The user needs at least **VIEW SERVER STATE** to collect performance metrics.	Obtainable from the SQL Server administrator.
password	The password (plaintext) of the user connecting to the SQL Server database.	Obtainable from the SQL Server administrator.

Properties	Description	Value
encrypted-password	The password (encrypted) of the user connecting to the SQL Server database.	Use the EncryptPassword.jsp utility (see "Password Obfuscation" on page 94) to encrypt the password.
probeName	The logical name to represent this Oracle instance in the Diagnostics UI. This name must be unique. When you have *n* databases in your instance you actually will have *n+1* probes; an extra probe for the totals of the instance which includes metrics such as wait events. The extra probe is shown in the UI as probeName, the probes for each database are shown as probeName_databaseName.	User-defined. If this value is not defined, the same value given for **instanceName** will be used here.
probeGroupName	The logical name of the Probe group under which this Probe appears in the Diagnostics UI. This can be an existing Probe group, or you can define a new one.	User-defined. For example: Existing: **Default** New: **SQL Server**

Password Obfuscation

Create an obfuscated password using the web application included with Diagnostics: **http://<host name>:2006/security/EncryptPassword.jsp**. The <host name> should be replaced with the name of the computer on which the Diagnostics server is installed.

The obfuscated password you generate can be used in the **r3config.xml** file used to configure the SAP NetWeaver - ABAP collector, in the **oracle-config.xml** file used to configure the Oracle collector or in the **sqlserver-config.xml** file used to configure the SQL Server collector.

Enter the plaintext password, re-enter the password to confirm and select the **Encrypt Password** button. The obfuscated password is then displayed. Copy the entire obfuscated password from this page, including the OBF: at the beginning, and paste that into the appropriate property file (the **r3config.xml** file, the **oracle-config.xml** file, the **sqlserver-config.xml**).

Note: You can continue to use the plaintext password property as well.

A **security.encrypted-password** property can also be used for the *mercury* user password in the following property files: **collector.properties**, **dispatcher.properties**, **server.properties**. The *mercury* user is used for authentication between the various diagnostics components. The following is a copy of the section of these properties files that is affected:

```
##########################################################################
# Remote Server Authentication Properties
##########################################################################

#
# This user name and password is used for communication between Diagnostics
# components (probes, and servers). You may want to change this password
# every so often to keep your system secure inside your enterprise. If you
# do change this password, you must first use
# http://<host name>:2006/security/EncryptPassword.jsp to encrypt the password.
# Plaintext passwords can be used by replacing the security.encrypted-password
# with security.password. You must also change the encrypted password in the
# <install-dir>/etc/.htaccess file, as well as all the Diagnostics probe, and
# servers, that communicate with each other in your enterprise.
#
security.username=mercury
security.encrypted-password=OBF:1c431jg81hv41k1d1l161wu81z0d1pyl1wmt1n6h1y
m71n511wnd1pw11z0h1wu61kxw1jyl1hse1jd21c2z
```

Verifying the Diagnostics Collector Installation

The Collector starts running automatically when the installation is complete. You can verify the Collector installation using the System Health Monitor. For detailed instructions about how to use the System Health Monitor, see Appendix D, "Using the System Health Monitor."

If you have been following the recommended installation sequence, after installing the Collector you can verify the following:

➤ The Diagnostics Server in Commander mode was successfully installed.

➤ Where relevant, additional Diagnostics Servers in Mediator mode were successfully installed and are communicating with the Diagnostics Server in Commander mode.

➤ The Collector was successfully installed and has established connectivity with the relevant Diagnostics Server.

Depending on your deployment, the Collector is shown on the System Health Monitor as a child of either the Diagnostics Server in Commander mode or a Diagnostics Server in Mediator mode. The Collector is represented by an icon labelled **<name of Collector>.** The name of the collector is assigned during the installation procedure.
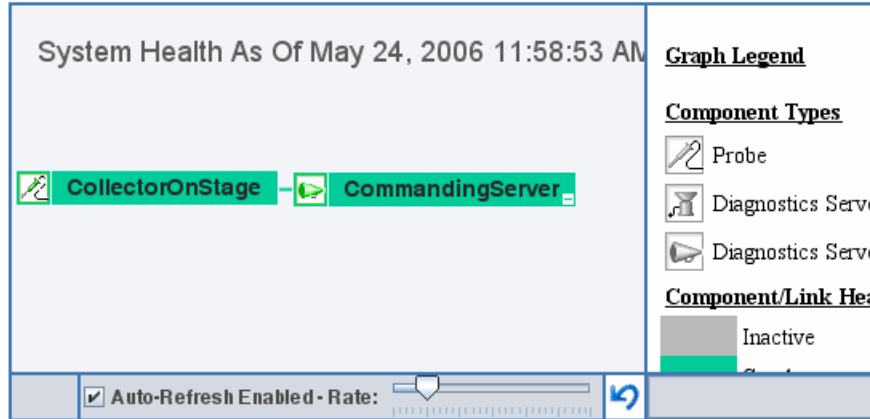
When the Collector has been stopped, the Collector node in the System Health monitor is grey. When the Collector has been started the node is green.

Starting and Stopping the Diagnostics Collector

Instructions for a Windows Machine

To start the Collector on a Windows machine:

➤ Select **Start** > **Programs** > **HP Diagnostics Collector** >
Start HP Diagnostics Collector. Or enter **net start "HP Diagnostics Collector"**
at the command line.

To stop the Collector on a Windows machine:

➤ Select **Start** > **Programs** > **HP Diagnostics Collector** >
Stop HP Diagnostics Collector. Or enter **net stop "HP Diagnostics Collector"**
at the command line.

Instructions for a UNIX Machine (using the Nanny)

The *nanny* is a process that runs as a daemon to ensure that the Collector is
always running. The following procedures start and stop the Collector using
the nanny.

To start the Collector on a UNIX machine:

1 Ensure that the **M_LROOT** environment variable is defined as the root
directory of the Collector.

For example, in *ksh*, you could enter the following:

```
export M_LROOT=<collector_install_dir>/nanny/solaris
```

If the **M_LROOT** environment variable is not defined as the root directory,
you will encounter the following error:

```
Warning : MDRV: cannot find Irun root directory . Please check your M_LROOT
Unable to format message id [-10791]
m_agent_daemon ( is down )
```

2 Change directories to **$M_LROOT/bin**.

3 Run **m_daemon_setup** with the **-install** option, as in the following example:

```
cd $M_LROOT/bin
./m_daemon_setup -install
```

To stop the Collector on a UNIX machine:

1 Change directories to **$M_LROOT/bin** as set in the start procedure above.

2 Run **m_daemon_setup** with the **-remove** option, as in the following example:

```
cd $M_LROOT/bin
./m_daemon_setup -remove
```

Instructions for a UNIX Machine (without the Nanny)

The following procedures start and stop the Collector without using the nanny.

To start the Collector on a UNIX machine:

➤ Run **<collector_install_dir>/bin/collector.sh**.

To stop the Collector on a UNIX machine:

➤ Terminate the process using a utility such as **kill**.

Determining the Version of the Diagnostics Collector

When you request support, it is useful to know the version of the Diagnostics Collector.

The version number of the Collector can be found:

➤ in the **<collector_install_dir>\version.txt** file.

➤ among the details of the Collector on the System Health Monitor.

Uninstalling the Diagnostics Collector

To uninstall the Collector:

➤ On a Windows machine, choose **Start** > **All Programs** > **HP Diagnostics Collector** > **Uninstall Diagnostics Collector**.

Alternatively, you can run **uninstaller.exe** which is located in the **<collector_install_dir>_uninst** directory.

➤ On a Linux or Solaris UNIX machine, run **uninstall*** which is located in the **<collector_install_dir>/_uninst** directory.

➤ On other UNIX machines, choose a 1.5 or later JVM and run **java -jar <collector_install_dir>/_uninst/uninstall.jar** to uninstall the collector.

Part III

Installation and Setup of the Java and .NET Agent (Probes)

5

Installing Java Agents

Instructions are provided for installing a Java Agent on Windows machines, UNIX machines, and z/OS mainframe machines. Instructions have also been provided for using a generic installer to install the Java Agent on other platforms.

This chapter includes:

About the Java Agent Installer

The functionality of the Java Agent is provided by the HP Diagnostics/ TransactionVision Java Agent (Java Agent). The Java Agent combines the capabilities of the Diagnostics Java Agent and the TransactionVision Java Sensors (JMS, Servlet, JDBC and EJB) into a single component.

The Java Agent is configured to serve as a Java Agent in a Diagnostics environment or as a Java Sensor in a TransactionVision environment and for combined environments, the agent simultaneously serves as both the probe and the sensor.

Before you can use a Java Agent to monitor an application in HP Diagnostics, you need to 1) Install the Java Agent, 2) Instrument the JRE and 3) configure the application server startup scripts to work with the Java Agent. For information about instrumenting the JRE see Chapter 6, "Running the JRE Instrumenter". For information about modifying the application startup script, see Chapter 7, "Configuring Application Server Startup Scripts to Work with the Java Agent".

The agent is installed on the machine hosting the application that you want to monitor.

Note: The location in which the agent is installed becomes the Diagnostics <probe_install_dir>. By default, the location is C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent on Windows and /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent on UNIX.

Note: For information about the recommended system configurations for hosting the Java Agent, see "Requirements for the Diagnostics Probe for Java Host" on page 31.

For more information about installing the Java Agent, see one of the following sections, depending on your operating system and method of installation:

Installing and Configuring the Java Agent on Windows

The following steps provide detailed instructions for installing the Java Agent on a Windows machine. These instructions also apply when you are installing the Java Agent on a UNIX machine using the graphical installer.

Note: If there is a pre-existing installation of the Java Agent on the host machine, see "Upgrading Diagnostics and Other HP Software Products" on page 691.

This section includes:

➤ "Launching the Java Agent Installer on Windows" on page 105

➤ "Running the Installation" on page 107

➤ "Configuring the Java Agent as a Profiler Only" on page 108

➤ "Configuring the Java Agent to Work with a Diagnostics Server" on page 111

Launching the Java Agent Installer on Windows

The Java Agent installer can be launched from the Diagnostics installation disk, another location, or from the Business Availability Center Diagnostics Downloads page.

To launch the installer from the product installation disk:

1 Run the **setup.exe** file in the root directory of the installation disk. The HP Diagnostics main installation menu is displayed.

2 From the installation menu page, select **Diagnostics Agent for Java** to launch the installer.

3 Continue with "Running the Installation" on page 107.

To launch the installer from another location:

1 From the **<HP Diagnostics Installation Disk>\Diagnostics_Installers** directory, copy the executable file **JavaAgentSetup_win_<version>.exe** to the new location, and then run it.

2 Continue with "Running the Installation" on page 107.

To launch the installer from the Business Availability Center Diagnostics downloads page:

1 In Business Availability Center select **Admin > Diagnostics** from the top menu and click the **Downloads** tab.

2 On the Downloads page, click the appropriate link to download the Java Agent installer for Windows.

Note: The Java Agent installers are available in Business Availability Center only if put into the required directory for Business Availability Center to access. You can enable this during the installation of the Diagnostic Server, or the Java Agent installers can be copied manually from the installation disk to the required location.

Continue with "Running the Installation" on page 107.

Running the Installation

After you have launched the installer, the software license agreement opens and you are ready to run the installation.

Note: If there is a pre-existing installation of the Java Agent on the host machine, see Appendix G, "Upgrading Diagnostics and Other HP Software Products."

To install the Java Agent on a Windows machine:

1 Accept the end user license agreement.

Read the agreement and select **I accept the terms of the license agreement**.

Click **Next** to proceed.

2 Specify the location where you want to install the agent.

Accept the default directory or select a different location either by typing the path to the installation directory into the **Installation Directory Name** box, or by clicking **Browse** to navigate to the installation directory.

Note: This location becomes the <probe_install_dir>. By default, the location is C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent on Windows and /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent on UNIX.

Click **Next** to proceed.

3 Review the summary information.

The installation directory and size requirement are listed.

Click **Next** to proceed.

4 Review the installation summary information. If the summary information panel indicates no errors, click Next to proceed.

The Java Agent Setup Module is started.

Configuring the Java Agent as a Profiler Only

The Java Agent can be configured as a standalone Java Agent that works with the Diagnostics Profiler or as a Java Agent that works with a Diagnostics Server. When the probe is configured as a Profiler only, you can reconfigure the probe in the future to work with the Diagnostics Server.

To configure the Java Agent as a Diagnostics Profiler only, you use the Java Agent Setup Module.

The Java Agent Setup Module starts automatically at the end of the Java Agent Installation. You can start the Java Agent Setup Module at any time by choosing **Start** > **All Programs** > **HP Java Agent** > **Setup Module**.

To configure the Java Agent to be a Diagnostics Profiler:

1 **Select the Diagnostics Profiler only option.**
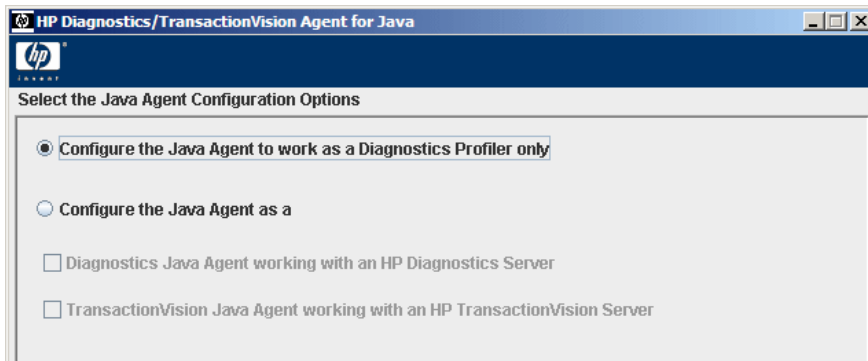
There are no other configuration options.

Click **Finish** to complete the configuration.

The Post Setup Summary dialog is displayed.

2 **Review the Post Setup Summary and then indicate whether or not you want to run the JRE Instrumenter at this time.**
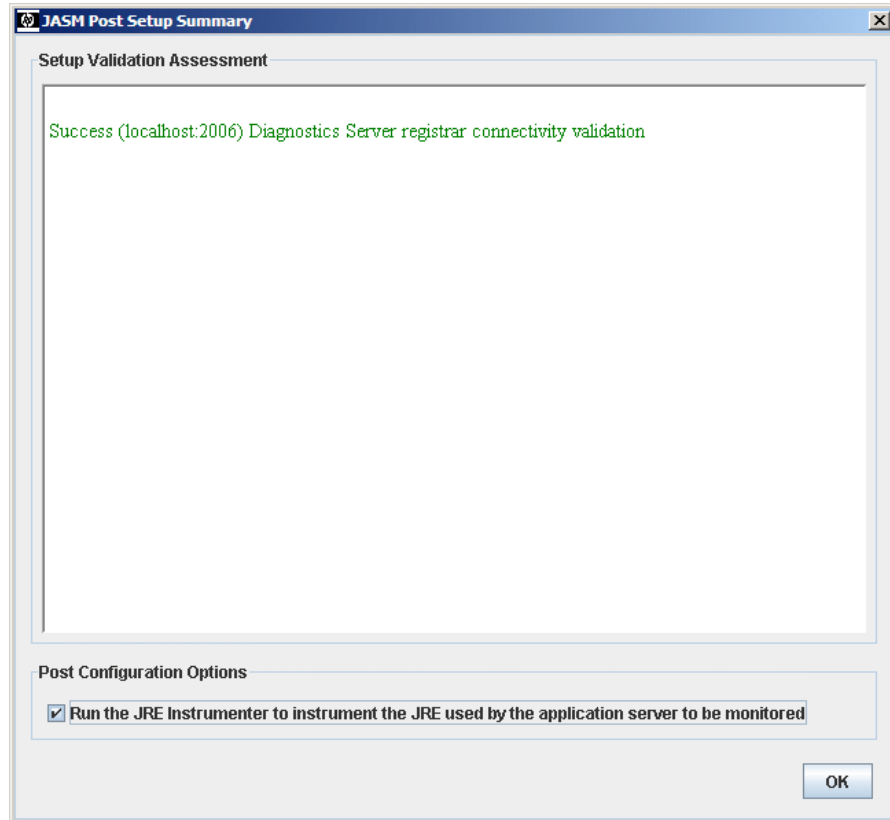
To run the instrumenter now, set the check box and click **OK.**

Note: If you are installing this agent to work with WebSphere in an Integrated Development Environment (IDE), you must run the JRE Instrumenter using a slightly different procedure than is described here. See "Running the JRE Instrumenter for WebSphere IDE" on page 164.

If you want to skip running the JRE Instrumenter now and run it later instead, clear the check box and click **OK**. The JRE Instrumenter window closes and the Java Agent configuration is complete.

For information about running the instrumenter separately from the Java Agent Setup program, see Chapter 6, "Running the JRE Instrumenter.".

3 Instrument the JVM and copy the startup parameters.

The Instrumentation window appears.

a Select the JRE to be instrumented and then click **Instrument**.
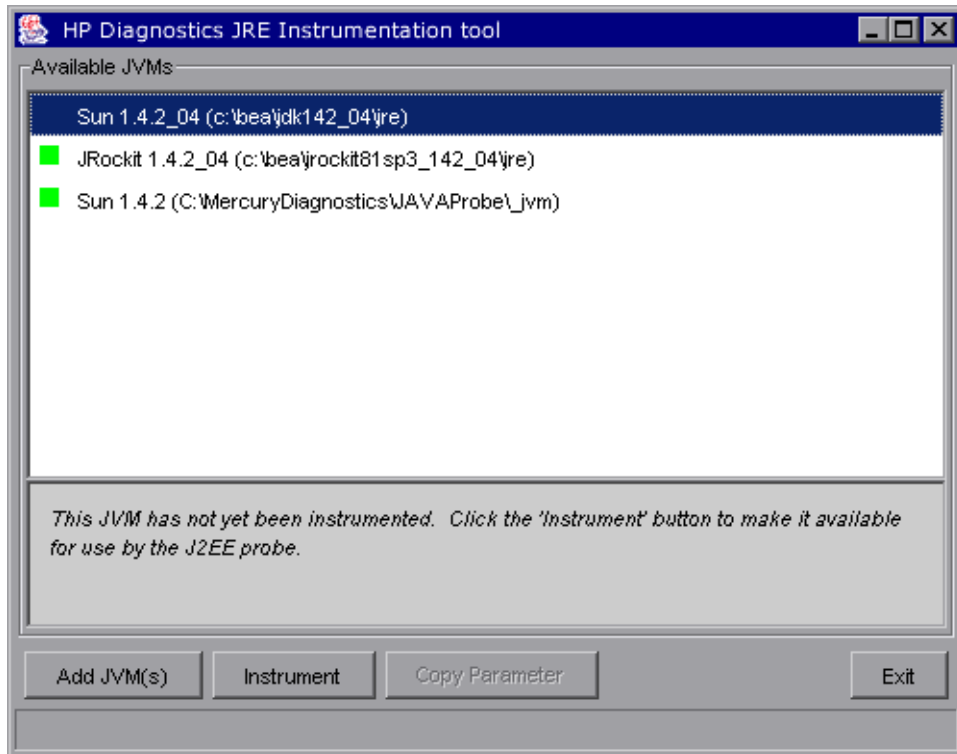
If the **JRE Directory** is not listed on the dialog, click the **Add JVM** button to browse to the JRE. For example, if you have installed WebLogic 8.1 (using Sun JDK) in your **D:\bea** directory, the **java.exe** file can be found in the bin folder of the JRE directory, **D:\bea\jdk142_05\JRE**.

b Click **Copy Parameter** to place the corresponding parameter on the clipboard.

Important: You will need to paste the clipboard contents in a later step so be careful to not overwrite the clipboard contents.

   c Click **Exit** to close the JRE Instrumenter window.

4 Proceed with post-installation tasks.

Once you have installed the agent, configured it as a Java Agent and instrumented the JRE, you need to

➤ Modify the startup script for the application server so that the probe is started together with the monitored application.

   For detailed instructions, see Chapter 7, "Configuring Application Server Startup Scripts to Work with the Java Agent."

➤ Perform additional probe configurations.

   Determine which advanced configurations of the probe apply to your environment. See Chapter 15, "Advanced Java Probe and Application Server Configuration."

Configuring the Java Agent to Work with a Diagnostics Server

The Java Agent can be configured as a standalone Java Agent that works with the Diagnostics Profiler or as a Java Agent that works with a Diagnostics Server.

To configure the Java Agent as a Java Agent reporting to a Diagnostics Server, you use the Java Agent Setup Module.

The Java Agent Setup Module starts automatically at the end of the Java Agent Installation. You can start the Java Agent Setup Module at any time by choosing **Start** > **All Programs** > **HP Java Agent** > **Setup Module**.

To configure the Java Agent to be a Java Agent reporting to a Diagnostics Server:

 1 **Select the Diagnostics Java Agent working with an HP Diagnostics Server option.**
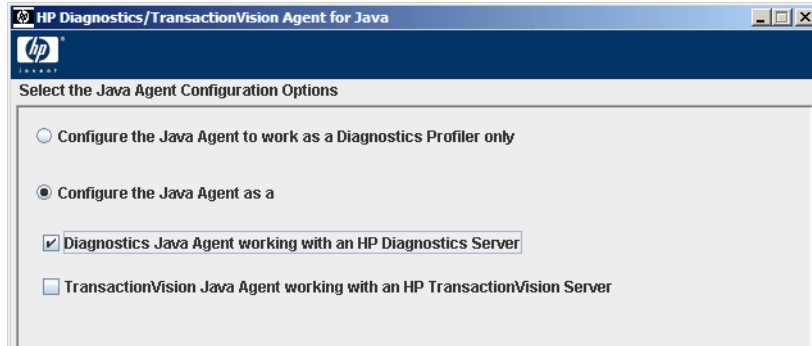
The Java Agent can be configured as a TransactionVision Sensor as well as a Diagnostics Java Agent. If you are configuring the Java Agent to be a TransactionVision Sensor, the Java Agent Setup Module will show dialogs that are not described in this manual (see the *HP TransactionVision Deployment Guide*).

Click **Next** to proceed.

 2 **Assign a name to the Java Agent and specify the group to which it belongs.**

➤ For the Java Agent name, enter a name that uniquely identify the agent within HP Diagnostics. The following characters can be used in the name: - , _ , and all alphanumeric characters. The agent name is assigned to be the probe name.

When assigning a name to an agent, choose a name that will help you recognize the application that the agent is monitoring, and the type of probe (in this case, a PetWorld application agent).

➤ For the Java Agent group name, enter a name for an existing group or for a new group to be created. The agent group name is case-sensitive.

Probe groups are logical groupings of probes that report to the same Diagnostics Server. The performance metrics for a probe group are tracked and can be displayed on many of the Diagnostics views.

For example, you may want to assign all of the probes for a particular enterprise application to a probe group so that you can monitor both the performance of the group and the performance of the individual probes.

Click **Next** to proceed.

3 Enter the configuration information for the Diagnostics Server, and additional options.

a In the **Diagnostics Server Name** box, enter the host name or IP address of the host of the Diagnostics Server.

If there is only one Diagnostics Server in the Diagnostics deployment where the agent will run, enter the Diagnostics Server's host name and event port information here.

When there is more than one Diagnostics Server in your deployment, enter the information for the Diagnostics Server that is to receive the events from the agent.

You should specify the fully qualified host name; not just the simple host name. In a mixed OS environment, where UNIX is one of the systems, this is essential for proper network routing.

b In the **Diagnostics Server Port** box, enter the port number of the Diagnostics Server.

The default port for the Diagnostics Server is **2006**. If you have changed the port since the Diagnostics Server was installed, you should specify that port number here instead of the default.

c If you want to allow this agent to support a SAP NetWeaver Application Server, set the **Tune Diagnostics Java Agent for use in an SAP NetWeaver Application Server** check box.

d If you want to allow this agent to work in an HP SaaS environment, set the **Disable direct outbound RHTTP access to Diagnostics Java Agent** check box.

The Diagnostics Server uses rhttp to query metrics from the Java Agent. You should disable the probe's outgoing rhttp to prevent HP Software-as-a-Service (SaaS) from accessing the data directly from the probe.

Select the appropriate options and click **Finish**.

The configuration process begins, and a progress bar indicates how the configuration is proceeding.

The connectivity to the Diagnostics Server is tested, and if any connectivity problems were encountered, the Set Up Program provides you with the results of the connectivity check.

Once you have installed the agent, configured it as a Java Agent and instrumented the JRE, you need to perform these post-installation tasks.

4 Modify the startup script for the application server so that the probe is started together with the monitored application.

For detailed instructions, see Chapter 7, "Configuring Application Server Startup Scripts to Work with the Java Agent."

5 Verify the probe installation.

The probe installation is verified by using the System Health Monitor as described in "Verifying the Java Agent Installation" on page 136.

6 Perform additional probe configurations.

Determine which advanced configurations of the probe apply to your environment. See Chapter 15, "Advanced Java Probe and Application Server Configuration."

Installing and Configuring the Java Agent on UNIX

Java Agent installers have been provided for several UNIX platforms. The following instructions provide you with the steps necessary to install the Java Agent in most UNIX environments using a console mode installation.

If you are installing the Java Agent on a UNIX machine using the graphical installer see the "Installing and Configuring the Java Agent on Windows" on page 105 section as these instructions also apply when using the UNIX graphical installer.

Important: By default, the **<probe_install_dir>/log** directory is set to 777. This ensures that the Java Agent is able to collect metrics from monitored applications being run by any user.

Depending on your organization's security requirements, you may want to further restrict access to this directory.

For example:

chmod 775 /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/log

---

In some instances, you may not be able to use the regular UNIX installers. In these cases, you should use the Generic installer as described in "Installing the Java Agent Using the Generic Installer" on page 132.

The instructions and screen shots that follow are for an agent installation on a Solaris machine. These same instructions should apply for the other certified UNIX platforms.

Note: If there is a pre-existing installation of the Java Agent on the host machine, see Appendix G, "Upgrading Diagnostics and Other HP Software Products."

This section includes:

➤ "Setting File Permissions for Probe Files (UNIX Only)" on page 117

➤ "Launching the Java Agent Installer on UNIX" on page 117

➤ "Running the Installation" on page 118

➤ "Configuring the Java Agent as a Profiler Only" on page 120

➤ "Configuring the Java Agent to Work with a Diagnostics Server" on page 122

Setting File Permissions for Probe Files (UNIX Only)

To enable the Java Agent to operate correctly, the user that starts the JVM being monitored must have the following access permissions:

➤ Read access to the **<probe_install_dir>** directory and files.

➤ Execute access to the **<probe_install_dir>/bin** directory.

➤ Read/Write access to the **<probe_install_dir>/log** directory.

To fully enable the Configuration tab in the Diagnostics Profiler, the user that starts the JVM being monitored must have the following access permissions:

➤ Read/Write access to the **<probe_install_dir>/etc** directory.

By default, the **<probe_install_dir>/log** directory is set to 777. This ensures that the Java Agent is able to collect metrics from monitored applications being run by any user.

Depending on your organization's security requirements, you may want to further restrict access to this directory.

For example:

chmod 775 /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/log

Launching the Java Agent Installer on UNIX

The Installer can be launched from the Diagnostics installation disk or from the Diagnostics Downloads page in Business Availability Center.

To launch the installer from the product installation disk:

1 From the **<HP Diagnostics Installation Disk>/Diagnostics_Installers** directory, copy the appropriate installer **JavaAgentSetup<platform>_<version>.bin** to the machine where the Java agent is to be installed.

2 Continue with "Running the Installation" on page 118.

To launch the installer from the Business Availability Center Diagnostics downloads page:

1 In Business Availability Center select **Admin > Diagnostics** from the top menu and click the **Downloads** tab.

2 On the Downloads page, click the link to the installer that is appropriate for your environment and save it to the machine where the Java agent is to be installed.

Note: The Java Agent installers are available in Business Availability Center only if enabled during the installation of the Diagnostic Server. See Step 3 on page 50 of Chapter 2, "Installing the Diagnostics Server."

Continue with "Running the Installation" on page 118.

Running the Installation

After you have copied the installer to the machine where the probe is to be installed, you are ready to run the installation.

If there is a pre-existing installation of the Java Agent on the host machine, see Appendix G, "Upgrading Diagnostics and Other HP Software Products."

Note: The following instructions assume an understanding of UNIX console screens and commands. For more information about UNIX screens and commands, see Appendix I, "Using UNIX Commands."

To install the Java Agent on a UNIX machine:

1 Run the installer.

Where necessary, change the mode of the installer file to make it executable.

➤ To run the installer in console mode, enter the following at the UNIX command prompt:

```
./<installer> -console
```

The installer displays the installation prompts in console mode as shown in the steps that follow.

➤ To run the installer in the graphical mode enter the following at the UNIX command prompt:

```
./<installer>
```

The installer displays the same screens that are displayed for the Windows installer, as shown in "Installing and Configuring the Java Agent on Windows" on page 105.

2 Accept the end user license agreement.

The end user software license agreement is displayed.

Read the agreement. As you read, you can press **Enter** to move to the next page of text, or type q to jump to the end of the license agreement.

Accept the terms of the agreement, and select **Next** to continue with the installation.

3 Specify the location where you want to install the agent.

At the **Installation Directory Name** prompt, accept the default installation location shown in brackets, or enter the path to a different location.

Select **Next** to continue with the installation.

4 Verify the installation location.

The installation location and the estimated size are listed.

If these are acceptable, select **Next** to start the installation.

The installation may take a few minutes.

The Java Agent Setup Module is launched.

Configuring the Java Agent as a Profiler Only
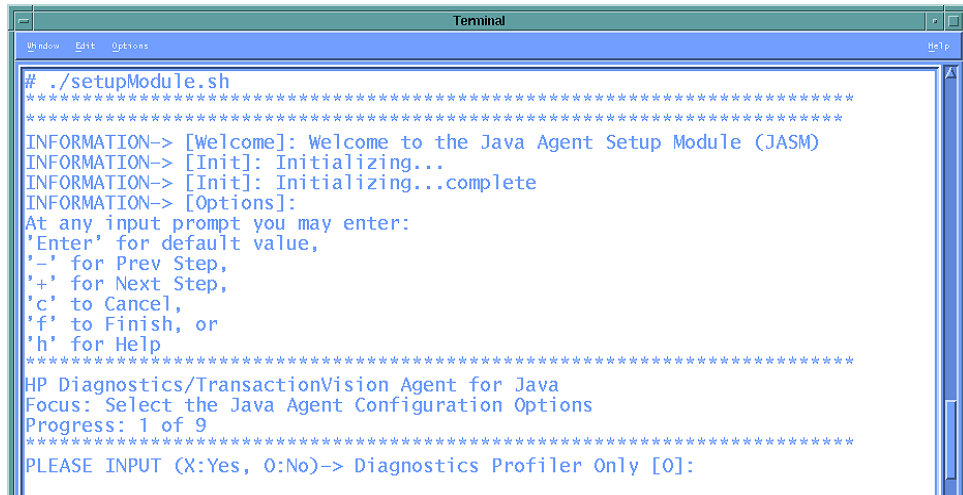
The Java Agent can be configured as a standalone Java Agent that works with the Diagnostics Profiler or as a Java Agent that works with a Diagnostics Server. When the probe is configured as a Profiler only, you can reconfigure the probe in the future to work with the Diagnostics Server.

To configure the Java Agent as a Diagnostics Profiler, you use the Java Agent Setup Module. The Java Agent Setup Module starts automatically at the end of the Installation program.

 You can start the Java Agent Setup Module at any time by running **<probe_install_dir>/bin/setupModule.sh**.

To configure the Java Agent to be a Diagnostics Profiler:

1 Enter O to select the Diagnostics Profiler Only option.
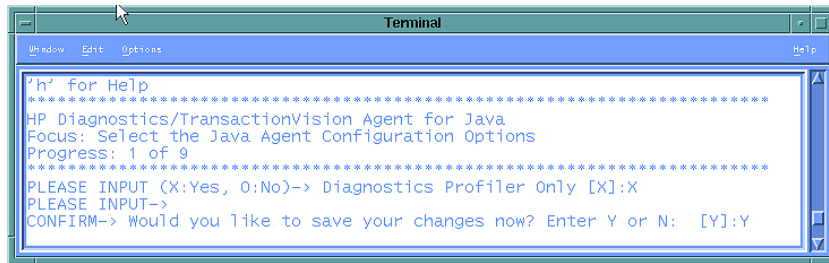
There are no other configuration options.

 2 **Enter Y to save the changes to the Java Agent Setup Module.**

 3 **Run the JRE Instrumenter.**

Enter an instrumentation command as follows:

➤ To instrument the JVM in a specific directory and copy the classloader information to /classes/boot, enter jreinstrumenter.sh -b <jvm_directory>.

<jvm_directory> is the path to the JRE bin folder where the JRE used by the monitored application server can be found. For example, if you have installed WebLogic 8.1 (using Sun JDK) in your **/opt/bea** directory, enter:

 jreinstrumenter.sh -b /opt/bea/jdk142_05/JRE

➤ To instrument the JVM in a specific directory, enter jreinstrumenter.sh -i <jvm_directory>.

➤ To list JVMs below a specific directory, enter jreinstrumenter.sh -a <jvm_directory>.

➤ To list all known JVMs, enter jreinstrumenter.sh -l.

Note: If you are installing this agent to work with WebSphere in an Integrated Development Environment (IDE), you must run the JRE Instrumenter using a slightly different procedure than is described here. See "Running the JRE Instrumenter for WebSphere IDE" on page 164.

---

4 Enter 0 (zero) to complete the configuration.

Once you have installed the agent, configured it as a Java Agent and instrumented the JRE, you need to proceed with the post-installation tasks.

➤ Modify the startup script for the application server so that the probe is started together with the monitored application.

For detailed instructions, see Chapter 7, "Configuring Application Server Startup Scripts to Work with the Java Agent."

➤ Perform additional probe configurations.

Determine which advanced configurations of the probe apply to your environment. See Chapter 15, "Advanced Java Probe and Application Server Configuration."

Configuring the Java Agent to Work with a Diagnostics Server

The Java Agent can be configured as a standalone Java Agent that works with the Diagnostics Profiler or as a Java Agent that works with a Diagnostics Server.

To configure the Java Agent to work as a Probe that reports to a Diagnostics Server, you use the Java Agent Setup Module. You start the Java Agent Setup Module in console mode by entering <**probe_install_dir**>/**bin**/**setupModule.sh -console**.

To configure the Java Agent to be a Java Agent reporting to a Diagnostics Server:

1 Enter O to skip the Diagnostics Profiler only option, and then X to select the Diagnostics Java Agent working with an HP Diagnostics Server option.
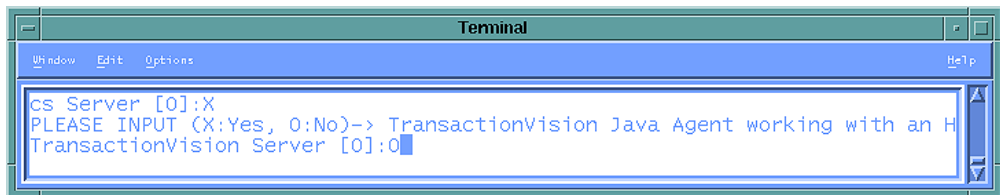
2 Specify whether the agent should also be configured to run as a TransactionVision sensor.

➤ Choose O to specify that the Agent is not to be configured as a TransactionVision sensor.

➤ Choose X to specify that the Agent is to be configured as a TransactionVision sensor.

The Java Agent can be configured as a TransactionVision Sensor as well as a Diagnostics Java Agent. If you are configuring the Java Agent to be a TransactionVision Sensor, the Java Agent Setup Module will show prompts that are not described in this manual.

If you are configuring the Java Agent to work as a TransactionVision Sensor, see the *HP TransactionVision Deployment Guide*.

3 Assign a name to the Java Agent and specify the group to which it belongs.

➤ For the Java Agent name, enter a name that uniquely identify the agent within HP Diagnostics. The following characters can be used in the name: - , _ , and all alphanumeric characters. The agent name is assigned to be the probe name.

When assigning a name to an agent, choose a name that will help you recognize the application that the agent is monitoring, and the type of probe (in this case, a PetWorld application agent).

➤ For the Java Agent group name, enter a name for an existing group or for a new group to be created. The agent group name is case-sensitive.

Probe groups are logical groupings of probes that report to the same Diagnostics Server. The performance metrics for a probe group are tracked and can be displayed on many of the Diagnostics views.

For example, you may want to assign all of the probes for a particular enterprise application to a probe group so that you can monitor both the performance of the group and the performance of the individual probes.

4 **Enter the configuration information for the Diagnostics Server.**
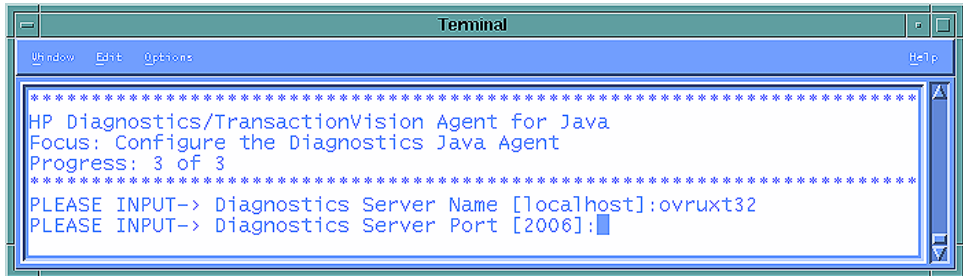
```
*********************************************************************************
HP Diagnostics/TransactionVision Agent for Java
Focus: Configure the Diagnostics Java Agent
Progress: 3 of 3
*********************************************************************************
PLEASE INPUT-> Diagnostics Server Name [localhost]:ovruxt32
PLEASE INPUT-> Diagnostics Server Port [2006]:
```

a Enter the host name or IP address of the host of the Diagnostics Server.

If there is only one Diagnostics Server in the Diagnostics deployment where the agent will run, enter the Diagnostics Server's host name and communication port information here.

When there is more than one Diagnostics Server in your deployment, enter the information for the Diagnostics Server that is to receive the events from the agent.

You should specify the fully qualified host name; not just the simple host name. In a mixed OS environment, where UNIX is one of the systems, this is essential for proper network routing.

b Enter the port number of the Diagnostics Server.

The default port for the Diagnostics Server is **2006**. If you have changed the port since the Diagnostics Server was installed, you should specify that port number here instead of the default.

5 **Specify whether to tune the agent for running in a SAP NetWeaver Application Server.**

```
PLEASE INPUT (X:Yes, O:No)-> Tune for SAP NetWeaver Application Server s
upport [O]:
```
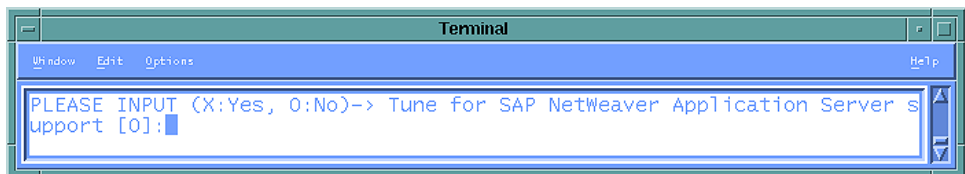
➤ To support the SAP NetWeaver Application Server, specify **X**.

➤ To disable support of the SAP NetWeaver Application Server, specify **O**.

6 Specify whether the agent will be running in an HP SaaS environment.

The Diagnostics Server uses rhttp to query metrics from the Java Agent. You should disable the probe's outgoing rhttp to prevent HP Software-as-a-Service (SaaS) from accessing the data directly from the probe
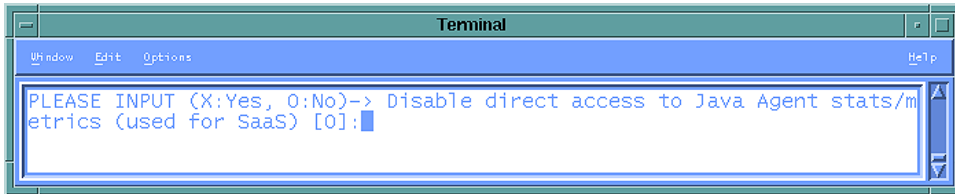
➤ To disable outbound rhttp, specify **X**.

➤ To enable outbound rhttp, specify **O**.

7 When prompted to save your changes to the Java Agent Setup Module, enter Y.

8 Instrument the JRE.

```
Terminal
Window  Edit  Options                                                    Help

CONFIRM-> Would you like to save your changes now? Enter Y or N:  [Y]:Y
INFORMATION-> [Save]: Saving Dialog Select the Java Agent Configuration
Options
INFORMATION-> [Save]: Saving Dialog Identify the Java Agent
INFORMATION-> [Save]: Saving Dialog Configure the Diagnostics Java Agent
INFORMATION-> [Save]: Saving Dialog Configure the TransactionVision Java
 Agent (page 1 of 2)
INFORMATION-> [Save]: Saving Dialog Configure the TransactionVision Java
 Agent (page 2 of 2)
INFORMATION-> [Save]: Saving Dialog Configure the TransactionVision Java
 Agent Transport Options for WebSphere MQ
INFORMATION-> [Save]: Saving Dialog Configure the TransactionVision Java
 Agent Transport Options for Sonic MQ
INFORMATION-> [Save]: Saving Dialog Configure the TransactionVision Java
 Agent Transport Options for Tibco EMS
INFORMATION-> [Save]: Saving Dialog Configure the TransactionVision Java
 Agent Transport Options for WebLogic JMS
INFORMATION-> [Save]: Saving Diagnostics Agent property files
INFORMATION-> [JASM Post Setup Summary]:
Success (ovruxt32:2006) Diagnostics Server registrar connectivity valida
tion
INFORMATION-> [Currently Instrumented VMs]:
HP 1.5.0.04 (/opt/bea10/jdk150_04/jre)
PLEASE INPUT-> Option? ('Command', H: Help, or 0: Exit) [0]:
```

Enter an instrumentation command as follows:

➤ To instrument the JVM in a specific directory and copy the classloader information to /classes/boot, enter jreinstrumenter.sh -b <jvm_directory>.

<jvm_directory> is the path to the JRE bin folder where the JRE used by the monitored application server can be found. For example, if you have installed WebLogic 8.1 (using Sun JDK) in your **/opt/bea** directory, enter:

   jreinstrumenter.sh -b /opt/bea/jdk142_05/JRE

➤ To instrument the JVM in a specific directory, enter jreinstrumenter.sh -i <jvm_directory>.

➤ To list JVMs below a specific directory, enter jreinstrumenter.sh -a <jvm_directory>.

➤ To list all known JVMs, enter jreinstrumenter.sh -l.

127

Note: If you are installing this agent to work with WebSphere in an Integrated Development Environment (IDE), you must run the JRE Instrumenter using a slightly different procedure than is described here. See "Running the JRE Instrumenter for WebSphere IDE" on page 164.

---

 9 **Enter 0 (zero) to complete the setup.**

Once you have installed the agent, configured it as a Java Agent and instrumented the JRE, you need to perform post-installation tasks.

10 **Modify the startup script for the application server so that the probe is started together with the monitored application.**

For detailed instructions, see Chapter 7, "Configuring Application Server Startup Scripts to Work with the Java Agent."

11 **Verify the probe installation.**

The probe installation is verified by using the System Health Monitor as described in "Verifying the Java Agent Installation" on page 136.

12 **Perform additional probe configurations.**

Determine which advanced configurations of the probe apply to your environment. See Chapter 15, "Advanced Java Probe and Application Server Configuration."

Installing the Java Agent on a z/OS Mainframe

This section provides instructions for installing the Java Agent from the .tar file that is included on the Diagnostics installation disk.

Important Considerations when Installing the Java Agent on z/OS

Consider the following before you install a Java Agent and configure it to be a Java Agent in a z/OS environment:

Editing Property Files on a z/OS Mainframe

When installed in a z/OS environment, the probe expects the Diagnostics property files to be in EBCDIC format. Use an EBCDIC editor to update the property files and make sure to store the updates in the same format.

Viewing the System Logs in z/OS

You can view the system log by accessing the primary operator's console in SDSF.

Capturing Metrics on the z/OS Mainframe

Load test metrics are not captured for z/OS. The probe can be configured to capture a limited number of system level metrics.

For more information on capturing system metrics in z/OS, see "Enabling z/OS System Metrics Capture" on page 546.

Installing the Java Agent on z/OS from the Diagnostics Installation Disk

A .tar file containing the Java Agent files has been included on the Diagnostics installation disk.

To install the Java Agent on a z/OS mainframe:

1 Upload **JavaAgentInstall_zOS.tar.gz** from the **Diagnostics_Installers** folder on the HP Diagnostics installation disk to the directory on the z/OS machine where you want to unzip the installer.

2 Unzip **JavaAgentInstall_zOS.tar.gz** using gzip as shown in the following example:

```
gzip -d JavaAgentInstall_zOS.tar.gz
```

This command creates the unzipped file, **JavaAgentInstall_zOS.tar**.

3 To unpack the .tar file, run the .tar command as shown in the following example:

```
tar -xvf JavaAgentInstall_zOS.tar
```

This command creates the unpacked directory, **JavaAgentInstall.**

4 Prepare to run **jreinstrumenter.sh** by setting the permissions on the script so that it can be run (o+x).

5 Run the JRE Instrumenter. For z/OS, use the following command:

```
<path_to_java> -jar <path_to_jre_instrumenter> -b <jvm_directory>
```

where:

➤ <path_to_java> is the path to the Java executable

➤ <path_to_jre_instrumenter> is **<probe_install_dir>/lib/ jreinstrumenter.jar**

➤ <jvm_directory> is the path to the JRE for the java installation that your application is using

6 Configure the probe properties to enable the probe to work with the other Diagnostics components as described in "About Advanced Java Agent Configuration" on page 139.

7 Once you have installed the probe and instrumented the JRE, you need to manually modify the startup script for the application server so that the probe is started together with the monitored application. For detailed instructions, see Chapter 7, "Configuring Application Server Startup Scripts to Work with the Java Agent."

8 Verify the probe installation as described in "Verifying the Java Agent Installation" on page 136.

9 If you want to collect z/OS system metrics, see "Enabling z/OS System Metrics Capture" on page 546.

Installing Java Agents on Multiple z/OS Machines

If you will be installing Java Agents on more than one z/OS machine, you may want to create a pax archive of the agent implementation on the first machine and then use the pax archive to install the agent onto the other machines. Contact your system administrator for more information.

Installing the Java Agent Using the Generic Installer

Important! This section only applies if you are going to configure the Java Agent as a Java Probe that reports to a Diagnostics Server.

The installers for the Java Agent have been built to support installing the agent on all of the platforms for which the component has been certified. However, the agent may work on other platforms that have not yet been certified. A generic installer has been provided on the product installation disk to allow you to install the agent on these uncertified platforms.

To get the agent to work on the platforms that are not supported by the regular installer, you must run the generic installer, and manually configure the agent as a Java Probe so that it will be able to communicate with the other Diagnostics components and be able to monitor the processing of your application.

To install and configure the Java Agent on an uncertified platform:

1 Locate **JavaAgentInstall.tar.gz** from the **Diagnostics_Installers** folder on the HP Diagnostics installation disk.

2 Unzip **JavaAgentInstall.tar.gz** using gzip as shown in the following example:

```
gzip -d JavaAgentInstall.tar.gz
```

When this command completes, the unzipped file is called **JavaAgentInstall.tar**.

3 To unpack the tar file, execute the tar command as shown in the following example:

```
tar -xvf JavaAgentInstall.tar
```

This command creates the unpacked **JavaAgentInstall** directory.

4 Configure the probe to enable the probe to work the other Diagnostics components as described in "About Advanced Java Agent Configuration" on page 139.

5 Configure the Application Server to allow the probe to monitor the application.

    a Run the JRE Instrumenter as described in "Running the JRE Instrumenter" on page 141.

    b Once you have installed the probe and instrumented the JRE, you need to manually modify the startup script for the application server so that the probe is started together with the monitored application. For detailed instructions, see Chapter 7, "Configuring Application Server Startup Scripts to Work with the Java Agent."

6 Verify the probe installation as described in "Verifying the Java Agent Installation" on page 136.

Silent Installation of the Java Agent

A *silent installation* is an installation that is performed automatically, without the need for user interaction. In place of user input, the silent installation accepts input from a response file for each step of the installation.

For example, a system administrator who needs to deploy a component on multiple machines can create a response file that contains all the prerequisite configuration information, and then perform a silent installation on multiple machines. This eliminates the need to provide any manual input during the installation procedure.

Before you perform silent installations on multiple machines, you need to generate a response file that will provide input during the installation procedure. This response file can be used in all silent installations that require the same input during installation.

The response file has the suffix **.rsp**. You can edit the response file with a standard text editor if desired.

The silent installation uses two response files: one for the Java Agent installation and one for the Java Agent Setup Module.

To generate a response file for the Agent installation:

➤ Perform a regular installation with the following command-line option:

```
<installer> -options-record <installResponseFileName>
```

This creates a response file that includes all the information submitted during the installation.

To generate a response file for the Java Agent Setup Module:

➤ Run the Java Agent Setup Module with the following command-line options.

On Windows:

```
<probe_install_dir>\bin\setupModule.cmd -createBackups -console -recordFile
   <JASMResponseFileName>
```

On UNIX:

```
<probe_install_dir>/bin/setupModule.sh -createBackups -console -recordFile
   <JASMResponseFileName>
```

Either command creates a response file that includes all the information submitted during the setup.

To perform a silent installation of the Java Agent:

➤ Perform a silent installation using the relevant response file.

You first set an environment variable and then run the installer with the **-silent** command-line option as follows.

```
set HP_JAVA_AGENT_SETUP=-DoNotRun
<installer> -options <installResponseFileName> -silent
```

On UNIX systems you must use quotes when specifying the environment variable.

```
set HP_JAVA_AGENT_SETUP="-DoNotRun"
```

To perform a silent configuration using the Java Agent Setup Module:

➤ Perform a silent installation using the relevant response file.

You must unset the environment variable and then run the Java Agent Setup Module with the **-silent** command-line option as follows.:

```
set HP_JAVA_AGENT_SETUP=
<setupModule>  -createBackups  -console  -installFile <JASMResponseFileName>
```

On UNIX systems you must use quotes when specifying the environment variable.

```
set HP_JAVA_AGENT_SETUP="-DoNotRun"
```

About Configuring the Application Server

In order to allow the Probe to monitor an application, you need to instrument the JRE for some JRE versions, manually modify the application server startup script and configure SOAP message handlers for some application servers.

If the JRE is 1.5 or later you can skip instrumenting the JRE. For earlier versions of the JRE you have to instrument the JRE manually.

Instrumenting the JRE

The instrumentation process defines for the Probe, the classes and methods that are to be monitored in the application. It also controls the layers that will be used to report the performance metrics that were gathered.

You use the JRE Instrumenter to run the instrumentation. The JRE Instrumenter automatically runs at the end of the Probe installation to prepare the application for instrumentation unless you indicate otherwise.

If you elect to skip running the JRE Instrumenter during the Probe installation, you must run it manually before you can use the Probe to monitor the application. For more information, see Chapter 6, "Running the JRE Instrumenter."

Modifying the Application Startup Script

Once you have installed the Probe and instrumented the JRE, you need to manually modify the startup script for the application server so that the Probe is started together with the monitored application.

For detailed instructions about how to modify the application server's startup script, see Chapter 7, "Configuring Application Server Startup Scripts to Work with the Java Agent."

Configuring SOAP Message Handlers

The Diagnostics SOAP message handler is required for probes to support the following features:

➤ Collect payload for SOAP faults.

➤ Determine SOA consumer ID from SOAP header, body, or envelope.

For most application servers, the instrumentation points and code snippets have been written to automatically configure the Diagnostics handlers for web services being monitored.

For WebSphere 5 JAX-RPC and Oracle 10g JAX-RPC, manual steps are required to configure the SOAP handler. See Chapter 7, "Configuring Application Server Startup Scripts to Work with the Java Agent."

The process for configuring the Java Agent and the application servers when there are multiple JVMs on a single machine is described in "Configuring the Probes for Multiple Application Server JVM Instances" on page 387.

Verifying the Java Agent Installation

You can verify the Probe installation using the System Health Monitor. This section only applies if you installed the Probe to run with a Diagnostics Server.

The Probe does not register with the Diagnostics Server until it is started. The Probe is started when the instrumented application server is started. Therefore, you cannot verify the installation of the Probe using the System Health Monitor until you have configured the Probe and Application Server.

If you have been following the recommended installation sequence, after you have installed the Java Agent you are able to verify the following:

➤ The Diagnostics Server in Commander mode was successfully installed.

➤ Where relevant, additional Diagnostics Servers were successfully installed and are communicating with the Diagnostics Server (CommandingServer).

➤ The Java Agent was successfully installed and has established connectivity with the Diagnostics Server.

The Java Agent is not displayed in System Health when first installed because the Probe does not register with the Diagnostics Server until it is started. The Probe is started and is registered with the Diagnostics Server when you start the monitored application server.

Depending on your deployment, the new Java Agent is shown on the System Health Monitor as a child of either the Diagnostics Server in Commander mode or the Diagnostics Server in Mediator mode.
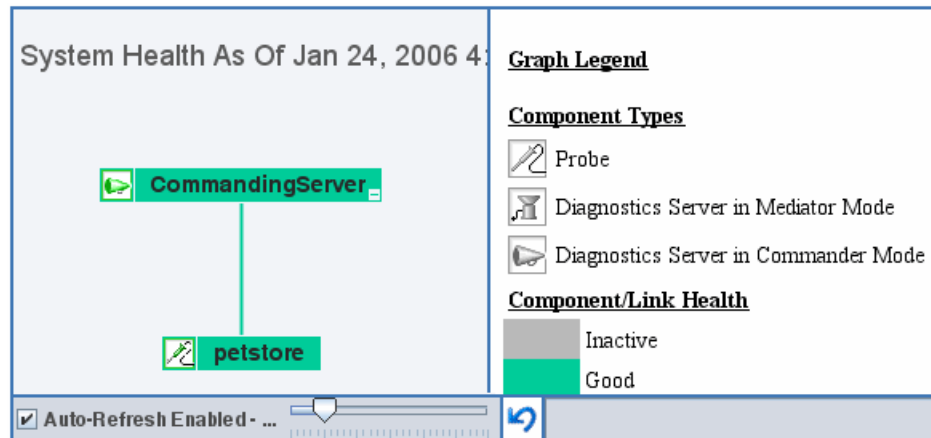
The Java Agent is colored grey in the System Health Monitor when the monitored application is closed. Once the application has been started the Probe is colored green.

For detailed instructions about how to use the System Health Monitor, see Appendix D, "Using the System Health Monitor".

Determining the Version of the Java Agent

When you request support, it is useful to know the version of the Diagnostics component that you have a question about.

You can find the version of the Probe in one of the following ways:

➤ Locate the version file **<Probe_install_dir>\version.txt**. The file contains the 4-digit version number, as well as the build number.

➤ The version number is available in the Probe log file (**<Probe_install_dir>/ log/<probe_id>/probe.log**)

➤ The version number is available among the details of the Probe on System Health Monitor.

Uninstalling the Java Agent

To uninstall the Java probe:

➤ On a Windows machine, choose Start > All Programs > HP Java Agent > Uninstaller.

Alternatively you can run **uninstaller.exe** which is located in the **<probe_install_dir>_uninst** directory.

➤ On a Linux or Solaris UNIX machine, run **uninstall*** which is located in the **<probe_install_dir>/_uninst** directory.

➤ On other UNIX machines, choose a 1.5 or later JVM and run **java -jar <probe_install_dir>/_uninst/uninstall.jar** to uninstall the Java probe.

About Advanced Java Agent Configuration

You should review the topics in Chapter 15, "Advanced Java Probe and Application Server Configuration" to determine if there are Probe configuration settings that will enable the Probe to work more efficiently in your environment.

About Custom Instrumentation for Java Applications

In addition, you can create custom instrumentation points to handle unique situations in your application environment. For general information on custom instrumentation you should review the topics in Chapter 9, "Custom Instrumentation for Java Applications".

6

Running the JRE Instrumenter

Instructions are provided for how to run the JRE Instrumenter to enable the Java Probe to monitor an application.

This chapter includes:

About the JRE Instrumenter

When you install a Java Probe, you must run the JRE Instrumenter to allow the Probe to gather the performance metrics for the application that it is to monitor. The JRE Instrumenter instruments the ClassLoader class for the JVM that the application is using and places the instrumented ClassLoader in a folder under the <probe_install_dir> /classes directory. It also provides you with the JVM parameter that must be used when the application server is started so that the application server will use the instrumented class loader.

The JRE Instrumenter runs automatically during the Probe installation unless you elect not to run it at that time. If you elect to skip the JRE Instrumenter during the installation of the Probe, you must run it manually.

Also you must run the JRE Instrumenter again if the JDK (java.exe executable) used by the application server changes so that the Probe will be able to continue to monitor the processing.

Notes:

➤ If you want to instrument IBM's 1.4.2 J9 JRE, you must instrument the correct ClassLoader and add the -Xj9 option on the application's command line. The correct ClassLoader is located in the **<java dir>\jre\lib\jclSC14** directory (e.g., jreinstrumenter.sh -i \usr\java14_64\jre\lib\jclSC14).

➤ If a Probe is being used to monitor multiple JVMs, the JRE Instrumenter must be run once for each JVM so that the Probe can be prepared to instrument the applications that are running on each JVM. For details, see "Configuring the Probes for Multiple Application Server JVM Instances" on page 387.

JRE Instrumenter Processing

The JRE Instrumenter performs the following functions:

➤ Identifies JVMs that are available to be instrumented.

➤ Searches for additional JVMs in directories that you specify.

➤ Instruments the JVMs that you specify and provides the parameter that you must add to the startup script for the JVM to point to the location of the instrumented ClassLoader class.

The Instrumenter puts the instrumented ClassLoader in different places depending on how it is executed.

   ➤ When the Instrumenter is run from the Probe installer, the Instrumenter places the instrumented ClassLoader in a folder under the **<probe_install_dir> /classes/boot** directory.

   ➤ When the Instrumenter is run using the graphical interface in a Windows or UNIX environment, the Instrumenter places the instrumented ClassLoader in a folder under the **<probe_install_dir> / classes/<JVM_vendor>/<JVM_version>** directory.

   ➤ When the Instrumenter is run in a UNIX environment in console mode, the Instrumenter places the instrumented ClassLoader in either a folder under the **<probe_install_dir> /classes/boot** directory or the **<probe_install_dir> /classes/<JVM_vendor>/<JVM_version>** directory depending on the processing option specified. For more information on the UNIX processing options see "Instrumenting a Listed JVM" on page 151.

Running the JRE Instrumenter

Instructions for running the JRE Instrumenter in a Windows environment and in a UNIX environment in console mode are provided below.

Note: If you chose to not run the JRE Instrumenter during the Probe installation, you must run it manually to enable the Probe to gather the performance metrics for your application.

This section includes:

➤ "Running the JRE Instrumenter on a Windows Machine" on page 144

Running the JRE Instrumenter on a Windows Machine

When the JRE Instrumenter is run in a Windows environment, the Instrumenter displays the dialogs of its graphical user interface. The same dialogs are displayed when running the installer on a UNIX machine when the Instrumenter is not running in console mode.

Starting the JRE Instrumenter on a Windows Machine

Open **<probe_install_dir>\bin** to locate the JRE Instrumenter executable. Run the command **jreinstrumenter.cmd**. When the Instrumenter starts, it displays the JRE Instrumentation Tool dialog.
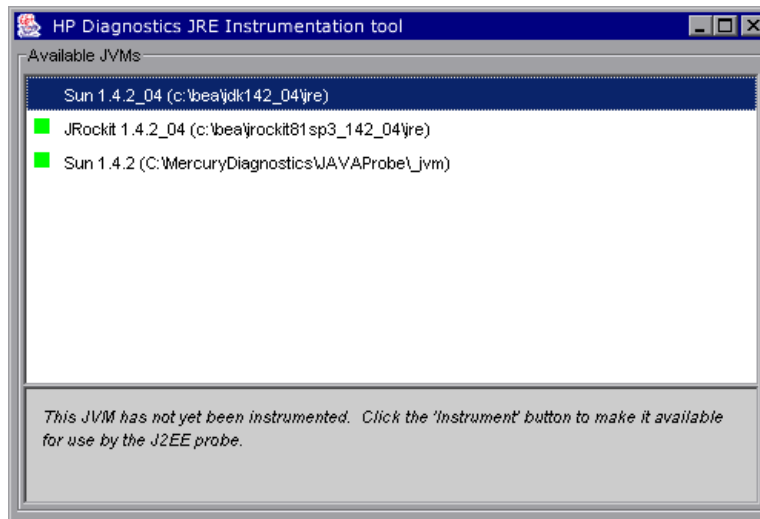
The Instrumenter lists the JVMs that were discovered by the Instrumenter and are available for instrumentation. The JVMs that have already been instrumented are listed with a green square preceding the name of the JVM.

From the JRE Instrumentation Tool dialog you can perform the following tasks:

➤ If the JVM that you want to instrument is not listed in the Available JVMs list in the dialog, you can add JVMs to the list as described in "Adding JVMs to the Available JVMs List" on page 145.

➤ If the JVM that you want to instrument is listed but has not yet been instrumented, you can instrument the JVM as described in "Instrumenting a Selected JVM" on page 148.

➤ If the JVM that you want to instrument is listed and has already be instrumented, you can copy the JVM parameter that must be inserted into the start script for the JVM to activate the Probe's monitoring as described in "Including the JVM Parameter in the Application Server's Startup Script" on page 148.

➤ If you have finished using the JRE Instrumenter you can click **Exit** to close the JRE Instrumentation Tool.

Adding JVMs to the Available JVMs List

1 In the JRE Instrumentation Tool dialog, click **Add JVM(s)** to search for other JVMs and add them to the Available JVMs list.

The Instrumenter displays the Choose the Directory dialog box.
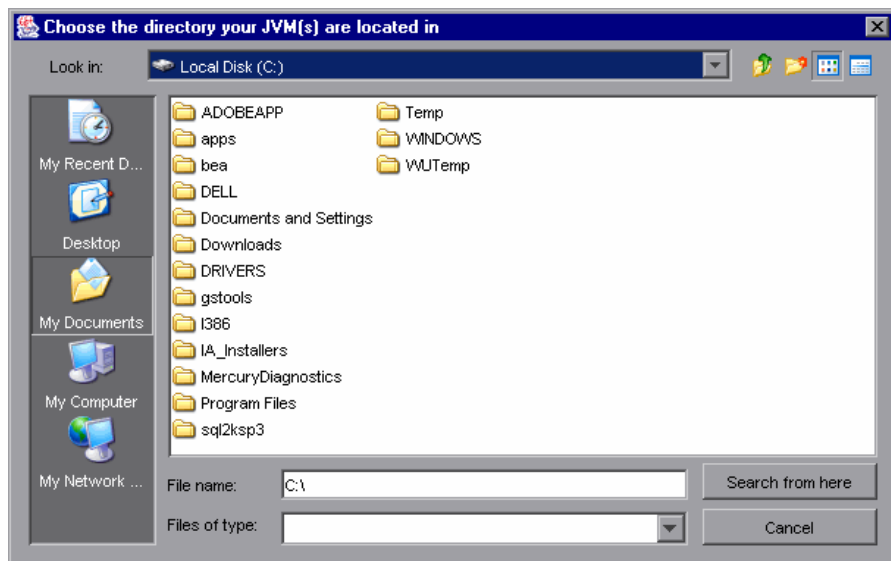
2 Navigate to the directory location where you would like the Instrumenter to begin searching for JVMs using the standard Windows Explorer type navigation controls.

3 Select the file where you would like to begin the search so that its name appears in the **File name** box.

4 Click **Search from here** to start searching for JVMs.

The Instrumenter closes the dialog box and displays the JRE Instrumentation tool dialog box once more. The command buttons on the dialog are disabled while the Instrumenter searches for JVMs. A progress bar at the bottom of the dialog indicates that the Instrumenter is searching and shows how far along it is in the search process.

As the tool locates JVMs, it lists them in the **Available JVMs** list.
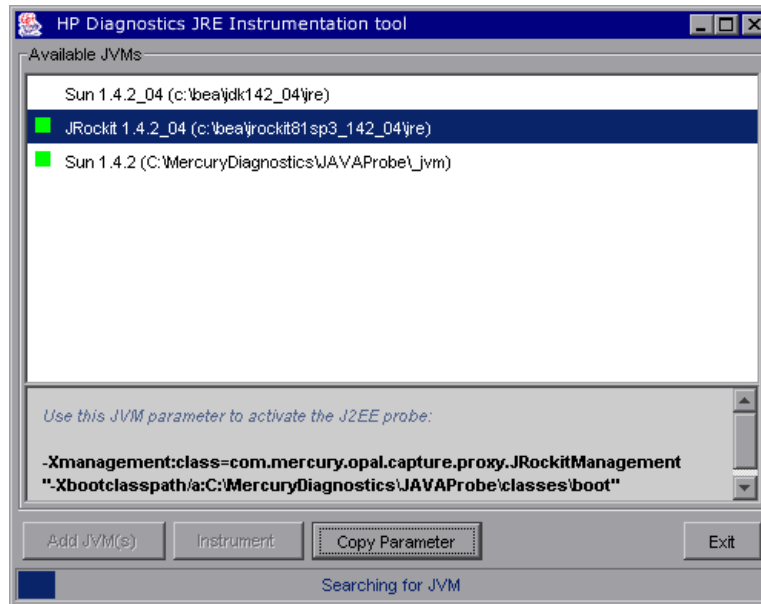
When the Instrumenter has completed the search, it enables the command buttons on the dialog as shown below. If the selected row is a JVM that has already been instrumented, the Instrument button is disabled.
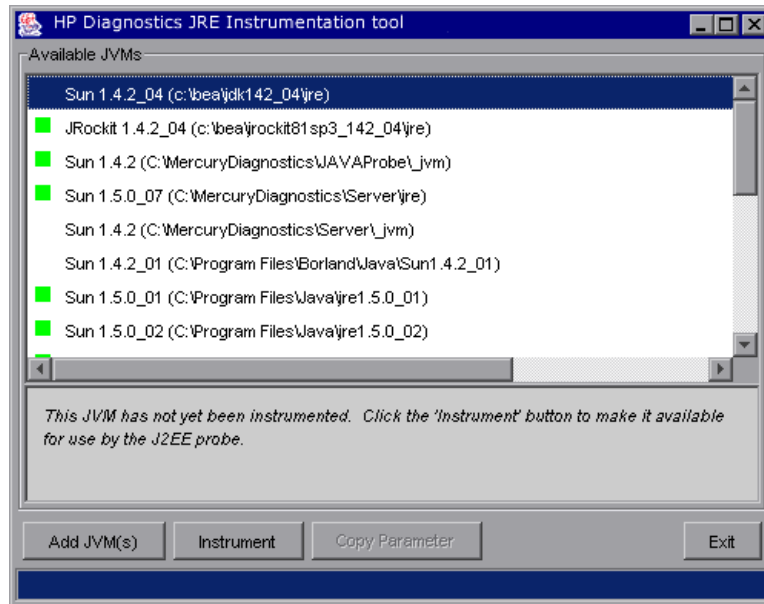
Instrumenting a Selected JVM

Select a JVM that has not been instrumented from the **Available JVMs** list and click **Instrument**.
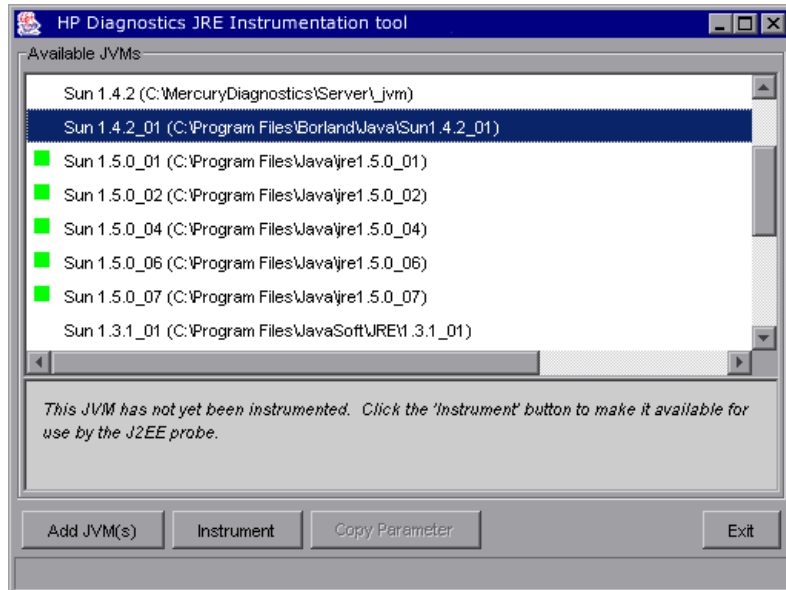
The JRE Instrumenter instruments the ClassLoader class for the selected JVM and places the instrumented ClassLoader in a folder under the **<probe_install_dir> /classes** directory. It also displays the JVM parameter that must be used when the application server is started in the box below the Available JVMs list.

Including the JVM Parameter in the Application Server's Startup Script

When the JRE Instrumenter instruments a JVM it also creates the JVM parameter that you must include in the startup script for the application server in order to cause your application to use the instrumented class loader. When you select an instrumented JVM from the Available JVMs list the JVM parameter is displayed in the box below the list.

To copy the JVM parameter displayed in this box to the clipboard, click **Copy Parameter**. The JVM parameter is copied to the clipboard so that you can paste it into the location that allows it to be picked up when your application server starts.

Running the JRE Instrumenter on a UNIX Machine

The following instructions provide you with the steps necessary to run the JRE Instrumenter using either a graphical mode installation or a console mode installation.
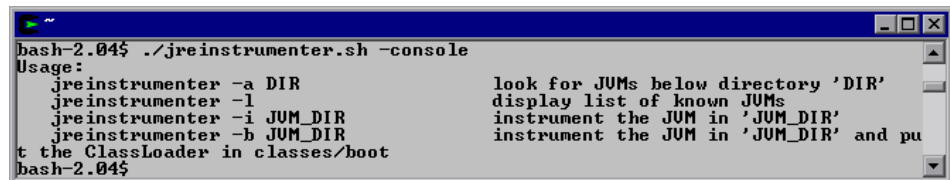
The JRE Instrumenter screens that are displayed in a graphical mode are the same as those documented for Windows installer in "Running the JRE Instrumenter on a Windows Machine" on page 144.

Starting the JRE Instrumenter on a UNIX Machine

Open <**probe_install_dir**>**bin** to locate the JRE Instrumenter executable. Run the following command:

```
./jreinstrumenter.sh -console
```

When the Instrumenter starts, it displays a list of the processing options that are available:

You can redisplay the list of options by specifying the -x option when you run the jreinstrumenter.sh command:

```
./jreinstrumenter.sh -x
```

149

The following table directs you to the documentation for each of the processing options:

Instrumenter Function	Documentation Section
jreinstrumenter -l	"Displaying the List of Instrumented JVMs" on page 150
jreinstrumenter -a DIR	"Adding JVMs to the Available JVMs List" on page 150
jreinstrumenter -i JVM_DIR jreinstrumenter -b JVM_DIR	"Instrumenting a Listed JVM" on page 151

Displaying the List of Instrumented JVMs

To display a list of the JVMs that are known to the JRE Instrumenter enter the following command:

```
./jreinstrumenter.sh -l
```

The Instrumenter lists the JVMs that it is aware of in rows containing the JVM vendor, JVM version, and the location where the JVM is located.

Adding JVMs to the Available JVMs List

To search for JVMs within a specific directory and add any JVMs that are found to the list of the JVMs that are known to the JRE Instrumenter enter the following command:

```
./jreinstrumenter.sh -a DIR
```

Replace DIR with the path to the location where you would like the Instrumenter to begin searching.

The Instrumenter searches the directories from the location specified including the directories and subdirectories. When it has completed its search, it displays the updated list of Available JVMs.

Instrumenting a Listed JVM

To instrument a JVM listed in the Available JVMs list, use one of the following two commands:

➤ Explicit path to ClassLoader

```
./jreinstrumenter.sh -i JVM_DIR
```

Replace JVM_DIR with the path to the location of the JVM as specified in the Available JVM list.

This command instructs the JRE Instrumenter to instrument the ClassLoader class for the selected JVM and places the instrumented ClassLoader in a folder under the **<probe_install_dir>** **/classes/ <JVM_vendor>/<JVM_version>** directory.

This is the command that you should use; especially if you want to instrument multiple JVM to be monitored by a single Probe.

➤ Generic path to ClassLoader
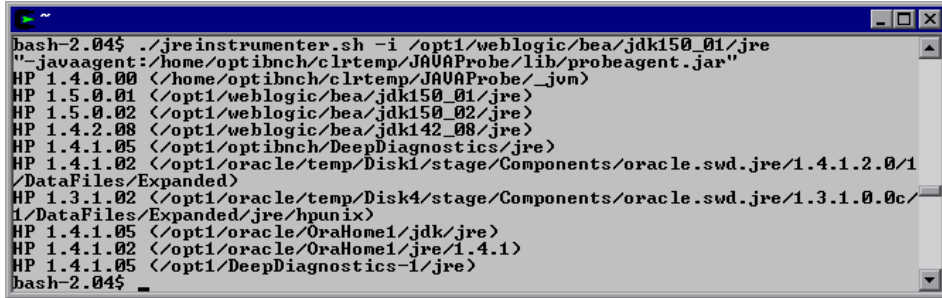
```
./jreinstrumenter.sh -b JVM_DIR
```

Replace JVM_DIR with the path to the location of the JVM as specified in the Available JVM list.

This command instructs the JRE Instrumenter to instrument the ClassLoader class for the selected JVM and places the instrumented ClassLoader in a folder under the **<probe_install_dir>** **/classes/boot** directory.

You should only use this command if you are monitoring a single JVM with the Probe and there is some reason that you do not want to use the more explicit path generated when you use the -i command option.

When the Instrumenter has finished instrumenting the JVM, it displays the JVM parameter that must be used to activate the instrumentation and enable the Probe to monitor the performance of your application. Following the JVM parameter, the Instrumenter lists the Available JVM list again as shown in the following example:

Including the JVM Parameter in the Application Server's Startup Script

When the JRE Instrumenter instruments a JVM it also creates the JVM parameter that you must include in the startup script for the application server in order to cause your application to use the instrumented class loader. When the Instrumenter has finished instrumenting the JVM, it displays the JVM parameter.

Copy the JVM parameter to the clipboard and paste it into the location that allows it to be picked up when your application server starts.

7

Configuring Application Server Startup Scripts to Work with the Java Agent

Instructions are provided for how to configure the application servers to allow the probe to monitor the application.

This chapter includes:

About Configuring the Application Server Startup Scripts

Once you have executed the JRE Instrumenter for the Java Probe, you must modify the startup script for the application so that the probe that is to monitor the application will be started when the application is started.

You can configure the application servers by updating the application server startup scripts manually. The following sections provide instructions for updating the application servers manually.

Important: Example procedures are shown for a given version of the application server.
For the most recent information on what application server versions are supported on what platforms refer to the Diagnostics Product Availability Matrix at
http://support.openview.hp.com/sc/support_matrices.jsp. or contact HP customer support.

It is possible that your site administrator has site-specific methods for making configuration modifications. In this case, the generic procedure described in "Configuring a Generic Application Server" on page 187 should provide the information that the site administrator needs to implement the required configuration changes.

Note: If there are no instructions for your specific type of application server in the sections that follow, follow the procedure in "Configuring a Generic Application Server" on page 187.

The process for configuring the Java Probe and the application servers when there are multiple JVMs on a single machine is described in "Configuring the Probes for Multiple Application Server JVM Instances" on page 387.

Notes:

➤ **<probe_install_dir>** is used to indicate the directory where the Java Probe was installed.

➤ When modifying the **-Xbootclasspath** parameter, be sure to use quotes if there are spaces in the path that you specify.

Configuring WebSphere Application Servers

WebSphere servers are controlled using the WebSphere Application Server Administrative Console. The Console has control over the JVM command line and allows you to add classpath elements, define runtime variables (-**D** variables), and configure the bootclasspath for WebSphere. You use the Administrative Console to add the Xbootclasspath property and any additional arguments that are needed for the JVM command line.

The appearance of the Console may differ for different versions of WebSphere. The way that changes are implemented in each version of WebSphere may vary slightly. As a result, the examples shown in this section may not correspond exactly to your WebSphere version. The examples should provide the information that you need to enter the required parameters in the appropriate location in the Console.

Note: WebSphere applies the changes that are made on each tab only when you click **Apply** on the tab. Your changes will not be applied until you click **Apply.**

This section includes examples for:

➤ WebSphere 5.x and 6.0

➤ WebSphere 6.1

➤ Running the JRE Instrumenter for WebSphere IDE

WebSphere 5.x and 6.0

To configure a WebSphere 5.x and 6.0 application server:

1 Use your Web browser to access the WebSphere Application Server Administrative Console for the application server instance for which the probe was installed:

http://<App_Server_Host>:9090/admin

Replace <**App_Server_Host**> with the machine name for the application server host.

The Websphere Application Server Administrative Console opens.
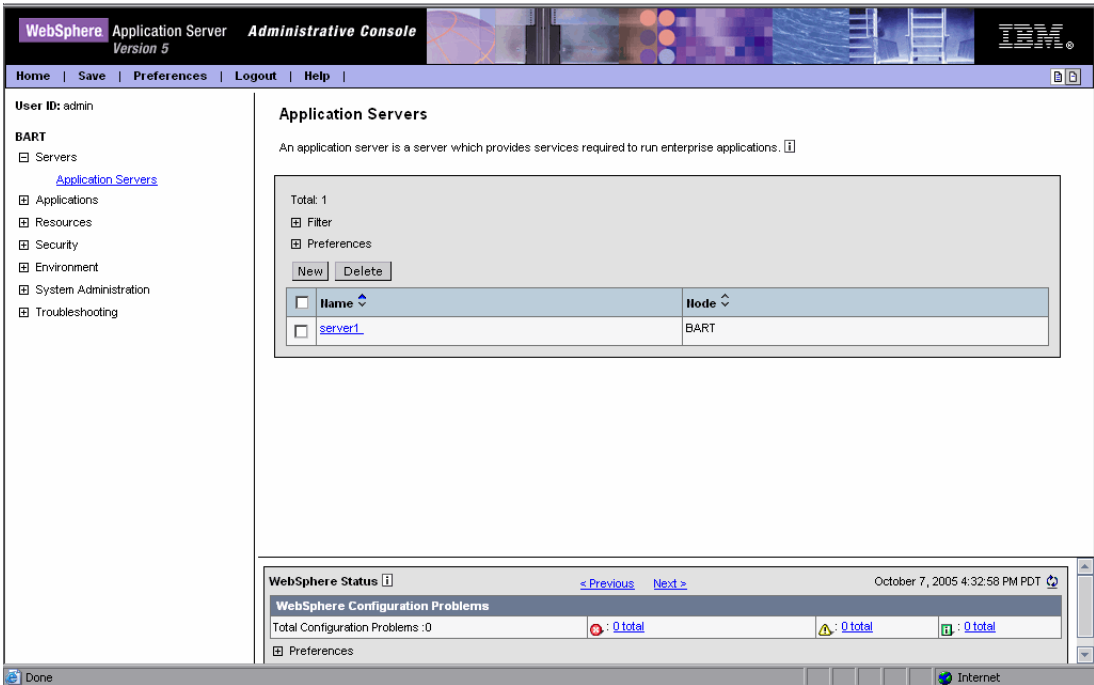
2 In the left panel, select **Servers** > **Application Servers**.

3 From the list of application servers in the right panel, select the name of the server that you want to configure so that it will be monitored by the probe.

The Configuration tab for the selected application server is displayed.
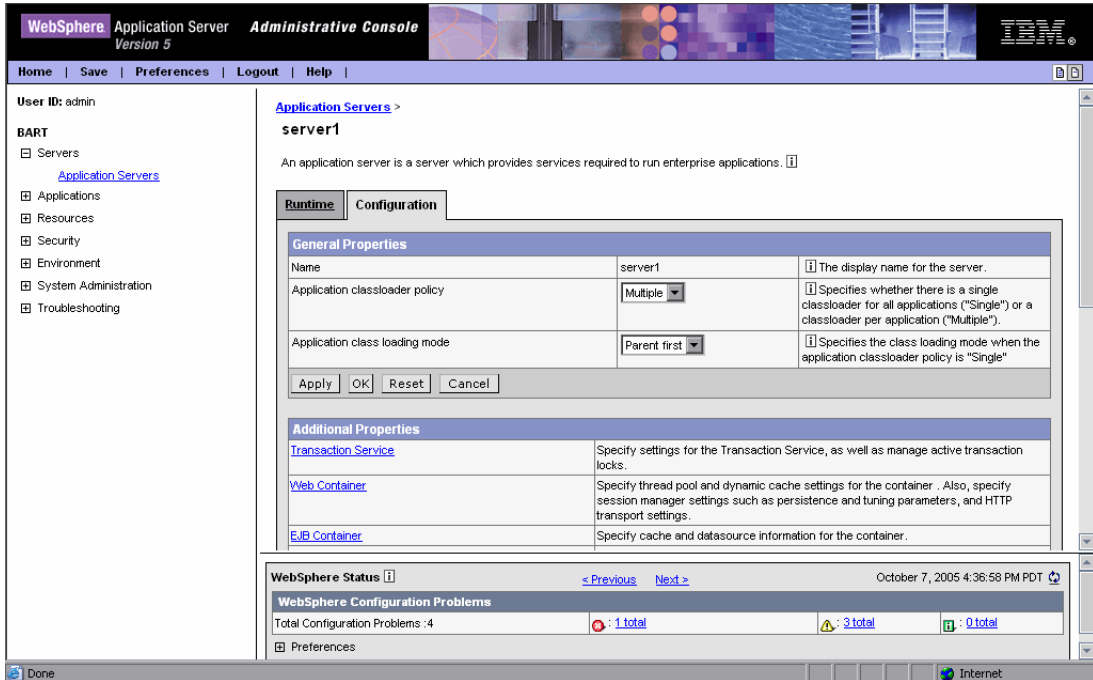
4 Scroll down in the Configuration tab, and in the **General Properties** column, look for the **Process Definition** property.
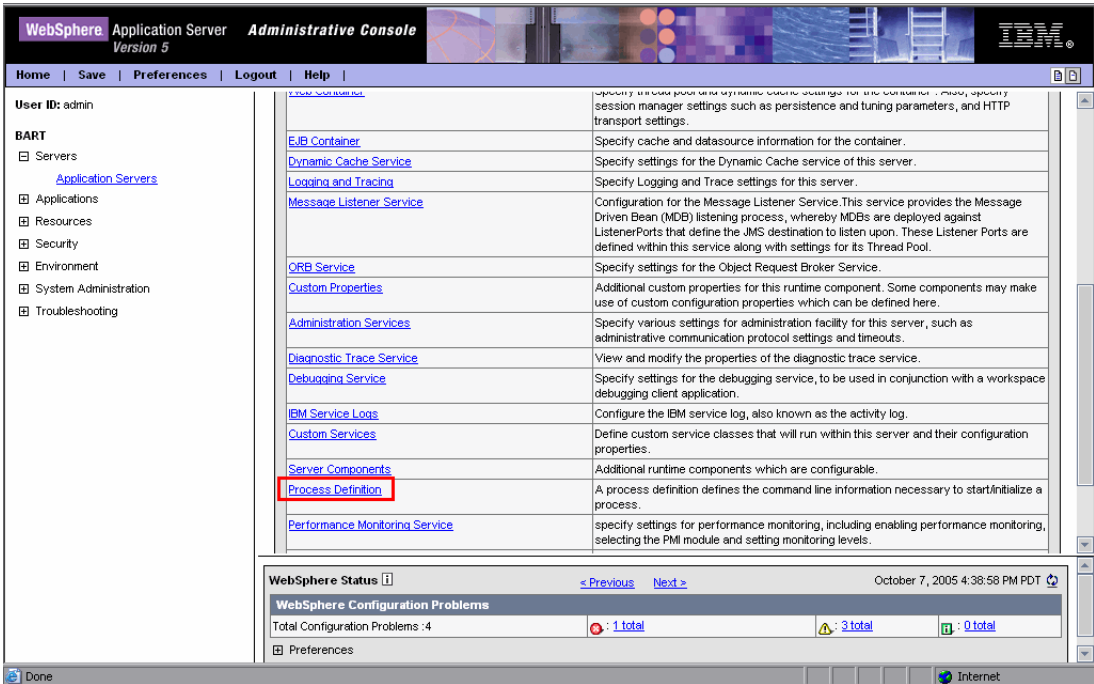
5 Click **Process Definition**.

6 Scroll down in the right panel, and look for **Java Virtual Machine**.
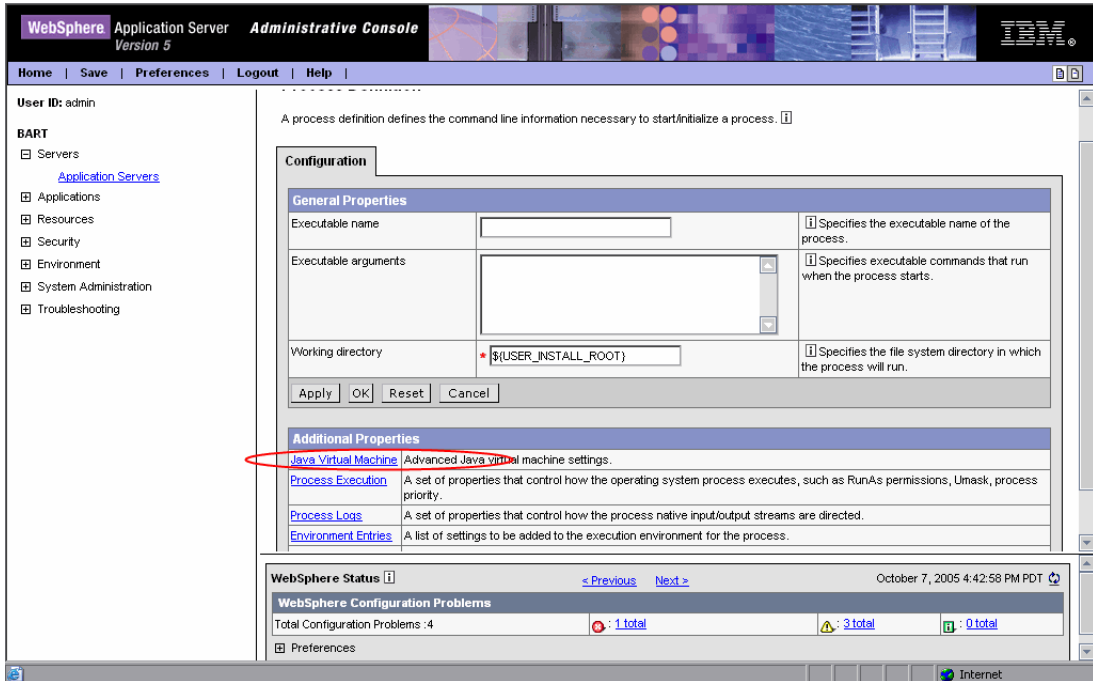
7 Click **Java Virtual Machine**.

8 The Configuration tab for the Java Virtual Machine is displayed.
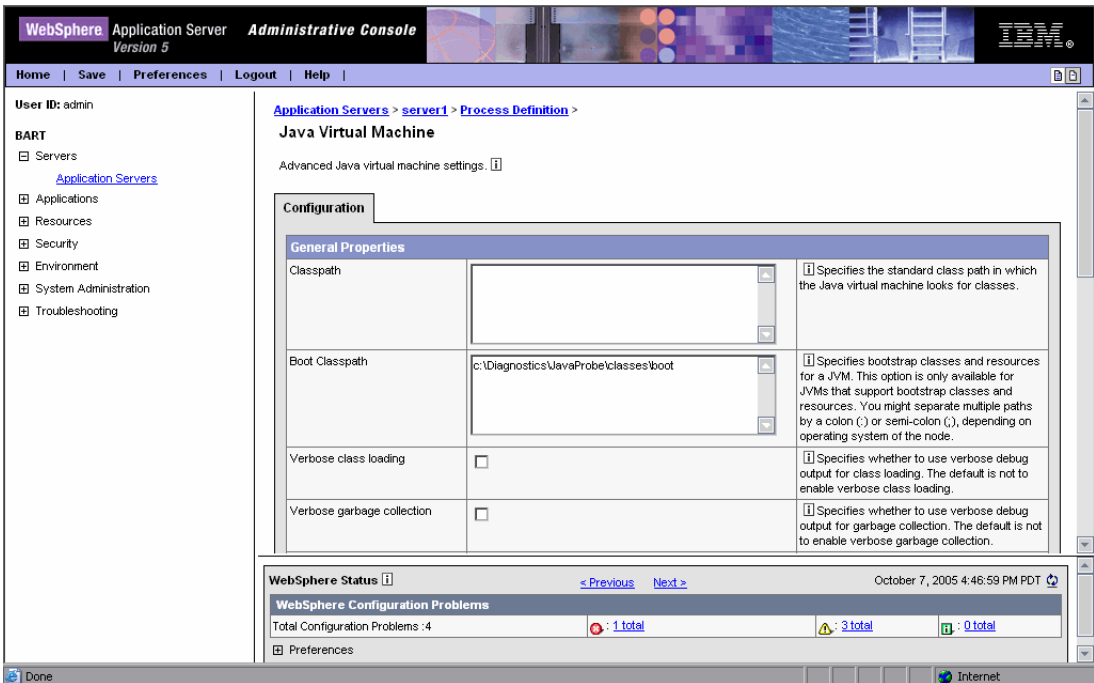
9 In the **Boot Classpath** box, type the path to the boot directory for the probe as follows:

```
<probe_install_dir>\classes\IBM\1.4.2_06;<probe_install_dir>\classes\boot
```

where **<probe_install_dir>** is the path to the location where the probe was installed.

10 Scroll to the bottom of the Configuration tab until the command buttons are visible.

Click **Apply**.

11 A message confirms that your changes have been applied.
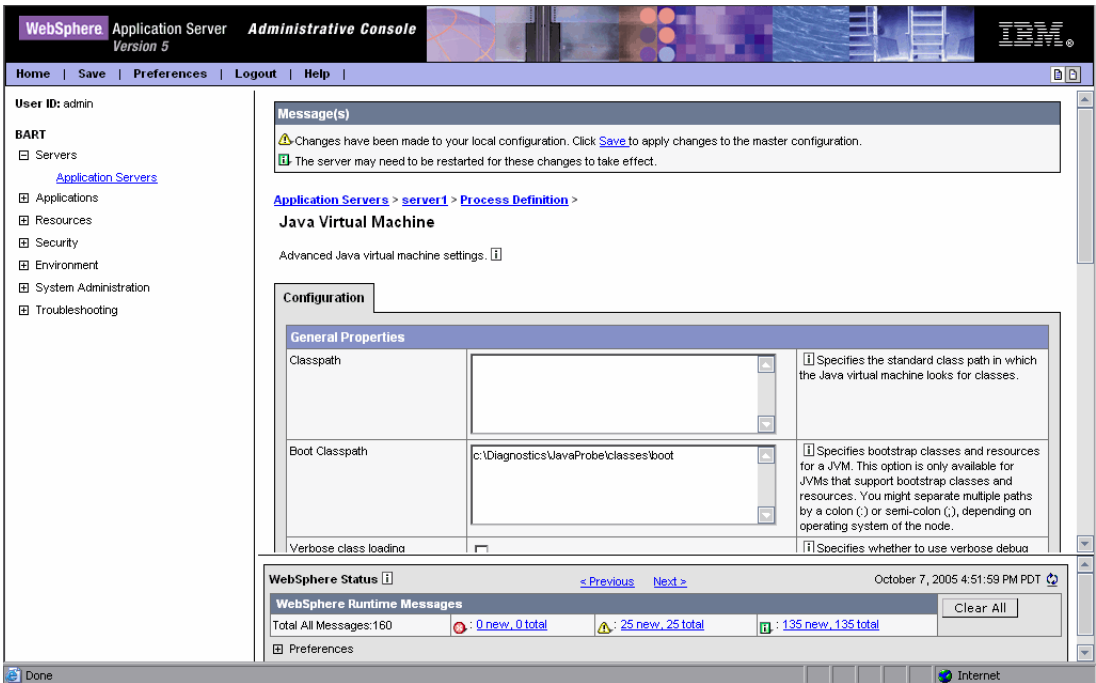
Click **Save** to apply the changes to the master configuration.

12 In the **Save to Master Configuration** area, click **Save**.
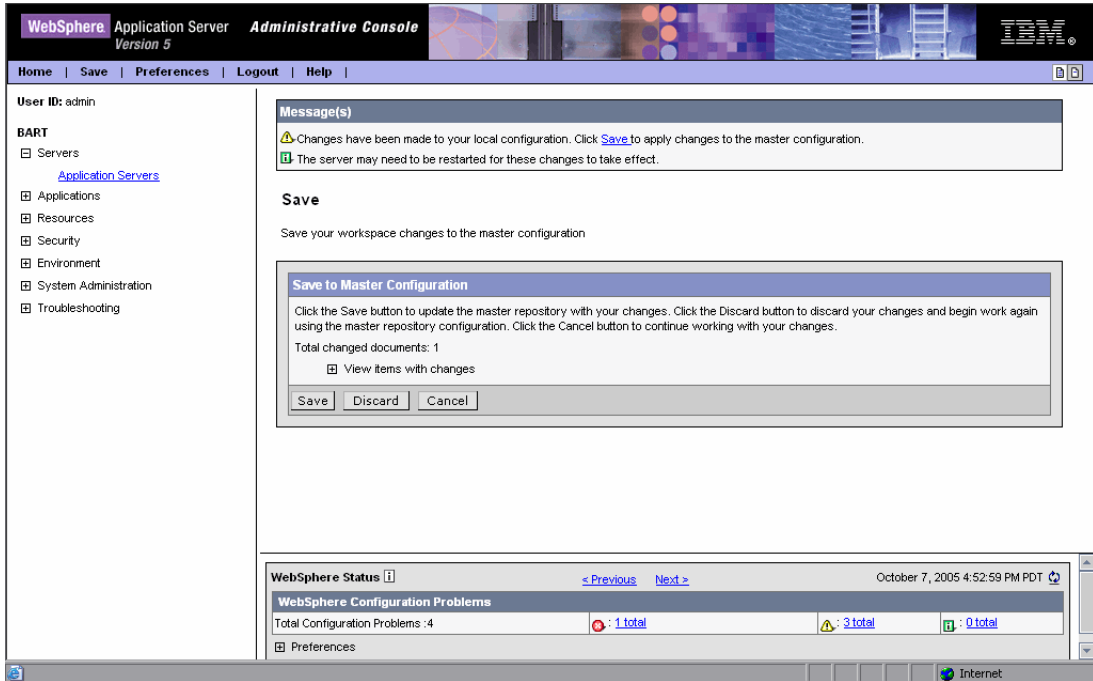
13 Restart the WebSphere application server. You do not need to restart the host for the application server.

14 To verify that the probe was configured correctly, check for entries in the **<probe_install_dir>\log\<probe_id>.log** file. If there are no entries in the file, this indicates that either you did not run the JRE Instrumenter or you did not enter the **Xbootclasspath** correctly. For details on running the JRE Instrumenter, see "Running the JRE Instrumenter" on page 141.

WebSphere 6.1

To configure a WebSphere 6.1 application server:

1 Open the Websphere Application Server Administrative Console.

2 Navigate to the Java Virtual Machine page as follows:

Application servers > **Application server instance name (e.g. server1)** > **Process Definition** > **Java Virtual Machine page**

3 On the Java Virtual Machine page, in the **Generic JVM Arguments** box, enter the following JVM parameter:

```
-javaagent:<probe_install_dir>\lib\probeagent.jar
```

where **<probe_install_dir>** is the path to the location where the probe was installed.

4 Apply and save your changes

5 Restart the WebSphere application server. You do not need to restart the host for the application server.

6 To verify that the probe was configured correctly, check for entries in the **<probe_install_dir>\log\<probe_id>\probe.log** file. If there are no entries in the file, it indicates that java probe was not started correctly.

Running the JRE Instrumenter for WebSphere IDE

If you are using WebSphere IDE, you must run the JRE Instrumenter manually in order to make sure that the correct JAVA executable for the WSAD IDE has been instrumented.

The WSAD IDE has 10 different java.exe executables to choose from. You must make sure to instrument the one that is used to run the IDE.

To instrument the correct java.exe:

1 Determine the version of WebSphere that you are using.

2 Determine the location of the appropriate java.exe. See the table below.

3 Run the JRE Instrumenter as described in "Running the JRE Instrumenter" on page 141.

Version	Executable
WAS 5.0	IDE INSTALL\runtimes\base_v5\java\bin\java.exe
WAS 5.1	IDE INSTALL\runtimes\base_v51\java\bin\java.exe

Notes:

➤ Before you can capture data, you must modify the **Xbootclasspath** for the application's JVM startup parameters per your IDE's instructions.

➤ If you are NOT running the application inside the IDE, follow the instructions below to configure the JVM parameters using the WebSphere Administrative Console.

Configuring WebLogic Application Servers

WebLogic application servers are configured by adding the Xbootclasspath property or JVM parameter to the script that is used to start the application server. WebLogic is started by running shell scripts in a UNIX environment, or command scripts in a Windows environment. Because the startup scripts that WebLogic provides are frequently customized by a site administrator, it is not possible to provide detailed configuration instructions that apply to all situations. Instead, the following sections provide instructions for each of the certified versions of the WebLogic application server for a generic implementation. Your site administrator should be able to use these instructions to show you how to make these changes in your customized environment.

Note: Make sure you understand the structure of the startup scripts, how the property values are set, and how to use environment variables before you make any configuration changes for the probe. Always create a backup copy of any file that you are going to update prior to making the changes.

This section includes examples for:

➤ WebLogic 8.1
➤ WebLogic 9.x and 10.0

WebLogic 8.1

To configure a WebLogic 8.1 application server for the Sun JVM:

1 Run the JRE Instrumentor and add the Sun JVM that WebLogic is using.

2 Once the JVM is added, click on the **Copy Parameter** button. This will copy the **Xbootclasspath** parameter into the clipboard. For example:

```
JAVA_OPTIONS="-Xbootclasspath/p:<probe_install_dir>
 \classes\Sun\1.4.2_04;<probe_install_dir>\classes\boot"
```

where **<probe_install_dir>** is the path to the directory where the probe was installed.

3 Locate the startup script used to start WebLogic for your domain. This file is typically located in a path similar to this example:

```
D:\bea\weblogic81\config\<Dom_Name>\start<Dom_Name>.cmd
```

Replace **<Dom_Name>** by the name of the script that starts the application.

For example, if your domain name is medrec, the path looks like this:

```
D:\bea\weblogic81\samples\domains\medrec\startMedRecServer.cmd
```

4 Create a backup copy of the startup script prior to making any changes to the script.

5 Use your editor to open the startup script.

6 Paste the **Xbootclasspath** parameter saved in the clipboard to the Java command line that starts the application server. The parameter must be placed at the beginning of the Java parameters following any JIT options, such as **-hotspot** or **-classic**.

Following is an example of a WebLogic startup script before adding the
Xbootclasspath parameter:

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m"-Xbootclasspath/
p:<probe_install_dir>\classes\Sun\1.3.1_04;<probe_install_dir>\classes\boot"
-classpath "%CLASSPATH%"
-Dweblogic.Domain=petstore -Dweblogic.Name=petstoreServer
-Dbea.home="C:\\bea"
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Dcloudscape.system.home=./samples/eval/cloudscape/data
-Djava.security.policy=="C:\bea\weblogic81/lib/weblogic.policy" weblogic.Server
```

Note: The startup script examples are shown with line breaks. The actual
scripts do not have line breaks and the text of the commands will wrap on
your screen as necessary.

Following is an example of a WebLogic startup script after adding the
Xbootclasspath parameter:

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m -Xbootclasspath/p:"C:\Program
Files\HP\common\JavaProbe\classes\boot"-classpath "%CLASSPATH%"
-Dweblogic.Domain=petstore -Dweblogic.Name=petstoreServer -Dbea.home="C:\\bea"
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Dcloudscape.system.home=./samples/eval/cloudscape/data
-Djava.security.policy=="C:\bea\weblogic81/lib/weblogic.policy" weblogic.Server
```

7 Save the changes to the startup script.

8 Restart the WebLogic application server. You do not need to restart the
application server host machine.

9 To verify that the probe was configured correctly, check for entries in the **<probe_install_dir>\log\<probe_id>\probe.log** file. If there are no entries in the file, this indicates that you did not run the JRE Instrumenter, or did not enter the **Xbootclasspath** correctly.

Note: For details on running the JRE Instrumenter, see "Running the JRE Instrumenter" on page 141.

To configure a WebLogic 8.1 application server for the JRockit JVM:

1 Run the JRE Instrumentor and add the JRockit JVM that WebLogic is using.

2 Once the JVM is added, click on the **Copy Parameter** button. This will copy the **Xbootclasspath** parameter into the clipboard.

Following is an example of the **Xbootclasspath** parameter:

```
-Xbootclasspath/p:<probe_install_dir>\classes\boot
```

where **<probe_install_dir>** is the path to the directory where the probe was installed.

3 Locate the command file that invokes the WebLogic application server (for example, startWLS.cmd). This file is typically located in a path similar to the following example:

```
C:\bea\weblogic81\server\bin\ startWLS.cmd
```

4 Create a backup copy of the command file prior to making any changes to the script. You may want to give the new copy a name such as startWLSWithJRockit.cmd, and use this as the new version of the command file that will be manipulated in the following steps.

5 Use your editor to open the startup script.

6 Set the JAVA executable invoked by WebLogic to JRockit.

a Locate the line in the command file where the value of the **JAVA_VENDOR** parameter is set.

b Change the value of the **JAVA_VENDOR** variable to point to the JRockit folder as follows:

```
set JAVA_VENDOR=<BEA_HOME_DIR>\jrockit
```

For example:

```
set JAVA_VENDOR=BEA
```

7 Modify the Java command line that starts the application server.

a Locate the line in the command file which begins as follows:

```
%JAVA_HOME%\bin\java %JAVA_VM% %JAVA_OPTIONS% .....
```

b Indicate the JRockit management URL by specifying the **Xmanagement:class** parameter immediately following the %JAVA_OPTIONS% variable.

The following is an example of the **Xmanagement:class** parameter:

```
-Xmanagement:class=com.mercury.opal.capture.proxy.JRockitManagement
```

c Allow the probe to hook into the application server process by adding the **Xbootclasspath** parameter you saved in the clipboard to immediately following the %JAVA_OPTIONS% variable.

The following is an example of a WebLogic startup script before adding the **Xmanagement:class** and **Xbootclasspath** parameters:

```
"%JAVA_HOME%\bin\java" %JAVA_VM% %MEM_ARGS% %JAVA_OPTIONS%
-Dweblogic.Name=%SERVER_NAME%
-Dweblogic.management.username=%WLS_USER%
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.management.server=%ADMIN_URL%
-Dweblogic.ProductionModeEnabled=%PRODUCTION_MODE%
-Djava.security.policy="%WL_HOME%\server\lib\weblogic.policy" weblogic.Server
```

Note: The startup script examples are shown with line breaks. The actual scripts do not have line breaks and the text of the commands will wrap on your screen as necessary.

The following is an example of a WebLogic startup script after adding the **Xbootclasspath** parameter:

```
"%JAVA_HOME%\bin\java" %JAVA_VM% %MEM_ARGS% %JAVA_OPTIONS%
-Xmanagement:class=com.mercury.opal.capture.proxy.JRockitManagement
-Xbootclasspath/p:"C:\Program Files\HP\common\JavaProbe\classes\boot"
-Dweblogic.Name=%SERVER_NAME%
-Dweblogic.management.username=%WLS_USER%
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.management.server=%ADMIN_URL%
-Dweblogic.ProductionModeEnabled=%PRODUCTION_MODE%
-Djava.security.policy="%WL_HOME%\server\lib\weblogic.policy" weblogic.Server
```

8 Save the changes to the command file.

9 Restart the WebLogic application server (not the computer; just the application server).

10 To verify that the probe was configured correctly, check for entries in the **<probe_install_dir>\log\<probe_id>\probe.log** file. If there are no entries in the file, this indicates that you did not run the JRE Instrumenter or did not enter the **Xbootclasspath** parameter correctly.

Note: For details on running the JRE Instrumenter, see "Running the JRE Instrumenter" on page 141.

WebLogic 9.x and 10.0

To configure a WebLogic 9.0 and 10.0 application server:

 1 Run the JRE Instrumentor and add the JVM that WebLogic 9.x or 10.0 is using.

 2 Once the JVM is added, click on the **Copy Parameter** button. This will copy the JVM parameter into the clipboard. For example:

> JAVA_OPTIONS="-javaagent:<probe_install_dir>\lib\probeagent.jar"

where **<probe_install_dir>** is the path to the directory where the probe was installed.

 3 Locate the startup script used to start WebLogic for your domain. For example, if your domain name is Medrec, the path looks like this:

> D:\bea\wlserver_10.0\samples\domains\medrec\bin\startWebLogic.cmd

 4 Create a backup copy of the startup script prior to making any changes to the script.

 5 Use your editor to open the startup script.

 6 Paste the **JVM** parameter saved in the clipboard to the Java command line that starts the application server. The parameter must be placed at the beginning of the Java parameters following any JIT options, such as **-hotspot** or **-classic**.

Note: The startup script examples are shown with line breaks. The actual scripts do not have line breaks and the text of the commands will wrap on your screen as necessary.

Following is an example of a WebLogic startup script after adding the **JVM** parameter:

> set JAVA_OPTIONS= "-javaagent:C:\MercuryDiagnostics\JAVAProbe\lib\probeagent.jar"
> %SAVE_JAVA_OPTIONS%

7 Save the changes to the startup script.

8 Restart the WebLogic application server. You do not need to restart the application server host machine.

9 To verify that the probe was configured correctly, check for entries in the **<probe_install_dir>\log\<probe_id>\probe.log** file. If there are no entries in the file, this indicates that you did not run the JRE Instrumenter, or did not enter the JVM parameter correctly.

Note: For details on running the JRE Instrumenter, see "Running the JRE Instrumenter" on page 141.

Configuring Oracle Application Servers

This section provides instructions for configuring the following Oracle application servers:

➤ "Configuring Oracle 9i" on page 172

➤ "Configuring Oracle 10.1.3" on page 176

➤ "Configuring Oracle 10.1.2" on page 177

Configuring Oracle 9i

Oracle9i application servers are configured by adding the Xbootclasspath property to the XML file used to start the application server. Because the files that Oracle9i provides are frequently customized by the site administrator, it is not possible to provide detailed configuration instructions that will apply exactly for each situation. Instead, the following sections provide instructions for configuring an Oracle9i application server for a generic implementation. Your site administrator should be able to use these instructions to guide you through making these changes in your customized environment.

Note: Make sure that you understand the structure of the startup scripts, how the property values are set, and the use of environment variables before you make any configuration changes for the probe. Always create a backup copy of any file that you are going to update, prior to making the changes.

To configure an Oracle9i application server:

1 Locate the XML file that is used to control the configuration of the application server when the server is started. The file is typically located at **<Oracle 9iAS_Install_Dir>/opmn/conf/opmn.xml**.

2 Create a backup copy of the **opmn.xml** file prior to making any changes.

3 Open the **opmn.xml** file to be edited using your editor.

4 Add the **Xbootclasspath** property. The property must be added to the "java-option value."

The following is an example of the **Xbootclasspath** parameter:

```
-Xbootclasspath/p:<probe_install_dir>\classes\Sun\1.4.2_04;
 <probe_install_dir>\classes\boot
```

where **<probe_install_dir>** is the path to the directory where the probe was installed.

Note: When modifying the -**Xbootclasspath** parameter. Be sure to use quotes if there are spaces in the path that you specify.

The following image is an example of an Oracle 9iAS startup script before adding the **Xbootclasspath** parameter:

The following image is an example of an Oracle 9iAS startup script after adding the **Xbootclasspath** parameter:

5 Save the changes to the XML file.

6 Restart the Oracle application server. You do not need to reboot the host for the application server.

7 To verify that the probe was configured correctly, check for entries in the **<probe_install_dir>\log\<probe_id>\probe.log** file. If there are no entries in the file, this indicates that you did not run the JRE Instrumenter or did not enter the **Xbootclasspath** parameter correctly. For details on running the JRE Instrumenter, see "Running the JRE Instrumenter" on page 141.

Configuring Oracle 10.1.3

This section provides instructions for configuring an Oracle 10.1.3 application server.

To configure an Oracle 10.1.3 application server:

1 Locate the XML file that is used to control the configuration of the application server when the server is started. The file is typically located at **<Oracle_Install_Dir>/opmn/conf/opmn.xml**.

2 Create a backup copy of the **opmn.xml** file prior to making any changes.

3 Open the **opmn.xml** file to be edited using your editor.

4 Add the following parameter to the "java-option value."

```
-javaagent:<probe_install_dir>\lib\probeagent.jar
```

where **<probe_install_dir>** is the path to the directory where the probe was installed.

5 Save the changes to the XML file.

6 Restart the Oracle application server. You do not need to reboot the host for the application server.

7 To verify that the probe was configured correctly, check for entries in the **<probe_install_dir>\log\<probe_id>\probe.log** file. If there are no entries in the file, it indicates that java probe was not started correctly.

Configuring Oracle 10.1.2

This section provides instructions for configuring an Oracle 10.1.2 application server.

To configure an Oracle 10.1.2 application server:

1 Open Oracle's **Application Server Control Console**

2 Double-click on the **home** System Component.

3 On the **OC4J: home** page select **Administration**.

4 On the Administration page, select **Server Properties**.

5 In the Server Properties window, under **Command Line Options,** add the Xbootclasspath property to the **Java Options** box.

Note: In Oracle 10.1.2 it is required to add a (^) prior to the switch. Otherwise Oracle will change the (/) switch option to a (\).

The following is an example of the Xbootclasspath parameter with the (^) inserted.

```
-Xbootclasspath^/p:<probe_install_dir>/classes/boot
```

where **<probe_install_dir>** is the path to the directory where the probe was installed.

6 Apply the changes and restart the Oracle server.

Configuring the JBoss Application Server

This section explains how to configure the JBoss application server.

Important: The instructions for configuring JBoss 4.0.5 are different from the other versions. For more information, see "JBoss Version 4.0.5 and Later" on page 183.

You configure JBoss application servers by adding the Xbootclasspath property or JVM parameter to the script that is used to start the application server. JBoss is started by running shell scripts in a UNIX environment, or command scripts in a Windows environment. Because the startup scripts that JBoss provides are frequently customized by the site administrator, it is not possible to provide detailed configuration instructions that will apply exactly for each situation. Instead, the following sections provide instructions for each of the certified versions of the JBoss application server for a generic implementation. Your site administrator should be able to use these instructions to guide you to make these changes in your customized environment.

Note: Make sure that you understand the structure of the startup scripts, how the property values are set, and the use of environment variables before you make any configuration changes for the probe. Always create a backup copy of any file that you are going to update prior to making the changes.

This section provides instructions for configuring the following JBoss application servers:

➤ "JBoss Versions Earlier Than 4.0.5" on page 182
➤ "JBoss Version 4.0.5 and Later" on page 183

JBoss Versions Earlier Than 4.0.5

To configure a JBoss application server for versions earlier than 4.0.5:

1 Locate the startup script that is used to start JBoss for the application. This file is typically located in path similar to the following example:

```
D:\JBoss\bin\run.bat
```

Note: For UNIX the file extension is ".sh".

2 Create a backup copy of the startup script prior to making any changes to the script.

3 Open the startup script to be edited using your editor.

4 Add the **Xbootclasspath** parameter to the Java command line that starts the application server. The parameter must be placed at the beginning of the Java parameters following any JIT options, such as **-hotspot** or **-classic.**

The following is an example of the **Xbootclasspath** parameter:

```
-Xbootclasspath/p:<probe_install_dir>\classes\Sun\1.4.2_04;
 <probe_install_dir>\classes\boot
```

where **<probe_install_dir>** is the path to the directory where the probe was installed.

Note: When modifying the **-Xbootclasspath** parameter. Be sure to use quotes if there are spaces in the path that you specify.

The following is an example of a JBoss startup script before adding the **Xbootclasspath** parameter:

```
"%JAVA%" %JAVA_OPTS% -classpath "%CLASSPATH%" org.jboss.Main %ARGS%
```

The following is an example of a JBoss startup script after adding the
Xbootclasspath parameter:

```
"%JAVA%" "-Xbootclasspath/p:<probe_install_dir>\classes\Sun\1.4.2_04;
 <probe_install_dir>\classes\boot" %JAVA_OPTS%
-classpath "%CLASSPATH%" org.jboss.Main %ARGS
```

Note: The startup script examples are shown with line breaks. The actual
scripts do not have line breaks and the text of the commands will wrap on
your screen as necessary.

5 Save the changes to the startup script.

6 Restart the JBoss application server with the probe by running the modified
script. You do not need to restart the application server host.

7 To verify that the probe was configured correctly, check for entries in the
<probe_install_dir>/log/<probe_id>/probe.log file. If there are no entries in
the file, this indicates that you did not run the JRE Instrumenter or did not
enter the **Xbootclasspath** parameter correctly. For details on running the JRE
Instrumenter, see "Running the JRE Instrumenter" on page 141.

JBoss Version 4.0.5 and Later

To configure a JBoss application server version 4.0.5 and later:

1 Locate the startup script that is used to start JBoss for the application. This
file is typically located in path similar to the following example:

```
D:\JBoss\bin\run.bat
```

Note: For UNIX the file extension is ".sh".

2 Create a backup copy of the startup script prior to making any changes to
the script.

3 Open the startup script to be edited using your editor.

4 Add the following JVM parameter:

```
-javaagent:<probe_install_dir>\lib\probeagent.jar
```

where **<probe_install_dir>** is the path to the directory where the probe was installed.

5 Save the changes to the startup script.

6 Restart the JBoss application server with the probe by running the modified script. You do not need to restart the application server host.

7 To verify that the probe was configured correctly, check for entries in the **<probe_install_dir>\log\<probe_id>\probe.log** file. If there are no entries in the file, it indicates that java probe was not started correctly.

Note: When you add the **-agentpath:<probe_install_dir>\lib\x86-windows\jvmti.dll** for the heap dump, you need to start Jboss with following option: **run.bat -Djboss.platform.mbeanserver**

Otherwise, you will receive the following error: Failed to locate MBeanServer via ManagementFactory." for jBoss 4 (InvocationTargetException).

Configuring the SAP NetWeaver Application Server

The following instructions describe how to configure the SAP NetWeaver application server so that the applications can be monitored by the probe.

Configuring the NetWeaver application server means instrumenting the JVM and adding the Xbootclasspath property to the script that is used to start the application server. The following sections provide instructions for a generic NetWeaver implementation. Your site administrator should be able to use these instructions to guide you in making the changes that are appropriate to your specific environment.

Note: Make sure that you understand the structure of the startup scripts, how the property values are set, and the use of environment variables before you make any changes to the configuration of the application server for the probe. You should always create a backup of files before making any changes.

To configure a SAP NetWeaver application server:

1 Add the JVM that runs the NetWeaver application server.

2 Instrument the JVM. The JRE Instrumenter provides the **Xbootclasspath** parameter. This parameter must be added to the NetWeaver Configtool's JVM parameters window as described in the next step.

3 Run the NetWeaver application server configuration tool. The configuration tool is called **configtool.bat** and it is located in the **usr\sap\j2e\jc00\j2ee\configtool** directory.

4 Add the -**Xbootclass** parameter into the Java parameters text window. This window is in the General tab when you select your server instance. For example, cluster-data | instance_ID70323 | server_ID7032350.

5 Save your changes and exit the configuration tool.

6 Assign the following values to these properties in the **etc/capture.properties** file:

> **minimum.buffer.size** = 250000
>
> **initial.private.buffer.count** = 50
>
> **maximum.private.buffer.count** = 200
>
> **gentle.reserve.buffer.count** = 50
>
> **hard.reserve.buffer.count** = 50

7 Restart the NetWeaver application server.

8 To verify that the probe was configured correctly, check for entries in the **<probe_install_dir>/log/<probe_id>/probe.log** file. If there are no entries in the file, this indicates that you did not run the JRE Instrumenter or did not enter the **Xbootclasspath** parameter correctly.

Configuring a Generic Application Server

Note: You should only use the instructions for a generic application server when you do not find configuration instructions for your specific application server in this document.

Your site administrator may configure the application server using an alternative, site-specific method. In this case, the generic procedure may be sufficient for the administrator to understand what changes must be made.

Important: Before making any changes, back up all startup scripts.

To update the application server configuration:

1 Locate the application server startup script or the file where the JVM parameters are set.

2 Create a backup copy of the application server startup script before you make any changes to the script.

3 Use an editor or the application server console to open the startup script.

4 Add the **Xbootclasspath** parameter to the Java command line that starts the application server, using the following syntax:

```
-Xbootclasspath/p:<probe_install_dir>\classes\Sun\1.4.2_04;
 <probe_install_dir>\classes\boot
```

where **<probe_install_dir>** is the path to the directory where the probe was installed.

This connects the probe to the application. The parameter must be placed at the beginning of the Java parameters, following any JIT options such as **-hotspot** or **-classic**.

Following is an example of a WebLogic Java command line in a startup script before adding the **Xbootclasspath** parameter:

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m -classpath "%CLASSPATH%"
-Dweblogic.Domain=petstore -Dweblogic.Name=petstoreServer -Dbea.home="C:\\bea"
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Dcloudscape.system.home=./samples/eval/cloudscape/data
-Djava.security.policy=="C:\bea\wlserver6.1/lib/weblogic.policy" weblogic.Server
```

Note: The startup script examples are shown with line breaks. The actual scripts do not have line breaks and the text of the commands will wrap on your screen as necessary.

The following is an example of a WebLogic Java command line in a startup script after adding the **Xbootclasspath** parameter:

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m
"-Xbootclasspath/p:<probe_install_dir>\classes\Sun\1.4.2_04;
<probe_install_dir>\classes\boot"
-classpath "%CLASSPATH%"
-Dweblogic.Domain=petstore -Dweblogic.Name=petstoreServer
-Dbea.home="C:\\bea" -Dweblogic.management.password=%WLS_PW%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Dcloudscape.system.home=./samples/eval/cloudscape/data
-Djava.security.policy=="C:\bea\wlserver6.1/lib/weblogic.policy" weblogic.Server
```

 5 Save the changes to the startup script.

 6 Restart the application server under test. You do not need to restart the application server host machine.

 7 To verify that the probe was configured correctly, check for entries in the **<probe_install_dir>/log/<probe_id>/probe.log** file. If there are no entries in the file, this indicates that you did not run the JRE Instrumenter or did not enter the **Xbootclasspath** parameter correctly.

Note: Alternatively, you can configure the JVM process definitions of the application server by going into the Administrative Console. However, this does not apply to WebLogic servers.

For details on running the JRE Instrumenter, see "Running the JRE Instrumenter" on page 141.

Adjusting the Heap Size for the Java Probe in the Application Startup Script

The size of the heap can impact the performance of the Java Probe.

The default value for the heap size is 64 MB. The heap size is set in the application's start up script using the following VM argument:

```
-Xmx<size>
```

You can increase the heap size by updating the value specified in the **-Xmx<size>** option. Refer to your JVM documentation for more help on setting this parameter.

Configuring the SOAP Message Handler

The Diagnostics SOAP message handler is required for probes to support the following features:

➤ Collect payload for SOAP faults.

➤ Determine SOA consumer ID from SOAP header, body, or envelope.

For most application servers, the instrumentation points and code snippets have been written to automatically configure the Diagnostics handlers for web services being monitored.

Important: For some application servers, special instrumentation is provided in Diagnostics to automatically load the Diagnostics SOAP message handler.

However, some manual configuration is required for WebSphere 5.1 JAX-RPC and Oracle 10g JAX-RPC, see "Configuring the SOAP Message Handler" on page 189 for details.

In addition, the Diagnostics SOAP message handler is not available for all application servers nor is custom instrumentation available to capture SOAP faults or consumer IDs from SOAP payloads, and therefore this feature is not available on all versions of all application servers. For the most recent information on Diagnostics SOAP message handler support refer to the Diagnostics Product Availability Matrix at http://support.openview.hp.com/sc/support_matrices.jsp.

Disabling the SOAP Message Handler

1 By default, the SOAP message handler is enabled. You can disable the handler as follows:

```
<probe_install_dir>\etc\inst.properties
...
\mercury.enable.autoLoadSOAPHandler = false
```

If the SOAP message handler is disabled you will need to manually configure where in the chain the handler gets installed.

Loading the Diagnostics SOAP Message Handler

The SOAP message handler is loaded automatically on most application servers but requires manual configuration on these application servers:

WebSphere 5.1 JAX-RPC

To configure the SOAP message handler on WebSphere 5.1, follow these steps.

Note: For WebSphere 6.1 JAX-WS web services, Diagnostics handlers are not supported. You must recompile their application with the Diagnostics SOAP handler classes.

---

1 Locate the Web service deployment descriptor (webservices.xml) for the application. The directory path should like something like:

<install_root>\config\cell\<Server>\applications\
<WebServiceEAR>\deployments\<WebServiceName>\
<WebServiceJAR|WARName>\WEB-INF

For example:

```
C:\Program Files\WebSphere\AppServer\config\
cells\MyServer1\application\WebServicesSamples.ear\
deployments\WebServicesSamplea\AddressBookJ2WB.war\ WEB-INF
```

2 Edit the webservices.xml and add the Diagnostics handler for each <port-component>:

```
<port-component>
 ......
 <handler>
  <handler-name>Diagnostics RPC Handler</handler-name>
   <handler-class>
    com.mercury.opal.javaprobe.handler.soap.ProbeRPCHandler
   </handler-class>
 </handler>
 ......
</port-component>
```

3 Copy the Diagnostics handler jar (<probe_install_dir>\lib\probeSOAPHandler.jar) to the WebSphere "lib" directory.

For example:

```
cp C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\ lib\probeSOAPHandler.jar
C:\Program Files\WebSphere\AppServer\lib
```

These steps were developed with IBM WebSphere 5.1.0 Application Server on Windows.

Oracle 10g JAX-RPC

To configure the SOAP message handler on Oracle 10g, follow these steps.

1 Locate the Web service deployment descriptor (webservices.xml) for the application. The directory path should like something like:

<OC4J install root>\j2ee\home\applications\<app name>\ <deployment name>\WEB-INF\webservices.xml

2 Edit the webservices.xml and add the Diagnostics handler for each <port-component>:

```
<port-component>
......
 <handler>
  <handler-name>Diagnostics RPC Handler</handler-name>
  <handler-class>
   com.mercury.opal.javaprobe.handler.soap.ProbeRPCHandler
  </handler-class>
 </handler>
......
</port-component>
```

3 Copy the Diagnostics handler jar (<probe install dir>\lib\probeSOAPHandler.jar) to the <OC4J install root>\j2ee\home\applib directory.

These steps were developed with Oracle Containers for J2EE (OC4J) 10g Release 3 (10.1.3.3) on Windows.

8

Installing .NET Agents (Probes)

The .NET Agent combines the capabilities of the Diagnostics .NET probe and the TransactionVision .NET sensor into a single component. The .NET Agent can simultaneously serve as both the TransactionVision sensor and the Diagnostics probe on a .NET host.

The .NET Agent provides a low-overhead capture solution that works with HP Software's Business Availability Center applications. The .NET Agent captures events from a .NET application and sends the event metrics to the TransactionVision Analyzer or to the Diagnostics Server, or both.

This chapter includes:

➤ About the .NET Agent Installer on page 194

➤ Installing the .NET Agent on page 198

➤ Verifying that the .NET Agent Is Connected on page 215

➤ About Custom Configuration and Instrumentation for the .NET Agent on page 216

➤ Enabling and Disabling Standard Instrumentation for Applications on page 216

➤ Restarting IIS on page 218

➤ Verifying the .NET Agent Installation on page 219

➤ Determining the Version of the .NET Agent on page 221

➤ Uninstalling the .NET Agent on page 221

➤ Enabling and Disabling the Diagnostics .NET Agent on page 221

➤ Disabling Logging on page 222

About the .NET Agent Installer

During the .NET Agent installation the following setup and configuration is done for you:

➤ Discovery of ASP.NET applications. The installer attempts to automatically detect the ASP.NET applications on the system where the agent is installed. See "Discovering ASP.NET Applications" on page 196.

➤ Default agent configuration.

➤ The installer configures the agent to capture basic ASP.NET/ADO/WCF workload for each of the ASP.NET application detected. The agent configuration is controlled using the **probe_config.xml** file. See ".NET Agent Automatic Configuration for Discovered ASP.NET Applications" on page 197.

➤ Default **Asp.Net.points**, **Ado.points** and **WCF.points** files are installed and enabled providing standard instrumenation to allow you to start monitoring ASP.NET applications. See "Preparing for the Installation" on page 198 for .NET and .NET WCF requirements and limitations. The points files control the workload that the agent will capture for the application. (To generate .NET Remoting Events you will also need Remoting.points and will have to setup the application for instrumentation.)

➤ Separate instrumentation points file for each detected application. Instrumentation points files are created for each IIS installed ASP.NET application domain detected (<applicationDomin>.points files). The probe_config.xml file contains an appdomain reference for each of the detected ASP.NET applications. Each of these appdomain references in the probe_config.xml file contain an instrumentation points file reference. The .NET Agent uses this runtime instrumentation to capture method latency information from specified applications.

➤ Additional instrumentation points files are created but not enabled (see Chapter 10, "Custom Instrumentation for .NET Applications"). The installer creates the following default files:

Asp.Net.IExecutionStep.points, IIS.points, lwmd.points, msmq.points, webservices.points (not used for WCF web services, use WCF.points file instead). You can enable the above points files by adding a reference to them in the **probe_config.xml** file (see "Enabling and Disabling Standard Instrumentation for Applications" on page 216 for more information on how to enable this instrumentation).

> ➤ A default set of .NET application layers is configured. See Chapter 12, "Instrumentation Layers" for more information.

➤ Optional custom instrumentation. Customize the instrumentation points files to capture application specific methods (see Chapter 10, "Custom Instrumentation for .NET Applications").

➤ Optional configuration. Modify the agent configuration in the probe_config.xml file (see Chapter 16, "Understanding the .NET Probe Configuration File" and Chapter 17, "Advanced .NET Probe Configuration").

➤ After installing the .NET agent and modifying the configuration or creating custom instrumentation, as needed, restart IIS before using the .NET Agent with ASP.NET applications. See "Restarting IIS" on page 218.

➤ After you restart IIS and a URL in the application has been accessed at least once, the .NET Agent can begin to collect performance data using the default instrumentation points and configuration.

Discovering ASP.NET Applications

The .NET Agent installer automatically discovers the ASP.NET applications that you may want to instrument. After you have installed the .NET Agent, you can request that the agent re-scan your IIS configuration to catch any additions or changes.

Discovering ASP.NET Applications During Installation

The .NET Agent installer detects ASP.NET applications on the machine when the agent is installed. The .NET Agent installer discovers applications by inspecting the IIS configuration and looking for virtual directory entries that might refer to ASP.NET applications.

In some instances, the ASP.NET applications are installed in a manner that prevents them from being detected. An example of a situation that could cause the installer to fail to detect an ASP.NET application is when an ASP.NET application has been installed as a Web directory instead of a virtual directory.

Discovering ASP.NET Applications After Installation

You can request a re-scan of the IIS configuration when you have modified an existing ASP.NET application deployment or installed new ASP.NET applications.

To request that the agent re-scan the IIS configuration and update the **probe_config.xml** file, select **Start** > **HP Diagnostics .NET Probe** > **Rescan ASP.NET Applications**.

.NET Agent Automatic Configuration for Discovered ASP.NET Applications

The .NET Agent installer configures the agent to capture basic ASP.NET/ADO/WCF workload for each of the ASP.NET applications detected. The agent performs the following configuration steps:

➤ Creates an application-specific capture points file template.

The capture points file defines the instrumentation that controls the workload that the agent captures for each application. You can modify the instrumentation in the capture points file to provide instructions that allow the agent to capture performance data for application-specific custom methods.

➤ Creates an **appdomain** tag in the **probe_config.xml** file which is located in the **<probe_install_dir>/etc** directory. The attributes of the **appdomain** tag direct the behavior of the .NET Agent (points and enabled attributes). See Chapter 16, "Understanding the .NET Probe Configuration File" for details.

Notes:

Diagnostics enables the instrumentation for all discovered applications by setting the **enablealldomains** attribute in the **process** tag to true which overrides the appdomain tag's enabled attribute. For information on enabling and disabling instrumentation for applications see "Disabling Logging" on page 222.

Non ASP.NET Applications

The .NET Agent installation will automatically discover your ASP.NET applications and create setting for the applications in the probe_config.xml and create template points file for them. For each non-ASP.NET application, e.g. NT Service, console application, UI client, you will need to create the appropriate settings in the probe_config.xml settings to configure the .NET Agent to monitor your applications as well as create points files indicating which points in your application you wish to monitor.

Example of probe_config.xml setting for an application called SimpleConsoleHost.exe:

```
<process name=="SimpleConsoleHost.points">
```

```
        <points file="SimpleConsoleHost.points"/>
        <logging level=" "/>
</process>
```

Example of points file setting for an application called
SimpleConsoleHost.exe:

```
[SimpleConsoleHost]
class = MyNamespace.SimpleConsoleHost
method = !.*
ignoreMethod = Main
layer = SimpleConsoleHost
```

See Chapter 10, "Custom Instrumentation for .NET Applications" for more
details.

Installing the .NET Agent

The following section provides detailed instructions for installing the .NET
Agent.

This section includes:

➤ "Preparing for the Installation" on page 198

➤ "Launching the .NET Agent Installer" on page 200

➤ "Running the Installation" on page 201

➤ "Installing the Agent as a Profiler Only" on page 205

➤ "Installing the Agent to Work with a Diagnostics Server" on page 207

Preparing for the Installation

You install the .NET Agent on the host machine of the application that you
want to monitor. The default configuration of the .NET Agent has been set
to monitor the application effectively, with very little impact on the
performance of the application.

For instructions on advanced .NET Agent (Probe) configuration, see Chapter 17, "Advanced .NET Probe Configuration."

Important: .NET Framework 1.1 or later needs to be installed on your machine before you run the .NET Agent installation.

Also, the .NET Agent includes a SOAP Extension Handler. Installing the .NET Agent may cause existing Web Applications that are using SOAP to restart.

WCF Requirements and Limitations: Monitoring .NET Windows Communication Foundation (WCF) services requires .NET Framework 3.0 SP1 or greater and only the following bindings are supported:

➤ BasicHttpBinding

➤ WSHttpBinding

➤ NetTcpBinding

If your application uses a binding that is not supported then the probe will only create a generic server request for each WCF method. It will not be a web Service and there will be no XVM correlation.

The recommended system configurations for hosting the .NET Agent:

Requirements for the .NET Agent Host

The overhead that the .NET Agent imposes on the system being monitored is extremely low. The following are the recommendations for memory and disk space that will support the Agent's processing:

Platform	All Supported Platforms.
Memory	60 MB Additional RAM
Free Hard Disk Space	200 MB Additional Space
.NET Framework	1.1 or later

Requirements for the .NET Diagnostics Profiler User Interface Host

The user interface for the .NET Diagnostics Profiler is presented using DHTML/XML/XSLT/JScript technology that requires IE6 or later. The machine that is to be used to present the UI must be able to access the .NET Diagnostics Profiler URL: http://<probehost>:<probeport>/profiler. The probes are assigned to the first available port within the range defined during the Probe installation. The default port range is 35000 - 35100.

Launching the .NET Agent Installer

The .NET Agent installer can be launched from the Diagnostics installation disk, another location, or from the Business Availability Center Diagnostics Downloads page.

To launch the Installer from the product installation disk:

1 Run the **setup.exe** file in the root directory of the installation disk. The Diagnostics setup program begins and displays the installation menu page.

2 From the menu select **Diagnostics Agent for .NET**. This will install the 32-bit Windows version of the .NET agent. If you want to install the 64-bit version of the .NET agent you will need to **Browse the DVD** to locate the **Diagnostics_Installers** directory and double-click the DotNetAgentSetup_x64_<version>.msi file.

To launch the Installer from another location:

1 From the **<HP Diagnostics Installation Disk>\Diagnostics _Installers** directory, copy the file **DotNetAgentSetup_x86_<version>.msi** (for 32-bit Windows) or **DotNetAgentSetup_x64_<version>.msi** (for 64-bit Windows) to the new location, and then double-click on the file.

2 Continue with "Running the Installation" on page 201.

To launch the Installer from the Business Availability Center Diagnostics downloads page:

1 In **Business Availability Center**, select **Admin > Diagnostics** from the top menu in and click the **Downloads** tab.

2 On the Downloads page, click the appropriate link to download the .NET Agent installer for either 32-bit Windows or 64-bit Windows.

Note: The .NET Agent installers are available in Business Availability Center if put into the required directory for Business Availability Center to access. You can enable this during the installation of the Diagnostic Server, or the .NET agent installers can be copied manually from the Diagnostics installation disk to the required location.

Continue with "Running the Installation" on page 201.

Running the Installation

After you have launched the installer, you are ready to begin the main installation procedure.

To install the .NET Agent on a Windows machine:

1 Accept the end user license agreement.

Read the agreement and select **I accept the terms of the License Agreement**.

Click **Next** to proceed.

2 Provide the location where you want the Agent installed.

By default, the Agent is installed in **C:\MercuryDiagnostics\.NET Probe**.

Accept the default directory or select a different location either by typing in a different path, or by clicking **Browse** to navigate to the installation directory.

Click **Next** to continue.

3 Select the .NET Agent features that you want to install.
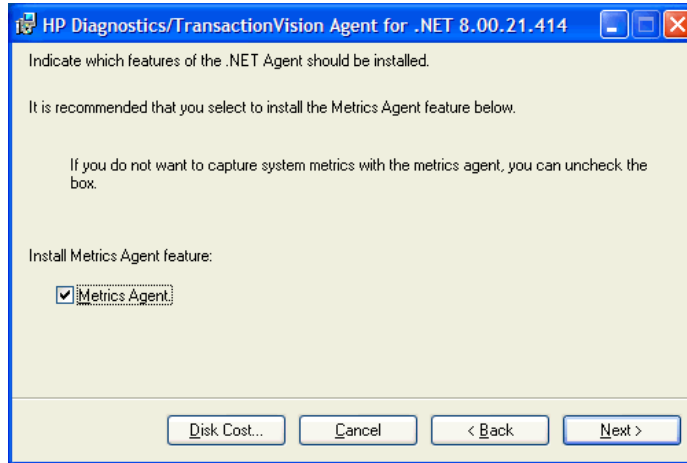
It is recommended that you install the **Metrics Agent** feature which is checked by default.

But if you do NOT want to capture system metrics on the host machine you can uncheck the **Metrics Agent** box.

If you want to check the amount of available disk space on the drives of the host, click the **Disk Cost** button. Use this functionality to make sure that there is enough room for the Agent installation.

Click **Next** to continue.

4 Select whether you want to install the Agent as the Diagnostics Profiler only, as an agent reporting to a Diagnostics Server, as an agent reporting to the TransactionVision Server or reporting to both the Diagnostics and TransactionVision servers.
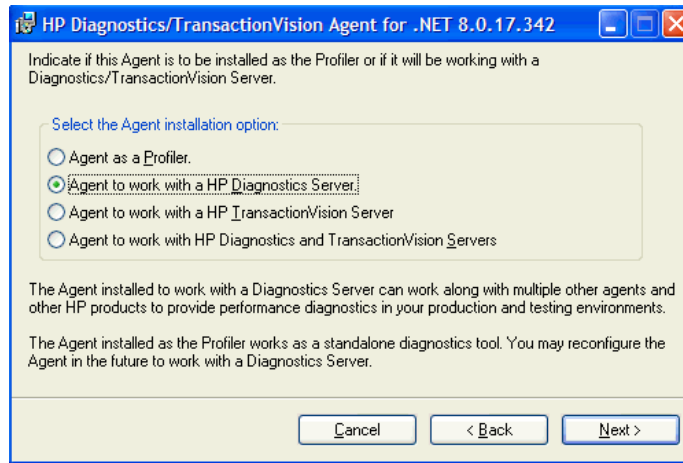
Make the selection that is appropriate for the environment where you will be using the Agent:

➤ To use the Agent as a Profiler in a Diagnostics environment, select **Agent as a Profiler**.

➤ To use the Agent as a probe in a Diagnostics environment, select **Agent to work with an HP Diagnostics Server**.

➤ To use the Agent in a TransactionVision environment, select **Agent to work with an HP TransactionVision Server**.

➤ To use the Agent in both a TransactionVision environment and HP Diagnostics environment, select **Agent to work with HP Diagnostics and TransactionVision Servers**.

Click **Next** to continue.

Next Step: At this stage, the installation procedure differs, depending on the environment in which you are installing the Agent.
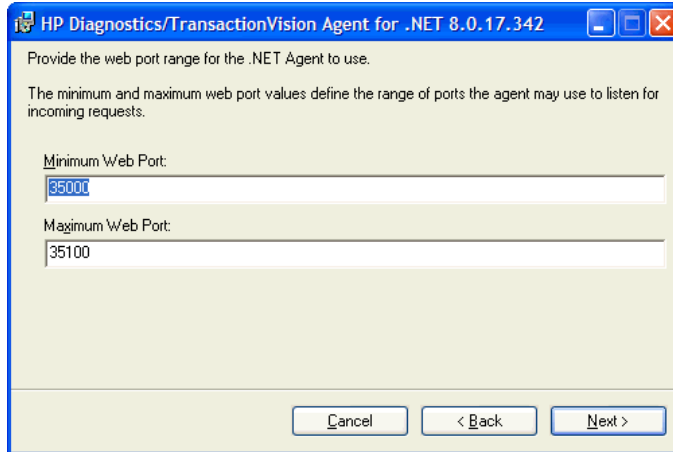
➤ If you are installing the Agent as the Diagnostics Profiler only, continue with "Installing the Agent as a Profiler Only" on page 205.

➤ If you are installing the Agent to work with a Diagnostics Server, continue with "Installing the Agent to Work with a Diagnostics Server" on page 207.

➤ If you are installing the Agent to work in a TransactionVision environment, continue with "Installing the Agent to Work in a TransactionVision Environment" on page 213.

➤ If you are installing the Agent to work in both a TransactionVision environment and HP Diagnostics environment, continue with "Installing the Agent to Work with a Diagnostics Server" on page 207 and then "Installing the Agent to Work in a TransactionVision Environment" on page 213.

Click **Next** to continue.

Installing the Agent as a Profiler Only

If you are installing the Agent to work as a Diagnostics Profiler only, continue with the following procedure:

1 Provide the Web port range for the .NET Agent to use.

➤ **Minimum Web Port.** Type the lowest port number, in a range of ports on the Agent host, that you want to assign to the Agent.

➤ **Maximum Web Port.** Type the highest port number, in a range of ports on the Agent host, that you want to assign to the Agent.

Note: The default range is from 35000 to 35100 (inclusive).

The upper and lower limits of the Web Port Range are defined by the **Minimum Web Port** and **Maximum Web Port** fields. The Web Port Range contains the ports that the Agent can use.

When an Agent is started, it attempts to find an unused port from within this range; starting from the lowest port number in the range and working its way up to the highest. Ports within the range may already be in use if another Agent or application has previously claimed them.

The minimum size for the port range is equal to the maximum number of Agents that will be concurrently running on the Agent's host.

Considerations when setting the Web Port Range:

➤ If the Agents are working with ASP.NET applications, it is recommended that you double the number of ports to account for ASP.NET's appdomain recycling.

➤ If you have a firewall between the Agent and a component that will be communicating with the Agent, you must open the firewall for the ports within the range. For this reason you may want to adjust the range to be just big enough.

Click **Next** to continue.

2 The pre-installation summary screen opens. Click **Back** to make any changes. Click **Install** to start the .NET Agent installation.
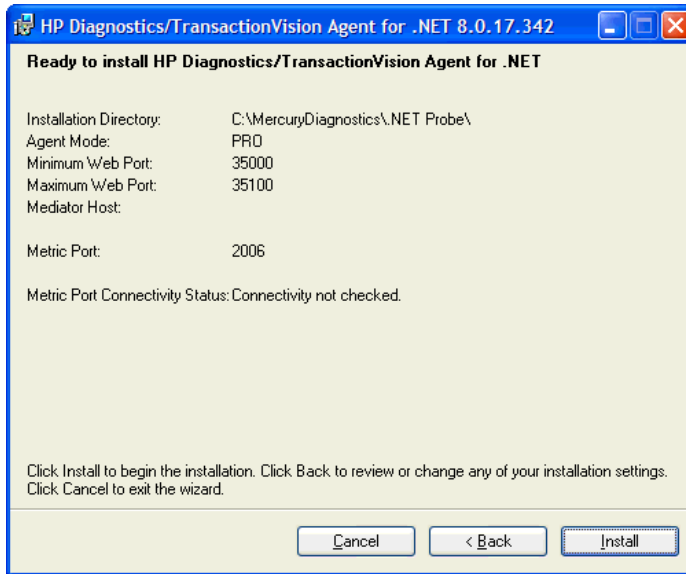
Note: When installing the Profiler only, there is no test for Metric Port connectivity.

3 When the .NET Agent installation completes, instructions for post installation tasks are displayed in the installer window. See "Enabling and Disabling Standard Instrumentation for Applications" on page 216 for more information.

Click **Finish** to exit the installer.

4 After you exit the installer you must restart either the IIS or the Web publishing service before you can use the .NET agent with ASP.NET applications. See "Restarting IIS" on page 218.

Installing the Agent to Work with a Diagnostics Server

If you are installing the Agent to work with a Diagnostics Server, continue with the following procedure.

1 Enter the Agent name and Agent group name.

➤ **Agent Name.** The name that identifies the Agent within HP Diagnostics. If you leave this field blank, the .NET Agent will auto-generate an Agent name based on the application's domain name.

207

Important: It is recommended that you leave **Agent Name** blank and allow the Agent to auto-generate the Agent name. Read the following information carefully if you decide to enter your own Agent name.

Considerations when entering an Agent name:

➤ Valid characters that can appear in the Agent name are: letters, digits, dashes, underscores, and periods.

➤ Assign an Agent name that will help you recognize the application that the Agent is monitoring, and the type of Agent it is.

For example, the Agent name for .NET Agent installed to monitor the application named PetWorld can be:

 PetWorld_Dotnet_Agent

➤ When you specify an Agent name, all of the Agents on the host are forced to use the same Agent name. If you want to override the default names, use the following substitution macros to enhance the name at run time:

$(MACHINENAME): Machine's host name

$(APPDOMAIN): Application's domain name

$(PID): Application's process ID

The default Agent name auto-generated by the Agent when the Agent name field is left blank is equivalent to specifying $(APPDOMAIN).NET.

➤ **Agent Group Name:** Enter a name for an existing group or for a new group to be created. The default value for the Agent group name is Default. The agent group name is case-sensitive. In Diagnostics this name is used as the probe group name.

Probe groups are logical groupings of probes that report to the same Diagnostics Server. The performance metrics for a probe group are tracked, and can be displayed on many of the Diagnostics views.

For example, you may want to assign all of the Agents for a particular enterprise application to a single Agent group so that you can monitor the performance of the group as well as the individual Agents.

Click **Next** to continue.

2 Provide the information needed to enable the .NET Agent to communicate with the Diagnostics Server in Mediator mode.

a In the **Diagnostics Server Mediator Host or IP Address** box, type the host name or IP address of the host for the Diagnostics Server in Mediator mode.

You should specify the fully qualified host name, not just the simple host name. In a mixed OS environment, where UNIX is one of the systems, this is essential for proper network routing.
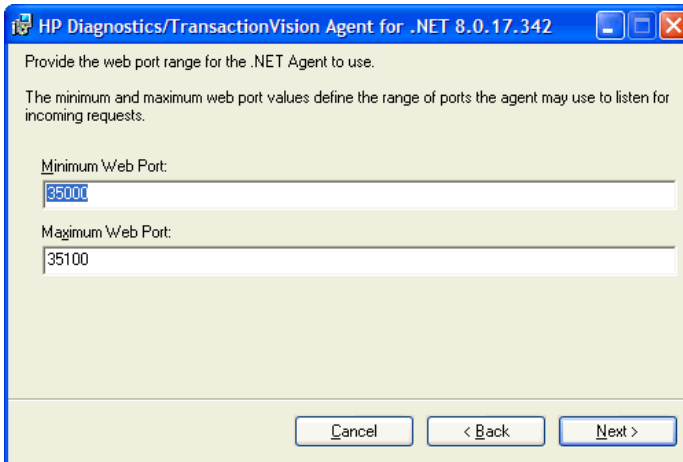
b In the **Diagnostics Server Data Port** box, type the port number where the Diagnostics Server is listening for Agent communication. The default port number is 2612. If you have changed the port since the Diagnostics Server was installed, you should specify that port number here instead of the default.

c In the **Diagnostics Server Metric Port** box, type the port number where the Diagnostics Server is listening for communications from the System Metrics Agent. The default port number is **2006**. If you have changed the port since the Diagnostics Server was installed, you should specify that port number here instead of the default.

d If you want to perform a connectivity check to make sure that the Diagnostics Server is running and accessible from the installation host, click **Test**.

The connectivity check lets you know right away if you have made an error in the information that you provided about the Diagnostics Server in Mediator mode, or if there is a connection problem between the Diagnostics Server's host and the Agent's host. If the connection to the Diagnostics Server in Mediator mode host cannot be resolved, an error message is displayed.

e Click **Next** to proceed.

3 Provide the Web port range for the .NET Agent to use.

➤ **Minimum Web Port.** Type the lowest port number, in a range of ports on the Agent host, that you want to assign to the Agent.

➤ **Maximum Web Port.** Type the highest port number, in a range of ports on the Agent host, that you want to assign to the Agent.

Note: The default range is from 35000 to 35100 (inclusive).

The upper and lower limits of the Web Port Range are defined by the **Minimum Web Port** and **Maximum Web Port** fields. The Web Port Range contains the ports that the Agent can use.

When an Agent is started, it attempts to find an unused port from within this range; starting from the lowest port number in the range and working its way up to the highest. Ports within the range may already be in use if another Agent or application has previously claimed them.

The minimum size for the port range is equal to the maximum number of Agents that will be concurrently running on the Agent's host.

Considerations when setting the Web Port Range:

➤ If the Agents are working with ASP.NET applications, it is recommended that you double the number of ports to account for ASP.NET's appdomain recycling.

➤ If you have a firewall between the Agent and a component that will be communicating with the Agent, you must open the firewall for the ports within the range. For this reason you may want to adjust the range to be just big enough.

Click **Next** to continue.

4 The pre-installation summary screen opens. Click **Back** to make any changes. Click **Install** to start the .NET Agent installation.
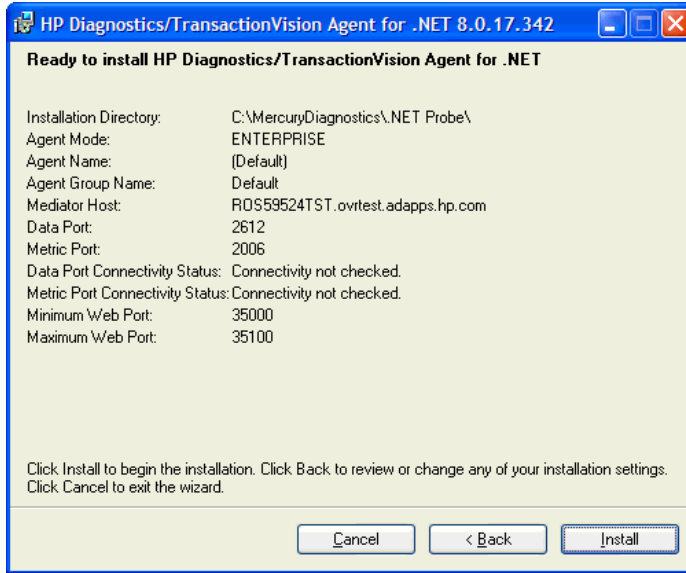
5 When the .NET Agent installation completes, instructions for post installation tasks are displayed in the installer window.

Click **Finish** to exit the installer.

See the following topics for details:

➤ "About Custom Configuration and Instrumentation for the .NET Agent" on page 216

➤ "Enabling and Disabling Standard Instrumentation for Applications" on page 216 for more information.

➤ "Restarting IIS" on page 218

➤ "Verifying the .NET Agent Installation" on page 219

6 After you exit the installer and finish any modification/customization to the .NET Agent configuration and .NET application instrumentation you must restart the IIS service before you can use the .NET Agent with ASP.NET applications. See "Restarting IIS" on page 218.

Installing the Agent to Work in a TransactionVision Environment

If you are installing the .NET Agent to work with in a TransactionVision environment, continue with the following procedure.

1 The Configure the .NET Agent for TransactionVision dialog appears.
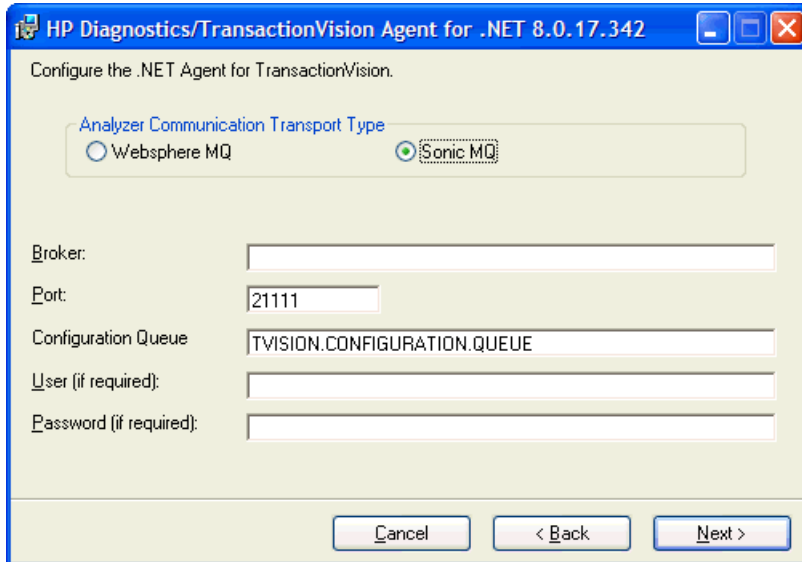
2 Choose the Messaging Middleware Provider (Analyzer Communication Transport Type). Options are: WebSphere MQ and SonicMQ.

SonicMQ is included with the .NET Agent. If you specify this option, the Sonic MQ .NET client (Sonic.Client.dll - Progress SonicMQ .NET Client, version 7.6.0.112) is installed as part of the Agent installation.

A third-party WebSphere MQ installation can be used instead. In this cae, you must install the MQ series .NET client (amqmdnet.dll - WebSphere MQ Classes for .NET, version 1.0.0.3) on the host being monitored.

By default, SonicMQ is selected.

3 For SonicMQ, enter the following:

Broker. The broker on which the TransactionVision configuration queue is hosted, usually the host name.

Port. The port on which the broker communicates.

Configuration Queue. Name of the configuration queue.

User. User id if required by Sonic MQ installation for connection.

Password. Password if required by Sonic MQ installation for connection. This is in the obfuscated form created by using the PassGen utility. For more information about PassGen, see Chapter 16, "Understanding the .NET Probe Configuration File."

4 For WebSphere MQ, enter the following:

Host. Host on which the WebSphere MQ queue manager resides.

Port. Port number for WebSphere MQ queue manager.

ConfigurationQueue. Name of the configuration queue.

User. User ID if required by WebSphere installation for connection.

Password. Password if required by the WebSphere MQ installation for connection. This is in the obfuscated form created by using the PassGen utility. For more information about PassGen, see Chapter 16, "Understanding the .NET Probe Configuration File."

WebSphere MQ Channel. Channel name for WebSphere MQ queue manager.

WebSphere MQ Q Manager. Name of the queue manager.

Click **Next** to continue.

5 The pre-installation summary dialog appears.

6 Click **Install** to proceed with the installation.

7 After you exit the installer and finish any modifications/customization to the .NET Agent configuration and .NET application instrumentation you must restart the IIS service before you can use the .NET Agent with ASP.NET applications. See "Restarting IIS" on page 218.

Verifying that the .NET Agent Is Connected

If you are running the .NET Agent in a production environment (with Business Availability Center or Diagnostics Standalone), or in a test environment (with LoadRunner or Performance Center), you can verify that the agent is connected using the following instructions.

To verify that the Agent is connected:

1 Open web browser and browse to http://
<diagnostics_server_host>:<diagnostics_server_port>/registrar for the
Diagnostics Server in Commander mode. You may be prompted to log on to the server.

2 Under **View** click on **Registered Components**.

3 Look for your application listed in the **Name** column. To refine the list of Matching Components, select the **Probe** Component Type and click the **Submit** button for a list of Agents (probes) only.

4 Look for the agent's status in the **Active** column. An Active status of True indicates that the agent is connected.

About Custom Configuration and Instrumentation for the .NET Agent

If you installed the .NET Agent to work with Diagnostics then by default it is configured to work as a Diagnostics probe - capturing performance metrics for applications. You can customize the configuration to suit your environment and the performance issues that you would like to diagnose.

The installer configures your ASP.NET applications and the .NET Agent so that they can work together to capture the basic workload of the applications. It is possible that one or more of your ASP.NET applications has been deployed in a manner that prevents the installer from detecting it, or you may want to enhance the standard instrumentation to capture the performance metrics for the custom classes in the application.

In HP Diagnostics you can do additional configuration using the probe_config.xml file. For details on this file see Chapter 16, "Understanding the .NET Probe Configuration File". For instructions on advanced .NET Agent configuration, see Chapter 17, "Advanced .NET Probe Configuration."

Also in HP Diagnostics you can create custom instrumentation points to handle unique situations in your application environment. For general information on custom instrumentation see Chapter 10, "Custom Instrumentation for .NET Applications".

Enabling and Disabling Standard Instrumentation for Applications

When the .NET Agent is first installed, the standard ASP.NET/ADO instrumentation for all discovered applications is enabled; but no application specific instrumentation is enabled. You control which applications have their instrumentation enabled or disabled using the attributes in the **probe_config.xml** file for the .NET Agent.

Disabling instrumentation for an application allows you to avoid the processing overhead and distracting information in the Diagnostics views for applications that are not relevant to the environment whose performance you want to monitor.

Enabling instrumentation for all application allows the .NET Agent to monitor the performance of all detected applications so that you can see the performance metrics for all of the applications in the views of the Diagnostics and Profiler user interfaces.

To enable or disable the instrumentation for an application:

1 Set the **enablealldomains** attribute in the **process** tag to **false**. This allows the attributes of each appdomain tag to control the state of the instrumentation for each application. If there are no appdomain entries, then no applications are enabled.

2 Set the **enabled** attribute in the **appdomain** tag to **true** for each application that you wish to enable the instrumentation.

3 Set the **enabled** attribute in the **appdomain** tag to **false** for each application that is to have its instrumentation disabled.

The following example shows the instrumentation has been enabled for one application and disabled for another.

```
<process name="ASP.NET", <enablealldomains="false">
    <points file="ASP.NET.points" />
    <points file="ADO.points" />
    <appdomain name="myApplication", enabled="true">
        <points file="myApplication.points" />
    </appdomain>
    <appdomain name="myApplicationTwo", enabled="false">
        <points file="myApplicationTwo.points"/>
    </appdomain>
</process>
```

217

To enable the instrumentation for ALL applications:

➤ Set the **enablealldomains** attribute in the **process** tag to **true**. This overrides the settings of the attributes in each appdomain tag so that the instrumentation can be enabled without going in and setting numerous attributes.

The following example shows the instrumentation has been enabled for all applications:

```
<process name="ASP.NET", <enablealldomains="true">
    <logging level=""/>
    <points file="ASP.NET.points"/>
    <appdomain name="myApplication", enabled="false">
        <points file="myApplication.points"/>
    </appdomain>
    <appdomain name="myApplicationTwo", enabled="false">
        <points file="myApplicationTwo.points"/>
    </appdomain>
</process>
```

Restarting IIS

After you have installed the .NET Agent and modified the instrumentation or configuration for the Agent, you must restart IIS to cause your changes to take effect.

To restart IIS from the command line or from the **Start** > **Run** menu, type iisreset and press **Enter**.

This command restarts the Web publishing service but does not immediately start the .NET Agent. The next time that a Web page in the application is requested, the agent is started, the applications are instrumented, and the agent registers with the Diagnostics Server in Commander mode.

Note: ASP.NET automatically restarts applications under various circumstances, including when it has detected that applications have been redeployed, or when applications are exceeding the configured resource thresholds.

When ASP.NET restarts an application that is being monitored by a .NET Agent, the agent is deactivated and a new agent is started. While this is occurring, there may be a period of overlap where there are multiple agents simultaneously registered with the Diagnostics Server in Commander mode and connected to the Diagnostics Server in Mediator mode. This condition could cause LoadRunner / Performance Center and Business Availability Center to report errors during the application restart sequence.

Verifying the .NET Agent Installation

This section applies only if you installed the Agent to work with a Diagnostics Server.

Use the System Health Monitor to verify the installation of the .NET Agent. For detailed instructions about how to use the System Health Monitor, see Appendix D, "Using the System Health Monitor".

If you have been following the recommended installation sequence, after you have installed the .NET Agent and restarted the instrumented application, you will be able to verify the following:

➤ The Diagnostics Server in Commander mode was successfully installed.

➤ Where relevant, additional Diagnostics Servers in Mediator mode were successfully installed and are communicating with the Diagnostics Server in Commander mode.

➤ The .NET Agent was successfully installed and is communicating with the Diagnostics Server.

Note:
The .NET Agent will not be displayed in System Health when first installed because the Agent does not register with the Diagnostics Server until it is started. The Agent is started and is registered with the Diagnostics Server when the instrumented application is run. For ASP.NET applications, this happens the first time that a page is requested for the instrumented application.
The default logging level for the .NET Agent has been set to **info** so that if the Agent is not displayed in System Health when you believe that it should be, you can check the log file to help troubleshoot the problem. If you do not see a log file you can check the Windows Event Viewer.

Depending on your deployment, the new .NET Agent is shown on the System Health Monitor to be reporting to either the Diagnostics Server in Commander mode or the Diagnostics Server in Mediator mode.
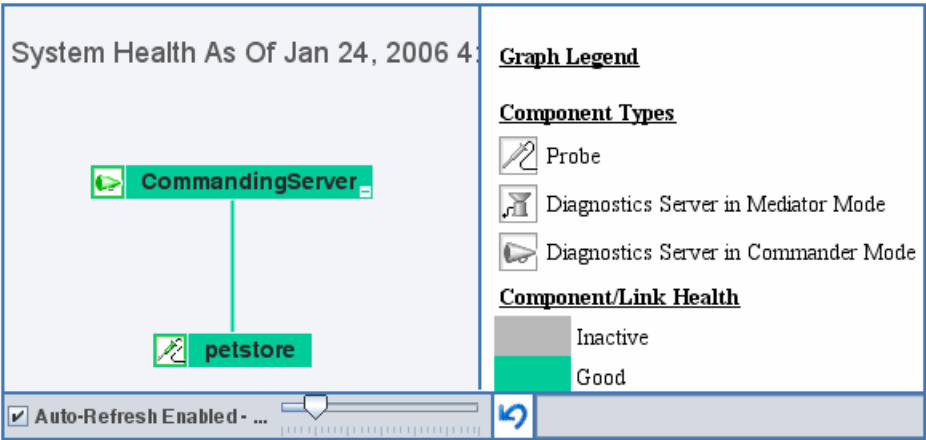
Determining the Version of the .NET Agent

When you request support, it is useful to know the version of the Diagnostics components that you have installed.

To determine the version of the .NET Agent:

➤ Right-click the file **<Agent_install_dir>\bin\HP.Profiler.dll** and select **Properties** from the menu. In the Properties dialog, select the Version tab to display the component version information.

Alternately you can determine the version of the .NET Agent using the System Health Monitor in Diagnostics.

Uninstalling the .NET Agent

To uninstall the .NET Agent, follow these steps:

1 Stop all Web applications that are using SOAP.

2 From the Windows Control Panel select **Add/Remove Programs** and then select **HP Diagnostics/TransactionVision Agent for .NET** to uninstall.

3 Restart the Web applications.

Enabling and Disabling the Diagnostics .NET Agent

The .NET Agent is enabled when it is installed. After you restart your Web server and a URL in the application has been accessed, the .NET Agent begins to gather performance information.

Note: Enabling the .NET Agent re-enables the SOAP Extension Handler which may cause some Web applications that use SOAP to restart.

Also, before you disable the .NET Agent, stop all Web applications on the host that are using SOAP.

You may disable the .NET Agent so that it does not start and does not gather performance metrics.

To disable a .NET Agent:

➤ Select **Start** > **Programs** > **HP Diagnostics .NET Probe** > **Disable HP .NET Probe**.

To enable a .NET Agent that was disabled:

➤ Select **Start** > **Programs** > **HP Diagnostics .NET Probe** > **Enable HP .NET Probe**.

Note: Disabling the .NET Agent only disables the probe metrics collector and the active probes. It does not disable the system metrics collector. The process of enabling or disabling system metrics is controlled through the standard Windows services manager. The effect of enabling or disabling probes will only happen the next time the probed application restarts, that is, it has no affect on currently running applications.

Disabling Logging

You can disable probe application logging by changing the **logging level** tag of the ASP.NET process section of the **probe_config.xml** file, as shown in the following example:

```
<process name="ASP.NET">
     <logging level="off"/>
</process>
```

You can disable probe instrumentation logging by changing the **logging level** tag of the instrumentation section, as shown in the following example:

```
<instrumentation>
     <logging level="off" />
</instrumentation>
```

Part IV

Custom Instrumentation for Monitoring Java and .NET applications

9

Custom Instrumentation for Java Applications

Information is provided for how to control the instrumentation that HP Diagnostics applies to the classes and methods of the applications to enable it to gather the performance metrics.

This chapter includes:

➤ About Instrumentation and Capture Points Files on page 226

➤ Locating the Capture Points Files on page 227

➤ Coding Points in the Capture Points File on page 228

➤ Defining Points With Code Snippets on page 237

➤ Instrumentation Examples on page 252

➤ Understanding the Overhead of Custom Instrumentation on page 268

➤ Instrumentation Control on page 269

➤ Advanced Instrumentation on page 270

About Instrumentation and Capture Points Files

Instrumentation refers to bytecode that the Probe inserts into the class files of the application as they are loaded by the class loader of your virtual machine. Instrumentation enables a Probe to measure execution time, count invocations, retrieve arguments, catch exceptions and correlate method calls and threads. The instrumentation points for each Probe instance are specified in the capture points file.

The capture points file is an instrumentation mechanism that allows you to control the scope of the instrumentation so that Diagnostics can give you all of the information that you need to understand the performance of the applications without overwhelming you with costly or confusing extraneous information. The instrumentation definitions contained in the capture points file are called *points* that tell the Probe which methods to instrument, how they should be instrumented, and which instrumentation should be installed.

Points can include regular expressions that "wildcard" the instructions so that they apply to more than one method, class, and package or namespace specification. For more information about using regular expressions, see Appendix J, "Using Regular Expressions".

You can customize the points in the capture point file to include methods, classes, packages and namespaces - for areas of the application that do not fall within the default points. A common situation that may require custom points is when a Java EE application contains business logic that is not derived from the javax.ejb.SessionBean interface. Another situation for custom points is when you want to override a default point to alter its layer or to track it from a specific caller method.

The points in the capture points file are grouped into layers. Layers are used to organize the performance metrics into meaningful tiers of information that can be compared as part of a triage process and to control the collection behavior of the instrumentation.

The points in the capture points file that is installed with the Probe are grouped into default layers. You can customize the default layers and create new layers. For more information about layers see Chapter 12, "Instrumentation Layers."

Locating the Capture Points Files

When you install the Java agent (probe), predefined default capture points files are installed with a set of points for the platform you are using.

For Java, a default capture points file containing pre-defined Java EE points is located at: **<probe_install_dir>\etc\auto_detect.points**

Notes:

➤ You should you make a copy of the default **auto_detect.points** file, giving it a different name and use the copy to make all of your instrumentation customizations. This is a precaution to prevent you from losing your custom instrumentation when you upgrade to a new version of the probe and the installer overlays the auto_detect.points file.

➤ The default file name is specified in **<probe_install_dir>\etc\probe.properties**.

➤ To override the default file name so that the copy will be used instead, use the -Dprobe.points.file.name=<newPointsFileName_NoExtension> JVM property.

Coding Points in the Capture Points File

The following is a list of the arguments that can be used to define a point in the capture points file:

```
[Point-Name]            =<unique name for the point>
;-------------------------------------------------------------------------
class                   = <class name or regular expression>
method                  = <method name or regular expression>
signature               = <method signature or regular expressions>
ignore_cl               = <list of class names or regular expressions>
ignore_method           = <list of method names or regular expressions>
ignore_tree             = <list of class names or regular expressions>
method_access_filter    = <list of class names or regular expressions>
deep_mode               = <soft or hard mode>
scope                   = <list of methods or regular expressions>
ignoreScope             = <list of methods or regular expressions>
detail                  = <list of specifiers>
layer                   = <layer name>
layerType               = <layer type>
rootRenameTo            = <string>
keyword                 = <keyword>
priority                = <integer number>
active                  = <true, false>
```

The arguments are described in the following sections.

➤ Mandatory Point Arguments

➤ Optional Point Entries

Mandatory Point Arguments

Every point, except for the points for LWMD, RMI and SAP RFC, HttpCorrelation, and JDBC SQL, must contain the following arguments:

Argument	Description
Point-Name	A unique name for the point.
class	The class argument specifies the name of the class or interface to be instrumented. The name should include the full package/namespace name using periods between the package levels. Any valid regular expression may be used.
method	The method argument specifies the name of the method to be instrumented. To be successful, the method name must match a method defined in the class or interface specified by the class argument. Any valid regular expression may be used.
signature	The signature argument specifies the signature (parameter and result types) of the method using javap symbolic encoding for method signatures (<jdk_install>/ bin.javap -s).

Argument	Description
layer	The layer argument specifies a layer, sub-layer or tier under which the data from this point is grouped. Layers are a part of the instrumentation collection control.
	Layers in a point can be specified with nested layers or sub layers by separating the layer names with a / (slash). The layer specified following the slash is a sub layer of the layer specified before the slash. A sub-layer can be defined to have its own sub-layers by coding another slash and layer name following a sub-layer name.
	In the UI, the sub-layers for a layer are displayed under their parent layer. For example, if the sub-layers of the Web layer are JSP and Struts, they will be displayed under the Web layer and a drilldown will exist from Web to JSP and Struts.

The following is an example of a custom point that contains the mandatory arguments:

```
[MyCustomEntry_1]
; comments here….
class = myPackage.myClass.MyFoo
method = myMethod
signature = !.*
layer = myCustomStuff
```

Note: Regular expressions may be used for most of the arguments in a point. They must be prefaced with an exclamation point. For more information about using regular expressions, see Appendix J, "Using Regular Expressions."

Optional Point Entries

Point definition may contain one or more of the following arguments:

Argument	Description
keyword	The presence of the **keyword** indicates an instrumentation point handled by a special instrumentation class. The value of the keyword is looked up as a property in inst.properties, and the value of the found property is the instrumentation class name. The special instrumentation points may use implementation-specific arguments not documented here.
ignore_cl	The **ignore_cl** argument is used to specify a comma-separated list of class names or regular expressions to ignore. Any class matching one of the classes specified with **ignore_cl** is not instrumented.
ignore_method	The **ignore_method** is used to specify a comma-separated list of methods to ignore. Any method matching one of the methods specified with **ignore_method** is not instrumented.
Ignore_tree	The **ignore_tree** argument is a list of classes or regular expressions. Any subclass of a class matching one of the list items will be excluded from the instrumentation
method_access_filter	The **method_access_filter** is a list of method specifiers, separated by comma. The available specifiers are **static**, **private**, **protected**, **package**, and **public**. Any method matching this point will not be instrumented if its access specifier matches any of the listed values.

Argument	Description
deep_mode	The **deep_mode** argument specifies how subclasses are handled. This argument accepts the following three values: ➤ **none** - A value of "none" is similar to not specifying a **deep_mode** argument. It has no effect on how subclasses are handled. ➤ **soft** - A value of "soft" requests that for every class or interface matching the class, method, and signature entries, any subclasses or sub-interfaces that also implement the matching method and signature should also be instrumented. ➤ **hard** - A value of hard, requests that for every class or interface matching the class, method, and signature entries, any subclasses or sub-interfaces at any depth should have all their methods instrumented. Hard mode is typically used for points for interfaces. **Caution:** "hard" mode can lead to extensive instrumentation and very high Probe overhead.
scope	The **scope** argument constrains the context in which instrumentation is performed. If this entry is specified, the inserted bytecode will be caller side. Any valid regular expression may be used for the value of this argument. Scope values are a comma-separated list of package, class, and method names in standard Java notation.
ignoreScope	The **ignoreScope** argument takes the form of a list of method names or regular expressions and is used to exclude certain packages, classes, and methods from those included in the scope specified in scope argument.

Argument	Description
detail	The **detail** argument specifies more specific capture instructions. It is a comma-separated list of the following:
	➤ **caller** - causes caller side instrumentation to be performed. If this keyword is not specified, the default instrumentation, callee side instrumentation, is performed.
	➤ **args:n** - calls the **toString()** method of the **n-th** argument. The string that is returned is displayed in the method's argument field in the Diagnostics console. The captured string can be used as the aggregation parameter in the layer argument. The value for **n** can be **1** through **256**.
	➤ **args:0** - calls the **toString()** on the current class instance or callee object. Static methods return the class name of the callee object
	➤ **before:code:\<code-key>** -inserts the code-snippet specified in the key at the start for the bytecode for methods that comply with the point. The final string value on the stack when the code-snippet has executed is displayed in the method's argument field in the Diagnostics console and may also be used as the aggregation parameter in the layer argument. The code-key argument specifies that secure code key that you generated for the code snippet that you created for the point. See "Defining Points With Code Snippets" on page 237 for information about code snippets and see "Securing Code Snippets" on page 250 for information on code keys.
	➤ **after:code:\<code-key>**- inserts the code-snippet specified by the key at every exit point from the bytecode of methods that comply with the point. The "after" code-snippets should not leave any values on the stack after they are executed.

Argument	Description
detail (continued)	➤ **disabled** - prevents the instrumentation inserted into the bytecode from reporting data. A disabled point can be dynamically enabled using the Instrumentation control webpage so that it will begin reporting data. This webpage can be accessed using the Profiler URL http://<probe_install_dir>:<probe_port>/inst/layer. ➤ **outbound** - flags the method so it is listed on the Outbound Calls screen. Also, causes the Diagnostics argument for this instrumentation entry to be parsed to determine if additional information about the outbound request can be displayed in the Diagnostics dashboards. ➤ **no-correlation** – used with those "outbound" points which do not use correlation supporting technologies. ➤ **method-no-trim** – indicates that no latency-based trimming should take place when a method instrumented by this point is executed. ➤ **method-trim** – indicates that every invocation of the method instrumented by this point should be "trimmed", i.e. not reported. However, side effects of the corresponding code-snippets, if any, take place normally. ➤ **lifecycle** – identifies the instrumentation point as relevant for object lifecycle monitoring. ➤ **no-layer-recurse** – prohibits recording of any methods called from the method instrumented by this point, unless the callee belongs to a different layer. ➤ **is-statement** – marks calls into the java.sql.Statement class. ➤ **is-prepare-statement** – marks calls returning java.sql.Statement objects to capture ➤ **method-cpu-time** – causes the CPU inclusive time to be collected for this method in addition to latency, unless CPU collection is completely turned off (cpu.timestamp.collection.method = 0).

Argument	Description
detail (continued)	➤ **condition** – prohibits instrumentation by this point unless the specified condition is met. The conditions are static, and they are defined by the details.conditional.properties property in inst.properties (or on the command line).
	➤ **when-root-rename** - instructs the probe to rename the server request whenever the method instrumented by this point is the first one executed.
	➤ **diag** - marks the point as relevant for HP Diagnostics (default).
	➤ **tv:<key>** - marks the point as relevant for HP Transaction Vision.
	➤ **no-tv** - marks the point as conflicting with HP Transaction Vision. If Transaction Vision is configured to be active, such points will be prohibited from instrumenting the Java code at all.
	➤ **add-field:<access>:<type>:<name>** - causes adding the specified field to the instrumented class.
	➤ **gen-instrument-trace** - causes printing the thread stack trace onto stdout whenever this point is used for instrumentation.
	➤ **gen-runtime-trace** - causes printing the thread stack trace onto stdout whenever the methods instrumented by this point are executed.
	➤ **trace** - causes printing verbose instrumentation information into probe.log on each enter or exit from each method instrumented by this point.
	➤ **sub-point:<key>** - specifies additional processing during instrumentation; the key must be present in inst.properties and must identify a class name used for the processing.
	➤ **store-thread** - causes all special fields used in the corresponding code-snippet to be stored in a thread-local data structure.
	➤ **store-fragment** - causes all special fields used in the corresponding code-snippet to be stored as attributes of the current server request.

Argument	Description
detail (continued)	➤ **store-method** - causes all special fields used in the corresponding code-snippet to be stored as attributes of the invocation of the method instrumented by this point. ➤ **ws-operation** - specifies that the instrumentation entry is for an inbound Web services call. Also, causes the Diagnostics argument for this instrumentation entry to be parsed to determine if additional information about the Web service request can be displayed in the Diagnostics dashboards.
rootRenameTo	The **rootRenameTo** argument is used to identify server requests whenever the **when-root-rename** detail is in effect
layerType	The **layerType** argument specifies special handling for some instrumented methods. This argument accepts the following values: ➤ **method** - no special handling (default). ➤ **trended_method** - identifies methods to be displayed in the Trended Methods view. ➤ **Portlet** - identifies portlet lifecycle methods that will be used for the Portal Components views. These are set by HP Diagnostics and should not be modified. ➤ **sql** - identifies methods that will be used to capture SQL for the SQL views. These are set by HP Diagnostics and should not be modified.
priority	Whenever there's more than one instrumentation point which can be applied to a given method, and when the Diagnostics Agent cannot resolve the conflict on its own, it will use the point's **priority** to determine which point to use. Higher priority wins, the default is zero.
active	The active argument activates or deactivates a point. When set to "true" the point is activated. When set to "false" the point is inactive and is ignored by the Probe.

Defining Points With Code Snippets

Custom code arguments specify a snippet of code that is to be inserted into the bytecode for a point. Code snippets in a point are used when the value returned by calling an object's **toString()** method, as specified in the **args:n** argument, is not going to provide useful information for the Diagnostics console or when there is a requirement to display more than one argument for an instrumented method.

A code snippet in a point is declared using the keyword **before:code:<code-key>** or **after:code:<code-key>** in the detail argument of the point. The before and after is used for executing the code snippet before or after the instrumented method. The code snippet is typically secured using a code-key argument to prevent un-authorized modifications of the code snippet. The values for the code-key arguments can be generated using any running Probe instance's code-key generator page and are valid on any Probe installation. For more information on the code-key see "Securing Code Snippets" on page 250.

The actual code snippets for a point are entered into the **<probe_install_dir>/etc/code/custom_code.properties** file. The code snippets in this file are associated with the point in the capture points file using the value of the code-key. Code Snippets are created using pseudo java code that uses syntax similar to OGNL. Using code snippets, calls can be made from the instrumented byte code to methods that can be accessed by the instrumented method. Objects returned by code snippets can be cast and can have their methods executed as well. Code Snippets must end with a string or an object where **toString()** can be left on the stack of statements being parsed into bytecode. This final string of the Code Snippet is used for the returned argument value this is displayed in the Diagnostics console.

Code snippets can also be used for storing values for a particular fragment directly or storing a value that could be used in a later code snippet. These features can be used through special fields and key word details like **store-fragment** and **store-thread**.

Note: Code snippets are a very powerful tool that should be used carefully because of the potential impact to the overhead incurred by the Probe. For this reason, Diagnostics requires that a code-key be specified along with the code snippet before the Probe will use the code snippet during instrumentation.

This section includes:

➤ "Using Code Snippets" on page 238

➤ "Code Snippet Grammar" on page 239

➤ "Securing Code Snippets" on page 250

Using Code Snippets

To use code snippets when specifying a point in **<probe_install_dir>/etc/ auto_detect.points** you must use the detail as shown in the following example.

```
class     = javax.jms.TopicPublisher
method    = publish
signature = !\(Ljavax/jms/Topic.*
deep_mode = soft
layer     = Messaging/JMS/Producer
detail    = outbound,no-correlation,before:code:6d0f3088
```

The before:code entry in the detail argument indicates that a code snippet has been entered for the point. The code-key value is used to secure the code that is in the code snippet and to tie the point with the actual code snippet.

The code snippet associated with the point must be entered in
<probe_install_dir>/etc/code/custom_code.properties as shown in the
following example:

```
# Used by [JMS-TopicPublisher2]
6d0f3088 = #topic =
@ProbeCodeSnippetHelper@.checkForTempName(#arg1.getTopicName()); \
"DIAG_ARG:type=jms&name=topic:"+ #topic + "&target=topic://" + #topic;
```

The code snippet is associated with the point in the capture points file using
the value of the code-key.

Code Snippet Grammar

The following describes the syntax that can be used to create the code
snippets:

➤ **Literals**

Only the following literal types are supported in code snippets.

Literal Type	Syntax Example
string	"a string"
boolean	true, false
integer	42
Arrays *	[A][B][C]
	[D][E][F]
null constant	null

* Single and multi-dimensional arrays of all types

➤ **String concatenation**

Basic string concatenation is supported in code snippets.

Concatenation Type	Syntax Example
Two strings	"a string" + "another string"
A string and a literal	"a string" + 42

➤ **Local members**

Default local members provide a way for code snippets to reference the current instance or objects that were passed to the instrumented method. These local members are used to call methods or retrieve values from those references.

Variable	Use
#callee	References the callee object for an instance method. Equivalent to the java "this" reference. This variable may not be used when referencing a static method.
#arg1, #arg2, ..., #argN	References the arguments for the callee method call.
#return	References the return value of the method end for after code snippets.
#classloader	Reserved for HP Software internal use.

Note: Some instrumentation points support *special* variable references. For example, the **CLApplicationDiscoveryPoint** supports a #classloader variable.

➤ **DIAG_ARG strings**

Code snippets allow for concatenation of a series of values building up a single DIAG_ARG value. This value allows for instrumentation of some common types of support data like Web Services and JMS by returning all the data for a particular type in one DIAG_ ARG formatted string.

Type	Field *(required)*	Definition
ws	&ws_name	Web Service name
	&ws_op	Web Service Operation name
	&ws_ns	Web Service namespace
	&ws_port (inbound only)	Web Service Port Name
	&target (outbound only)	Outbound Web Service Target
jms	&name	Queue or Topic name
	&target	Target Queue or Topic name

The format of the DIAG_ARG string includes the type fields and values (local variables) concatenated into one string as follows:

```
"DIAG_ARG:type=ws&ws_name="+ #servicename +"&ws_op="+ #operation +\
"&ws_ns="+ #ns +"&ws_port="+ #port;
```

The DIAG_ARG string should not be used in combination with the store-fragment special fields for Web Service inbound data (special fields starting with ##WS_inbound_*). You should use ONLY one or other for collecting web service inbound data.

➤ **Special fields (store-fragment)**

Default special fields provide an easy way for code snippets to pass fragment related data for common events. This mechanism supplements the existing events, but is not expected to replace them. Fragment Local Storage has higher overhead cost than custom events. These variables must be used with the **store-fragment** detail setting.

Variable	Use
##WS_consumer_id	Used to store the consumer Id for a particular fragment.
##WS_SOAP_fault_code	Used to store the SOAP fault code.
##WS_SOAP_fault_reason	Used to store the SOAP fault reason.

##WS_SOAP_fault_detail	Used to store the SOAP fault detail.
##WS_inbound_service_name	Used to store the inbound web service name.
##WS_inbound_operation_name	Used to store the inbound web service operation name.
##WS_inbound_target_namespace	Used to store the inbound web service target namespace.
##WS_inbound_port_name	Used to store the inbound web service port name.

➤ **Special fields (store-thread)**

Additionally special fields provide an easy way for code snippets to store related data for the life of the thread. These thread local storage variables should be used with caution as they have overhead associated with them. These variables must be used with the store-thread detail setting.

These variables can be retrieved in later code snippets by making a call to the probe's ThreadContextProxy class reference with either the getThreadContextValue("string value") or getAndRemoveThreadContextValue("string value") methods with "string value" being the name of the variable with out the leaded ## signs. The last retrieval of the value should always call getAndRemoveThreadContextValue("string value") to clear the value from memory and to remove the value for the next thread.

Variable	Use
##SOAPHandler_wsname	Used to store the web service name for later use by the probe's SOAP Handler.
##<any_string>	Used to store any value for later retrieval in a following code snippet.

➤ **Class references and static members**

Static members/methods may be accessed by pre-pending the class with an @ symbol to identify it as a Static and then marking the method being accessed with an @ symbol as in the examples below:

```
@java.lang.System@.out ("Hello World");
```

```
@com.mercury.diagnostics.capture.metrics.countingCollector@.incrementCounter();
```

The arguments in the code snippets support Java class syntax when the Java class is surrounded with a marker that the parser can get hold of.

The following examples show how to use the @ symbol as a marker:

```
@java.lang.System@
```

```
@java.lang.System@out (Static field)
```

➤ **Code Snippet Helper**

Some functionality is very hard, or even impossible to get coded using the limited syntax available within the code-snippets. Therefore, the code-snippet environment offers two helper classes, ProbeCodeSnippetHelper, and ProbeCodeSnippetV5. The latter is available only with Java 5 or later. The ProbeCodeSnippetHelper offers functionality as depicted below.

```
/*
 * (c) Copyright 2008 Hewlett-Packard Development Company, L.P.
 */

package com.mercury.opal.capture.proxy;

/**
 * Used to help out Code Snippets
 */
public class ProbeCodeSnippetHelper {

  /**
   * When a Special Field holds a reference to the string below,
   * it will not be stored in the Fragment Local Storage,
   * or Invocation Local Storage
   */
  public static final String DO_NOT_STORE = ...

  /**
   * Helper to convert an int to an Integer
   * @param i
   * @return a new Integer object having the value of i
   */
  public static Object intToInteger(int i) {
   ...
  }

  /*
   * Mark the current thread, if not marked yet
   * @return true, if and only if the thread had been already marked
   */
  public static boolean testAndSetRecursiveFlag() {
   ...
  }

  /*
   * Unmark the current thread
   */
  public static void clearRecursiveFlag() {
   ...
  }

  /**
   * Helper method to call ResourceBundle.getString() and catch any exceptions that
   *  might be thrown
   * @param theBundle the ResourceBundle on which to call getString
   * @param key the key to pass getString
   * @return the value returned from getString, or null if an exception occurred
   */
  public static String getStringFromResourceBundle(ResourceBundle theBundle, String key) {
   ...
  }

}
```

The CodeSnippetHelperV5 uses some APIs available only with Java 5 or later. The methods offered by this class are shown below.

```
/*
 * (c) Copyright 2008 Hewlett-Packard Development Company, L.P.
 */

package com.mercury.opal.capture.jdk15.agent;

/**
 * Used to help out Code Snippets using Java 5 or later
 */
public class ProbeCodeSnippetHelperV5 {

  /**
   * Get the annotation of the specified type from the class or its superclass,
   * or its implemented interfaces
   * @param theClass  The class to get the annotation for
   * @param annClass  The annotation class to look for
   * @return
   */
  public static Object getEndpointClassAnnotation(Class theClass, Class annClass) {

   ...
  }

  /**
   * Get the method annotation of the specified type from the class
   * or its superclass, or its implemented interfaces
   * @param theClass  the class
   * @param methodName  the method name
   * @param argCount  the argument count
   * @param annClass  the class annotation type
   * @param methodAnnClass  the method annotation type
   * @return
   */
  public static Object getEndpointMethodAnnotation(Class theClass, String methodName,
     String argCount, Class annClass, Class methodAnnClass) {

   ...
  }

  /**
   * Helper method to get an annotation element value. If the annotation
   * does not have the element, return null.
   * @param annClass   The class of the annotation
   * @param instance   The annotation instance object
   * @param elementName  The element name
   * @return  The element value for the annotation instance, or null
   */
  public static String getAnnotationElementValue(Class annClass, Object instance, String elementName) {

   ...
  }

  /**
   * This helper method is used to serialize a DOM document.
   * This method uses APIs available in DOM Level 3 or newer, which are
   * available with a 1.5 or later JVM.
   * @param document
   * @return The serialized form (XML) of the input DOM document
   */
  public static String serializeDOMToString(Document document) {

   ...
  }

}
```

➤ **Spanning multiple lines with the stack of method calls**

The stack of method calls in a code snippet can span multiple lines. The parser which builds the bytecode requires a "\" (back slash) before each carriage return when it is supposed continue parsing the stack of statements. The final line of the Code Snippet stack of statements should not contain a "\" (back slash) and should simply end with carriage return.

```
@java.lang.System@.out ("Hello World");\
"Callee Name="+#callee.getName().toString();
```

➤ **Casting**

When calling a method that returns an object, casting is typically required to call members on the returned object. Casting is supported on object references. To cast an object to another type, place the casting reference between the symbols "<" and ">" following the reference to that object. The following are examples of casting.

```
#arg1<com.myCompany.myFoo>.myMethod();
```

This is equivalent to the Java statement:

```
((com.myCompany.myFoo)arg1).myMethod();
```

```
@some.class.Foo@foo<com.myCompany.myFoo>.myMethod();
```

Would be equivalent to the java statement:

```
((com.MyCompany.myFoo)some.class.Foo.foo).doSomething();
```

```
#foo = #arg1<bar>.b(); #foo.toString();
```

Creates the following java equivalent:

```
String foo = ((Bar)arg1).b(); ((Object)foo).toString();
```

Note: Casting is not supported for special types such as #classloader.

➤ **Method calls**

Method calls can be included in snippet arguments. The support of method calls includes calls with or without arguments and method chaining. The following are examples of method calls that have been included in code snippet arguments:

```
#arg1.toString()
```

```
#arg2.getSomething().getSomethingElse()
```

```
#callee.getSomething("foo", #arg1).somethingElse()
```

```
@some.Class@.staticMethod()
```

The dot still needs to appear after the static reference for the method call to be parsed properly.

```
@java.lang.System@out.println("Here I am!")
```

To speed up the generation of bytecode at runtime (by avoiding reflection), you can to specify the type that is returned from a method as shown in the following example:

```
#arg1.getSomething()<some.class.Here>
```

This will not help if the method takes arguments, or if a static field is used.

➤ **Multiple statements**

Code snippets can include multiple statements in a single code snippet. This is necessary for instrumentation, such as **CLApplicationDiscoveryPoint**, that expect multiple objects to be left on the stack and it can be handy in other situations.

```
@java.lang.System@out.println("Look out!");
#arg2.getSomething();
```

➤ **Local Member assignment**

In addition to the default supported "local" variables, you can create your own local members to hold object references returned by called methods.

To create a new Local Member enter the "#" symbol before the name of the local member and the parser creates the local member for you.

```
#myBar = #arg2.getName();\
#myUpperBar = #myBar.toUpper();\
"Target Name=http://"+myUpperBar+"/services";
```

➤ **Special Field assignment (store-fragment)**

You can use the pre-defined special fields to store the object references returned by called methods.

To use a special fields enter the "##" symbols before the name of the special field along with the **store-fragment** detail keyword on the instrumentation point.

```
##WS_SOAP_fault_code = #arg2;\
##WS_SOAP_fault_reason = #arg3;\
##WS_SOAP_fault_detail = (#arg4 == null ? null : #arg4.toString());"";
```

➤ **Special Field assignment (store-thread)**

You can use the special fields to store the object references returned by called methods.

To use a special fields enter the "##" symbols before the name of the special field along with the **store-thread** detail keyword on the instrumentation point.

```
# Used by [SOA_Broker_Payload_Handler]
##SOA_Manager_Inbound_Payload=#callee.getRequestDocument();"";
```

In a later code snippet you can retrieve the value stored by calling getThreadContextValue with the special field value above without the leading ## symbols.

```
#temp_soam_payload=@com.mercury.opal.capture.proxy.ThreadContextProxy@.get
ThreadContextValue("SOA_Manager_Inbound_Payload");
```

In a later code snippet you can retrieve and remove the special field value stored by calling getAndRemoveThreadContextValue method with the value same above without the leading ## symbols. It's very important that you call getAndRemoveThreadContextValue to free memory and clear the way for the next occurrence.

```
#temp_soam_payload=@com.mercury.opal.capture.proxy.ThreadContextProxy@.
getAndRemoveThreadContextValue( ("SOA_Manager_Inbound_Payload");
```

➤ **Conditional Logic**

Code snippets syntax allows for limited conditional logic which is equivalent to the java if-else statement. This syntax allows you to compare object references of the same type using both the **==** and != operators. Literal value and primitive comparisons are not valid using this syntax.

The following is an example of how to compare references:

```
(value1 == value2 ? <if_True_codeSnippet>:<if_False_codeSnippet>)
```

The following is an example of how to verify that an object is not null before calling a method:

```
(#arg1 == null ? "Unknown" : #arg1.getSomething())
```

This would be equivalent to the following java statement:

```
if (arg1==null) return "Unknown" else return arg1.getSomething();
```

Securing Code Snippets

By default, you must specify a valid code-key along with the code snippet before the Probe will use the code snippet during instrumentation. Requiring the code-key prevents someone from accidently introducing instrumentation that could significantly increase the overhead of the Probe.

When you generate the code-key, Diagnostics checks the syntax of the code snippet to make sure that it is valid before it generates the key. When Diagnostics instruments an application, it checks the value entered for the code-key argument to make sure that it matches the code-key that it calculates for the code snippet for the point. If the code-keys do not match, Diagnostics ignores the code snippet and does not create the instrumentation point.

Generating the Code Snippet Code-Key

The Java Probe is installed with a tool that generates the code-key from the code snippet that you input.

To generate a code-key:

1 Open the page at the following URL in your browser: http://
<probe-host>:<probe-port>/inst/code-key. Diagnostics displays the page where
you can validate the code snippet syntax and generate the code-key as
shown in the following example:

2 Enter the code snippet that you specified in the code argument in the
auto_detect.points file into the **Input your code snippet** text box and click
Submit.

Note: The code snippet must include all of the text following the code =
argument name.

---

3 Diagnostics presents the results of the code snippet validation and the code-key generation in the **Resulting point section** text box.

If the code snippet was valid, Diagnostics displays the value of both the code-key and code arguments. Enter these values into the capture points file.

If the code snippet was not valid, Diagnostics displays an error message that indicates the problem that was detected. Correct the problem and click **Submit** again to validate the corrected code.

Disabling the Code-Key Security Check

By default, Diagnostics verifies that the value of the code-key argument matches the value that it generates when it is instrumenting the application. It is possible to disable this security check by inserting the **require.code.security.key** property into the **<probe_install_dir>/etc/ inst.properties** file with a value of false.

Note: You should be very careful when using this property. If you disable this check, you could allow unexpected processing overhead and unpredictable performance monitoring results.

Instrumentation Examples

The examples in this section have been provided to illustrate how you can customize the instrumentation of an application by creating and modifying the points in the capture points file.

This section includes the following examples:

➤ Custom layer and sub-layer

➤ Wildcard method

➤ Ignore Specified Methods

➤ Capture Methods for the Trended Methods View

- ➤ Capture Only a Specific Method In a Class
- ➤ Capture a Specific Method That Returns a String
- ➤ Capture with a Controlled Scope
- ➤ Hard and Soft deep_mode
- ➤ Argument Capture
- ➤ Inbound and Outbound Web Services
- ➤ Renaming root methods
- ➤ Adding a field to a class
- ➤ Passing attributes to instance trees
- ➤ Filtering out methods by their access flag
- ➤ Not recording direct recursion
- ➤ Performing caller-side instrumentation
- ➤ Configuring allocation analysis
- ➤ Configuring light-weight memory diagnostics (LWMD)
- ➤ Enabling object lifecycle monitoring for JDBC result set
- ➤ Adding a disabled point and enabling it at runtime
- ➤ Specifying that a method should never be trimmed
- ➤ Specifying that a method should always be trimmed
- ➤ Enabling collection of CPU time for a method
- ➤ Printing instrumentation and runtime information for a point (debugging only)
- ➤ Using thread local storage to store the SOAP payload
- ➤ Using fragment local storage to store Web service field

Custom layer and sub-layer

➤ The point in the following example creates a custom sub-layer called "BAR" within the layer called "FOO" for the method myMethod in myCompany.myFoo class:

```
[myCompany.myFoo_customLayer]
class = myCompany.myFoo
method = myMethod
signature = !.*
layer = FOO/BAR
```

Wildcard method

➤ The point in the following example captures all methods in the MyCompany.MyFoo class:

```
[myCompany.myFoo_AllMethods]
class = myCompany.myFoo
method = !.*
signature = !.*
layer = FOO/BAR
```

Ignore Specified Methods

➤ The point in the following example captures all methods in the MyCompany.MyFoo class except for the methods setHomeInterface and getHomeInterface:

```
[myCompany.myFoo_AllMethodsExcept]
class = myCompany.myFoo
method = !.*
ignoreMethod = !setHomeInterface.*, !getHomeInterface.*
signature = !.*
layer = FOO/BAR
```

➤ The point in the following example captures all methods in the
MyCompany package/namespace except for those contained in the
MyCompany.logging class:

```
[myCompany_All_Methods_except_from_MyCompany_Logging]
class = !myCompany\..*
method = !.*
ignore_cl = MyCompany.logging
signature = !.*
layer = FOO/BAR
```

Capture Methods for the Trended Methods View

➤ The point in the following example captures the required data to populate
the Trended Methods View for the myMethod method:

```
[myCompany.myFoo_customLayer]
class = myCompany.myFoo
method = myMethod
signature = !.*
layer = FOO/BAR
layertype = trended_method
```

Capture Only a Specific Method In a Class

➤ The point in the following example captures all methods in the constructor
for the MyCompany.MyFoo class:

```
[myCompany.myFoo_Constructor]
class = myCompany.myFoo
method = <init>
signature = !.*
layer = FOO/BAR
```

➤ The point in the following example captures all methods in the singleton constructor for the MyCompany.MyFoo class:

```
[myCompany.myFoo_Singleton]
class = myCompany.myFoo
method = <clinit>
signature = !.*
layer = FOO/BAR
```

➤ The point in the following example captures the setFoo method in the MyCompany.MyFoo class:

```
[myCompany.myFoo_setFoo]
class = myCompany.myFoo
method = setFoo
signature = !.*
layer = FOO/BAR
```

➤ The point in the following example captures all "set" methods in the MyCompany.MyFoo class:

```
[myCompany.myFoo_AllSets]
class = myCompany.myFoo
method = !set.*
signature = !.*
layer = FOO/BAR
```

➤ The point in the following example captures all methods in the MyCompany package/namespace:

```
[myCompany_All_Methods]
class = !myCompany\..*
method = !.*
signature = !.*
layer = FOO/BAR
```

Capture a Specific Method That Returns a String

➤ The point in the following example captures the getFoo method that returns a java.lang.String in the MyCompany.MyFoo class:

```
[myCompany.myFoo_GetFoo_String]
class = myCompany.myFoo
method = getFoo
signature = ()Ljava\lang\String
layer = FOO/BAR
```

Capture with a Controlled Scope

➤ The point in the following example captures all methods in the MyCompany package/namespace that are called from the MyCompany.logging class. For more details see "Using Caller Side Instrumentation" on page 271.

```
[myCompany_All_Methods_from_MyCompany_Logging]
class = !myCompany\..*
method = !.*
signature = !.*
scope = MyCompany.logging
layer = FOO/BAR
```

➤ The ignoreScope argument is used to exclude certain packages, classes, and methods from those included in the scope specified in scope argument. The point in the following example captures all methods in the MyCompany package/namespace that are called from the MyCompany.logging class except for those called from the myMethod method. For more details see "Using Caller Side Instrumentation" on page 271.

```
[myCompany_All_Methods_except_from_MyCompany_Logging]
class = !myCompany\..*
method = !.*
signature = !.*
scope = MyCompany.logging
ignoreScope = MyCompany.logging\myMethod
layer = FOO/BAR
```

Hard and Soft deep_mode

The following interface definition will be used for both soft and hard deep_mode examples:

```
public interface Interface1 {

    public void callerMethod();

}
```

The following class will be used for both soft and hard deep_mode examples:

```
public class Class1 implements Interface1 {
    public void callerMethod(){
        calleeMethod();
        calleeMethod2();
    }

    public void calleeMethod(){
        System.out.println("hello world");
//more code lines here…
    }

    public void calleeMethod2(){
        System.out.println("hello world 2");
    }
}
```

➤ The point in the following example captures the "callerMethod" in the Class1 class:

```
[Training-1]
class     = Interface1
method    = !.*
signature = !.*
deep_mode = soft
layer     = Training
```

➤ The point in the following example captures all methods in Class 1 (i.e. "callerMethod", "calleeMethod1" and "calleeMethod2"):

```
[Training-1]
class    = Interface1
method   = !.*
signature = !.*
deep_mode = hard
layer    = Training
```

Argument Capture

The argument displayed in Diagnostics is the final string left on the stack by the code snippet. Code snippets must end with a string or an object where toString() can be left on the stack of statements to be parsed to the bytecode.

➤ Suppose that we are instumenting a method myCompany.myFoo.myMethod() and suppose that "myFoo" has another method called getComponentName() which returns a String. The following example will display the result of getComponentName() as the argument in Diagnostics (note that #callee refers to the callee object for an instance method, in this case).

```
[myCompany_componentName_as_argument]
class = myCompany.myFoo
method = myMethod
signature = !.*
detail = before:code: 8d2509eb
layer = FOO/BAR
```

The code snippet in the **custom_code.properties** file is entered as follows:

```
8d2509eb = #callee.getComponentName()
```

➤ The point in the following example captures the first argument to myMethod and shows it as the captured argument in Diagnostics. It also uses it as the sub-layer name. This is achieved by including ${ARG} in the layer. In this example, if the captured argument (in this case the first argument of "myMethod"), has a value "myArg", then the layer will be FOO/myArg.

```
[myCompany_capture_firstArg_and_also_show_as_layer]
class = myCompany.myFoo
method = myMethod
signature = !.*
detail = before:code: 358f05d6
layer = FOO/${ARG}
```

The code snippet in the **custom_code.properties** file is entered as follows. Note that if we used #arg2, we would be capturing the second argument instead.

```
358f05d6 = #arg1.toString()
```

Inbound and Outbound Web Services

When the detail argument in a point contains the "outbound" or "ws-operation" keyword, Diagnostics attempts to parse the final string on the Code Snippet stack for additional information to display about the method call.

➤ For inbound Web Services ("ws-operation" detail must be used), the string looks like this:

```
"DIAG_ARG:type=ws&ws_name="+<WebServiceName>+"&ws_op="+
<OperationName>+"&ws_ns="+<TargetNameSpace>+"&wsOport="+<wsPort>
```

➤ For outbound Web Services ("outbound" detail must be used), the string looks like this:

```
"DIAG_ARG:type=ws&ws_name="+<WebServiceName>+"&ws_op="+
<OperationName>+"&target="+<TargetName>
```

For example:

```
class    = weblogic.wsee.ws.WsStub
method   = invoke
signature = (Ljava/lang/String;Ljava/lang/String;Ljava/util/Map;Ljava/util/Map;)Ljava/
lang/Object;
layer    = Web Services
detail   = outbound,before:code:edd75e36
```

The code snippet in the **custom_code.properties** file is entered as follows

```
edd75e36 = #service = #callee.getService().getWsdlService();\
#qname = #service.getName();\
"DIAG_ARG:type=ws&ws_name="+ #qname.getLocalPart() +"&ws_op="+ \
#callee.getMethod(#arg1).getOperationName().getLocalPart() +"&target="+ \
#callee.getProperty("javax.xml.rpc.service.endpoint.address");
```

Renaming root methods

➤ Let's consider the following point:

```
class       = Statement
method   = execute
layer       = Database/JDBC/Execute
detail      = when-root-rename
rootRenameTo   = mySuffix
```

If such a method ends up being the root method, then the name of such a
server request will be "Background-mySuffix" and the type of the server
request will be "RootRename".

➤ Now let's consider this point instead:

```
class      = Statement
method   = execute
layer       = Database/JDBC/Execute
detail      = when-root-rename
```

Notice we skipped the rootRenameTo property. The name of such a server
request will be "Background - Database" (since Database is the first sub-layer)
and the server request type will again be "RootRename".

Adding a field to a class

➤ Let's consider the following point:

```
class    = com.corp.Foo
method   = bar
detail   = add-field:protected:Object:serviceName
```

The detail above will cause the following one field and two public setter/getter methods to be added to the class com.corp.Foo:

```
protected transient Object serviceName
public void _diag_set_serviceName(Object arg)
public Object _diag_get_serviceName()
```

Passing attributes to instance trees

➤ The following example will attach "my_attribute" name to every captured instance of com.corp.Foo.bar()

The above name (prefixed with "display_") and its corresponding value will be shown in the call profile.

```
class    = com.corp.Foo
method   = bar
detail   = store-method,code:f59f0c5c
```

Code snippet:

```
f59f0c5c = ##my_attribute="value-of-my-attribute";"";
```

Filtering out methods by their access flag

➤ The following example will instrument all methods in class com.corp.Foo (but not static methods).

```
class    = com.corp.Foo
method   = !.*
signature    = !.*
method_access_filter = static
```

Not recording direct recursion

➤ In the following example, if method com.corp.Foo.bar calls itself (or anything in the same layer), the second call isn't recorded. This is caused by the **detail = no-layer-recurse**.

However, note that this is only for direct recursion. If com.corp.Foo.bar calls a different instrumented method, which calls this method again, then all methods will be recorded.

```
class    = com.corp.Foo
method  = bar
layer    = Example/MyBar
detail   = no-layer-recurse
```

Performing caller-side instrumentation

➤ The following point will cause caller-side instrumentation to be performed (as opposed to the default callee instrumentation). This is caused by the **detail = caller**.

Note that another way to do caller-side instrumentation is to use the "scope" property as described earlier in the section. See "Using Caller Side Instrumentation" on page 271

```
class    = com.corp.Foo
method  = bar
detail    = caller
```

Configuring allocation analysis

Both of the following examples will track allocations of java.lang.Integer in the package com.mycompany.mycomponent.

There are however 2 differences:

➤ In the first example **(detail = leak)**, tracking is managed, i.e. starts when the user clicks "start" in the profiler and stops when the user clicks on "stop". In the second example **(detail = deallocation)**, tracking starts with application startup

➤ In the first example, the point is disabled when it comes to regular instrumentation. This means that you will not see "new Integer" show up on an instance tree. In the second example, you will.

Example 1 - Managed. Tracking starts when user clicks on "start" and stops when user clicks on "stop" in the profiler:

```
[Leak]
scope   = !com\.mycompany\.mycomponent\..*
class   = java.lang.Integer
keyword = allocation
detail  = leak
active  = true
```

Example 2 - Unmanaged. Tracking starts with application startup:

```
[Leak]
scope   = !com\.mycompany\.mycomponent\..*
class   = java.lang.Integer
keyword = allocation
detail  = deallocation
active  = true
```

Note that neither of these points will capture reflected allocation. To enable reflected allocation capture, simply append the detail "reflection" to the point (i.e. **detail = leak,reflection**)

Configuring light-weight memory diagnostics (LWMD)

➤ The following example turns on collection diagnostics for collections that happened inside of the com.mercury.mycomponent package. You can find this example in the **auto_detect.points** file. It is set to active = false by default.

You also need to set the property **lwm.diagnostics.capture=true** in the **dynamic.properties** file. For more information refer to the *HP Diagnostics User's Guide* chapter on the "Collections and Resources View".

```
[Light-Weight Memory Diagnostics]
scope   = !com\.mycompany\.mycomponent\..*
class   = java.lang.Integer
keyword = lwmd
active  = true
```

Enabling object lifecycle monitoring for JDBC result set

There are a few pre-configured instrumentation points which allow object lifecycle monitoring but they are disabled by default. Two of them are shown below for convenience. This example shows how to enable object lifecycle monitoring for JDBC Result Sets. For a more detailed discussion on object lifecycle monitoring, please refer to the *HP Diagnostics User's Guide*, chapter on "Analyzing Memory and Object Lifecycle" in the section on the Allocation /Lifecycle Analysis Tab.

For this example, there are two actions required:

1 Go to inst.properties and find details.conditional.properties. Set "mercury.enable.resourcemonitor.jdbcResultSet=true".

2 Specify the scope in the corresponding "open" instrumentation points (shown below).

In the below example, the probe will perform object lifecycle monitoring for JDBC Result Sets inside package com.mycompany.mycomponent.

```
[Lifecycle-JDBC-ResultSet-Open]
scope    = !com\.mycompany\.mycomponent\..*
class    = java.sql.Statement
method  = !(getResultSet.*)|(executeQuery.*)
signature = !.*\)Ljava/sql/.*ResultSet;
detail = condition:mercury.enable.resourcemonitor.jdbcResultSet,lifecycle,caller

[Lifecycle-JDBC-ResultSet-Close]
class    =
!(java\.sql\.ResultSet)|(weblogic\.jdbc\.wrapper\.ResultSet)|(com\.ibm\.ws\.rsadapter\.jd
bc\.WSJdbcResultSet)
method  = !(close)|(closeWrapper)
signature  = !.*
deep_mode  = soft
detail =
condition:mercury.enable.resourcemonitor.jdbcResultSet,before:code:513a2b36,metho
d-trim
```

Adding a disabled point and enabling it at runtime

➤ In the following example, the point is disabled. This does not mean that instrumentation did not happen. It just means that the instrumentation did happen but it does not collect any data. This significantly lowers the runtime overhead of such a point.

To enable the data collection while the application is running, go to the http://<probe-host>:<probe-port>/inst/layer page, look for layer **myLayer** and click **Enable**.

```
[My Example]
class    = Example
method = !.*
layer   = myLayer
detail   = disabled
```

If you don't want instrumentation to happen at all, then you need to use **active=false**. However, such a point cannot be enabled at runtime.

Specifying that a method should never be trimmed

➤ In the following example, latency trimming will not apply to
Example.myMethod().

```
[My Example]
class    = Example
method = myMethod
detail    = method-no-trim
```

Specifying that a method should always be trimmed

➤ In the following example, the method Example.myMethod() will not be
reported.

```
[My Example]
class    = Example
method = myMethod
detail    = method-trim
```

Enabling collection of CPU time for a method

➤ In the following example, the detail "method-cpu-time" will cause the CPU
time to be collected for method Example.myMethod().

```
[My Example]
class    = Example
method = myMethod
detail    = method-cpu-time
```

Printing instrumentation and runtime information for a point (debugging only)

The following example will print several pieces of debug information in
standard out and probe.log. In particular:

➤ The **gen-instrument-trace** detail will cause printing to stdout the thread
stack trace whenever this point is used to instrument a method.

➤ The **gen-runtime-trace** will cause printing to stdout the thread stack trace
whenever Example.myMethod() is executed.

➤ The **trace** detail will cause printing in the probe.log verbose instrumentation information whenever Example.myMethod() is executed.

```
[My Example]
class    = Example
method = myMethod
detail    = gen-instrument-trace, gen-runtime-trace, trace
```

Understanding the Overhead of Custom Instrumentation

When you are creating custom instrumentation you should beware of over-instrumenting the application because it can introduce excessive latency into the probed application. The excessive latency arises from an increase in the classloader latency as more and more classes are instrumented. The custom instrumentation does not have the same impact on the method latency or the CPU overhead because the overhead of instrumentation is nearly fixed for every method because the amount of bytecode is almost always the same. This means that the physical percentages of the CPU and latency overhead will vary in direct proportion to the length of time the method takes to execute.

For example if a method take 100ms and instrumentation makes it execute in 101ms, then overhead is 1%. If a method takes 10ms and instrumentation changes it's response to 11ms, then overhead is 10%. If this method is not called very often its overall latency effect on the application is minimal. However, the overall latency effect of an instrumented method that is called more frequently could have an impact on the latency of the application's response even though is overhead percentage is much smaller.

Unlike a traditional profiler, HP Diagnostics uses bytecode instrumentation. This allows the default instrumentation to be selective so as to minimize the overhead caused by instrumentation to an average of 3-5%. Methods with higher latency overhead introduced by instrumentation are only instrumented when they are called infrequently in relation to other components in the application and when the instrumentation provides specific information needed for triage activities (i.e. JNDI lookups).

You should also consider Diagnostics data overhead when you are customizing the instrumentation for an application. The more methods that you instrument the more data that the Probe must serialize and pass over the network to the Diagnostics Server. You can tune the Probe's default configuration so that it can adjust the volume of Diagnostics data to avoid any unnecessary effect on the performance of the system being monitored. Improper Probe tuning can cause CPU, Memory and Network overhead on the physical machine where your Probe resides. For more information about managing Latency, CPU, Memory and Network overhead, see Chapter 15, "Advanced Java Probe and Application Server Configuration."

Instrumentation Control

By default, the layers defined in the capture points file are enabled. If you include the details=disabled argument in a point, the layer is disabled when the Probe is started.

The classmap in JDK 1.5 provides the capability to dynamically instrument methods and classes using the JVMTI interface without restarting the JVM instance. All other virtual machines require that the JVM instance be restarted in order to apply changes that you make to the capture points files. Once instrumentation is placed within a method, its data collection and running CPU and method latency overhead can be controlled on a per layer basis using the Probe's Instrumented Layers page.

You can access the Instrumented Layers page from the URL: http://<probe-host>:<probe-port>/inst/layer.

To disable a layer from the Instrumented Layers page, click the **Disable** link associated with the layer on the page. The layer is disabled and the link toggles to **Enabled** so that you can enable the layer again when necessary.

Advanced Instrumentation

This section includes:

➤ "Using Caller Side Instrumentation" on page 271

➤ "URI Aggregation Instrumentation" on page 273

➤ "Using RMI Instrumentation" on page 274

➤ "Using thread local storage to store the SOAP payload" on page 275

➤ "Using fragment local storage to store Web service field" on page 276

➤ "Using Annotations for Custom Instrumentation" on page 280

Using Caller Side Instrumentation

By default all instrumentation in Diagnostics is Callee side instrumentation where the bytecode is placed within the method call. Caller side instrumentation refers to the process of placing the bytecode for measurement around the call to the method to be instrumented instead of within.

Caller side instrumentation allows finer control of instrumentation placement, but may increase application classloader time as each class specified in the scope must be checked for references to the class/method that is specified in the points.

A common use for caller side instrumentation is to instrument calls to methods in rt.jar. Classes loaded into the virtual machine using the bootclassloader and not from a conventional class loader cannot be directly instrumented. To instrument calls to these methods you must use caller side instrumentation.

In the point in the following example, the "parse" methods for the **javax.xml.parsers.SAXParser** and **javax.xml.parsers.DocumentBuilder** is instrumented by placing bytecode around the calls to "parse" in any (!.*) method from any class. Caller side instrumentation is required because both the **javax.xml.parsers.SAXParser** and **javax.xml.parsers.DocumentBuilder** classes are contained in the rt.jar and loaded into the virtual machine by the bootclassloader.

```
[XML-DOM-JDK14]
;---------- Interface -------------
Class = !javax\.xml\.parsers\.(SAXParser|DocumentBuilder)
method    = parse
signature  = !.*
scope = !.*
layer   = XML
```

➤ The point in the following example instruments calls to javax.naming.Context's "lookup" method that are called from the com.myCompany.myFoo classes and places them in the JNDI sublayer in the FOO layer.

```
[JNDI-lookup-FOO]
;------------- Server side JNDI hook --------------
class     = javax.naming.Context
method    = lookup
signature = (Ljava/lang/String;)Ljava/lang/Object;
detail    = args:1
scope = !com\.myCompany\.myFoo\..*
deep_mode  = soft
layer   = FOO/JNDI
```

Notes:

➤ This instrumentation should be placed above any other points that may be in conflict.

➤ To verify that the point has caused the bytecode to be properly placed, check the **<probe_Install_dir>/log/<probeName>/detailReport.txt** file for the entries Unique Header Name (i.e. [JNDI-lookup-FOO]).

➤ During final triage steps for a performance issue in an application it may be impractical to use the classmap and individually build points for every method called by a suspect area of the application. It is very common to use one or more levels of caller side instrumentation to identify the time spent within an individual method or methods that have a suspected bottleneck. This is a useful way to fill in the next level to a Call Profile in Diagnostics.

The point in the following example instruments any call to a method that is performed within the **com.myCompany.myFoo** class by the "myMethod" method:

```
[MethodsCalledByFoo.myMethod]
class = !.*
method = !.*
scope = !com\.myCompany\.myFoo\.myMethod.*
layer = FOO/other
```

The following is another example point that would capture the arguments to any "set" method called in **com.myCompany.myFoo** class by the "myMethod" method:

```
[SetMethodsCalledByFoo.myMethod]
class = !.*
method = !set.*
scope = !com\.myCompany\.myFoo\.myMethod.*
detail = args:1
layer = FOO/other
```

URI Aggregation Instrumentation

Applications typically use the same URL to access different workflow. If the application uses a URI (i.e. http://server/myApplication?page=home) argument to differentiate the between the workflow, Diagnostics can be configured to parse and treat the different URIs as different server requests.

URI aggregation is controlled from the [HttpCorrelation] point. A valid regular expression entry for "args_by_class" should be created for each URI pattern.

The point example below allows the following ServerRequests to appear uniquely in the Diagnostics console:

```
http://server/myApplication?page=home
http://server/myApplication?page=openReport
```

```
[HttpCorrelation]
args_by_class=!.*&page
```

More than one URI parameter can be used for URI parsing as shown in the following example:

```
args_by_class=!.*&page&role
```

Note: You should avoid using a session parameter or highly unique URI value because of the impact to overhead and data storage.

In a WebLogic environment you should set the **use.weblogic.get.parameter=true** in **<probe_install_dir>/etc/inst.properties** when using URI aggregation to prevent URI aggregation from consuming the ServletRequest's inputstream.

Using RMI Instrumentation

The RMI (Cross-VM) point in the capture points file is inactive by default. You must activate this point to capture the cross-vm processing in the application. If you have Probes with this point activated on both sides of an RMI call, Diagnostics can correlate the call tree data from both virtual machines.

```
[RMI]
keyword = rmi
layer   = CrossVM
active  = false
```

RMI Instrumentation In a Clustered Environment

The **weblogic.t3.rmi** property in the **<probe_install_dir>/etc/inst.properties** file controls how the RMI instrumentation captures Cross-VM RMI performance metrics. By default, **weblogic.t3.rmi** is set to false which causes the performance metrics to be gathered using the generic RMI instrumentation. In a clustered environment, all servers in a cluster must have RMI instrumentation turned on in order to avoid application failure when **weblogic.t3.rmi** is set to false.

When **weblogic.t3.rmi** is set to true, the generic RMI instrumentation is disabled and the RMI Cross VM is captured using only WebLogic's T3 protocol. This allows the Probe to function with only some of the servers in a cluster probed with RMI instrumentation enabled.

Using thread local storage to store the SOAP payload

This example demonstrates usage of thread local storage. In particular it shows how to store (and clean) the SOAP payload from thread local storage. Please note that SOAP payload is captured by default only for certain application servers. For more information on our support matrix, see "Configuring SOAP Fault Payload Data" on page 424.

This example is applicable only for those application servers where Diagnostics does not capture payload out of the box.

First we need to identify where to access the payload from. Assume that the payload is the second argument of a method called DispatchController.dispatch(). We would then have the following point.

The keyword store-thread is telling the probe to store the special fields in the corresponding code snippet (in this case My_Inbound_Payload) into thread local storage. This can be referenced from a different code snippet as long as both points are hit on the same thread. Looking up the payload is demonstrated in the next example ("Using fragment local storage to store Web service field" below).

```
[MyAppServer-SoapPayload-Capture]
class    = com.myCompany.DispatchController
method = dispatch
signature = !\(Ljava/lang/Object;Ljava/lang/Object;\).*
layer   = Web Services
detail   = before:code: ae7f0a58,store-thread


# Used by [MyAppServer-SoapPayload-Capture]
ae7f0a58 = ##My_Inbound_Payload=#arg2;"";
```

Using fragment local storage to store Web service field

This example demonstrates several features of points and code snippets:

➤ how to use fragment local storage to store web service specific fields (ws_name, ws_op, etc). This is an alternative to specifying the "DIAG_ARG" string.

➤ how to retrieve (and remove) the stored payload from thread local storage (which we stored in the previous example)

➤ how to extract the consumer ID out of the SOAP payload

➤ how to use fragment local storage to store the consumer ID

Web Services are treated in a special way within Diagnostics so several fields need to be captured. Those fields are described in "Code Snippet Grammar" on page 239.

The first step is to find the instrumentation points that will give us access to the required fields (Web Service name, operation, namespace, port name). Suppose that there is a single class in our instrumented application that has access to all these fields. Let's assume that this class is called com.myCompany.MyWSContext. We need to access an instance of this class when all the above fields are set. There can be many options. Suppose that one such option is when MyWSContext is passed as the first argument of a method MyWSFactory.create(). Then this is the method we want to instrument.

Here is our instrumentation point (each line is explained later):

```
class     = com.myCompany.MyWSFactory
method    = create
signature = !\(Lcom/myCompany/MyWSContext;.*
layer     = Web Services
detail    = ws-operation, before:code: f334f0b4,store-fragment
```

The first 3 lines of the point are telling the probe to instrument anything that matches com.myCompany.MyWSFactory.create(MyWSContext, *).

The 4th line is specifying the layer for this point.

The 5th line is giving additional information to the probe about this point (details):

➤ The first detail (ws-operation) is important because it tells the probe to treat this as an inbound Web Service.

➤ The second detail (before:code: f334f0b4) is telling the probe to insert the corresponding code snippet at the start of the methods that comply with this point. The actual code snippet is shown below. The number f334f0b4 was generated by going to http://<probe-host>:<probe-port>/inst/code-key and pasting the code snippet in the text box.

➤ The third detail (store-fragment) is telling the probe to store all special fields (##) found in the corresponding code snippet as attributes of the server request.

Here is the corresponding code snippet (each line of the below code snippet is explained later).

```
f334f0b4 = #wsContext=#arg1;\
##WS_inbound_service_name=#wsContext.getServiceName();\
##WS_inbound_operation_name=#wsContext.getOperationName();\
##WS_inbound_target_namespace=#wsContext.getNamespaceURI();\
##WS_inbound_port_name=#wsContext.getEndpoint();\
#soap_payload =
@com.mercury.opal.capture.proxy.ThreadContextProxy@.getThreadContextVal
ue("My_Inbound_Payload");\
#consumer_id = (#soap_payload == null ? null :
@com.mercury.opal.capture.proxy.ProbeCodeSnippetHelper@.getConsumerId
FromDocument(##WS_inbound_service_name<java.lang.String>,#soap_payloa
d<org.w3c.dom.Document>));\
##WS_consumer_id = (#consumer_id == null ?
@ProbeCodeSnippetHelper@DO_NOT_STORE : #consumer_id);"";
```

First line: f334f0b4 = #wsContext=#arg1;\

As mentioned earlier, the number f334f0b4 was generated by going to http://<probe-host>:<probe-port>/inst/code-key and pasting the code snippet in the text box. The actual code snippet starts after f334f0b4 =. The first expression is #wsContext=#arg1; and it simply assigns the first argument of the method (in this case a MyWSContext object) to a local variable (wsContext).

Second line: ##WS_inbound_service_name=#wsContext.getServiceName();\

This expression uses fragment local storage to store the service name. It is important to use the exact variable name (WS_inbound_service_name). These variable names are documented in the "Special Fields" section of "Code Snippet Grammar" on page 239.

Third line: ##WS_inbound_operation_name=#wsContext.getOperationName();/

This expression uses fragment local storage to store the ws operation. It is important to use the exact variable name (WS_inbound_operation_name). These variable names are documented in the "Special Fields" section of "Code Snippet Grammar" on page 239.

Fourth line: ##WS_inbound_target_namespace=#wsContext.getNamespaceURI();\

This expression uses fragment local storage to store the ws namespace. It is important to use the exact variable name (WS_inbound_target_namespace). These variable names are documented in the "Special Fields" section of "Code Snippet Grammar" on page 239.

Fifth line: ##WS_inbound_port_name=#wsContext.getEndpoint();\

This expression uses fragment local storage to store the ws port name. It is important to use the exact variable name (WS_inbound_port_name). These variable names are documented in the "Special Fields" section of "Code Snippet Grammar" on page 239.

The above first five lines are sufficient to successfully capture the inbound Web Service. The remaining of the code snippet deals with capturing the consumer ID out of the SOAP payload. This is optional and only if the instrumented application server is not one of the application servers we support for capturing SOAP payload out of the box. See previous example for details on this. In this example we refer to the SOAP payload that was captured in the previous example.

Sixth line: #soap_payload = @com.mercury.opal.capture.proxy.ThreadContextProxy@.getAndRemoveThreadContextValue("My_Inbound_Payload");\

This expression is retrieving and removing the stored payload from thread local storage (see previous example on how this was stored) and storing it on a local variable (soap_payload).

Seventh line: #consumer_id = (#soap_payload == null ? null : @com.mercury.opal.capture.proxy.ProbeCodeSnippetHelper@.getConsumerId FromDocument(##WS_inbound_service_name<java.lang.String>,#soap_payloa d<org.w3c.dom.Document>));\

This expression is setting a consumer_id local variable. If the payload was null, the consumer_id is set to null. Otherwise, we will use the service name to perform consumer ID matching based on the consumer.properties entries. For more information on consumer ID matching, please refer to "Configuring Consumer IDs" on page 415.

Eighth line: ##WS_consumer_id = (#consumer_id == null ? @ProbeCodeSnippetHelper@DO_NOT_STORE : #consumer_id);"";

In this final line, this consumer ID local variable becomes the consumer id for this server request. It is important to use the exact variable name (WS_consumer_id). These variable names are documented in the "Special Fields" section of "Code Snippet Grammar" on page 239.

Using Annotations for Custom Instrumentation

Applications which are using version 1.5 or greater of the JVM can "force" the instrumentation of methods by simply using a custom annotation (InstrumentationPoint) which is contained in the **annotation.jar** file found in the Diagnostics Java Agent lib directory. Put a copy of this file in your classpath when compiling your classes using the InstrumentationPoint annotation. The annotation is defined as follows (InstrumentationPoint.java):

```
/*
 * (c) Copyright 2008 Hewlett-Packard Development Company, L.P.
 * --------------------------------------------------------------------------
 */
@Retention(value = RetentionPolicy.RUNTIME)
@Target(value = ElementType.METHOD)
public @interface InstrumentationPoint {
  String layer();
  String keyword() default "";
  String layerType() default "method";
  String detail() default "";
  String code() default "";
  Boolean active() default true;
}
```

This feature requires that the **look.for.annotations** property in inst.properties is set to true (default).

Development

1 Add the path to the **annotation.jar** (or copy the jar into your application) file found in the Diagnostics Java Agent lib directory to your application build classpath.

2 Import the classes for any methods that need to be monitored:

import com.mercury.diagnostics.common.api.InstrumentationPoint;

3 Identify methods to be monitored and add the annotation:

@InstrumentationPoint(ARGS)

public void launchTest4()

where ARGS include the following (please refer to points file documentation for more information about what these arguments mean):

- ➤ layer="layer name"
- ➤ keyword= "keyword"
- ➤ layerType="type"
- ➤ detail="details"
- ➤ active="true/false"

Example

Here is an example of code using the InstrumentationPoint annotation:

```
/*
 * (c) Copyright 2008 Hewlett-Packard Development Company, L.P.
  ----------------------------------------------------------------------
 */

import  com.mercury.diagnostics.common.api.InstrumentationPoint;

…

@InstrumentationPoint(layer="my_app",detail="diag,method-no-trim,method-cpu-time")
public void myMethod1(Object x, String y) {
    …
}
```

In the above example, myMethod1 will get instrumented and will be visible as a node in all instance trees. It will not get trimmed, even if its latency goes below the minimum method latency threshold (51 ms by default). Furthermore, the inclusive (including children) CPU consumption by this method will be measured and reported.

10

Custom Instrumentation for .NET Applications

Provides instructions for how to control the instrumentation that HP Diagnostics applies to the classes and methods of the applications to enable it to gather the performance metrics.

This chapter includes:

About Instrumentation and Capture Points Files

Instrumentation refers to bytecode that the probe inserts into the class files of the application as they are loaded by the CLR. Instrumentation enables a probe to measure execution time, count invocations, catch exceptions and correlate method calls and threads. The instrumentation points for each probe instance are specified in the capture points file.

The capture points file is an instrumentation mechanism that allows you to control the scope of the instrumentation so that Diagnostics can give you all of the information that you need to understand the performance of the applications without overwhelming you with costly or confusing extraneous information. The instrumentation definitions contained in the capture points file are called *points* that tell the probe which methods to instrument, how they should be instrumented, and which instrumentation should be installed.

Points can include regular expressions that "wildcard" the instructions so that they apply to more than one method, class or namespace specification. For more information about using regular expressions, see Appendix J, "Using Regular Expressions."

You can customize the points in the capture point file to include methods, classes, and namespaces - for areas of the application that do not fall within the default points.

The Microsoft specification for .NET does not include a unified or recommended interface that business logic should implement except in the case of instrumentation for Web and WCF methods. This means that the .NET probe will almost always require custom points in the capture points file in order to enable it to gather meaningful metrics for the performance of the business logic classes and methods in .NET applications.

The points in the capture points file are grouped into layers. Layers are used to organize the performance metrics into meaningful tiers of information that can be compared as part of a triage process and to control the collection behavior of the instrumentation.

The points in the capture points files that are installed with the Probe are grouped into default layers. You can customize the default layers and create new layers (see Chapter 12, "Instrumentation Layers").

Locating the .NET Probe Capture Points Files

When you install the .NET Agent, predefined default capture points files are installed.

Default capture points files for ASP.NET applications are located at **<probe_install_dir>\etc** and include: **Asp.Net.points, Ado.points** and **WCF.points**.

In addition, the .NET Agent installer automatically creates a separate capture points file for each IIS deployed ASP.NET Application Domain that it detects. You must modify the automatically detected and created points file to enable custom instrumentation points for the Application Domain. These capture points files are located at **<probe_install_dir>\etc\<ApplicationDomain>.points** file. These points files are read along with the default points files by the .NET Agent.

At installation, only the **Asp.Net.points**, **Ado.points** and **WCF.points** default points files are enabled. The following default .NET points files are added but not enabled:

Default Point File (initially disabled)	Instrumentation Target
Asp.Net.IExecutionStep.points	IIS5, IIS6 and IIS7. This file makes the IIS points obsolete.
IIS.points	IIS5 and IIS6
Lwmd.points	Lightweight Memory Diagnostics
Msmq.points	Microsoft Message Queuing
Remoting.points	.NET Remoting
WebServices.points	ASP.NET Web Services

You can enable the above points files by adding a reference to them in the **probe_config.xml** file. See Chapter 16, "Understanding the .NET Probe Configuration File" for details on each element in the probe_config.xml file.

For information on .NET probe instrumentation specific to TransactionVision see the *HP TransactionVision Deployment Guide*.

Important:

➤ It is not recommended to modify any of the default points files, because in the case of an installation upgrade, your modifications will be lost. Application-specific instrumentation points should be stored in a custom capture points file that was either automatically or manually created.

Coding Points in the Capture Points File

The following is a list of the arguments that can be used to define a point in the capture points file:

```
[Point-Name]      =<unique name for the point>
;------------------------------------------------------------------------
class             = <class/package name/s to capture>
method            = <method name/s to capture>
signature         = <signature/s of method/s>
ignoreClass        = <classes to ignore>
ignoreMethod      = <method prototypes to ignore>
ignoreTree        = <class hierarchy to ignore>
deep_mode         = <soft or hard mode>
scope             = <comma separated list of methods>
ignoreScope       = <comma separated list of methods>
keyword            = <keyword>
layer             = <layer name>
layerType         = <layer type>
```

The arguments are described in the following sections.

Mandatory Point Arguments

Every point, except for the points for LWMD, HttpCorrelation, WSCorrelation and WCF, must contain the following arguments:

Argument	Description
Point-Name	A unique name for the point.
class	The class argument specifies the name of the class or interface to be instrumented. The name should include the full namespace name using periods between the namespace and class levels. Any valid regular expression may be used.
method	The method argument specifies the name of the method to be instrumented. To be successful, the method name must match a method defined in the class or interface specified by the class argument. Any valid regular expression may be used.
layer	The layer argument specifies a layer, sub-layer or tier under which the data from this point is grouped. Layers are a part of the instrumentation collection control.
	Layers in a point can be specified with nested layers or sub layers by separating the layer names with a / (slash). The layer specified following the slash is a sub layer of the layer specified before the slash. A sub-layer can be defined to have its own sub-layers by coding another slash and layer name following a sub-layer name.

The following is an example of a custom point that contains the mandatory arguments:

```
[MyCustomEntry_1]
; comments here….
class = myNameSpace.myClass.MyFoo
method = myMethod
layer = myCustomStuff
```

Note: Regular expressions may be used for most of the arguments in a point. They must be prefaced with an exclamation point. For more information about using regular expressions, see Appendix J, "Using Regular Expressions."

Optional Point Entries

Point definition may contain one or more of the following arguments:

Argument	Description
keyword	The presence of the keyword indicates special instrumentation. It can be used to enable specific features, for example, the WCF keyword turns on the WCF feature. It can also be used to relate point definitions to special functionality, for example, the RemotingServer keyword which is described in the section on "Remoting" on page 300. ➤ **HttpCorrelation.** Turns on correlation of client/server method calls via HTTP. ➤ **WsCorrelation.** Turns on Web service correlation logic on the client side. ➤ **WCF.** Turns on the WCF feature. ➤ **lwmd.** Turns on lwmd instrumentation. ➤ **Remoting.** Turns on .NET Remoting framework instrumentation. ➤ **RemotingServer.** Used to associate points in a .NET Remoting server to special .NET Remoting logic for these points.
ignoreClass	The **ignoreClass** argument is used to specify a comma-separated list of classes to ignore. Any class matching one of the classes specified with **ignoreClass** is not instrumented.

Argument	Description
ignoreMethod	The **ignoreMethod** argument is used to specify a comma-separated list of methods to ignore. Any method matching one of the methods specified with **ignoreMethod** is not instrumented.
ignoreTree	The **ignoreTree** argument ignores instrumenting any method that is implemented on a class that inherits from the specified class. Thus an entire class hierarchy tree of methods would be ignored.
deep_mode	The **deep_mode** argument specifies how subclasses are handled. This argument accepts the following three values: ➤ **none** - A value of "none" is similar to not specifying a **deep_mode** argument. It has no effect on how subclasses are handled. ➤ **soft** - A value of "soft" requests that for every class or interface matching the class, method, and signature entries, any subclasses or sub-interfaces that also implement the matching method and signature should also be instrumented. ➤ **hard** - A value of hard, requests that for every class or interface matching the class, method, and signature entries, any subclasses or sub-interfaces at any depth should have all their methods instrumented. Hard mode is typically used for points for interfaces. **Caution:** "hard" mode can lead to extensive instrumentation and very high Probe overhead.
scope	The **scope** argument constrains the context in which instrumentation is performed. If this entry is specified, the inserted bytecode will be caller side. Any valid regular expression may be used for the value of this argument. Scope values are a comma-separated list of class, and method names.
ignoreScope	The **ignoreScope** argument is used to exclude certain classes and methods from those included in the scope specified in scope argument.

Argument	Description
layerType	The **layerType** argument specifies special handling for some instrumented methods. This argument accepts the following three values: ➤ **trended_method** - identifies methods to be displayed in the Trended Methods view. ➤ **sql** - identifies methods that will be used to capture SQL for the SQL views. These are set by HP Diagnostics and should not be modified.
signature	The **signature** agrument specifies the signature (return and parameter types). Example: System.String(System.int32, System.String). Any valid regular expression may be used.

Instrumentation Examples

The examples in this section have been provided to illustrate how you can customize the instrumentation of an application by creating and modifying the points in the capture points file.

This section includes:

General Examples

Custom layer and sub-layer

➤ The point in the following example creates a custom sub-layer called "BAR" within the layer called "FOO" for the method myMethod in myCompany.myFoo class:

```
[myCompany.myFoo_customLayer]
class = myCompany.myFoo
method = myMethod
layer = FOO/BAR
```

Wildcard method

➤ The point in the following example captures all methods in the MyCompany.MyFoo class:

```
[myCompany.myFoo_AllMethods]
class = myCompany.myFoo
method = !.*
layer = FOO/BAR
```

Ignore Specified Methods

➤ The point in the following example captures all methods in the MyCompany.MyFoo class except for the methods setHomeInterface and getHomeInterface:

```
[myCompany.myFoo_AllMethodsExcept]
class = myCompany.myFoo
method = !.*
ignoreMethod = setHomeInterface,getHomeInterface
layer = FOO/BAR
```

➤ The point in the following example captures all methods in the
MyCompany namespace except for those contained in the
MyCompany.logging class:

```
[myCompany_All_Methods_except_from_MyCompany_Logging]
class = !myCompany\..*
method = !.*
ignoreClass = MyCompany.logging
layer = FOO/BAR
```

Capture Methods for the Trended Methods View

➤ The point in the following example captures the required data to populate
the Trended Methods View for the myMethod method:

```
[myCompany.myFoo_customLayer]
class = myCompany.myFoo
method = myMethod
layer = FOO/BAR
layertype = trended_method
```

Capture Only a Specific Method In a Class

➤ The point in the following example captures all non-static constructor
methods for the MyCompany.MyFoo class:

```
[myCompany.myFoo_Constructor]
class = myCompany.myFoo
method = .ctor
layer = FOO/BAR
```

➤ The point in the following example captures all static constructor methods
for the MyCompany.MyFoo class:

```
[myCompany.myFoo_Singleton]
class = myCompany.myFoo
method = .cctor
layer = FOO/BAR
```

➤ The point in the following example captures the setFoo method in the MyCompany.MyFoo class:

```
[myCompany.myFoo_setFoo]
class = myCompany.myFoo
method = setFoo
layer = FOO/BAR
```

➤ The point in the following example captures all methods in the MyCompany.MyFoo class whose name includes "set":

```
[myCompany.myFoo_AllSets]
class = myCompany.myFoo
method = !set.*
layer = FOO/BAR
```

➤ The point in the following example captures all methods in the MyCompany namespace:

```
[myCompany_All_Methods]
class = !myCompany\..*
method = !.*
layer = FOO/BAR
```

Capture a Specific Method That Returns a String

➤ The point in the following example captures the getFoo method that returns a System.String in the MyCompany.MyFoo class:

```
[myCompany.myFoo_GetFoo_String]
class = myCompany.myFoo
method = getFoo
signature = !System.String\(.*
layer = FOO/BAR
```

Caller Side Instrumentation

By default all the instrumentation in Diagnostics is Callee side instrumentation where the bytecode is placed within the method call. Caller side instrumentation refers to the process of placing bytecode for measurement around the call to the method to be instrumented, instead of within.

Caller side instrumentation allows for finer control of instrumentation placement, but may increase the application initialization time as each class specified in the scope must be checked for references to the class/method that is specified in the points.

The scope and ignoreScope arguments are used to specify what caller should be instrumented. The following two examples refer to Caller instrumentation.

➤ The point in the following example captures all methods in the MyCompany namespace that are called from the MyCompany.logging class.

```
[myCompany_All_Methods_from_MyCompany_Logging]
class = !myCompany\..*
method = !.*
scope = MyCompany.logging
layer = FOO/BAR
```

➤ The ignoreScope argument is used to exclude certain classes, and methods from those included in the scope specified in scope argument. The point in the following example captures all methods in the MyCompany namespace that are called from the MyCompany.logging class except for those called from the myMethod method.

```
[myCompany_All_Methods_except_from_MyCompany_Logging]
class = !myCompany\..*
method = !.*
scope = MyCompany.logging
ignoreScope = MyCompany.logging\myMethod
layer = FOO/BAR
```

Deep_mode Examples

The following interface definition will be used for both soft and hard deep_mode examples:

```
public interface Interface1 {

    public void callerMethod();

}
```

The following class will be used for both soft and hard deep_mode examples:

```
public class Class1 implements Interface1 {
    public void callerMethod(){
        calleeMethod();
        calleeMethod2();
    }

    public void calleeMethod(){
        Console.WriteLine("hello world");
  //more code lines here…
    }

    public void calleeMethod2(){
        Console.WriteLine("hello world 2");
    }
}
```

➤ The point in the following example captures the "callerMethod" in the Class1 class:

```
[Training-1]
class     = Interface1
method    = !.*
deep_mode = soft
layer     = Training
```

➤ The point in the following example captures all methods in Class 1 (i.e. "callerMethod", "calleeMethod1" and "calleeMethod2"):

```
[Training-1]
class     = Interface1
method    = !.*
deep_mode = hard
layer     = Training
```

How to Configure and Set Up Points for Non-ASP.NET or Windows Applications

This section provides instructions on how to configure both the **probe_config.xml** file and custom points files that enable instrumentation for Non-ASP.NET or Windows applications. Instrumentation for Windows Services, console applications, Windows Forms applications and WPF applications are considered Windows applications in the context of this section. For simplicity, we will refer to such applications as Windows applications.

Windows Application Design

The critical point to realize when contemplating how to configure a Windows application that you want to monitor, is that the .NET probe is designed to monitor long running processes. Therefore if your Windows application is designed to run for a few seconds and then exit; you will probably not be able to see any data for that run. The basic reason for this is that when the Windows application exits quickly, the appdomain is shut down and the probe is shut down before it can establish and maintain communication with a Diagnostics Server or a .NET Profiler.

The following example is a very simple Windows application we'll use to illustrate a number of crucial concepts to be considered when configuring the instrumentation for a Windows application.

```
namespace Hello_dotNet_nameSpace
{
    class someclass
    {
        static void Main(string[] args)
        {
// do something

            // read form commandline then exit
            clReader myClReader = new clReader();
            String cl;
            cl = myClReader.readCl();
        }
    }
    // Command Line Reader
    public class clReader
    {
        public String clread;

        public String readCl()
        {
            System.Console.WriteLine("Continue?");
            clread = Console.ReadLine();
            return clread;
        }
    }
}
```

The Hello_dotNet.exe Windows application has Main() that calls a method and then waits for the user to enter something on the command line and then exits. Until the application exits, the probe will be active.

Creating the Hello_dotNet.points File

In the **<probe_install_dir>\bin** folder there is a **Reflector.exe** command line utility that you can run against the Hello_dotNet.exe Windows application to obtain a suggested points file. See "Discovering the Classes and Methods in an Application" on page 485 for more information on the reflector utility.

Assuming that both the Reflector.exe and the Hello_dotNet.exe application are in the same folder you would run the following command:

Reflector.exe Hello_dotNet.exe

The output is sent to stdout, and among other information you will see the suggested Hello_dotNet.points as follows:

```
---------------------------------------------------------------------
Sample .points by Namespace
---------------------------------------------------------------------
[Hello_dotNet_nameSpace]
class = !Hello_dotNet_nameSpace.*
layer = Hello_dotNet_nameSpace
```

The suggested points can be used as is, except in the case where the Windows application has a method like Main(). That is, a method that if instrumented, would not return an exit until the application exits. In this case, the method would span the lifetime of the application so nothing would be reported until the application exits. Since the probe will be unloaded when the application exits, you will probably not get any data from the instrumentation point.

To fix this situation you construct a points file so that the Main() method, or any method like it, is not instrumented. The following Hello_dotNet.points file shows how to do this. It assumes that Main() is implemented in someclass.

Hello_dotNet.points:

```
[Hello_dotNet_nameSpace]
class = !Hello_dotNet_nameSpace.*
ignoreClass = Hello_dotNet_nameSpace.someclass
layer = Hello_dotNet_nameSpace

[ignore]
class = Hello_dotNet_nameSpace.someclass
ignoreMethod = Main
layer = Hello_dotNet_nameSpace
```

The crucial aspect of this type of points file is shown in bold. The [ignore] section would instrument other methods in Hello_dotNet_nameSpace.someclass if there were any; while ignoring the Main() method.

Configuring the Windows Application for Instrumentation

To configure the .NET probe to instrument the Hello_dotNet.exe Windows application, you will need to add the following XML to the **etc****probe_config.xml** file. You can add it to the bottom of the probe_config.xml file just above the </**probeconfig**> entry.

```xml
<process name="Hello_dotNet">
    <points file="Hello_dotNet.points" />
    <instrumentation>
      <logging level="" />
    </instrumentation>
    <logging level="" />
</process>
```

Note: You will need to place your **Hello_dotNet.points** file in the **<probe_install_dir>****etc** folder before you make the above changes to the probe_config.xml file.

The only required child element is the points file. The instrumentation, logging and modes are optional. The following instrumentation setting can be useful when diagnosing which methods are or are not being instrumented:

```xml
<instrumentation>
  <logging level="points ilasm" />
</instrumentation>
```

Remoting

You can configure the .NET probe to add custom instrumentation that supports the instrumentation of .NET Remoting Client and Server applications. Supported configurations are:

➤ Both HTTP and TCP bindings

➤ Both Binary and SOAP Formatting

➤ Both .NET 1.1 and 2.0 Remoting applications

Configuration

By default the .NET probe is not enabled to instrument Remoting applications. You will need to add custom instrumentation points for both the Client and Server applications.

There are two instrumentation keywords related to Remoting:

Remoting. The Remoting keyword is used to enable instrumentation for various points in the Remoting Framework.

RemotingServer. The RemotingServer keyword is used identify the class that implements the Remoting Methods and isolates the instrumentation of the methods on that class from unintended instrumentation of other similar methods.

Client Example

The following example is a very simple Windows application example we'll use to illustrate a number of crucial concepts the must be considered when configuring the instrumentation for a Remoting Client Application.

```
namespace HPSoftware.AM.Tests.Remoting.SimpleRemoting
{
   class SimpleConsoleClient
   {
      [STAThread]
      static void Main(string[] args)
      {
         const string msg1 = "How are you?";

         String filename =
AppDomain.CurrentDomain.SetupInformation.ConfigurationFile;
         RemotingConfiguration.Configure(filename, false);

         MyRemotableObject remoteObject = new MyRemotableObject();

         doit(remoteObject, myMsg);

         Console.WriteLine();
         Console.WriteLine("(Press any key to exit)");
         Console.ReadKey();
      }

      public static void doit(MyRemotableObject obj, String message)
      {
           Console.WriteLine(obj.GetEnlightenment(message));
      }
}
```

As described in the section "How to Configure and Set Up Points for Non-ASP.NET or Windows Applications" on page 296, you can use the Reflector utility to help you determine how to configure the Remoting Client points file.

To configure the Probe to instrument the SimpleConsoleClient Remoting Windows application, you will need to add the following XML to the **probe_config.xml** file:

```xml
<process name="SimpleConsoleClient">
        <points file="Remoting.points" />
        <points file="SimpleConsoleClient.points" />
        <instrumentation><logging level="" /></instrumentation>
        <logging level="" />
 </process>
```

It is required that you add the <points file="Remoting.points" /> entry.

If you are in the directory that holds the SimpleConsoleClient.exe, and the Reflector.exe is in the PATH, you can execute the Reflector on the command line to view an implementation decomposition of the SimpleConsoleClient.exe and suggested point file settings:

Reflector SimpleConsoleClient.exe

The output of this command will contain the following:

```
-----------------------------------------------------------------------
Sample .points by Namespace
-----------------------------------------------------------------------
[HPSoftware.AM.Tests.Remoting.SimpleRemoting]
class = !HPSoftware.AM.Tests.Remoting.SimpleRemoting.*
layer = HPSoftware/AM/Tests/Remoting/SimpleRemoting
-----------------------------------------------------------------------
(1 classes)  Namespace: HPSoftware.AM.Tests.Remoting.SimpleRemoting
-----------------------------------------------------------------------
HPSoftware.AM.Tests.Remoting.SimpleRemoting.SimpleConsoleClient (8
Methods)
   Equals              System.Boolean(System.Object)
   Finalize            System.Void()
   GetHashCode         System.Int32()
   GetType             System.Type()
   doit                (method signature information unavailable))
   Main                System.Void(System.String[])
   MemberwiseClone System.Object()
   ToString            System.String()
```

The suggested SimpleConsoleClient.points are:

```
[HPSoftware.AM.Tests.Remoting.SimpleRemoting]
class = !HPSoftware.AM.Tests.Remoting.SimpleRemoting.*
layer = HPSoftware/AM/Tests/Remoting/SimpleRemoting
```

While being a good suggestion, these settings would not create instrumentation that would produce any data. The reason is, as discussed in the section "How to Configure and Set Up Points for Non-ASP.NET or Windows Applications" on page 296, you need to ignore methods like Main(). If you factor in the need to ignore Main(), that would leave us with the following possible points file settings:

```
[HPSoftware.AM.Tests.Remoting.SimpleRemoting]
class = !HPSoftware.AM.Tests.Remoting.SimpleRemoting.*
ignoreMethod = Main
layer = HPSoftware/AM/Tests/Remoting/SimpleRemoting
```

While this results in settings that would be useful, and would produce data, it is best to refine these setting some more to make them more precise. The reason for this is primarily probe performance; the more methods that are instrumented the greater the probe's performance hit on the instrumented application. So, for example, if you can remove the wildcards "!.*" from the settings then the scope of your settings become explicit.

Notice from the Reflector output that there is actually only a single implemented class:

HPSoftware.AM.Tests.Remoting.SimpleRemoting.SimpleConsoleClient

We can remove the wildcards from the class setting as follows:

```
class = HPSoftware.AM.Tests.Remoting.SimpleRemoting.SimpleConsoleClient
```

Notice also, that the Reflector output does not contain a method setting. The default meaning of no method setting is that all methods are instrumented. Since most the following methods are only present because they are inherited from System.Object, it is unlikely that you really want to instrument these methods: Equals, Finalize, GetHashCode, GetType, MemberwiseClone, ToString. However, it is likely that you would want to instrument the doit method since it wraps the Remoting client call. The following settings are the recommended settings for the SimpleConsoleClient.points file:

```
[HPSoftware.AM.Tests.Remoting.SimpleRemoting]
class = HPSoftware.AM.Tests.Remoting.SimpleRemoting.SimpleConsoleClient
method = doit
layer = HPSoftware/AM/Tests/Remoting/SimpleRemoting
```

Server Example

The following Windows application example will be used to illustrate a number of crucial concepts the must be considered when configuring the instrumentation for a Remoting Server Application:

C# code snippets are shown for both the Remotable Object, which is shared between the Remoting Client and Server, and the SimpleConsoleServer.exe Remoting Server Application.

C# code snippet for the Remotable Object:

```
HPSoftware.AM.Tests.Remoting.SimpleRemoting
{

 public class MyRemotableObject : MarshalByRefObject
 {
     const string response = "I'm just fine!";

     public MyRemotableObject()
     {
     }

     public String GetEnlightenment(string message)
     {
         return response;
     }
 }
}
```

C# code snippet for the SimpleConsoleServer.exe:

```
namespace HPSoftware.AM.Tests.Remoting.SimpleRemoting
{
   class SimpleConsoleServer
   {
     [STAThread]
     static void Main(string[] args)
     {
       String filename =
AppDomain.CurrentDomain.SetupInformation.ConfigurationFile;
       RemotingConfiguration.Configure(filename, false);

       Console.WriteLine("Server is running...   press any key to exit");
       Console.ReadKey();
     }
   }
}
```

To configure the probe to instrument the SimpleConsoleServer Remoting WIndows application, you will need to add the following XML to the **probe_config.xml** file:

```
<process name="SimpleConsoleServer">
        <points file="SimpleConsoleServer.points" />
        <instrumentation><logging level="" /></instrumentation>
        <logging level="" />
</process>
```

Note that it is not required to add the <points file="Remoting.points" /> entry.

Point files for the Remoting Server may have one or more sections. The first section relates to the Remotable Object and is a required section. Optionally, a second section that relates to the Remoting Server instrumentation may be added. Finally, other optional sections may be added to instrument other methods that may be called by either the Remoting methods or the Remoting Server. We will construct the Remotable Object section first.

The Remotable Object will reside in some assembly; we will assume it is in the RemotableObjects.dll. Again, you will run the Reflector against the RemotableObjects.dll and see output that includes:

```
-----------------------------------------------------------------------
Sample .points by Namespace
-----------------------------------------------------------------------

[HPSoftware.AM.Tests.Remoting.SimpleRemoting]
class = !HPSoftware.AM.Tests.Remoting.SimpleRemoting.*
layer = HPSoftware/AM/Tests/Remoting/SimpleRemoting


-----------------------------------------------------------------------
(1 classes)  Namespace: HPSoftware.AM.Tests.Remoting.SimpleRemoting
-----------------------------------------------------------------------

HPSoftware.AM.Tests.Remoting.SimpleRemoting.MyRemotableObject (17
Methods)
    __RaceSetServerIdentity         System.Runtime.Remoting.ServerIden…)
    __ResetServerIdentity           System.Void()
    CanCastToXmlType                System.Boolean(System.String,System…)
    CreateObjRef                    System.Runtime.Remoting.ObjRef(Syste…)
    Equals                          System.Boolean(System.Object)
    Finalize                        System.Void()
    GetComIUnknown                  System.IntPtr(System.Boolean)
    GetEnlightenment                System.String(System.String)
    GetHashCode                     System.Int32()
    GetLifetimeService              System.Object()
    GetType                         System.Type()
    InitializeLifetimeService       System.Object()
    InvokeMember                    System.Object(System.String,System…)
    IsInstanceOfType                System.Boolean(System.Type)
    MemberwiseClone                 System.MarshalByRefObject(System…)
    MemberwiseClone                 System.Object()
    ToString                        System.String()
```

As with the Remoting Client example, you cannot just use the suggested point settings. First you need to be certain that you have identified the class that implements the Remotable Object. You can do this by observing that the Remotable Object is required to inherit from System.MarshalByRefObject and therefore it must have the following methods on it: CreateObjRef, GetLifetimeService, InitializeLifetimeService, MemberwiseClone. From the Reflector output above you can see that the HPSoftware.AM.Tests.Remoting.SimpleRemoting.MyRemotableObject class is an obvious candidate for the class that implements the Remotable Object.

The Remotable Object section must include the **keyword = RemotingServer** entry. This entry indicates that the Probe's Instrumenter should perform special processing for the point settings in this section. This special processing accomplishes two things. First, it instruments all methods on a class that inherits from System.MarshalByRefObject. Therefore, you need not specify which Remoting methods to instrument; all Remoting methods will be instrumented. This is also why there is no need for a method entry in this section. Second, this keyword isolates the instrumentation of methods that are implemented on a class that inherits from System.MarshalByRefObject to the specified class. This is important because there are many System classes and user classes that also inherit from System.MarshalByRefObject and we do not want to unintentionally instrument them.

Based on these observations, here is the recommended Remotable Object section:

```
[RemotableObject]
keyword = RemotingServer
class = HPSoftware.AM.Tests.Remoting.SimpleRemoting.MyRemotableObject
layer = RemotableObject
```

Now we will construct the optional Remoting Server section. You will only need to create this section if you wish to monitor the Server logic that is invoked independent of the Remoting methods. Again, you will run the Reflector against the SimpleConsoleServer.exe and you will see output that includes:

```
---------------------------------------------------------------------
Sample .points by Namespace
---------------------------------------------------------------------
[HPSoftware.AM.Tests.Remoting.SimpleRemoting]
class = !HPSoftware.AM.Tests.Remoting.SimpleRemoting.*
layer = HPSoftware/AM/Tests/Remoting/SimpleRemoting
```

```
---------------------------------------------------------------------
(1 classes)  Namespace: HPSoftware.AM.Tests.Remoting.SimpleRemoting
---------------------------------------------------------------------
HPSoftware.AM.Tests.Remoting.SimpleRemoting.SimpleConsoleServer (7
Methods)
    Equals              System.Boolean(System.Object)
    Finalize            System.Void()
    GetHashCode         System.Int32()
    GetType             System.Type()
    Main                System.Void(System.String[])
    MemberwiseClone     System.Object()
    ToString            System.String()
```

Again, as explained in the section "How to Configure and Set Up Points for Non-ASP.NET or Windows Applications" on page 296, you cannot just use the suggested points settings. You will need to ignore the Main() method.

Based on these observations, the following settings are the recommended settings for the SimpleConsoleServer.points file:

```
[RemotableObject]
keyword = RemotingServer
class = HPSoftware.AM.Tests.Remoting.SimpleRemoting.MyRemotableObject
layer = RemotableObject

[RemotingServer]
class = HPSoftware.AM.Tests.Remoting.SimpleRemoting.SimpleConsoleServer
ignoreMethod = Main
layer = RemotingServer
```

Finally, other optional sections may be added to instrument other methods that may be called by either the Remoting methods or the Remoting Server.

Understanding the Overhead of Custom Instrumentation

When you are creating custom instrumentation you should beware of over-instrumenting the application because it can introduce excessive latency into the probed application. The custom instrumentation does not have the same impact on the method latency or the CPU overhead because the overhead of instrumentation is nearly fixed for every method because the amount of bytecode is almost always the same. This means that the physical percentages of the CPU and latency overhead will vary in direct proportion to the length of time the method takes to execute.

For example if a method take 100ms and instrumentation makes it execute in 101ms, then overhead is 1%. If a method takes 10ms and instrumentation changes it's response to 11ms, then overhead is 10%. If this method is not called very often, its overall latency effect on the application is minimal. However, the overall latency effect of an instrumented method that is called more frequently could have an impact on the latency of the application's response even though Its overhead percentage is much smaller.

Unlike a traditional profiler which may profile every method called, HP Diagnostics uses bytecode instrumentation. This allows the default instrumentation to be selective so as to minimize the overhead caused by instrumentation to an average of 3-5%. Methods with higher latency overhead introduced by instrumentation are only instrumented when they are called infrequently in relation to other components in the application and when the instrumentation provides specific information needed for triage activities.

You should also consider Diagnostics data overhead when you are customizing the instrumentation for an application. The more methods that you instrument the more data that the Probe must serialize and pass over the network to the Diagnostics Server. You can tune the Probe's default configuration so that it can adjust the volume of Diagnostics data to avoid any unnecessary effect on the performance of the system being monitored. Improper Probe tuning can cause CPU, Memory and Network overhead on the physical machine where your Probe resides. For more information about managing Latency, CPU, Memory and Network overhead, see Chapter 17, "Advanced .NET Probe Configuration."

11

Instrumenting Java Applications and Configuring the Probe from the Profiler

You can use the Configuration tab in the Java Profiler to maintain instrumentation points and configure several properties of the Java probe.

This chapter includes:

➤ About the Java Profiler Configuration Tab on page 312

➤ Maintaining Instrumentation from the Configuration Tab on page 313

➤ Maintaining Probe Settings from the Configuration Tab on page 325

About the Java Profiler Configuration Tab

The Configuration tab in the Java Diagnostics Profiler provides a way for you to maintain the instrumentation points and the Probe configuration without having to manually edit the capture points file or property files.

You can access the Configuration tab from the Java Diagnostics Profiler whether profiling has been started on the Probe or not.

The Configuration tab in the Java Diagnostics Profiler is divided into the Instrumentation section and the Probe Settings section as shown in the following screen image:
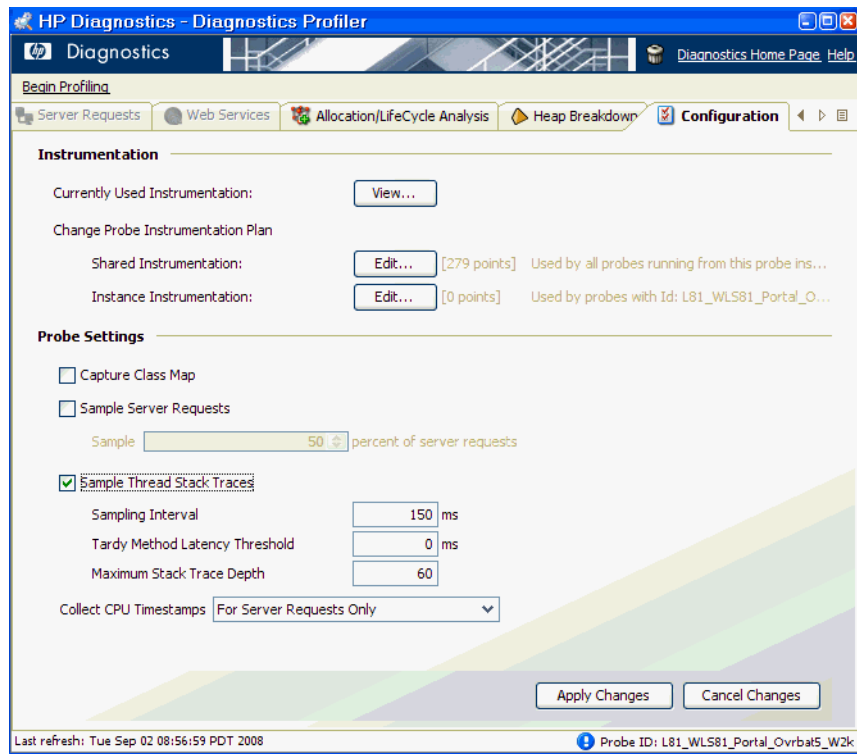
The Instrumentation section gives you access to view and update the instrumentation for the application that the Probe is monitoring. The edit dialogs that you access from the Instrumentation section allow you to view and edit the instrumentation points as defined in the capture points file that Diagnostics uses to instrument your applications.

The Probe Settings section of the Configuration tab allows you to configure class map capture, server request sampling, thread stack trace sampling and CPU timestamping the Probe uses as it captures the performance metrics for your application. For more information on the Probe Settings section see "Maintaining Probe Settings from the Configuration Tab" on page 325.

Maintaining Instrumentation from the Configuration Tab

From the Instrumentation section of the Configuration tab you can control the instrumentation that Diagnostics applies to your application.
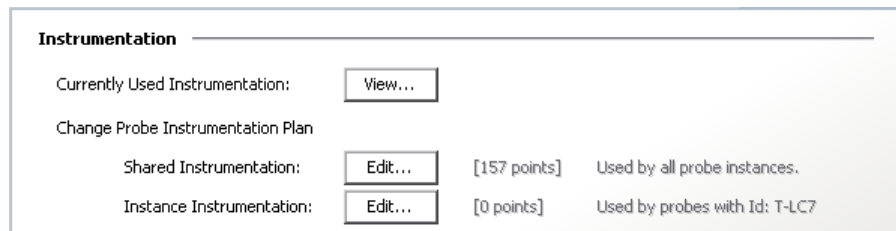
Reviewing the Current Instrumentation

To review the layers, classes, and methods that have been instrumented as a result of the points in the current capture points file click **View...** in the Instrumentation section of the Configuration tab. The Profiler displays the Instrumented Layers page as shown in the following image:

Instrumented layers			
Layer	**# Hits**	**Active Points**	**Actions**
(Other)	55591	18 / 18	[Disable] [Clear # Hits]
Business Tier/EJB/Entity Bean	0	38 / 38	[Disable] [Clear # Hits]
Business Tier/EJB/Session Bean	26399	30 / 30	[Disable] [Clear # Hits]
Database/JDBC/Connection	212	59 / 59	[Disable] [Clear # Hits]
Database/JDBC/Execute	111	41 / 41	[Disable] [Clear # Hits]
Directory Service/JNDI	148	5 / 5	[Disable] [Clear # Hits]
Messaging/JMS/Connection	19	8 / 8	[Disable] [Clear # Hits]
Messaging/JMS/Consumer	0	3 / 3	[Disable] [Clear # Hits]
Messaging/JMS/Listener	0	6 / 6	[Disable] [Clear # Hits]
Messaging/JMS/Session	13	33 / 33	[Disable] [Clear # Hits]
Messaging/JMS/Session/Queue	8	9 / 9	[Disable] [Clear # Hits]
Messaging/JMS/Session/Topic	2	9 / 9	[Disable] [Clear # Hits]
Web Services	16249	2 / 2	[Disable] [Clear # Hits]
Web Tier/JSP	1605	1 / 1	[Disable] [Clear # Hits]
Web Tier/Servlet	42021	7 / 7	[Disable] [Clear # Hits]

Mercury Diagnostics J2EE Probe "T-LC7", version 6.5.13.383

The Instrumented Layers page lists the layers that have been instrumented along with the number of times that the instrumentation points in the layer have been triggered and the number of points that are currently active in the layer. The following table describes the columns on this page:

Column	Description
Layer	Contains a list of the layers that have been instrumented. The layer names in this column are links that allow you to navigate to a page that provides the details about the processing in the layer that was monitored by the Probe. **Note:** Only the layers defined in points that were actually instrumented are listed.
# Hits	Contains a count of the number of times that the classes and methods that are monitored by the points in the listed layer have been invoked. You can reset the count using the **Clear # of Hits** link in the Actions column.
Active Points	Contains the count of the number of points that are currently active as well as the total number of points that have been defined for the particular layer.
Actions	Contains links that allow you to manipulate the information for the listed layers. The available action are: ➤ **Disable:** Disables all of the points in the selected layer so that they no longer capture data. The instrumentation stays in place and can be enabled again. Enabling or disabling points here is effective only until the next restart of your application. To change the default enabled state for a point see "Coding Points in the Capture Points File" on page 228. ➤ **Clear # Hits**: Resets the hit count displayed in the # Hits column for the selected layer.

Maintaining the Instrumentation Points

To maintain the points that provide the instrumentation instructions the tell the Probe what to monitor in your application, navigate to the Configuration tab in the Java Diagnostics Profiler and click **Edit...** for either the Shared Instrumentation or the Instance Instrumentation. The Instrumentation Points dialog is displayed as shown in the following image:
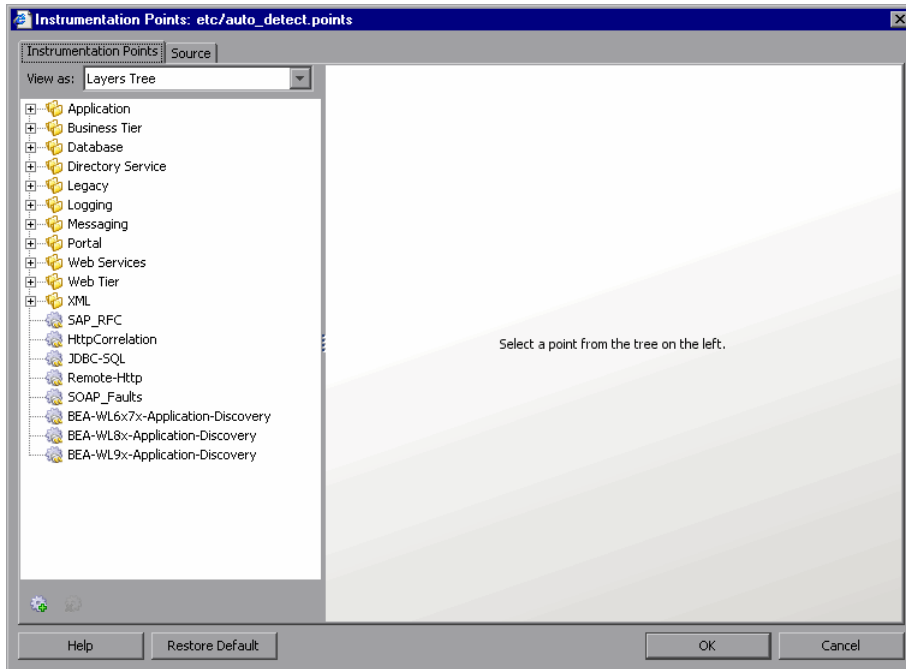
There are two ways you can edit the instrumentation: Visually using a list or tree of points on the Instrumentation Points tab or via the source of the capture points file on the Source tab.

Selecting and Viewing an Existing Point

The navigation bar in the Instrumentation Points dialog helps you to locate the points in the capture points file that you would like to maintain. By making a selection from the **View as** dropdown you can choose the format in which the points are listed.

When you select Layer Tree from the **View as** dropdown, the Probe lists the points in the capture points file in a tree structure according to the layers and sublayers that you assigned to the point as shown in the following image:
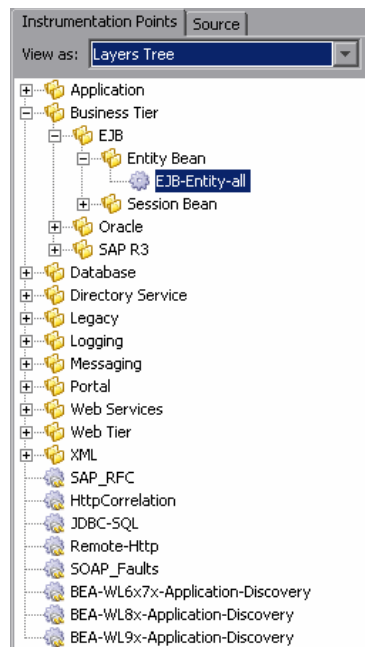
When you select **Points List** from the **View as** dropdown, the Probe lists the points in the capture points file in ascending alphabetical order as shown in the following image:
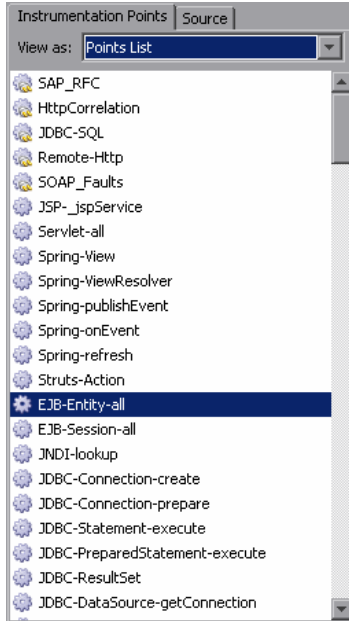
When you have located the point that you want to view or maintain, select the point in the navigation bar. The Probe displays the details of the selected point in the view/edit panel where you can maintain the point.

Updating an Existing Point

When you select a layer or sublayer from the navigation bar, the view/edit panel contains only a prompt to remind you to select a point.

To update an existing point, select the point from the navigation bar so that the Profiler displays the details for the point in the Instrumentation Points tab of the view/edit panel as shown in the following example:
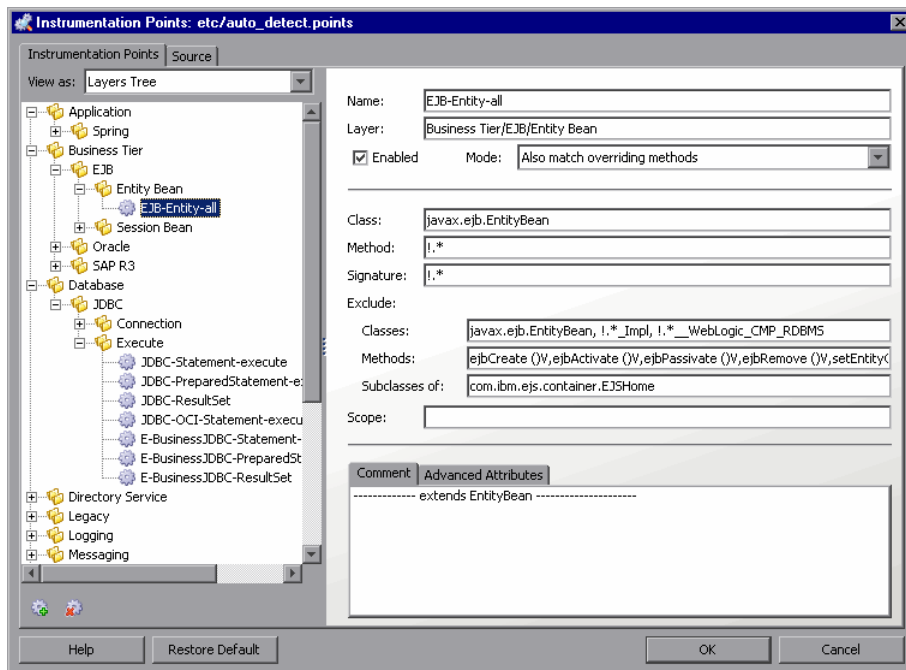
The arguments that are commonly used when defining a point in the capture points file are displayed as separate data fields to make it easier for you to make any necessary updates. Other, more advanced arguments are displayed in the Advanced Attributes tab at the bottom of the Instrumentation Points tab. Comments for the point are displayed in the Comments tab.

All of the arguments that can be used to define a point in the capture points file have been documented in "Coding Points in the Capture Points File" on page 228.

The following table will help you to cross reference the arguments displayed in the Instrumentation Points tab with the capture points file arguments documented in this section. You may also find this cross reference table useful when reviewing the points displayed in the Source tab.

Instrumentation Point Tab	Capture Points File Argument
Name	Point-Name
Layer	layer
Enabled	keyword = disabled
Mode	deep_mode
Class	class
Method	method
Signature	signature
Exclude Classes	ignore_cl
Exclude Methods	ignore_method
Exclude Subclasses	ignore_tree
Scope	scope
Advanced Attributes	detail, code-key, tierDefinition, layerType, merge_classes, merge_url_to, ignore_tree, ignoreScope

An example of the Source tab is shown in the following image:
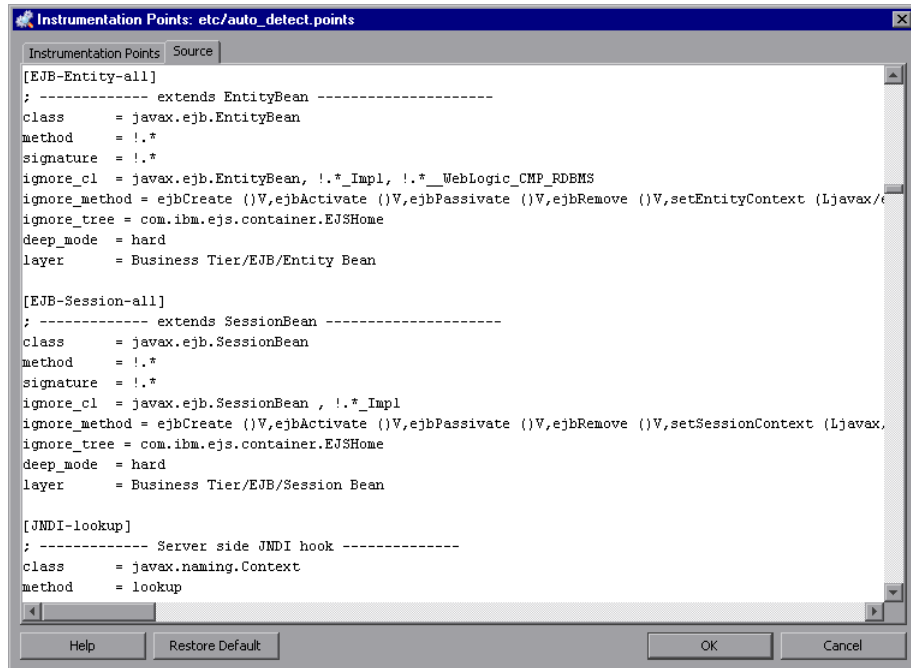
Deleting an Existing Point or Layer

You may delete a point or layer listed in the navigation bar.

To delete a point or layer:

1 Select the point or layer from the Navigation bar on the Instrumentation Points tab.

2 Click Delete Point. The Profiler deletes the selected entity from the list in the navigation bar.

The selected entity is not actually deleted from the capture points file until you apply all of your instrumentation points updates from the Configuration tab in the Profiler.

3 Close the Instrumentation Points dialog by clicking **OK**.

4 Apply all of the changes made using the Configuration tab by clicking **Apply Changes**.

Adding a New Point

You may add a point to the instrumentation.

To add a point:

1 Click New Point. The Profiler displays the Select New Point Type dialog box as shown in the following image:
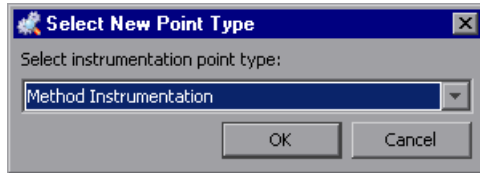
2 Select the appropriate point type from the dropdown and click **OK.**

The Profiler displays the Instrumentation Points tab with the view/edit section initialized for creating a new point for the selected point type.

3 Enter the arguments and comments for the new point into the appropriate locations on the tab.

When you enter the Layer information, the entry for the new point in the navigation bar is updated to show the point in the correct existing layer or, if the layer that you specified does not already exist, with a brand new layer.

The new point is not actually added to the capture points file until you apply all of your instrumentation points updates from the Configuration tab in the Profiler.

4 Close the Instrumentation Points dialog by clicking **OK**.

5 Apply all of the changes made using the Configuration tab by clicking **Apply Changes**.

Activating OVTA-like Points

Points are included in the Java probe instrumentation for Servlet Filters and EJB local home methods. These instrumentation points provide additional functionality similar to the OVTA (OpenView Transaction Analyzer) Java Monitor.

The ServletFilter point requires that the HttpCorrelation2 point also be activated for server filters to be monitored correctly. This is because servlet filters sometimes are the first time Diagnostics sees an HTTP server request.

The EJBLocalHome, ServletFilter and related HttpCorrelation2 instrumentation points are not active by default. Inactive points are indicated by a red symbol on the icon next to the instrumentation point, as shown in the screen below. To use these points, you must set active=true in the **auto_detect.points** file. This can be done through the UI or by directly editing the file.

Locate these points in the Profiler UI as described in "Selecting and Viewing an Existing Point" on page 317 and navigate to the **Business Tier**>**EJB**>**LocalHome**>**EJBLocalHome** point or the **Web Tier**>**Servlet**>**ServletFilter** point and **HttpCorrelation2** point.
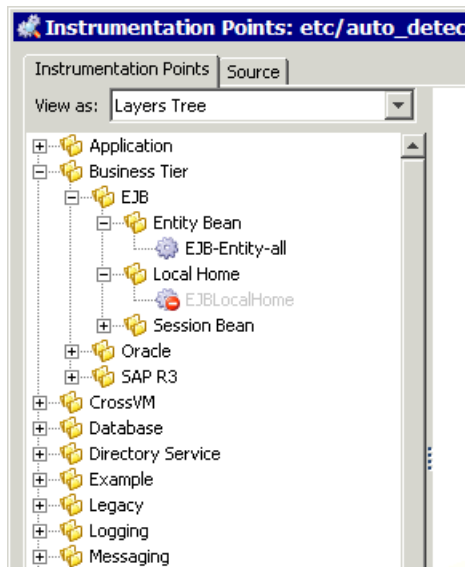
To set these points to active, select the point from the Instrumentation Points navigation bar so that the Profiler displays the details for the point. Check the **active** check box.

Close the Instrumentation Points dialog by clicking **OK**.

Apply all of the changes made using the Configuration tab by clicking **Apply Changes**. You will be prompted to restart your probe (application server) for the newly activated points to take effect.

Restoring Default Points

When you have finished diagnosing a problem using the Profiler or HP Diagnostics you may want to restore the default instrumentation to avoid incurring the overhead from a more robust instrumentation.

To restore the default settings to the instrumentation:

1 Click **Restore Defaults.**

The instrumentation points are not actually added to the capture points file until you apply all of your instrumentation points updates from the Configuration tab in the Profiler.

2 Close the Instrumentation Points dialog by clicking **OK**.

3 Apply all of the changes made using the Configuration tab by clicking **Apply Changes**.

Maintaining Probe Settings from the Configuration Tab

The Probe Settings section of the Configuration tab allows you to configure class map capture, server request sampling, thread stack trace sampling and CPU timestamping the Probe uses as it captures the performance metrics for your application.

Probe Settings

☐ Capture Class Map

☐ Sample Server Requests

    Sample   [50 ⬍] percent of server requests

☑ Sample Thread Stack Traces

    Sampling Interval   [150] ms

    Tardy Method Latency Threshold   [0] ms

    Maximum Stack Trace Depth   [60]

Collect CPU Timestamps [For Server Requests Only   ▼]

Controlling Class Map Capture

The class map allows Diagnostics to provide more details about the classes and methods that are invoked by a server request. This information can help you to narrow your search for the source of a performance issue. The class map information can be viewed by drilling down on the Layers listed on the Instrumented Layers page as described in "Reviewing the Current Instrumentation" on page 314. The cost for using class map comes from the additional overhead that creating the map places upon the Probe host.

To enable class map capture:

1 Check the **Capture Class Map** check box.

2 When you have completed making changes to the Configuration tab, click **Apply Changes**.

3 Restart the Probe to allow the Probe to begin class map capture.

To disable class map capture:

1 Uncheck the **Capture Class Map** check box.

2 When you have completed making changes to the Configuration tab, click **Apply Changes**.

3 Restart the Probe to allow the Probe to stop class map capture.

Controlling Server Request Sampling

Server Request sampling allows Diagnostics to gather detailed information for a subset of all of the server requests executed in your application. This reduces the overhead of the Probe because no information is captured for Server requests that are not sampled. As long as the sampling rate is not too low, the average results from all sampled server requests should provide results that are reasonably accurate and meaningful for understanding the performance of your applications.

You control when sampling is enabled, and what percentage of the server requests will be sampled. The properties controlling server request sampling are in **dynamic.properties**:

➤ enable.server.request.sampling

➤ server.request.sampling.rate

➤ server.request.sampling.method

➤ sample.colored.requests

To enable server request sampling from the Profiler:

1 Check the **Sample Server Requests** check box.

2 Set the sampling percentage using the spinner.

3 When you have completed making changes to the Configuration tab, click **Apply Changes**.

Your changes take effect immediately. There is no need to restart the Probe.

To disable server request sampling from the Profiler:

1 Uncheck the **Sample Server Requests** check box.

2 When you have completed making changes to the Configuration tab, click
Apply Changes.

Your changes take effect immediately. There is no need to restart the Probe.

Configuring Thread Stack Trace Sampling

When asynchronous thread sampling is enabled you can see, in the Call
Profile view, which methods were executed during long running fragments
even if no instrumented methods were hit during this time. See the *HP
Diagnostics User's Guide* chapter on Call Profiles for a screen shot showing
the additional nodes added based on thread sampling.

There are several properties used to enable and configure thread stack trace
sampling.

In **dynamic.properties** there are the following properties:

➤ **enable.stack.trace.sampling** - property to enable asynchronous thread stack
trace sampling. Possible values are: false, auto, true, where auto is the
default.

When the dynamic property enable.stack.trace.sampling is set to auto, stack
trace sampling will be enabled IF the probe is running on selected (certified)
platforms and JVMs. For other JVMs, the setting must be set to true
explicitly and you must use caution because the JVM could generate errors
or abort. See the Diagnostics Release Notes.

➤ **tardy.method.latency.threshold** - the minimum time that an instrumented
method must run without hitting any instrumentation points before stack
trace sampling will be attempted for this method. The purpose of this
property is mainly to control the overhead of sampling by limiting the stack
trace collection to only the most interesting cases.

➤ **stack.trace.sampling.rate** - this is the time that must elapse before the next
consecutive sampling attempt will be made.

Small values for **stack.trace.sampling.rate** cause frequent sampling, thus
providing rich data but at the cost of increased overhead.

The overhead caused by frequent sampling affects primarily the latency of server requests. The overall CPU usage by the probe may go up as well, but this effect is not as profound as the latency increase. For systems with many CPUs, the process CPU consumption may actually go down (and it is not a good thing).
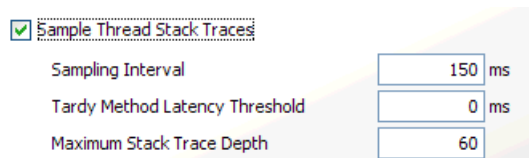
➤ **stack.trace.depth.max** - this is the limit for the depth of stack traces obtained from the JVM. You will most likely not need to adjust this value.

In **dispatcher.properties** there are the following properties:

➤ **enable.stack.trace.aggregation** - a boolean property allowing the correlation thread to merge together nodes observed on more than one consecutive stack trace collected, unless there's proof that the nodes must not represent a single method invocation. When set to true, it may decrease the number of additional call tree nodes created, but may create a false impression that the number of calls to the additional nodes is known and is small. When it is set to false, it creates a node for each method and each stack trace it was visible on, thus creating a false impression that the number of calls to the nodes is known and is large. In fact, stack trace sampling cannot reveal the number of calls at all.

➤ **aggregated.stack.trace.validity.threshold** - if the enable.stack.trace.aggregation property is set to true, only the call tree nodes that stem from more than the **aggregated.stack.trace.validity.threshold** number of individual stack traces will be reported. This setting controls noise elimination and memory footprint, especially on the server side.

All of the above properties can be dynamically changed.

You can change the first four properties (from dynamic.properties) remotely, using the Configuration tab in the Diagnostics Java Profiler.

Example Thread Sampling Configurations

Use Case 1: A particular method has average latency about 170 milliseconds, but from time to time it takes 1.4 second for this method to complete. Most of the methods visible in Call Profiles for any fragment execute in about 550 milliseconds or less. Since the method in question makes multiple calls to its callees, you don't want to instrument them.

Instead you enable stack trace sampling to find out what is the cause of long execution times. To minimize the overhead, you set tardy.method.latency.threshold to 600 milliseconds. This ensures that most of the methods will not get sampled at all, because they are likely to complete before this time elapses. However, any method running longer than this value, including our long running method, will get sampled, once the method runs for 600 milliseconds (or longer) without making any calls to any of the instrumented methods.

If you also set the value of **stack.trace.sampling.rate** to 100 milliseconds, theoretically, this should give up to 8 samples for each method invocation that lasts 1.4 seconds ((1400 - 600) / 100). Because you know that the method makes many calls to its callees, you could also set **aggregated.stack.trace.validity.threshold** to zero. This ensures that even if each collected stack trace is completely different, they will all be reported.

In this case you examine the Call Profile for long running instances of the server request and would see additional nodes revealed by stack trace sampling.

Use Case 2: You prepare a custom application for deployment and see that the default instrumentation provided with the Diagnostics probe does not work very well because many Call Profiles contain very few methods which does not give any insight about the application specific behavior. You are reluctant to add additional instrumentation for all classes and methods belonging to the custom application, because of the performance and memory consumption concerns.

You enable stack trace sampling. Assuming that a typical server request that does not have sufficiently detailed call tree information runs in about 2 seconds, you select a **stack.trace.sampling.rate** of 200ms. This can give up to 10 stack traces per typical server request. However, you do not want all the stack traces to be reported, because some of the methods visible in the stack traces can be very fast, and they do not substantially contribute to the server request's overall latency. Therefore, you set **aggregated.stack.trace.validity.threshold** to 2. This ensures that only methods visible in 3 or more consecutive stack traces, or methods with estimated latency of 300 milliseconds or more, will be reported.

After viewing the Call Profiles with the additional nodes obtained from sampling, you can make informed decision about adding additional instrumentation points to the probe configuration in deployment.

Troubleshooting Stack Trace Thread Sampling

Question 1: I do not see any new nodes in my Call Profile after I enabled stack trace sampling. Why?

Answer: Go through this checklist and see if any of the following applies to your case:

❏ Check if you are using Java 1.5 or newer. Stack trace sampling does not work for earlier versions of Java.

❏ Check if the last method visible in the Call Profile is an outbound call. Methods marked as outbound do not get sampled. (To reliably check if a method is marked as outbound, find this method in detailReport.txt file and check its corresponding instrumentation point detail for the "outbound" keyword).

❏ Check if the last method visible in the Call Profile has been marked as no-layer-recurse. Such methods do not get sampled. (Use the same procedure as in the previous point to check if a method is no-layer-recurse).

❏ Try to reduce tardy.method.latency.threshold and/or minimum.method.latency. It is possible that the last method visible in Call Profile makes calls that get trimmed, but they prohibit the sampling to kick in because there's never an inactive period of tardy.method.latency.threshold for the caller.

❑ Try to reduce aggregated.stack.trace.validity.threshold and/or check if there are warnings in the probe.log file about stack depth to shallow. It is possible that the observed stack traces change too quickly to get reported.

❑ Try to reduce stack.trace.sampling.rate. Perhaps your methods simply miss the opportunities to get sampled.

❑ Verify that the latency of the last visible method in Call Profile is not caused by running garbage collector. No Java code runs during garbage collection, and this includes the stack trace sampling code.

Question 2: What is the minimum value of stack.trace.sampling.rate I can use?

Answer: You can use any positive value, but please keep in mind that each platform will simply refuse to sample more frequently that it possibly can. The three factors playing a role here are: the minimum granularity of sleep() available, the timer resolution, and the time it actually takes to collect one set of samples.

Question 3: What is the maximum value of stack.trace.sampling.rate I can use?

Answer: There's no limit. The usefulness of a high setting depends entirely on the latency of the server requests for the application. To get any results, you should plan for at least a few samples for each server request you are concerned with, and even that may require tuning other sampling parameters as well.

Controlling CPU Timestamp Collection

The CPU timestamps are used to calculate the amount of exclusive CPU time that a method uses. This information can be viewed on the **Hotspots** tab in the Java Diagnostics Profiler.

Important: In VMware, the CPU time metric is from the perspective of the guest operating system and is affected by the VMware virtual timer. See the VMware whitepaper on timekeeping at http://www.vmware.com/pdf/vmware_timekeeping.pdf and refer to "Time Synchronization for Probes Running on VMware" on page 403.

By default, collection of CPU time metrics is enabled for server requests.

Collection of CPU time metrics can be configured in property files (see "Configuring Collection of CPU Time Metrics" on page 412) or using the the Java Diagnostics Profiler UI as described below.

1 In the **Profiler** UI select the **Configuration** tab. The profiler does not need to be started to make this probe configuration change.

2 In the Configuration screen select a **Collect CPU Timestamps** option from the drop down list.

CPU Timestamp Collection Method	Description
None	No CPU Timestamps.
For Server Requests Only	CPU timestamps will only be collected for server requests.
For Server Requests and Portlet Methods	CPU timestamps will be collected for ALL server requests and the lifecycle methods instrumented for portal components (layertype=portlet.
For Server Requests and All Methods	CPU timestamps will be collected for ALL server requests and ALL methods.

3 When you have completed making changes to the Configuration tab, click **Apply Changes.**

Note: Your changes take effect immediately. There is no need to restart the Probe.

Restarting the Probe

When you click **Apply Changes** on the Configuration tab all of the updates that you made in the Instrumentation section and Probe Settings sections of the Configuration tab are applied to the capture points file and the property files. If the changes that you made require that the Probe be restarted so that the changes that you made can take effect, the Profiler displays messages in the bottom left corner of the Configuration tab as a reminder. The only field that requires a Probe restart is the Capture Class Map check box.

12
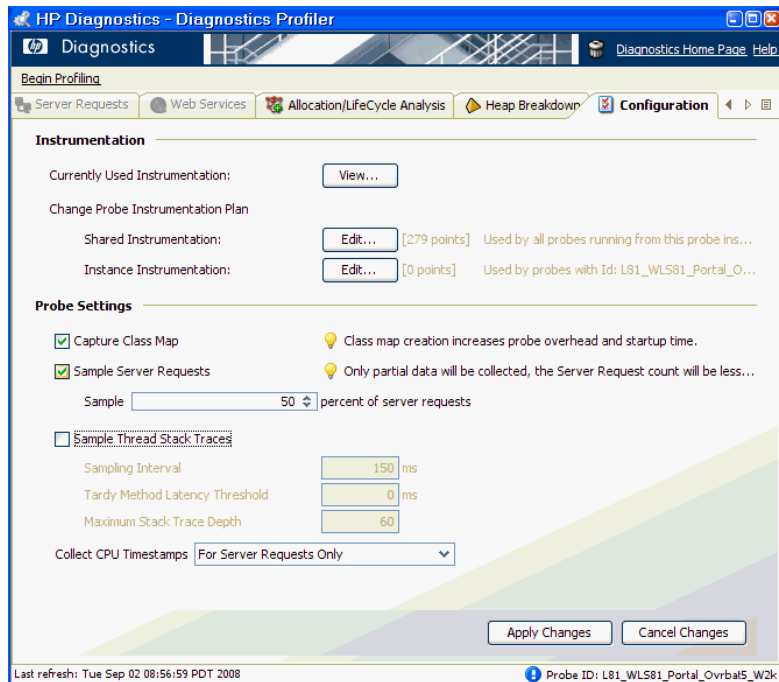
Instrumentation Layers

Default instrumentation layers are defined in the instrumentation points when a probe is first installed.

This chapter includes:

➤ About Instrumentation Layers on page 335

➤ Understanding Diagnostics Layers on page 336

About Instrumentation Layers

HP Diagnostics groups the performance metrics for classes and methods into *layers* and *sub-layers* based upon the instructions provided in the capture points file. The default layers were defined so that the performance metrics for processing in the application that used similar system resources could be reported together. The layers make it easier for you to isolate and identify the areas of the system that may be contributing to performance issues.

For information on maintaining points and layers see the Custom Instrumentation topics for the Probe.

Understanding Diagnostics Layers

The following tables list the default layers and sub-layers that have been defined for typical Java EE and .NET classes and methods, and indicate the points in the capture points file where the classes and methods are assigned to the layers. This information should help you interpret and customize the points in the capture points file.

Platform specific layers have also been defined in the capture points file. These layers are, for the most part, sub-layers of the top-level parent layers defined in the tables below. You can see performance data for layers in the Load View in the Diagnostics UI.

Java EE Layers

Layer	Sub-Layers	Parent Layer	Defined in Points
Web Tier	JSP Servlets Struts		JSP-_jspService Servlet-all Struts-Action
JSP		Web Tier	JSP-_jspService
Servlets		Web Tier	Servlet-all
Struts		Web Tier	Struts-Action
Business Tier	EJB		EJB-Entity-all EJB-Session-all
EJB	Entity Bean Session Bean	Business Tier	EJB-Entity-all EJB-Session-all
Entity Bean		EJB	EJB-Entity-all
Session Bean		EJB	EJB-Session-all
Directory Service	JNDI		JNDI-lookup
JNDI		Directory Service	JNDI-lookup

Layer	Sub-Layers	Parent Layer	Defined in Points
Database	JDBC		JDBC-Connection-create
			JDBC-Connection-prepare
			JDBC-Statement-execute
			JDBC-PreparedStatement-execute
			JDBC-ResultSet
			JDBC-DataSource-getConnection
			JDBC-XADataSource-getConnection
			JDBC-Driver-connect
			JDBC-OCI-Statement-execute
JDBC	Execute Connections	Database	JDBC-Connection-create
			JDBC-Connection-prepare
			JDBC-Statement-execute
			JDBC-PreparedStatement-execute
			JDBC-ResultSet
			JDBC-DataSource-getConnection
			JDBC-XADataSource-getConnection
			JDBC-Driver-connect
			JDBC-OCI-Statement-execute
Execute		JDBC	JDBC-Connection-create
			JDBC-Connection-prepare
			JDBC-Statement-execute
			JDBC-PreparedStatement-execute
			JDBC-ResultSet
			JDBC-OCI-Statement-execute

Layer	Sub-Layers	Parent Layer	Defined in Points
Connections		JDBC	JDBC-DataSource-getConnection JDBC-XADataSource-getConnection JDBC-Driver-connect
Messaging	JMS		JMS-MessageProducer JMS-MessageListener JMS-MessageConsumer
JMS	Producer Listener Consumer	Messaging	JMS-MessageProducer JMS-MessageListener JMS-MessageConsumer
Producer		JMS	JMS-MessageProducer
Listener		JMS	JMS-MessageListener
Consumer		JMS	JMS-MessageConsumer

.NET Layers

Layer	Sub-Layers	Parent Layer	Defined in Points
Web Tier	ASP.NET		
Database	ADO		ADO 1 ADO 2 ADO 3 ADO 4
ADO	Execute Connection	Database	ADO 1 ADO 2 ADO 3 ADO 4

Layer	Sub-Layers	Parent Layer	Defined in Points
Execute		ADO	ADO 1 ADO 2 ADO 3
Connection		ADO	ADO 4
Messaging	MSMQ		MSMQ Synchronous MSMQ Asynchronous MSMQ ReceiveCompleted EventHandler MSMQ PeekCompleted EventHandler
MSMQ		Messaging	MSMQ Synchronous MSMQ Asynchronous MSMQ ReceiveCompleted EventHandler MSMQ PeekCompleted EventHandler

Portal Layers

Diagnostics groups the performance metrics for the classes and method calls associated with processing for portals into Portal Component layers. Each Portal Component layer is broken down to layers for the portal lifecycle methods. For more information about portal layers, see the *HP Diagnostics User's Guide*.

13

Using Solution Templates

Solution templates are predefined instrumentation points included in the capture points file that have been created for several leading application middleware servers and frameworks. The solution templates allow Diagnostics to capture platform specific performance metrics and present them in the Diagnostics views.

This chapter includes:

➤ About Solution Templates on page 342

➤ Understanding Solution Template Data on page 342

➤ Viewing Solution Template Performance Metrics on page 342

About Solution Templates

HP Diagnostics provides solution templates for the following servers and frameworks:

➤ **WebLogic.** The WebLogic Portal Server solution template collects metrics for BEA WebLogic Portal Server WL 8.1 SP3, SP4, SP5, 9.2.

➤ **WebSphere.** The WebSphere Portal Server solution template collects metrics for WebSphere 5.1, 6.1.

➤ **Oracle 11i**. The Oracle 11i solution template collects metrics for the Web-based applications framework.

➤ **SAP.** The SAP solution template collects metrics for iViews in the SAP Web Application Server 6.0, 7.0.

Understanding Solution Template Data

HP Diagnostics groups the performance metrics for classes and methods into *layers* and *sub-layers* based upon the resources that the application invokes to perform the processing. These layers make it easier to isolate and identify the areas of the system that may be contributing to performance issues. For more information on layers see Chapter 12, "Instrumentation Layers."

The solution templates allow you to see performance metrics for business services that are associated with various middleware servers and frameworks in new layers across the Diagnostics application.

Viewing Solution Template Performance Metrics

HP Diagnostics displays unique performance metrics for each of the solution templates. The following discussion describes the performance metrics that are gathered by the instrumentation points in each of the solution templates.

WebLogic

The WebLogic Portal Server solution template lets you view the performance metrics for the following business services in layers across the Diagnostics application:

Service	Layer	Description
Entitlements	Portal/BEA/Entitlement	Entitlement API's
Content management	Portal/BEA/Content	Content management API's
User profiles	Portal/BEA/UserProfile	User profile API's
Personalization	Portal/BEA/Personalization	Personalization API's
Portal components	Portal/BEA/Portlet	Portal component rendering API's

Note: The Portal/BEA/Web Services layer, representing BEA Web Service, is commented out of the points file by default.

The following is an example of a Diagnostics view that displays the WebLogic business services layers with metrics gathered using the solution template.
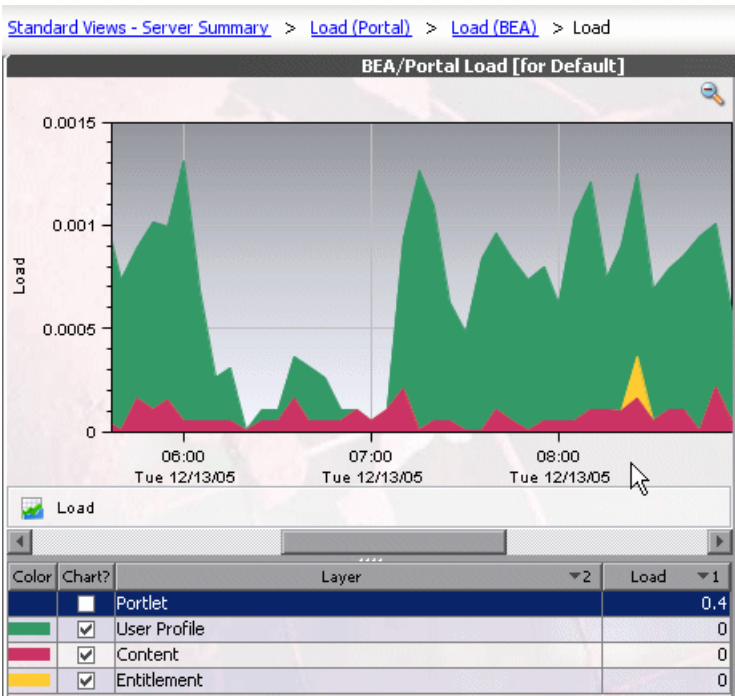
Note: You can also see WebLogic performance metrics on the Portal views in HP Diagnostics. For more information about using the Portal views, refer to the *HP Diagnostics User's Guide*.

WebSphere

The WebSphere Portal Server solution template lets you view the performance metrics for the following business services in layers across the Diagnostics application:

Service	Layer
WAS Portal Gateway Servlet	Portal/IBM/Gateway
WAS Portal Authentication	Portal/IBM/Authentication
WAS Portal PortalAuthorization	Portal/IBM/Authorization
WAS Portal Resource Manager	Portal/IBM/ResourceManager
WAS Portal Portlet Adapter Interface Actions	Portal/Jetspeed/Portlet/Action
WAS Portlet Lifecycle	Portal/Jetspeed/Portlet/Lifecycle
WAS Portal Rendering	Portal/IBM/Portlet/Rendering
WAS Portal Work Manager	Portal/IBM/WorkManager
WAS Portal Portlet Services	Portal/IBM/Portlet/Services
WAS Portal Phases	Portal/IBM/Portlet/Phases
WAS Portal Services	Portal/IBM/Portlet/Services
WAS Portal Portlet Filter	Portal/IBM/Portlet/Filter

Notes:

➤ The Portal/IBM/Portlet/Phases layer, representing WAS Portal Phases, is commented out of the points file by default. Portal Phases do not need to be captured unless you want to understand how the container processes each phase of the portlet's Init-Action-Render cycle.

➤ The Portal/IBM/Portlet/Services layer, representing WAS Portal Services, is also commented out of the points file by default. Many portal services such as the AccessControlService service and PortletFilterService are exposed in other layers, but you might want to uncomment this section if you are interested in some of the other services provided by the portal container.

The following is an example of a Diagnostics view that displays the WebSphere business services layers with metrics gathered using the solution template:
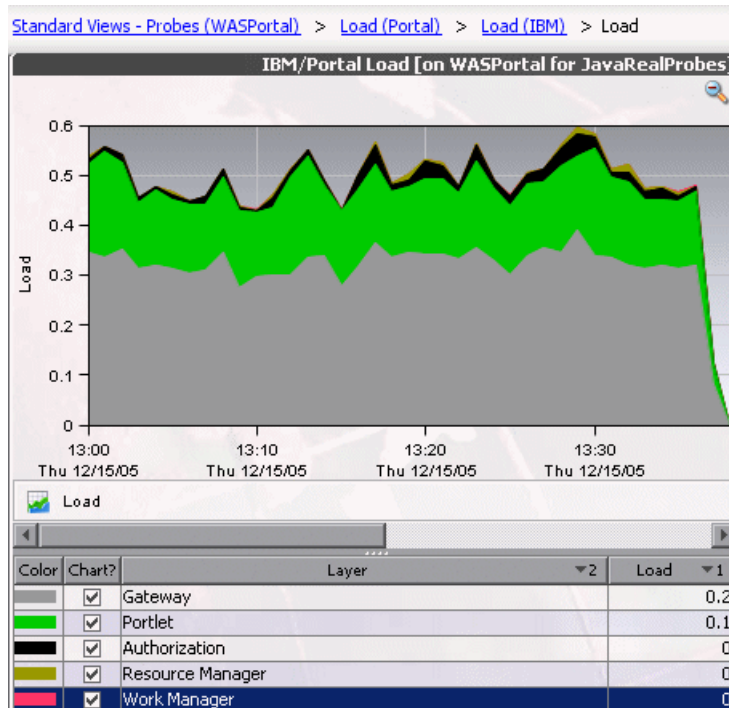
Oracle

The Oracle 11i solution templates enables you to view the performance metrics for the following two layers found underneath the root **Business Tier** layer:

➤ Caching

➤ OAFramework (Oracle Application Framework)

You can drill down further into the OAFramework layer to display the following three layers:

➤ Model

➤ View

➤ Controller

The MVC layers (Model, View, and Controller) relate to the different parts of the Oracle 11i application framework.

The following is an example of a Diagnostics view that displays the MVC layers with metrics gathered using the solution template.
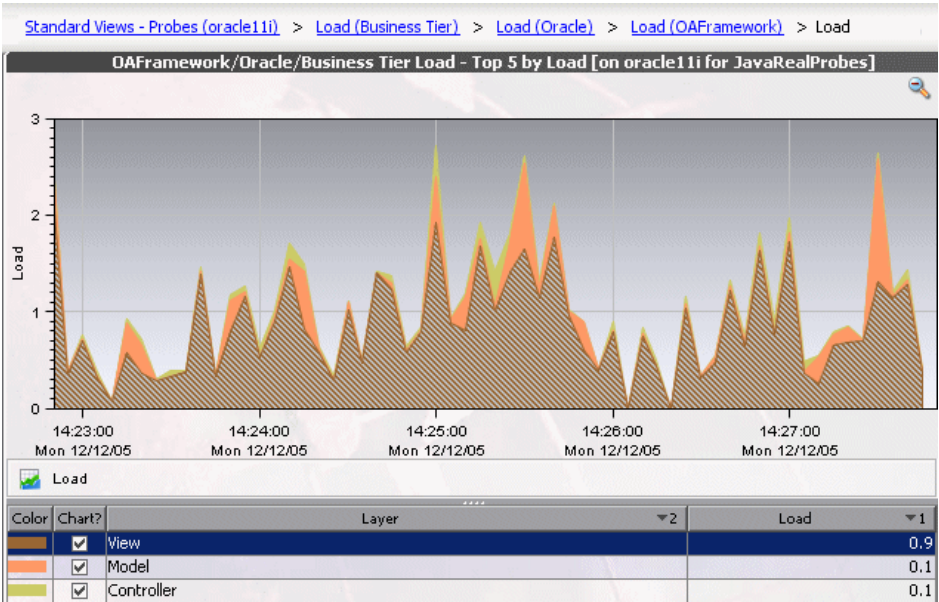
SAP

The SAP Portal Server Solution Template lets you view the performance metrics for the following business services in layers across the Diagnostics application:

Service	Layer	Comment
Runtime	Portal/SAP/Runtime	General runtime part of infrastructure
Authorization	Portal/SAP/Authorization	Core application component dedicated to authorization services
Authentication	Portal/SAP/Authentication	Core application component dedicated to authentication services
JNDI Services:	Portal/SAP/JNDI	Core application component dedicated to JNDI services
Components Runtime	Portal/SAP/Components/ Runtime	General runtime part of Portal components
Content Generation (Portal Components)	Portal/SAP/Components/ Content Generation	Content generation part of Portal components
Content Generation (Service Components)	Portal/SAP/Components/ Service	Content generation Service components (Note: Commented out. Optimize before use)
Content Generation (Request Object Specific)	Portal/SAP/Components/ Request	Content generation Request object specific (Note: Commented out. Optimize before use)
Content Generation (Profile Components)	Portal/SAP/Components/ Profile	Content generation Profile components
Content Generation (Config Components)	Portal/SAP/Components/ Config	Content generation configuration components

Service	Layer	Comment
Content Generation (Response Object Specific)	Portal/SAP/Components/ Response	Content generation Response object specific
R3 Content Generation	Business Tier/SAP R3	Content generation specific to R3 backend
Dynpage Content Generation	Web Tier/SAP/DynPage	Content generation specific to DynPage
JSP Dynpage Content Generation	Web Tier/SAP/JSPDynPage	Content generation specific to JSP DynPage
Page Processor Component	Web Tier/SAP/Page Processor	Content generation specific to the Page Processor

The following is an example of a Diagnostics view that displays the SAP business services layers with metrics gathered using the solution template. The names of the layers are equivalent to the names of the remote function calls (RFCs).
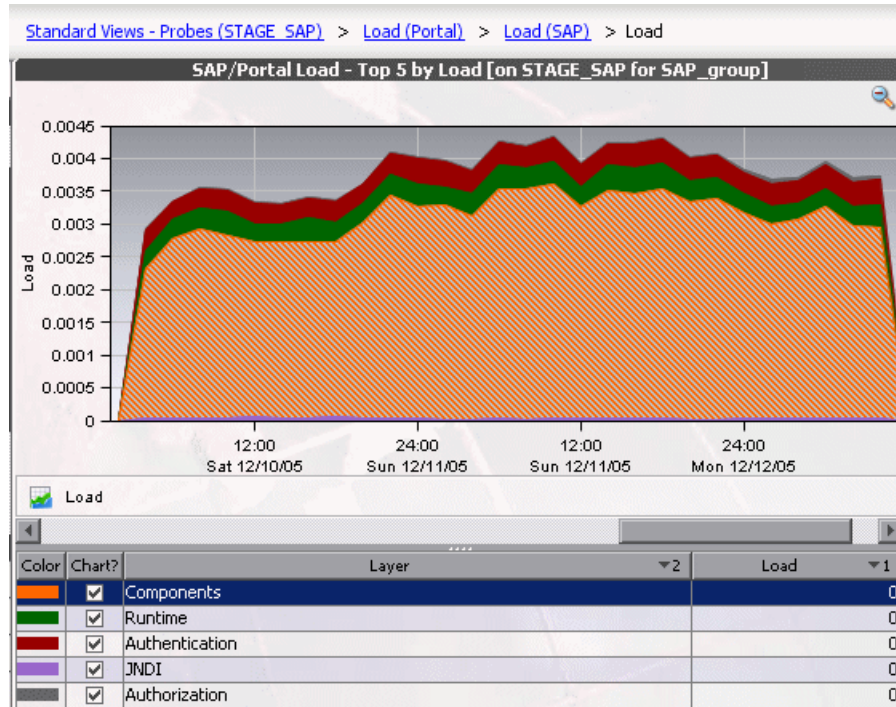
SAP R3 Layer

In a typical SAP Web Application Server deployment (WAS), the SAP WAS is connected to a backend SAP R3 Server. Communication with the SAP R3 Server takes place through RFCs, using SAP JCO (Java Connector). HP Diagnostics captures each distinct call and presents them in sub-layers.

You access these sub-layers by drilling down from the Business Tier layer. For example, if you start from the Server Summary view in HP Diagnostics, you can drill down to the SAP R3 sub-layers as follows: **Standard views - Server Summary** > **Server Requests** > **Layers (Business Tier)** > **Layers (SAP R3)** > **Layers**.

The following is an example of a Diagnostics view that displays the SAP R3 sub-layers with metrics gathered using the solution template.
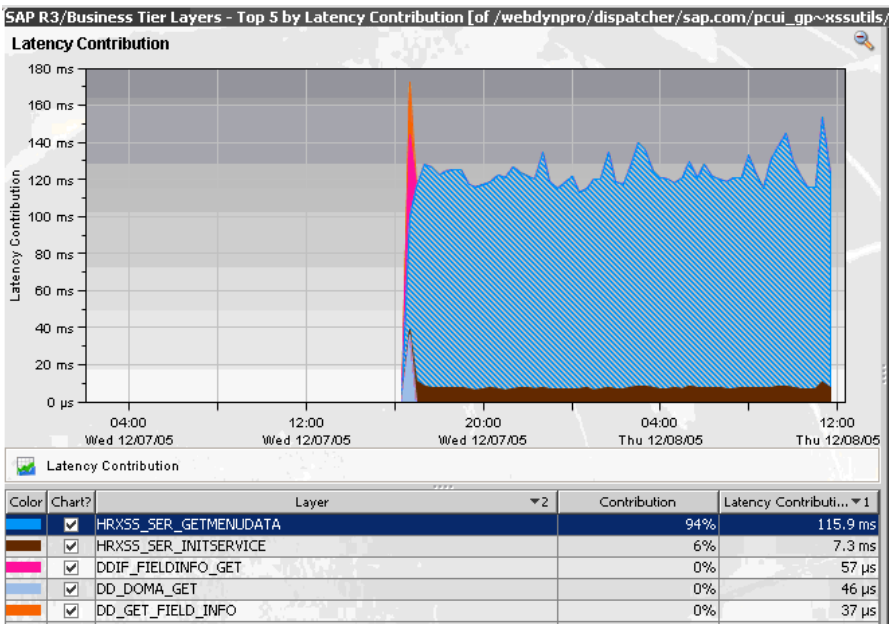
Note: You can also see SAP Portal performance metrics on the portal views in HP Diagnostics. For more information about using the Portal views, refer to the *HP Diagnostics User's Guide*.

Part V

Advanced Configuration of the Diagnostics Server and the Java and .NET Agents

14

Advanced Diagnostics Server Configuration

Instructions are provided for advanced configuration of the Diagnostics Server. Advanced configuration is intended for experienced users with in-depth knowledge of this product. Use caution when modifying any of the Diagnostics components' properties.

This chapter includes:

➤ Optimizing the Diagnostics Server in Production to Handle More Probes on page 379

Synchronizing Time Between Diagnostics Components

In order for Diagnostics data to be stored and correlated properly, it is critical that time is synchronized between the Diagnostics components. To facilitate synchronization of data, the Diagnostics data is adjusted and saved to the synchronized GMT time of the Diagnostics Server in Commander mode. Synchronization makes it possible to display the data correctly for any local time in which the UI may be located.

The sections that follow describe how time synchronization works, and how to configure the components properly so that the time will be synchronized.

Probes running in VMware hosts have additional time synchronization requirements. See "Time Synchronization for Probes Running on VMware" on page 403.

Understanding Time Synchronization

Time synchronization in Diagnostics begins with the Diagnostics Server in Commander mode determining the difference between its time and the GMT time provided by a designated **Time Source**. The **Time Source** to be used is set using the **timemanager.time_source** property in **<diagnostics_server_install_dir>/etc/server.properties**.

The valid values for the **timemanager.time_source** property are:

➤ **NTP.** Indicates that an NTP Server is to be used as the source of GMT time. This is the default value.

The NTP servers that are to be used are listed as values of the **timemanager.ntp.servers** property in **<diagnostics_server_install_dir>/etc/server.properties**.

Note: Ensure that one of the NTP servers in the list can be contacted from the Diagnostics Server, or add your local NTP server as the first server in the list.

➤ **BAC.** Indicates that the registered Business Availability Center core server is to be used as the source of GMT time.

Note: If Business Availability Center is configured to use Database time, you should also configure the Diagnostics Server in Commander mode to use this setting as the time source.

➤ **SERVER.** Indicates that the Diagnostics Server in Commander mode is to be used as the **Time Source**.

This should only be used when the Diagnostics Server is being used in Standalone mode.

The Diagnostics Servers that are in Mediator mode synchronize their time by establishing the time difference between the Diagnostics Server in Mediator mode and the Diagnostics Server in Commander mode.

If the Diagnostics Server in Commander mode has not yet synchronized with the **Time Source**, then the Diagnostics Servers in Mediator mode are considered to be "unsynched". The Diagnostics Servers in Mediator mode that are unsynched will attempt to synchronize their time every 15 seconds until they succeed.

When a Diagnostics probe connects to a Diagnostics Server in Mediator mode or to a Diagnostics Server in Commander mode, the time difference is established between the Diagnostics Server and the probe.

If the probe attempts to connect to a Diagnostics Server that is still "unsynched," then the probe connection is not allowed, and is dropped.

Since the data is stored based upon the GMT, differences in time zones or daylight savings times for the various components are not an issue. For example, the data that is displayed in the Diagnostics UI can be adjusted to display correctly for the time zone in which the UI is running.

Note: All data has been adjusted and saved to the synchronized GMT time of the Diagnostics Server in Commander mode. So, if the UI is running on a machine whose time has not been synchronized properly with the **Time Source**, the data displayed in the UI will appear shifted by the amount of time the UI machine is off from the synchronized GMT time.

Configuring the Time Synchronization on the Diagnostics Server

You synchronize the Diagnostics Server in Commanding mode by performing the procedure below.

Note: Time Synchronization settings for Diagnostics Servers in Mediator mode are ignored because their time is automatically synchronized with the Diagnostics Server in Commander mode.

To ensure that time on the Diagnostics Server in Commander mode can be synchronized:

1 The default configuration for the Diagnostics Server is set such that the value of the **timemanager.time_source** property in **<diagnostics_server_install_dir>/etc/server.properties** is **NTP**.

If the Diagnostics Server has an internet connection and the ability to connect to a server in the list of available NTP servers specified in the **timemanager.ntp.servers** property, then the default configuration will work and no changes are necessary.

Since Business Availability Center also uses NTP for time synchronization by default, this is the recommended setting.

2 If the Diagnostics Server does not have an internet connection or the ability to connect to the list of available NTP servers specified in **timemanager.ntp.servers** property, then you *must* do one of the following:

➤ Setup a local NTP server that can be contacted by the Diagnostics Server in Commander mode. List this local NTP server as the first entry in the **timemanager.ntp.servers** property in **<diagnostics_server_install_dir>/ etc/server.properties**.

Note: It is recommended to have backup NTP servers in case the primary NTP server is not available.

➤ If you are using Diagnostics in a Business Availability Center or HP Software as a Service (SaaS) environment, then you can set the **timemanager.time_source** property in **<diagnostics_server_install_dir>/ etc/server.properties** to **BAC** to indicate Business Availability Center. This causes the Diagnostics Server to connect to the registered Business Availability Center core server to establish the time.

Note: To set up Business Availability Center to use Diagnostics, see Chapter 23, "Setting Up Business Availability Center to Use Diagnostics."

➤ If the Diagnostics Server in Commander mode is to be used in Standalone mode, with no intention of using it with Business Availability Center, and you have no internet connection that will allow you to synchronize time with an NTP server, then you can set the **timemanager.time_source** property in **<diagnostics_server_install_dir>/ etc/server.properties** to **SERVER**. This causes the Diagnostics Server to use its own time as the **Time Source**.

> **Note:** It is recommended that you do not change the value of the **timemanager.time_source** property in **<diagnostics_server_install_dir>/ etc/server.properties** once data has been captured and persisted using the designated **Time Source**. Changing the **Time Source** after data has been captured could result in a significant corruption to the data that has been captured and persisted. This is because the data that was persisted may have been captured while the Diagnostics Server was not synchronized with GMT. If the data that is captured later is captured while the Diagnostics Server is synchronized with GMT, it is possible that data could get re-aggregated multiple times or could get recorded into time buckets where it does not belong.

Configuring the Diagnostics Server for a Large Installation

If you are using a Diagnostics Server in Mediator mode with more than 20 probes, it is recommended that you make modifications to the default configuration of the Diagnostics Server.

> **Note:** These changes to the configuration are not needed for the Diagnostics Server in Commander mode unless it also has probes assigned to it so that it also serves as a Diagnostics Server in Mediator mode.

Adjusting the Heap Size

The size of the heap can impact the performance of the Diagnostics Server in Mediator mode. If the heap is too small you may experience problems where the Diagnostics Server in Mediator mode "hangs" for periods of time. If the heap is too large it is possible that the Diagnostics Server in Mediator mode will experience long garbage collection delays.

The default value for the heap size is 512 MB. The heap size is set in the **server.nanny** file located at:
<diagnostics_server_install_dir>\nanny\windows\dat\nanny
for Windows, or **<diagnostics_server_install_dir>/nanny/solaris/ launch_service/dat/nanny/** for Solaris.

Use the following VM argument to set the size, where **???** is the size in MB:

-Xmx???m

If you encounter problems with the Diagnostics Server in Mediator mode "hanging", you can increase the heap size specified by updating the value specified in the **-Xmx???m** option.

To adjust the heap size of the Diagnostics Server in Mediator mode:

1 Determine the amount of heap that the Diagnostics Server in Mediator mode will need based upon the following table:

Number of Probes	Recommended Heap Size
Up to 50 Java Probes	512 MB
Up to 100 Java Probes	750 MB
Up to 200 Java Probes	1280 MB
Up to 10 .NET Probes	350 MB
Up to 20 .NET Probes	700 MB
Up to 50 .NET Probes	1400 MB

Note: The recommended heap size should not exceed more than 75% of the physical memory of the machine. If a machine has 1 G, then the heap size should not exceed 768 MB.

2 Open the **server.nanny** file that is to be edited. This file is located at:

<diagnostics_server_install_dir>\nanny\windows\dat\nanny
for Windows, or **<diagnostics_server_install_dir>/nanny/solaris/**
launch_service/dat/nanny/ for Solaris.

3 On the start_<platform> line that is appropriate, replace the heap size
specified in the **-Xmx???m** option with the optimal heap size that you
calculated:

-Xmx???m

Continuing the previous example, the current heap size, represented by **???**
is replaced with 768 MB.

-Xmx768m

Before you modify this line in the **server.nanny** file it will look like this:

```
start_nt="C:\MercuryDiagnostics\Server\jre\bin\javaw.exe" -server -Xmx512m
-Dsun.net.client.defaultReadTimeout=70000
-Dsun.net.client.defaultConnectTimeout=30000
"-javaagent:C:\MercuryDiagnostics\Server\probe\lib\probeagent.jar"
-classpath "C:\MercuryDiagnostics\Server\lib\mediator.jar;
C:\MercuryDiagnostics\Server\lib\loading.jar;
C:\MercuryDiagnostics\Server\lib\common.jar;
C:\MercuryDiagnostics\Server\lib\mercury_picocontainer-1.1.jar"
com.mercury.opal.mediator.util.DiagnosticsServer
```

After you modify this line in the **server.nanny** file it will look like this:

```
start_nt="C:\MercuryDiagnostics\Server\jre\bin\javaw.exe" -server -Xmx768m
-Dsun.net.client.defaultReadTimeout=70000
-Dsun.net.client.defaultConnectTimeout=30000
"-javaagent:C:\MercuryDiagnostics\Server\probe\lib\probeagent.jar"
-classpath "C:\MercuryDiagnostics\Server\lib\mediator.jar;
C:\MercuryDiagnostics\Server\lib\loading.jar;
C:\MercuryDiagnostics\Server\lib\common.jar;
C:\MercuryDiagnostics\Server\lib\mercury_picocontainer-1.1.jar"
com.mercury.opal.mediator.util.DiagnosticsServer
```

Adjusting the Amount of Data Pulled from the Probe

Large call profiles will require significant network bandwidth between the probe and server, and significant CPU resources on the server.

If the network becomes a bottleneck (e.g. network utilization above 25% on a mediator as observed in Windows task manager and/or probes report less than 100% availability although they were up), it is recommended to reduce the data that is generated via trimming, to enable compression if the probe system's CPU is not fully utilized and/or reduce the frequency of the data that the server pulls from the probe.

The main trimming parameters on the probe are:

➤ In the **capture.properties** file:

   ➤ maximum.stack.depth = 25

   ➤ maximum.method.calls = 1000 (e.g. can be set to 25 to limit overall number of methods in a Call Profile)

   ➤ minimum.method.latency = 51ms

➤ In the **dispatcher.properties** file:

   ➤ minimum.fragment.latency = 51ms (e.g. can be increased to 101ms)

See the following for more information on trimming: "Configuring Fragment Name Based Trimming" on page 371 for the server, "Configuring Latency Trimming and Throttling" on page 488 and "Configuring Depth Trimming" on page 493 for .NET probes, "Controlling Automatic Method Trimming on the Probe" on page 396 and "Controlling Probe Throttling" on page 398 for Java probes.

To enable compression, on the probe set **webserver.properties: rhttp.gzip.replies = true**. This reduces network traffic on the server significantly. However, the probe (and server) will require additional CPU for compression.

Another way of decreasing the network traffic is to change the frequency that data is pulled from the probe. By default, trends are pulled every 5s and trees (Call Profiles) are pulled every 45s. To lower the frequency for call trees, change **probe.trees.pull.interval** on the mediator in the **server.properties** file (for example: 90s or 240s depending how many methods a Call Profile contains). It is recommended to first lower the pull frequency of call trees. If this is not enough, the trend pull frequency can be lowered as well by changing **probe.trends.pull.interval** (for example to 10s).

Note, changing any of these parameters requires a restart of the probe or server.

Additional Adjustments

If more than 50 probes are connected, it is recommended to increase the number of threads used for pulling data from the probe. For each mediator, set **probe.pull.max.threads=30** and restart the server.

In addition, you can increase the number of threads available for jetty by setting **webserver.properties, jetty.threads.max=500**.

In case call tree and trend files (see also Appendix E, "Diagnostics Data Management") become quite large in their uncompressed state (greater than 4GB), it is recommended to offload some of the probes to a new mediator. Otherwise, the aggregation and compression of the files may start to lag due to the large amount of data.

When a large number of probes are connected to a server, the default purging setting of 5GB might not be enough. For more information see "Data Storage Size & Data Retention" on page 679.

Overriding the Default Diagnostics Server Host Name

In situations where a firewall or NAT is in place, or where the host for the Diagnostics Server in Mediator mode has been configured as a multi-homed device, it may not be possible for the Diagnostics Server in Commander mode to communicate with the Diagnostics Server in Mediator mode using the host name that was assigned when the Diagnostics Server in Mediator mode was installed. The **registered_hostname** property allows you to override the default host name that the Diagnostics Server in Mediator mode uses to register itself with the Diagnostics Server in Commander mode.

To override the default host name for a Diagnostics Server in Mediator mode set the **registered_hostname** property located in **<diagnostics_server_install_dir>/etc/server.properties** to an alternate machine name or IP address that will allow the Diagnostics Server in Commander mode to communicate with the Diagnostics Server in Mediator mode.

Changing the Default Diagnostics Server Port

If the configuration of the Diagnostics Server host will not allow the default Diagnostics port to be used, you may choose a different port for the Diagnostics Server communications with the probes and other Diagnostics Servers.

Important: Make sure that the new port number is not already used by another application and that the other Diagnostics components can communicate with this port.

If you decide to use an alternative port number after you have deployed Diagnostics, you must update the properties in the following table for each of the indicated components in your deployment with the new port number in order to ensure that the proper communications can take place.

Component Type	Properties
Diagnostics Server in Commander mode	**<diagnostics_server_install_dir>/etc** ➤ webserver.properties - jetty.port ➤ mediator.properties - commander.url ➤ probe/etc/dispatcher.properties - registrar.url
Diagnostics Server in Mediator mode	**<diagnostics_server_install_dir>/etc** ➤ mediator.properties - commander.url **<diagnostics_server_install_dir>/probe/etc** ➤ dispatcher.properties - registrar.url
Probes	**<probe_install_dir>/etc** ➤ dispatcher.properties - registrar.url

Configuring the Diagnostics Server for Multi-Homed Environments

The machines that host the Diagnostics Server can be configured with more than one Network Interface Card (NIC), and the Diagnostics Server process listens on all interfaces on its host. Some customer environments do not allow applications to listen on all network interfaces on a machine. If your environment has this restriction, use the following instructions to configure the Diagnostics Server to listen on specific network interfaces.

Setting the Event Host Name

If the Diagnostics Server host has multiple network interfaces, and you want to specify the hostname that the Diagnostics Server will listen on, you need to set the **event.hostname** property.

This property can be found in:

<diagnostics_server_install_dir>/etc/server.properties

Uncomment the property, **event.hostname**, and specify the hostname value.

By default, the **event.hostname** property is not set. This means that the Diagnostics Server will listen on all hostnames.

Modifying the jetty.xml File

The **jetty.xml** file has a section which defines the interfaces on which the Diagnostics Server is permitted to listen. By default, the **jetty.xml** file included with the Diagnostics Server has no listeners defined and the Diagnostics Server listens on all of the interfaces.

To configure the Diagnostics Server to listen on specific network interfaces on a machine:

1 Open <**diagnostics_server_install_dir**>/**etc/jetty.xml** and locate the following line:

```
<Configure class="org.mortbay.jetty.Server">
```

2 Add the following block of code after this line, changing the <Set name="Host">……</Set> to contain the NIC's IP address.

```
<Call name="addListener">
  <Arg>
   <New class="org.mortbay.http.SocketListener">
    <Set name="Host">127.0.0.1</Set>
    <Set name="Port"><SystemProperty name="jetty.port" default="2006"/></Set>
    <Set name="MinThreads">1</Set>
    <Set name="MaxThreads">5</Set>
    <Set name="MaxIdleTimeMs">30000</Set>
    <Set name="LowResourcePersistTimeMs">5000</Set>
    <Set name="ConfidentialPort">8443</Set>
    <Set name="IntegralPort">8443</Set>
   </New>
  </Arg>
 </Call>
```

3 Repeat the previous step adding a new copy of the block of code and setting the IP address for the NIC for each interface on which the Diagnostics Server is to listen.

Make sure that the **</Configure>** tag follows the listener code for the last NIC.

Note: Make sure that components that access the Diagnostics Server can resolve the hostnames of the Diagnostics Server to the IP address that you specify in the **jetty.xml** file for the host values. It is possible that some systems may resolve the host name to a different IP address on the Diagnostics Server host. For more information, see "Overriding the Default Diagnostics Server Host Name" on page 365.

Sample jetty.xml File

The following example shows the **jetty.xml** file for the Diagnostics Server, where the Diagnostics Server will listen on loopback and one IP address on the system.

```xml
<!-- Configure the Jetty Server                              -->
<!-- ===================================================== -->
<Configure class="org.mortbay.jetty.Server">
<!--===================================================== -->
<!-- Configure the Request Listeners                       -->
  <!--===================================================== -->
<Call name="addListener">
   <Arg>
    <New class="org.mortbay.http.SocketListener">
     <Set name="Host">127.0.0.1</Set>
     <Set name="Port"><SystemProperty name="jetty.port" default="2006"/></Set>
     <Set name="MinThreads">1</Set>
     <Set name="MaxThreads">5</Set>
     <Set name="MaxIdleTimeMs">30000</Set>
     <Set name="LowResourcePersistTimeMs">5000</Set>
     <Set name="ConfidentialPort">8443</Set>
     <Set name="IntegralPort">8443</Set>
    </New>
    </Arg>
  </Call>

<-Listen on specific IP Address on this machine for incoming Commander calls->
<Call name="addListener">
  <Arg>
   <New class="org.mortbay.http.SocketListener">
    <Set name="Host">10.241.3.109</Set>
    <Set name="Port"><SystemProperty name="jetty.port" default="2006"/></Set>
    <Set name="MinThreads">1</Set>
    <Set name="MaxThreads">5</Set>
    <Set name="MaxIdleTimeMs">30000</Set>
    <Set name="LowResourcePersistTimeMs">5000</Set>
    <Set name="ConfidentialPort">8443</Set>
    <Set name="IntegralPort">8443</Set>
   </New>
  </Arg>
 </Call>
</Configure>
```

Reducing Diagnostics Server Memory Usage

The Transaction Timeout Period is a safety mechanism that is used to prevent the Diagnostics Server from using excessive amounts of memory because it is holding on to old data for too long. The Diagnostics Server holds on to all of the information that it receives for a transaction until it receives the End of Transaction Notification (ELT), which tells the Diagnostics Server the transaction is complete. The timeout period for a transaction is reset each time the Diagnostics Server receives data for the transaction.

If the machine that the Diagnostics Server in Commander mode is running on is overloaded (CPU is heavily loaded or there are too many transactions per second for it to handle), or if there are network connectivity issues between the Load Generators or Business Availability Center and the Diagnostics Server in Commander mode, or between Business Process Monitor and Business Availability Center, the Diagnostics Server may not receive the ELT that lets it know when a transaction has ended. If the ELT is not received by the time the transaction timeout period expires, the Diagnostics Server assumes that the ELT is not coming and proceeds to process the data for the transaction and free up the memory that the transaction data is using.

The **correlation.txn.timeout** property sets the duration of the transaction timeout period. If you are experiencing out of memory conditions in the Diagnostics Server, you may want to reduce the transaction timeout period so that the Diagnostics Server waits less time for the end of a transaction. Use caution when adjusting the value of this property because multiple probes may be sending data to the Diagnostics Server, and an active transaction may be idle in one Diagnostics Server, so setting the value of this property too low could cause transactions to be reported incorrectly. It is recommended that if you need to reduce the value of this property, set it to 90 seconds more than the longest transaction in your test.

Configuring Fragment Name Based Trimming

Fragment name based trimming lets you configure Diagnostics to filter out server requests that appear to be causing Diagnostics Server performance issues without changing the configuration or the instrumentation in the probes.

Note: Fragment name based trimming is not intended to be used instead of the latency and depth trimming that you configure on the probes.

Using the **trim.fragment** properties in the
<diagnostics_server_install_dir>\etc\trimming.properties file you can specify the names of the fragments that Diagnostics is to trim. Diagnostics trims the fragments for both Real User and Virtual User server requests.

By default, the properties **trim.fragment.1** and **trim.fragment.2** are commented out in **trimming.properites.** To specify a fragment to be trimmed, uncomment one of the properties and type the fragment name that is to be trimmed as it is listed in the Diagnostics views. If you have more than two fragments to be trimmed you can create additional **trim.fragment** properties being sure to increment the number at the end to ensure that each property name is unique. For example, the next **trim.fragment** property would be named **trim.fragment.3**.

Events and fragments that are trimmed as a result of these property settings are counted in the dropped event and dropped fragment counts.

Preparing a High Availability Diagnostics Server

If your Diagnostics deployment requires that the Diagnostics Server have a high availability, you can use the following instructions to create a standby Diagnostics Server for each Diagnostics Server. The standby is then ready to be used in the event of a hardware failure or some other problem with the host of the Diagnostics Server.

Creating a Standby Diagnostics Server

You can create a standby for each Diagnostics Server by installing the Diagnostics Server onto a standby machine and then periodically replicating the primary Diagnostics Server data into the standby Diagnostics Server.

To configure a standby Diagnostics Server:

1 Install the Diagnostics Server onto the standby machine. Make sure that the version of the Diagnostics Server to be installed on the standby server is the same as the Diagnostics Server on the primary server.

2 Schedule a periodic remote backup of the primary server into the standby server using the following commands from the host of the standby Diagnostics Server:

```
% cd /opt/MercuryDiagnosticsServer/
% ./bin/remote-backup.sh -h <primary_server_host> -o .
```

Replace **<primary_server_host>** with the host name for the Diagnostics Server that is being replicated.

These commands perform an incremental replication of the Diagnostic data, configuration files, customized views, alerts and comments onto the standby Diagnostics Server. You can schedule the periodic backup using a cron job or a scheduled task on Windows.

Note: The **wget** utility is used to download the backup over HTTP. For Windows, you must have an installation of **cygwin** on the host for the Diagnostics Server. You can get a copy of **cygwin** at http://www.cygwin.com/.

Failover to the Standby Diagnostics Server

If the host for the primary Diagnostics Server fails, you must configure the standby Diagnostics Server so that it can begin to function as the primary Diagnostics Server.

To make the standby Diagnostics Server the primary Diagnostics Server:

1 Change the hostname of the standby Diagnostics Server to match the hostname of the failed host of the primary Diagnostics Server. This allows the probes to reconnect to the Diagnostics Server when it is started.

2 Start the standby Diagnostics Server as a Windows Service, or by using the **bin/server.sh** or **bin\server.cmd** scripts.

3 The probes reconnect to the Diagnostics server. Whenever a probe loses its connection to its Diagnostics Server it attempts to reconnect approximately every 30 seconds.

4 The standby Diagnostics Server is now the primary Diagnostics Server. Configure a new standby Diagnostics Server as described in "Creating a Standby Diagnostics Server" on page 372.

Note: When the failed Diagnostics Server host is recovered, you should not make it the primary Diagnostics Server because you will lose any data that was gathered from the probes while the new primary Diagnostics Server was being used.

LoadRunner / Performance Center Diagnostics Server Assignments

Default Diagnostics Server Assignment

By default, a probe that is selected for a LoadRunner or Performance Center run will use the Diagnostics Server that is specified in its **<probe install directory>/etc/dynamic.properties**.

Diagnostics Server Assignment Override

It is possible to override the probe configuration when the probe is started for a run. To do so, you modify a mapping file on the Diagnostics Server in Commanding Mode, allowing you to override the Diagnostics Server assignment for a probe.

This can be useful when you are running Diagnostics in a combined LoadRunner / Performance Center and Business Availability Center environment. In such a case, you may want the probes to use different Diagnostic Servers when they are in a LoadRunner / Performance Center run than they do for Business Availability Center monitoring. You can do so by following these Diagnostics Server override instructions.

It may also be more convenient to use this mechanism than to edit the probe configuration file.

Note: When the probe is not in a run, it will always use the Diagnostics Server specified in its **<probe_install_dir>etc/dynamic.properties file**.

To override the Diagnostics Server Assignment for a probe, modify the **server_assignment.properties** file in the **<diagnostics_server_install_dir>\etc** directory on the Diagnostics Server in Commander mode host machine.

The format of the **server_assignment.properties** file is:

```
<ProbeID> = <Server.id>
```

- Replace <**ProbeID**> with the ID of the probe.
- Replace <**Server.id**> with the ID of the Diagnostics Server.

The **server_assignment.properties** file is dynamically read at the start of each LoadRunner / Performance Center run. This means that changes made to this file become effective without having to restart the Diagnostics Server in Commander mode.

Configuring the Diagnostics Server for LoadRunner Offline Analysis File Size

For each LoadRunner scenario or Performance Center test that is run, the Diagnostics Server in Mediator mode produces a file that is needed for LoadRunner Offline analysis which contains the Java data captured during the scenario. The size of this file can grow to be quite large. You must ensure that you have enough disk space to hold the LoadRunner Offline file on both the Diagnostics Server in Mediator mode host machine where the file is stored temporarily while the scenario is running and the Load Runner controller host machine where the file is stored when the scenario has ended.

Estimating the Size of the LoadRunner Offline File

Estimating the size of the offline file is highly dependent upon the data that is being captured and the rate at which the data is captured.

To estimate the size of the LoadRunner offline file:

1 Run a load test for 5 minutes and monitor the size of the offline file created by the Diagnostics Server in Mediator mode when Load Runner scenario is started.

Locate the offline file on the Diagnostics Server in Mediator mode host machine in **<diagnostics_server_install_dir>/data/<newest directory>**. The offline file has an extension of **.inuse**.

2 After 5 minutes note the size of the offline file.

3 Extrapolate the size of the offline file after one hour by multiplying the size of the offline file from the previous step by 12.

4 Determine the anticipated size of the offline file at the end of your load test by multiplying the one hour file size calculated in the previous step by the number of hours that you expect your actual load test to run.

5 Determine if the Diagnostics Server in Mediator mode host machine and the Controller host machine have enough disk space to accommodate the anticipated offline file size.

Reducing the Size of the LoadRunner Offline File

If you are concerned about the size of the offline file, you can reduce the file size by increasing the offline aggregation periods for the Diagnostics Server in Mediator mode. This will reduce the level of granularity in the offline data and thus reduce the size of the offline files.

The default settings for these properties are **5s** (5 seconds) which causes the Diagnostics Server in Mediator mode to aggregate all data into 5 second time slices. Increasing the value of these properties will make the offline file smaller because fewer data points are required to be stored when the aggregation period is longer. For example, increasing the offline aggregation period properties to **45s** should reduce the file size by roughly 50-75%.

Note: The impact on the size of the offline file size that will be achieved by adjusting the offline aggregation period is highly dependent upon the behavior of the application and the specifics of your load test.

Use the following steps modify the Diagnostics Server in Mediator mode offline aggregation period properties **bucket.lr.offline.duration** and **bucket.lr.offline.sr.duration** in **<diagnostics_server_install_dir>/etc/mediator.properties**.

To reduce the size of the offline files by increasing the Diagnostics Server in Mediator mode offline aggregation periods:

1 Ensure that the Diagnostics Server in Mediator mode is not participating in any active LoadRunner / Performance Center runs. This is necessary because the Diagnostics Server in Mediator mode must be restarted before the property changes described in the following steps will take effect.

2 Access the Mediator Configuration Page from your browser by navigating to the following URL:

http://<diagnostics_server_hostname>:8081/configuration/Aggregation?level=60

3 Increase the Offline VU Aggregation Period by increasing the setting for the **Load Runner / Performance Center Offline VU Aggregation Period** property. The value of this property must be a multiple of 5. For example set it to 45s.

4 Increase the Offline Server Request Aggregation Period by increasing the value of the **Load Runner / Performance Center Offline Server Request Aggregation Period** property. The value of this property must be a multiple of 5. For example set it to 45s.

5 Update the Diagnostics Server in Mediator mode with the revised property values by clicking **Submit** at the bottom of the page.

A message will appear at the top of the page to indicate that the changes were saved along with a reminder to restart the Diagnostics Server in Mediator mode. As a further reminder the **Restart Mediator** button is displayed.

For more information on updating property values from the Configuration Page and a screen image showing the command buttons, see "Configuring Diagnostics Using the Diagnostics Server Configuration Pages" on page 585.

To cause the configuration changes to take effect restart the Diagnostics Server in Mediator mode by clicking **Restart Mediator**.

Configuring Business Availability Center Samples Queue Size and Web Services CI Frequency

These configurations are applicable to Business Availability Center integrations.

Configuring Business Availability Center Samples Queue Size

BAC Samples queue size, by default, is set to 100. In cases where more than 100 samples are created at once, some of the samples will be dropped resulting in missing data in BAC for SOA. You will see the following message in the log: BAC samples being dropped since too many are waiting for delivery.

You can increase the samples queue size by setting the server property, **bac.webservice.delivery.max.queue.size** to configure the WDEDelivery queue size.

Frequency of Web Service CIs

The Web Services CIs are created and added to the uCMDB automatically by Diagnostics using a default frequency.

In some cases you may want to change the timing of the process that adds the Web Services CIs to the uCMDB. The Web Service CI population process has these configuration properties defined in server.properties;

➤ **bac.webservice.CI.create.runfrequency** - the number of seconds between population runs (default=360, 5 minutes)

➤ **bac.webservice.CI.create.query.granularity** - the granularity of the Diagnostics query used to identify Web Service CIs to create (default=1d)

Configuring Diagnostics Using the Diagnostics Server Configuration Pages

The Diagnostics Server Configuration pages provide a way for you to set the property values that control how the Diagnostics Server communicates with the other Diagnostics components, and how it processes the data that it receives from the probes.

Note: To ensure that you are entering valid property values you should use these pages to update the Diagnostics Server properties rather than editing the property files directly.

For information about viewing and modifying Diagnostics using the Diagnostics Server Configuration pages, see Appendix A, "Diagnostics Server Administration Page."

Optimizing the Diagnostics Server in Production to Handle More Probes

The number of probes that a single diagnostic server process can handle depends largely on the number of unique server requests per 5-min interval, and the number of methods and layers in each server request. The following are some of the optimizations that will help to increase the number of probes that can be handled per server process.

➤ The default setting is for the diagnostic server to pull the trends from each probe every 5 seconds, and the trees from each probe every 45s. If a single diagnostic server process is handling more than 25 probes, then this could be optimized such that the trends and trees are pulled less often. A suggested optimal setting in production is a 30 sec trend pull interval, and a 120 second tree pull interval. These values can be configured in **<diagnostics_server_install_dir>\Server\etc\server.properties** as follows:

```
# The interval at which to pull trends from probes
probe.trends.pull.interval = 30s

# The interval at which to pull trees from probes
probe.trees.pull.interval = 120s
```

➤ The maximum heap size of the server process is determined by the **-Xmx** parameter in the server's startup script. The default setting is 512MB for maximum heap size. Increase the maximum heap size as appropriate based on the load from the probes. The suggested values for maximum heap size, based on the number of probes to be handled is available in Chapter 1, "Preparing to Install HP Diagnostics."

➤ A 1Gbps link is strongly recommended in production for the diagnostic server, when the server is handling more than 30 probes.

➤ If a single server process is handling more than 75 probes, increase the number of jetty threads. The general rule of thumb for sizing the number of threads is twice the number or probes + 40. The default value is 200. The number of jetty threads can be increased by modifying the **jetty.threads.max** property in **<diagnostics_server_install_dir>\Server\etc\ webserver.properties**. For example,

```
jetty.threads.max=300
```

15

Advanced Java Probe and Application Server Configuration

Instructions are provided for advanced configuration of the Java Probe and application servers. Advanced configuration is intended for experienced users with in-depth knowledge of this product. Use caution when modifying any of the Diagnostics components' properties.

This chapter includes:

➤ Authentication and Authorization for Java Diagnostics Profilers in Standalone Mode on page 409

➤ Configuring Collection of CPU Time Metrics on page 412

➤ Configuring Consumer IDs on page 415

➤ Configuring SOAP Fault Payload Data on page 424

➤ Configuring REST Services on page 425

➤ Grouping JMS Temporary Queue/Topics on page 425

Advanced Configuration Directory

The following bullet points will help guide you to the probe configuration topic that will enable you to configure the probe to address particular issues in your environment.

➤ If you have a probe that was installed to work with one HP Software product and you now would like to work with other products, see "Configuring the Probe to Work with Other HP Software Products" on page 383.

➤ If your application server is using multiple JVMs or if you want to capture data for multiple JVMs you must perform additional probe configuration steps described in "Configuring the Probes for Multiple Application Server JVM Instances" on page 387.

➤ If you have a probe that you want to prevent others from using in Profiler mode, see "Disabling the Java Diagnostics Profiler" on page 392.

➤ If you have more than one JVM using a single probe installation, you may need to set some of the probe properties in the Java system properties in the application start up commands. For more information, see "Specifying Probe Properties as Java System Properties" on page 392.

➤ If you want to have log messages posted to the probe logs for lower level messages, you can adjust the log level as described in "Controlling Probe Logging" on page 393.

➤ If you have more than one probe installed on the same host you can make sure that the log messages for each probe are stored in a different file as explained in "Changing the Log Directory for a Probe" on page 394.

➤ If you want to examine the performance of processing that would normally be trimmed from the metrics reported in Diagnostics you can reduce the level of trimming, or turn off trimming completely as described in "Controlling Automatic Method Trimming on the Probe" on page 396.

➤ In some situations, you may want to turn off probe throttling so that you can be sure that you are getting all of the available performance information without regard to the performance impact on the application itself. For information on throttling and how to control it see, "Controlling Probe Throttling" on page 398.

➤ If there is a proxy present between the probe and the Diagnostics Server in Commander mode you must set the correct property to tell the probe the URL of the Diagnostics Server in Commander mode. For more information, see "Configuring a Probe for a Proxy Server" on page 401.

➤ If you have installed a Java Probe that is to be used in a HP Software as a Service (SaaS) environment you are encouraged to disable the reverse http (rhttp) communication between the probe and the Diagnostics Server in Mediator mode. For more information, see "Configuring Reverse HTTP for a Probe in SaaS" on page 401.

Configuring the Probe to Work with Other HP Software Products

This section includes:

➤ "Adding an HP Software Product to the Probe Configuration" on page 383

➤ "Removing a Product from the Probe Configuration" on page 386

Adding an HP Software Product to the Probe Configuration

The Java Probe is a lifecycle probe that can be can be configured to monitor your applications from development through implementation and production. The Java Probe can be configured to work with several HP Software products or to work as a Diagnostics Profiler without other Diagnostics components or HP Software products.

The products with which the Java Probe can work is determined by the value of the **active.products** property located in the property file **<probe_install_dir>/etc/probe.properties**.

The value of the **active.products** property is set at the time that you install the Java Probe. For information on probe installation see Chapter 5, "Installing Java Agents.". To enable the probe to capture data for different products you must set the value of the **active.products** property by editing the property file and restarting the application server.

Note: If you want to use the Java Probe that was downloaded with the Diagnostics Profiler for Java with other HP Software products, contact HP Software Customer Support. The instructions that follow will not work for a probe installed for the Java Diagnostics Profiler without intervention from HP Software Customer Support.

To see Diagnostics data in the user interface of the interfacing HP Software products there are additional configuration steps that must be completed. See the chapters in this guide that correspond to the HP Software product that you are using to complete the configuration.

The sections that follow provide instructions for configuring each product mode.

PRO Product Mode – Diagnostics Profiler for Java

For Diagnostics Profiler for Java, the product mode is PRO. In this mode, the probe gathers additional metrics and presents those metrics on a user interface that it makes available through a URL on the probe host.

If you are running the Java Probe as part of the Java Diagnostics Profiler in Development Mode, there are restrictions placed upon the probe to limit the load that the probe can handle.

If you are running the Java Probe as part of Diagnostics Standalone, or along with another HP Software product, the Profiler is enabled without the load restrictions.

Enterprise Product Mode

When configured in Enterprise mode, the probe works with the HP Software products such as Business Availability Center, LoadRunner, Performance Center, and as Diagnostics Standalone.

To configure the probe for Enterprise mode:

1 Set the **active.products** property to **Enterprise** and restart the Application Server.

2 Configure the probe to register with the Diagnostics Server in Commander mode.

One of the functions of the Diagnostics Server in Commander mode is to keep track of the Diagnostics components so that it can facilitate communication between them and keep you informed about the status and health of the components.

To configure the probe to register with the Diagnostics Server, set the host name and port using the **registrar.url** property which can be found in the property file: **<probe_install_dir>\etc\dispatcher.properties**.

Below is an excerpt from **dispatcher.properties** showing the **registrar.url** property:

```
## the URL of the registrar
registrar.url=http://host01.company.com:2006/registrar/
```

3 Configure the probe to connect to the Diagnostics Server in Mediator mode.

The probe must be able to transmit the processing metrics that it gathers to the Diagnostics Server in Mediator mode in order for Business Availability Center, LoadRunner, and Performance Center to be able to receive, process, and display the reports. In LoadRunner and Performance Center, Diagnostics assigns a Diagnostics Server to the probe. In Business Availability Center, you must tell the probe which Diagnostics Server to work with.

To configure the probe to communicate with the Diagnostics Server in Mediator mode, set the host name and the port using the **mediator.host.name** and **mediator.port.number** properties which can be found in **<probe_install_dir>\etc\dynamic.properties**.

The default port for the Diagnostics Server in Mediator mode is 2006. If, when installing the Diagnostics Server in Mediator mode, you set the port for the Diagnostics Server in Mediator mode to a value other than the default, make sure to use that same port number when you set the **mediator.port.number** property for the probe.

Below is an excerpt from the **dynamic.properties** file showing these properties:

```
#the host the Topaz mediator is running on
mediator.host.name=host01.company.com

#the port the Topaz mediator is listening on (default is 2612)
mediator.port.number=2612
```

AM Product Mode

To protect a probe in a production Business Availability Center deployment from accidently being included in a LoadRunner or Performance Center run, set **active.products** to AM. When in AM mode the probe is not listed as an available probe in LoadRunner or Performance Center.

AD Product Mode

To prevent a probe in a QA environment from using additional resources and continually reporting data to the Diagnostics console dataset when a load test is not running set **active.products** to AD.

Removing a Product from the Probe Configuration

The product mode for the probe is configured using the **active.products** property located in the file **<probe_install_dir>\etc\probe.properties**.

To configure the probe so that it will no longer support a product, update the **active.products** property so that the product is no longer included in the property value.

Configuring the Probes for Multiple Application Server JVM Instances

When your application server is using multiple JVMs, or when you want to capture data for multiple JVMs, you must perform additional probe configuration steps. There are two options; you may install one probe for each JVM on a host, or you may install one probe that will be shared by all of the JVMs.

This section includes:

Configuring Multiple JVMs to Use a Single Probe Installation

To allow multiple JVMs to share a single probe installation, you must configure a separate instance of the probe for each JVM. This configuration enables the following:

➤ Establishment of communication between the JVM and the probe

➤ Identification of the probe by the JVM

➤ Instrumentation of the JVM to instruct the probe about the performance metrics it will monitor

To configure a Java Probe to work with multiple JVMs:

1 When a probe is being used to monitor multiple JVM versions, the JRE Instrumenter must be run once for each JVM version to enable the probe to monitor the events of the applications that are running on the JVMs.

If you have not already run the JRE Instrumenter for each of the JVM versions that the probe will be working with, do so now. See "Running the JRE Instrumenter" on page 141.

Notes:

➤ You must run the JRE Instrumenter even if you let the installer instrument the JRE when the probe was installed. The JRE Instrumenter prepares the applications for multi-JVM support.

➤ If you are using multiple JVM versions and have run the JRE Instrumenter multiple times, be sure to include two paths in the **-Xbootclasspath** parameter exactly as indicated in the output from the Instrumenter. The first path is for the specific JVM and the second path for the default "boot" directory. For example: -Xbootclasspath/p:/path/to/javaprobe/classes/IBM/1.3.1:/path/to/javaprobe/ classes/boot

---

2 Specify the range of ports from which the probe can automatically select. The Java Probe communicates using the mini Web server. A separate port is assigned for probe communications for each JVM that a probe is monitoring. By default, the port number range is set to **35000 - 35100**. You must increase the port number range when the probe is working with more than 100 JVMs.

Note: If a firewall separates the probe from the other Diagnostics components, you must configure the firewall to allow communications using the ports in the range that you specify. For more information, see Appendix , "Configuring Diagnostics to Work in a Firewall Environment."

If you have configured the firewall to allow probe communications on a range of ports that is different than the default, make sure to update the port range values discussed in the following bullets accordingly.

---

    a Locate the **webserver.properties** file in the folder **<probe_install_dir>/ javaprobe/etc**.

    b Set the following properties to adjust the range of ports that are available for probe communications.

        ➤ The minimum port in the port number range is set using the following property:

          jetty.port=35000

        ➤ The maximum port in the port number range is set using the following property:

          jetty.max.port=35100

3 Assign a custom probe Identifier to the probe for each JVM, using the Java command line.

-Dprobe.id=<Unique_Probe_Name>

The probe ids defined on the Java command line override the probe names that are defined in the **probe.properties** file using the probe's **id** property.

The following is an example of a WebLogic startup script before adding the **probe.id** parameter:

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m -classpath "%CLASSPATH%"
-Dweblogic.Domain=petstore -Dweblogic.Name=petstoreServer -Dbea.home="C:\\bea"
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Dcloudscape.system.home=./samples/eval/cloudscape/data
-Djava.security.policy=="C:\bea\wlserver6.1/lib/weblogic.policy" weblogic.Server
```

The following is an example of a WebLogic startup script after adding the **probe.id** parameter:

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m"
-Xbootclasspath/
p:C:\MercuryDiagnostics\JAVAProbe\classes\Sun\1.4.1_03;C:\MercuryDiagnostics\JA
VAProbe\classes\boot"
-classpath "%CLASSPATH%"
-Dprobe.id=<Unique_Probe_Name> -Dweblogic.Domain=petstore
-Dweblogic.Name=petstoreServer
-Dbea.home="C:\\bea" -Dweblogic.management.password=%WLS_PW%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Dcloudscape.system.home=./samples/eval/cloudscape/data
-Djava.security.policy=="C:\bea\wlserver6.1/lib/weblogic.policy" weblogic.Server
```

Note: The command-line properties must be entered on one line, without any breaks.

---

4 Specify the points file that the probe will use. By default LoadRunner/ Performance Center will set the points.file.name to auto_detect.points. You may want to specify that a custom points file be used when you need to use more than one custom instrumentation plan, or where you have several JVMs on the same machine using a single probe instance, and one or more of the JVMs needs specific methods and classes included in a layer support custom instrumentation.

```
-Dprobe.points.file.name="<Custom_AutoDetect_Points_File>"
```

Configuring Separate Probe Installations For Each JVM

When there are multiple JVMs on a single host, you may install a separate probe for each JVM instance. To use a separate probe for each JVM, you must install the probe multiple times, and define an instance of each probe by setting the probe's **id** property in the **probe.properties** file in each probe's installation directory.

To define an instance of each installed probe:

1 If you have not already manually run the JRE Instrumenter for the JVMs that the probe will be working with, do so now. To run the JRE Instrumenter, see "Running the JRE Instrumenter" on page 141.

2 Locate the **probe.properties** file in the **<probe_install_dir>/javaprobe/etc** directory.

For example:

```
C:\\MercuryDiagnostics\JAVAProbe\etc\probe.properties
```

3 Assign a name to the **id** property that will be unique on the server and on the Diagnostics Server as follows:

```
id=<uniqueProbeName>
```

When the probe instance is started, a log file is created in the **<probe_install_dir>/javaprobe/log** directory where the log messages for the probe are stored.

Note: This configuration enables you to perform the diagnostics; but you will have to spend some time configuring each probe installation. Using a single probe for multiple JVMs can provide you with the same diagnostics detail, while allowing you to spend less time configuring the components.

Disabling the Java Diagnostics Profiler

You can disable the Diagnostics Profiler for Java on a Java Probe so that it cannot be accessed accidently. When the Java Diagnostics Profiler is disabled, the user interface cannot be accessed from the Java Diagnostics Profiler URL: http://<probe_host>:<probeport>/profiler.

To disable the Java Diagnostics Profiler, set the **disable.profiler** property in **<probe_install_dir>/etc/probe.properties** to true.

The default value for **disable.profiler** is **false**. To enable the Java Diagnostics Profiler once it has been disabled, change the value of the **disable.profiler** property from **true** to **false**.

Specifying Probe Properties as Java System Properties

All of the Java Probe properties, except for those defined in the **dynamic.properties** property file, can be specified as Java System properties on the startup command-line for the application server. This is very useful when there is more than one JVM using a single probe installation.

To specify a probe property as a Java System property, pre-pend the letter **D** and the first part of the properties file name to the property name. This is best explained using some examples:

➤ To set the **id** property in **probe.properties** from the startup command, you concatenate the **D** and **probe** from the property file name, and then tack on the name of the property that you are specifying, that is, **id**, as follows:

```
-Dprobe.id=SomeId
```

➤ To set the **active.products** property in **probe.properties** from the startup command, you concatenate the **D** and **probe** from the property file name, and then tack on the name of the property that you are specifying, that is, **active.products**, as follows:

```
-Dprobe.active.products=AD,AM
```

➤ To set the **registrar.url** property in **dispatcher.properties** from the startup command, you concatenate the **D** and **dispatcher** from the property file name, and then tack on the name of the property that you are specifying, that is, **registrar.url**, as follows:

```
-Ddispatcher.registrar.url=http://host01.company.com:2006/commander/registrar
```

Controlling Probe Logging

You can control the level of the messages that the probe logs and change the location where the log messages are posted using the probe properties.

Controlling the Log Message Level

The level of messages from the probe that are logged to the standard output is controlled by the **lowest_printing_level** property in the property file **<probe_install_dir>/etc/logging.properties**. The default setting for this property is **OFF** which prevents almost all probe messages from being logged to the console.

You can adjust the logging level dynamically by changing the value assigned to the **lowest_printing_level** property. The level of messages that are logged changes as soon as you save the property file.

The valid values for the **lowest_printing_level** property are:

Property Value	Description
OFF	No messages are logged.
DEBUG	All messages are logged.
INFO	Info, Severe, and Warning messages are logged.
WARN	Warning and Severe messages are logged.
SEVERE	Severe messages are logged.

Changing the Log Directory for a Probe

The default location for the log directory for a probe is **<probe_install_dir>/log.** When you have more than one probe on the same host, you can change the location of the log directory for each probe using the **log.dir** property. This property can be set in two ways:

➤ The value of the **log.dir** property can be set in the property file **<probe_install_dir>/etc/probe.property.**

➤ The value of the **log.dir** property can be specified on the startup command-line for the application server as a JAVA system property as shown in the following example:

```
-Dprobe.log.dir=/path/to/log
```

For more information on specifying the log.dir property on the startup command line, see "Configuring a Probe for a Proxy Server" on page 401.

Setting the Probe Host Machine Name

The probe's host name is used to register the probe with the Diagnostics Server in Commander mode. The Diagnostics Server in Commander mode uses the probe's host name to communicate with the probe and displays it along with the system metrics for the server that the probe is monitoring in the Diagnostics views.

The probe normally can detect the host name of the machine that is its host. In some situations, the server configuration is faulty and the probe cannot detect the correct host name. In situations where a firewall or NAT is in place or where your probe host machine has been configured as a multi-homed device, it may not be possible for the probe to properly detect its host.

If the probe cannot detect its host name, you can instruct the probe to get the host name via a reverse DNS lookup based on the socket connection or you can specify the host name using a probe property.

Instructing the Probe to Use Reverse DNS Lookup

If the configuration of the probe's host prevents the probe from detecting the host name, you can instruct the probe to detect the host name using a reverse-DNS lookup by setting the **server.host.name.lookup** property. This property is located in the **<probe_install_dir>/etc/dispatcher.properties** file.

The default value for the **server.host.name.lookup** property is 'false'. This tells the probe to do the lookup with out using reverse-DNS. Set this property to 'true' to instruct the probe to use reverse-DNS lookup.

Manually Specifying the Probe Host Name

The **registered_hostname** property allows you to manually set host machine name for the probe and stop the probe from doing the automatic lookup.

To set default host machine name for a probe, set the **registered_hostname** property, located in the Java Probe property file, **<probe_install_dir>/etc/dispatcher.properties**, to a machine name or IP address.

When you set the **registered_hostname** property the automatic lookup of the host name is disabled.

Note: Setting the **registered_hostname** property because of a NAT or firewall is only an issue for a test environment where you are using LoadRunner, Performance Center, or Diagnostics Standalone.

However, if you should set the **registered_hostname** in a production environment where you are using Business Availability Center or Diagnostics Standalone, the name that you specify will be shown as the host name in System Health.

Controlling Automatic Method Trimming on the Probe

The default configuration for the probe includes settings that control the trimming of methods. Trimming can be controlled based upon how long the method takes to execute, which is known as *latency*, and by the *stack depth* of the method call. The default configuration instructs the probe to trim both by latency and by depth.

You may want to reduce the level of trimming, or turn off trimming completely, for certain Diagnostics situations. You can control the trimming using the **minimum.method.latency** and **maximum.stack.depth** properties in **<probe_install_dir>/etc/capture.properties**.

Controlling Latency Trimming

Methods that complete with latency greater than or equal to the value of the **minimum.method.latency** property are captured, and those that complete with latency less than this limit are trimmed to avoid incurring the overhead for methods that are less likely to be of interest.

Note: There a are a couple of situations where a method latency is not trimmed when its latency is less than the trimming property.

➤ Methods that are the root for a call tree are not trimmed.

➤ Methods that threw an exception are not trimmed, unless the probe is facing a severe throttling situation due to an inability to send events to the Diagnostics Server in Mediator mode.

Because of threading and buffering behavior, partial information about a method that was trimmed may be saved in the Profiler buffers (Development mode) or transmitted to the Diagnostics Server (Production and Test mode). When the Diagnostics Server detects that it only received partial information for a method, it issues a warning message. You should ignore this message unless your run requires that the information for all methods be captured.

If the information for all methods must be captured, lower the value of the **minimum.method.latency** property or set it to zero.

Consider the following when setting the **minimum.method.latency** property:

➤ The lower the value of the **minimum.method.latency** property, the greater the chance that the performance of your application will be adversely impacted.

➤ Depending on your platform, and whether native timestamps are being used (**use.native.timestamps** = false), it may not be useful to specify this value in increments of less than 10 ms.

➤ For Business Availability Center, LoadRunner, and Performance Center capture, the throttle mechanism will automatically increase the effective minimum method latency value if the Diagnostics Server is unable to keep up with the number of events being sent.

Controlling Depth Trimming

Methods that are called at a stack depth less than or equal to the value of the **maximum.stack.depth** property will be captured. Those that are called at a stack depth greater than this limit will be trimmed to avoid incurring the overhead for methods that are less likely to be of interest.

For example:

➤ If **maximum.stack.depth** is 3

➤ /login.do calls a() calls b() calls c()

➤ Only /login.do, a, and b will be captured.

The following should be considered when setting the **maximum.stack.depth** property.

➤ Setting a low **maximum.stack.depth** can significantly reduce the overhead of capture.

➤ In AM mode, it is not useful to have a **maximum.stack.depth** configured higher than the Diagnostics Server's depth trim. The Diagnostics Server's depth trimming is set using **trimming.type**=depth in **<Diagnostics Server_install_dir>/etc/server.properties.** This property is disabled by default.

Controlling Probe Throttling

The default configuration for the Java Probe is set to minimize the performance impact of collecting Diagnostics information. In addition, the probe is able to scale back automatically on the amount of information that it captures when it detects that its processing has started to have a negative impact on the performance of your application. The process for scaling back the amount of information that the probe captures is called *throttling*.

About Throttling

The probe throttles the amount of Diagnostics information that it collects by skipping over method calls that occur relatively quickly. The probe determines exactly what "relatively quickly" means based upon the amount of data that it is collecting. The skipped method calls can have a duration that starts at the configured minimum latency trim value, which defaults to 51 milliseconds. As load increases, the minimum latency trim value is increased. When the load decreases, the trim value is reduced to the configured minimum value.

The probe properties that are used to control throttling are listed below. These properties are found in **<probe_install_dir>/etc/capture.properties**.

➤ **gentle.reserve.buffer.count**

The gentle reserve buffers are temporarily allocated for workload spikes, while the load throttle measures are deploying. By default, the **gentle.reserver.buffer.count** property is set to the same value as the **maximum.private.buffer.count** property.

➤ **hard.reserve.buffer.count**

The hard reserve buffers are allocated for workload spikes when it is determined that the gentle reserve buffering cannot keep up. By default, the **hard.reserve.buffer** count is set to the same value as the **maximum.private.buffer.count** property.

➤ **buffer.wait.time**

The **buffer.wait.time** property is used to control the length of time that the probe will wait for an event to be buffered. This property tells the probe what to do when all throttle attempts are exhausted and an event cannot be buffered:

➤ -1 = indicates that the processing should wait for as long as it takes

➤ 0 = indicates that the event should be dropped immediately

➤ **#** = indicates that the processing should wait for # milliseconds before dropping the event

Turning Off Throttling

When you are using the probe to monitor a production system, the preferred behavior is to allow the probe to throttle automatically. However, when diagnosing some performance issues, you may want to sacrifice the performance of your application so that you can be sure that every diagnostic event is captured.

To configure the probe so that it will not throttle, you must edit the following properties in the **<probe_install_dir>/etc/capture.properties** file:

➤ Set **gentle.reserve.buffer.count** to 0

➤ Set **hard.reserve.buffer.count** to 0

➤ Set **buffer.wait.time** to -1

Note: When you are turning off throttling, be sure to set all three of the properties as instructed.

Tuning Throttling

By reviewing the probe logs, you can determine when throttling is being started and stopped. If throttling is being engaged more than you would like you should check to make sure that the probe thread configuration is synchronized with the number of threads that the application server has been configured to allow. If the application server has more threads than the probe can handle, throttling is engaged sooner and may not be disengaged.

You can check the maximum number of threads that can generate events into the probe by checking the log messages after the probe is started. A message similar to the following message should be logged:

> 2006-02-01 12:37:30,203 INFO class com.mercury.opal.capture.util.BufferPool [main] BufferPool ready: buffer size=6600000 (max 66 threads), # events in throttle overflow=4000000

In this example the maximum number of threads is 66.

The maximum number of threads that can generate events can be influenced using the **maximum.private.buffer.count** property, in property file **<probe_install_dir>/etc/capture.properties.**

If the maximum number of threads indicated by the log message is less than the number of threads that your application server can allocate, you can increase the value of the property. Be careful when increasing this value because this causes the memory overhead of the probe to increase. If the memory overhead is of concern, the value of the **maximum.buffer.size** and **minimum.buffer.size** properties may need to be decreased by an amount proportionate to the increase in the maximum buffer count.

Configuring a Probe for a Proxy Server

Important! This section only applies if you are using the probe with a Diagnostics Server.

There are two properties that are used tell the probe the URL of the Diagnostics Server in Commander mode. The property that you set depends upon whether there is a proxy present or not.

➤ **registrar.url** in **dispatcher.properties**

The **registrar.url** property in **<probe_install_dir>\etc\dispatcher.properties** is set when you install the probe. When there is a direct connection between the probe and the URL of the Diagnostics Server in Commander mode, the value of this property is used by the probe.

➤ **registrar.url** in **webserver.properties**

In the presence of a proxy, you must set the **registrar.url** property in the **<probe_install_dir>\etc\webserver.properties** file to indicate the URL of the Diagnostics Server in Commander mode.

Configuring Reverse HTTP for a Probe in SaaS

When you install a Java Probe that is to be used in a HP Software as a Service (SaaS) environment you are encouraged to disable the reverse http (rhttp) communication between the probe and the Diagnostics Server in Mediator mode. This configuration prevents other customers in the SaaS environment from inadvertently getting the metrics from your applications.

The following instructions have been provided so that you can disable or enable rhttp after the probe has been installed.

Disabling Reverse HTTP

If you neglected to disable rhttp when the probe was installed you can disable it manually using the following instructions.

To disable rhttp, comment out the **dispatcher.objects** property value in **<probe_install_dir>/etc/modules.properties** as shown in the following example:

```
####################################################################
## Dispatcher Module properties
####################################################################
dispatcher.objects=\
  com.mercury.diagnostics.capture.correlation.CorrelationSink, \
  com.mercury.opal.capture.dispatcher.ac.AppCriticTarget, \
  com.mercury.diagnostics.capture.correlation.CacheTarget, \
  com.mercury.diagnostics.capture.onlinecache.JavaprobeCacheStructure, \
  com.mercury.diagnostics.common.onlinecache.ProbeTreeSelector, \
  com.mercury.diagnostics.common.net.InternalCommSecurityManager, \
  com.mercury.diagnostics.common.modules.HostNameResolver, \
  com.mercury.diagnostics.probe.enterprise.RegistrarCommunication, \
  com.mercury.diagnostics.probe.enterprise.ServerCommunication, \
  com.mercury.diagnostics.probe.enterprise.ServerUpdate, \
  com.mercury.diagnostics.probe.DataPullingTokenProvider, \
  com.mercury.diagnostics.probe.enterprise.MediatorManager
#  , \
#  com.mercury.diagnostics.probe.enterprise.ReverseClientProbeImpl
```

Enabling Reverse HTTP

If you inadvertently disabled rhttp for a probe, you can enable rhttp manually using the following instructions.

To enable rhttp, uncomment the **dispatcher.objects** property value in **<probe_install_dir>/etc/modules.properities** as shown in the following example:

```
####################################################################
## Dispatcher Module properties
####################################################################
dispatcher.objects=\
```

```
com.mercury.diagnostics.capture.correlation.CorrelationSink, \
com.mercury.opal.capture.dispatcher.ac.AppCriticTarget, \
com.mercury.diagnostics.capture.correlation.CacheTarget, \
com.mercury.diagnostics.capture.onlinecache.JavaprobeCacheStructure, \
com.mercury.diagnostics.common.onlinecache.ProbeTreeSelector, \
com.mercury.diagnostics.common.net.InternalCommSecurityManager, \
com.mercury.diagnostics.common.modules.HostNameResolver, \
com.mercury.diagnostics.probe.enterprise.RegistrarCommunication, \
com.mercury.diagnostics.probe.enterprise.ServerCommunication, \
com.mercury.diagnostics.probe.enterprise.ServerUpdate, \
com.mercury.diagnostics.probe.DataPullingTokenProvider, \
com.mercury.diagnostics.probe.enterprise.MediatorManager , \
com.mercury.diagnostics.probe.enterprise.ReverseClientProbeImpl
```

Time Synchronization for Probes Running on VMware

For probes running in a VMware guest, time must be synchronized between the VMware guest and the underlying VMware host. If time is not synchronized properly, the Diagnostics UI may display inaccurate metrics or no metrics at all from a probe running in a VMware guest.

Time should be synchronized according to the recommendations given in the VMware whitepaper on timekeeping (http://www.vmware.com/pdf/vmware_timekeeping.pdf) in the section "Synchronizing Hosts and Virtual Machines with Real Time." In summary, VMware Tools must be installed in each VMware guest operating system that hosts a Diagnostics probe and the time synchronization option in VMWare Tools should be turned on. Note that this option in VMware Tools will only work if the guest operating system time is initially set earlier than that of the VMware host. For instructions on how to install VMware Tools, refer to "Basic System Administration" document for VMware ESX Server. In addition, if any non-VMware time synchronization software (such as Network Time Protocol) is used, it should be run in the VMware ESX server service console.

Limiting Exception Tree Data

The probe collects exception information and uses it to build exception instance trees. Exception instance trees provide the data for the exception information that appears in the Diagnostics UI such as a stack trace.

By default, every exception that occurs in the monitored application is a candidate for the exception instance trees. Collecting all exception information is usually undesirable however, because exceptions that are not of interest overload the display as well as the data collection and transfer operations. You can therefore limit the exception types for which data is collected by the probe. For example, filtering application server-based errors such as **javax.naming.AuthenticationException** would allow the exception trees to contain more application-specific errors.

The exception tree data that is collected is controlled in two ways: limiting specific exception types or limiting the number of exception types.

Limit Specific Exception Types

You can control which specific exception types are excluded and included from collection by setting the **exception.types.to.exclude** and **exception.types.to.include** properties in the **<probe_install_dir>\etc\dispatcher.properties** file as follows:

➤ **exception.types.to.exclude**

Set this property to ignore exceptions of one or more specified types. All subtypes of each specified type are also ignored unless the subtype is specified by the **exception.types.to.include** property.

➤ **exception.types.to.include**

Set this property to specify which, if any, of the specified excluded exceptions (or their subtypes) are to be included. Subtypes of any exception type specified to be included are also included.

Both properties take lists of fully-qualified exception type names, separated by commas. Changes to the **dispatcher.properties** file take effect immediately; it is not necessary to restart the application.

Limit the Number of Exception Types

The exception tree data that is collected can also be limited by specifying the number of different types of exceptions by setting the **exception.instance.tree.count** property in **server.propertie**s. By default, this property is set to 4 which indicates that only the first 4 exceptions types encountered during the probe's data collection cycle are used in building the exception trees. You can raise or lower this setting as needed.

Examples

The following example causes exceptions of type ClassNotFoundException and all its subtypes to be ignored.

```
...
exception.types.to.exclude=javax.naming.AuthenticationException
```

The following example causes some subtypes of the java.lang.IOException class to be excluded, as indicated by the diagram that follows:

```
...
exception.types.to.exclude=java.io.IOException,java.io.InvalidClassException
exception.types.to.include=java.io.ObjectStreamException
```

The following diagram shows the excluded and included exception types on the java.io class hierarchy:
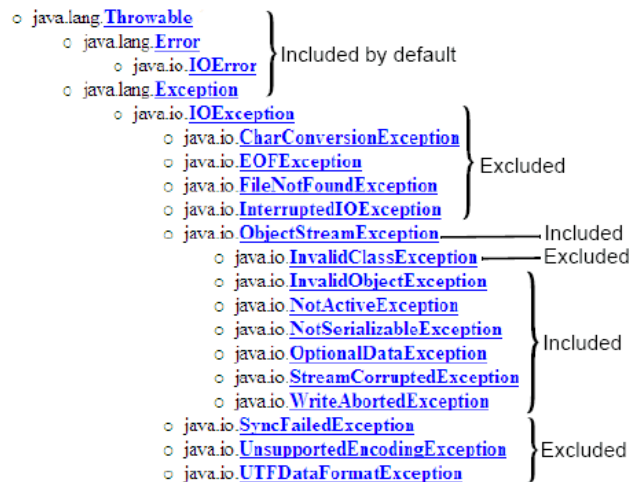
# Diagnostics Probe Administration Page

You can use the Diagnostics Probe Administration page to configure Java Probe and Profiler settings. You access the Diagnostics Probe administration page directly from your browser.

Accessing the Diagnostics Probe Administration Page

You open the Diagnostics Probe administration page inside your browser.

To access the Diagnostics Probe administration page:

1 In your browser, navigate to http://<probe_host>:<probeport>.

A probe is assigned to the first available port, beginning at **35000**.

The Administration page opens.
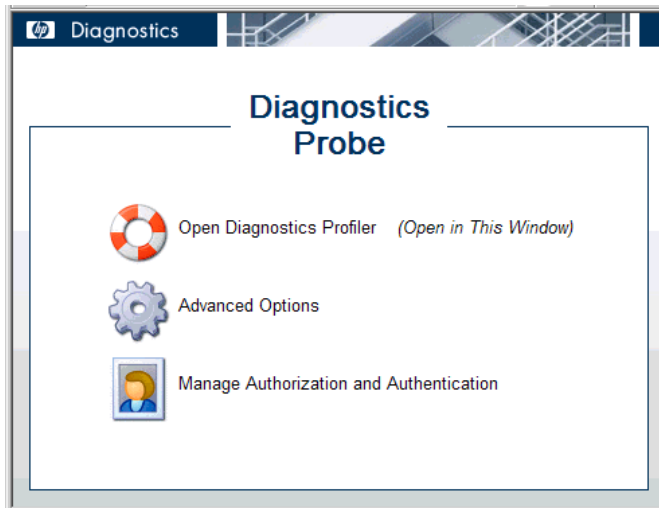
2 Select the menu option for the activity that you want to perform.

➤ **Open Diagnostics Profiler.** Opens the Java Diagnostics Profiler.

➤ **Advanced Options.** Opens the Components pages. For more information, see "Diagnostics Probe Components Page" on page 407.

➤ **Manage Authorization and Authentication.** Depending on how your probe is configured, you will access a different pages from this option

➤ If your probe is configured to work with a Diagnostics Server, the probe (Profiler) authorization and authentication settings are managed from the Diagnostics Server in Commander mode to which this probe is connected. In this case, when you click this option, you are redirected to that Diagnostics Server in Commander mode. For more information, see Appendix B, "User Authentication and Authorization."

➤ If your probe is configured to work as a Profiler only and is not connected to any Diagnostics Server, this option opens the User Administration page, where you can create, edit and delete users and change their privileges. For more information, see "Authentication and Authorization for Java Diagnostics Profilers in Standalone Mode" on page 409.

Diagnostics Probe Components Page

From the Components page you can open the Java Diagnostics Profiler, and access the User Administration page.

To access the Components page:

1 Open the Diagnostics Probe Administration page as described in "Accessing the Diagnostics Probe Administration Page" on page 406.

2 Click **Advanced Options**.

3 If prompted, enter your user name and password.

The Components page opens.

4 Click one of the listed options, as follows:

➤ **query.** For internal use by developers.

➤ **inst.** Includes various instrumentation options. For more information about probe instrumentation, see "Custom Instrumentation for Java Applications" on page 225.

➤ **security.** Depending on how your probe is configured, you access a different page from this option.

➤ If your probe is configured to work with a Diagnostics Server, the probe (Profiler) authorization and authentication settings are managed from the Diagnostics Server in Commander mode to which this probe is connected. In this case, when you click this option, you are redirected to that Diagnostics Server in Commander mode. For more information, see "User Authentication and Authorization" on page 593.

➤ If your probe is configured to work as a Profiler only and is not connected to any Diagnostics Server, this option opens the User Administration page, where you can create, edit and delete users and change their privileges. For more information, see "Authentication and Authorization for Java Diagnostics Profilers in Standalone Mode" on page 409.

➤ **scheduler.** Allows you to see and control regularly scheduled background tasks. For the ServerCommunication scheduler or the sharedInfrequentEventScheduler you can see the state and the number of tasks inside each. For each task, you can select an action such as RUN NOW or DELETE.

➤ **infrequentLogger.** See the current status of entries in the infrequent logging table.

➤ **files.** Installation directory browser - upload and download property files, log files, etc.

Authentication and Authorization for Java Diagnostics Profilers in Standalone Mode

When you install the Java Probe as a Profiler only (not connected to any Diagnostics Server), you manage the authentication and authorization of users of the Profiler from the Diagnostics Probe User Administration page.

Note: If the Java Probe is configured to work with a Diagnostics Server, the probe (Profiler) authorization and authentication settings are managed from the Diagnostics Server in Commander mode to which this probe is connected. For more information, see "User Authentication and Authorization" on page 593.

To manage authentication and authorization for users of the standalone Java Diagnostics Profiler:

1 Access the Diagnostics Probe administration page

In your browser, navigate to http://<probe_host>:<probeport>. A probe is assigned to the first available port, beginning at 35000.

The Diagnostics Probe administration page opens.

2 Select Manage Authorization and Authentication to open the User Administration page.
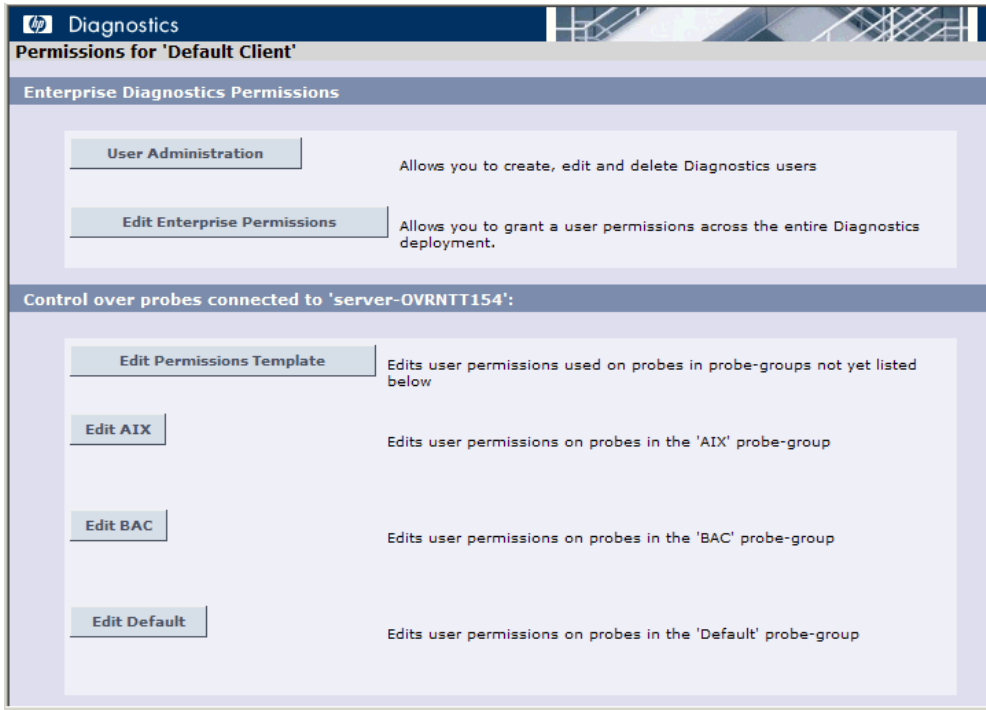
On the User Administration page, you can create new users, assign privileges to users, change passwords of existing users, and delete users.

To create a new user:

1 Click **Create User**, enter a user name in the **New User Name** box, and click **OK.** The new user appears in the list of user names.

2 In the row representing the new user, type a password in the **Password** box and confirm it by retyping it in the **Confirm Password** box.

3 Type the password of the user currently logged on, in the **Password for <current user>** box and click **Save Changes**.

To assign privileges to a user:

1 Go to the row representing the relevant user and select the appropriate check boxes representing the different privileges.

The following privilege levels can be assigned to Java Diagnostics Profiler users:

Privilege	Description
View	The user can view Profiler data from the UI.
Execute	The user can perform garbage collection and clear the performance data held by the Profiler.
Change	The user can run potentially risky operations, such as taking a heap-dump or changing instrumentation.

Note: The privilege levels, **rhttpout** and **system** are for internal purposes only.

Each privilege level stands alone. There is no inheritance of privileges from one level to the next. You must grant a user all of the privilege levels that are necessary to perform the functions that they need to perform.

2 Type the password of the user currently logged on, in the **Password for <current user>** box and click **Save Changes**.

To change the password of an existing user:

1 Go to the row representing the relevant user, type a password in the **Password** box, and confirm it by retyping it in the **Confirm Password** box.

2 Type the password of the user currently logged on, in the **Password for <current user>** box and click **Save Changes**.

To delete a user:

1 Type the password of the user currently logged on, in the **Password for <current user>** box.

2 Click the **Delete user** button corresponding to the user you want to delete.

A message box opens asking if you want to delete the selected user.

3 Click **OK** to delete the user.

Configuring Collection of CPU Time Metrics

The CPU Time metrics appear in the Details pane for various views such as the Transaction view, the Probes view, the Call Profile view and the Portal Components view. You can enable, disable and configure the collection of CPU time metrics. The CPU time metrics are **CPU (Avg)** and **CPU (Total)**. If collection of CPU time metrics is disabled or not configured for methods, you will see N/A for these metrics.

The CPU Time metrics rely on CPU timestamping which is generally supported on the following platforms.

➤ Windows

➤ Solaris 10+

➤ AIX 5.2+

➤ HP-UX 11i Itanium

➤ Linux 5.x and SUSE Linux 10.x

Note: Support for CPU timestamping can vary however, not only by operating system, but also by platform architecture (for example SPARC versus x86).
For the most recent information on support for CPU Time on specific platform versions and architecture refer to the Diagnostics Product Availability Matrix at
http://support.openview.hp.com/sc/support_matrices.jsp.

Important: In VMware, the CPU time metric is from the perspective of the guest operating system and is affected by the VMware virtual timer. See the VMware whitepaper on timekeeping at http://www.vmware.com/pdf/vmware_timekeeping.pdf and refer to "Time Synchronization for Probes Running on VMware" on page 403.

By default, collection of CPU time metrics is enabled for server requests. You can disable CPU time metric collection and configure collection of CPU time metrics in property files or using the Java Diagnostics Profiler UI.

For a Java Probe the collection of CPU Time metrics is controlled by two properties:

➤ **use.cpu.timestamps** property in **<probe_install_directory>\etc\capture.properties**.

This property is set to **true** by default which enables collection of CPU time metrics. But what, if any, CPU timestamps are collected is controlled by a second property listed below. If you set the use.cpu.timestamps property to false, the CPU time metrics will not be collected for any server request or method reported by the probe

➤ **cpu.timestamp.collection.method** property in **<probe_install_directory>\etc\dynamic.properties**.

Note: Use caution when configuring the collection of CPU timestamps because of the increase in Diagnostics overhead. The increased overhead is caused by an additional call for each method that is needed to collect the timestamp.

Cpu.timestamp.collection.method can be set to one of the following:

➤ **0 –** No CPU timestamping

➤ **1 –** CPU timestamps will be collected only for server requests.

➤ The default value is **1** which means that CPU times can be reported at the server request level but not at the transaction level. However, if the setting is removed or commented out of the properties file, the default is then 0.

➤ **2 –** CPU timestamps will be collected for All server requests and ALL methods.

➤ **3 -** CPU timestamps will be collected for ALL server requests and the lifecycle methods instrumented for Portal Components.

Another way to set the cpu.timestamp.collection.method property is using the Configuration tab in the Java Diagnostics Profiler as follows:

1 In the **Profiler** UI select the **Configuration** tab. The profiler does not need to be started to make this probe configuration change.

2 In the Configuration screen select a **Collect CPU Timestamps** option from the drop down list.

CPU Timestamp Collection Method	Description
None	No CPU Timestamps.
For Server Requests Only	CPU timestamps will only be collected for server requests.
For Server Requests and Portlet Methods	CPU timestamps will be collected for ALL server requests and the lifecycle methods instrumented for portal components.
For Server Requests and All Methods	CPU timestamps will be collected for ALL server requests and ALL methods.

3 When you have completed making changes to the Configuration tab, click **Apply Changes.**

Note: Your changes take effect immediately. There is no need to restart the Probe.

Configuring Consumer IDs

Web service metrics can be grouped by particular consumers of the Web service. The metrics are then aggregated for that consumer and displayed in SOA Services views such Services by Consumer ID or Operations by Consumer ID.

There are several ways of defining the consumer ID:

➤ a value that appears in the SOAP header

➤ a value that appears in the SOAP envelope or body

➤ a value that appears in an HTTP header

➤ a JMS queue name (or topic name) for SOAP over JMS web services

➤ a JMS message property or header for SOAP over JMS web services

➤ a specific IP address or a range of IP addresses

Important: Defining consumer ID based on SOAP header, envelope or body requires the Diagnostics SOAP message handler. For some application servers, special instrumentation is provided in Diagnostics to automatically load the Diagnostics SOAP message handler.

However, some manual configuration is required for WebSphere 5.1 JAX-RPC and Oracle 10g JAX-RPC, see "Configuring the SOAP Message Handler" on page 189 for details.

In addition, the Diagnostics SOAP message handler is not available for all application servers nor is custom instrumentation available to capture SOAP faults or consumer IDs from SOAP payloads, and therefore this feature is not available on all versions of all application servers. For the most recent information on Diagnostics SOAP message handler support refer to the Diagnostics Product Availability Matrix at http://support.openview.hp.com/sc/support_matrices.jsp.

Aggregating the data by consumer ID is useful if you want to determine who is using a particular service and how frequently they are using it. Consumer IDs are also useful for Business Availability Center. Business Availability Center users can look at the performance of the same application based on consumers to compare their performance characteristics.

Note: For environments in which Diagnostics is integrated with BAC, the consumer ID aggregation feature is supported for BAC version 7.5 or greater only.

Configuring Consumer IDs is optional. IP address is used as the default consumer ID for SOAP over HTTP/S web services and inbound queue name (or topic name) is used as the default consumer ID for SOAP over JMS web services.

Basic Procedure for Consumer ID Configuration

The basic procedure to configure consumer IDs is as follows:

1 For each Java Probe that you want metrics grouped by consumer, update the consumer.properties file as described in "Rules Syntax and Examples for Java Probes" on page 417.

2 If you plan on tracking more than 5 consumer types, update the max.tracked.ids.per.probe setting in the dispatcher.properties file.

3 Review the <probe_name>_id.properties file located in the probe/files/log directory. The <probe_name>_id.properties file may need to be completed deleted or modified to match the consumer.properties changes made in the previous step. The file goes hand in hand with the max.tracked.ids.per.probe (dispatcher.properties) setting, once the limit is reached per probe all other consumers will be classified as "Other".

About Consumer ID Rules

The assignment of consumer IDs is controlled by consumer ID rules in a configuration file, **consumer.properties**.

Each category of consumer IDs has its own rules: SOAP rules, HTTP header rules, JMS web service rules and IP rules. The rules are applied in an order no matter which order the rules are defined. The SOAP header rules are applied first, the HTTP headers rules are applied next, then the JMS rules are applied, and lastly the IP rules are applied.

Important: ALL configuration items in the rules are case sensitive.

All rule types do not need to be used. There may be SOAP rules, no HTTP rules and IP rules. If there is no match on any of these rules then the original IP address or queue name for JMS is used as the consumer ID.

The SOAP rules allow for the consumer ID to be obtained from an XML element in the SOAP header, SOAP envelope or body as well.

The rule specifies a regular expression that is used to match against the web service name being called by the consumer. If there is a match, then the probe will attempt to find the text element that is also specified in the rule. If the text element is not found in the SOAP header, then this rule is skipped and the probe will go on to the next rule that is defined.

The HTTP header rules allow for the consumer ID to be obtained from a header in the collection of HTTP headers in a HTTP request.

The JMS web service rules allow for the consumer ID to be JMS queue/topic name, and JMS Message properties or Message Header (JMSReplyTo only).

The IP rules allow for the consumer ID to be obtained from the mapping of IP addresses to a consumer ID. The rule is used to define an IP address, or a range of addresses, to be assigned to a consumer ID.

Rules Syntax and Examples for Java Probes

The assignment of consumer IDs is controlled by specifying rules in **consumer.properties**.

Important: ALL configuration items are case sensitive. So for example if you enter a <pattern-name> of TraderService, the Web service name must have a capital T and a capital S for the pattern to match.

A value in a SOAP Header

To assign a consumer ID based on a value in a SOAP header, use the following format:

<rule-name>**;HTTP_WS;**<pattern-name> **= soap-header;**<element-value>

<rule-name> is a String that identifies the rule. The name must be unique to the consumer.properties file.

<pattern-name> is a regular expression to match on the Web service name or you can use the exact Web service name.

<element-value> the element in the SOAP envelope whose value you want to use as the Consumer ID.

For example, the following rule matches on a Web service with service name TraderService and uses the CallerA element's value as the consumer IDs:

```
SoapRule1;HTTP_WS;TraderService = soap-header;CallerA
```

As long as the callers of the TraderService Web service have a value defined for CallerA, the metrics will be grouped by the different values for CallerA. Here is an excerpt from the soap header that would map to a consumer ID of "Customer2" for this caller of the TraderService:

```
SoapTest1;WS<env:Envelope  xmlns:env="http://schemas.xmlsoap.org/soap/
envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <env:Header>
    <CallerA>Customer2</CallerA>  <---- The consumer id returned would be
                                         "Customer2"
   </env:Header>
   <env:Body   env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <m:sell    xmlns:m="http://www.bea.com/examples/Trader">
         <string    xsi:type="xsd:string">sample string</string>
         <intVal    xsi:type="xsd:int">100</intVal>
      </m:sell>
   </env:Body>
</env:Envelope>
```

A value in a SOAP Envelope

To assign a consumer ID based on a value in a SOAP envelope, use the following format:

<rule-name>**;HTTP_WS;**<pattern-name> **= soap-envelope;**<element-value>

<rule-name> is a String that identifies the rule. The name must be unique to the consumer.properties file.

<pattern-name> is a regular expression to match on the Web service name or you can use the exact Web service name.

<element-value> the element in the SOAP envelope whose value you want to use as the Consumer ID.

A value in the SOAP Body

To assign a consumer ID based on a value in the SOAP body, use the following format:

> <rule-name>;**HTTP_WS;**<pattern-name> **= soap-body;**<element-value>

<rule-name> is a String that identifies the rule. The name must be unique to the consumer.properties file.

<pattern-name> is a regular expression to match in the Web service name or you can use the exact Web service name.

<element-value> the element in the SOAP body whose value you want to use as the Consumer ID.

A value in an HTTP header

To assign a consumer ID based on a value in an HTTP header, use the following format:

<rule-name>;**HTTP_WS;**<pattern-name> **= attribute;**<header-value>

<rule-name> is a String that identifies the rule. The name must be unique to the consumer.properties file.

<pattern-name> is a regular expression to match on, in the URI.

<header-value> is the HTTP header whose value you want to use as the Consumer ID.

For example, the following rule matches on a web service with a URI of "/webservice/.*" and uses the "User-Agent" header's value as the consumer ID:

```
WsRule1;HTTP_WS;/webservice/.* = attribute;User-Agent
```

As long as the callers of the Web service have a value defined for User-Agent, the metrics will be grouped by the different values for User-Agent. Here is an excerpt from the HTTP header that would map to a consumer ID in the heading:

```
 GET /service/call HTTP/1.1
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 2000)
Host: ovrntt1
Caller: ovrntt1
Connection: Keep-Alive
```

A JMS Queue Name

To assign a consumer ID based on the matching the JMS queue/topic name, use the following format:

<rule-name>;**JMS_WS**;<queue-name>=<consumerID-string>

<rule-name> is a String that identifies the rule. The name must be unique to the consumer.properties file.

<queue-name> is a regular expression to match on, in the JMS queue/topic name.

<consumerID-string> is a literal string to use as the Consumer ID.

For example, the following rule matches on a JMS queue name that starts with queue://sca_soapjms.***** and uses the string "myJMSConsumer" as the consumer ID:

```
JMSTest3;JMS_WS;queue\://sca_soapjms.*=myJMSConsumer
```

You must use a backslash "\:" to escape the ":" after queue or topic.

The priority used in matching is determined by the order specified in the consumer.properties file. JMS_WS queue matching takes priority over IP matching; JMS_WS property matching takes priority over JMS_WS Header matching; JMS_WS Header matching takes priority over JMS_WS queue name matching.

A JMS Message Property

To assign a consumer ID based on matching a JMS queue/topic name and use the value from the JMS message property as the consumer ID, use the following format:

<rule-name>;**JMS_WS**;<queue-name>=**jms-property**;<property-value>

<rule-name> is a String that identifies the rule. The name must be unique to the consumer.properties file.

<queue-name> is a regular expression to match on in the JMS queue/topic name.

<property-value> is the JMS property whose value you want to use as the Consumer ID.

For example, the following rule matches on a JMS queue name that starts with queue://MedRec.***** and uses the value from the JMSXDeliveryCount property as the consumer ID:

```
JMSTest1;JMS_WS;queue\://MedRec.*=jms-property;JMSXDeliveryCount
```

You must use a backslash "\:" to escape the ":" after queue or topic.

A JMS Message Header

To assign a consumer ID based on matching the JMS queue/topic name and JMS message header, use the following format:

<rule-name>;**JMS_WS**;<queue-name>=**jms-header**;<header-value>

<rule-name> is a String that identifies the rule. The name must be unique to the consumer.properties file.

<queue-name> is a regular expression to match in the JMS queue/topic name.

<header-value> must be JMSReplyTo.

For example, the following rule matches on a JMS queue name that starts with queue://MedRec.***** and uses the value from the JMSReplyTo header as the consumer ID:

```
JMSTest1;JMS_WS;queue\://MedRec.*=jms-header;JMSReplyTo
```

You must use a backslash "\:" to escape the ":" after queue or topic.

A specific IP address

To assign a consumer ID based on an IP Address, use one of the following formats:

<rule-name>; **IP;** <IP-address> = <consumerID-string>

For example, the following rule matches on IP address 123.456.567.8 and uses the name "CustomerA_IP" as the consumer ID:

```
IPRule1;IP;123.456.567.8 = CustomerA_IP
```

A range of IP addresses

To assign a consumer ID based on a range of IP addresses, use the following format:

<rule-name>; **IP**; <IP-address-wildcards> = <consumerID-string>
<rule-name>; **IP**; <IP-address-ranges> = <consumerID-string>

where <IP octet> = integer, *****, or integer range specified with -

For example, the following rule matches all IP addresses whose first octet is 15 and uses the name "mySuperCluster" as the consumer ID:

```
IPRule2;IP;15.*.*.* = mySuperCluster
```

The following rule matches all IP addresses whose first octet is 15 and whose second octet is between 200 and 300; it uses the name "Customer_IP" as the consumer ID:

```
IPRule3;IP;15.200-300.*.* = Customer_IP
```

Configuring SOAP Fault Payload Data

If a SOAP fault is detected, the SOAP payload is added to the SOAP fault data. In the Diagnostics UI, you can view the payload information as part of the instance tree. Both JAX-WS and JAX-RPC Web services are supported.

Important: Capturing SOAP payload requires the Diagnostics SOAP message handler. For some application servers, special instrumentation is provided in Diagnostics to automatically load the Diagnostics SOAP message handler.

However, some manual configuration is required for WebSphere 5.1 JAX-RPC and Oracle 10g JAX-RPC, see "Configuring the SOAP Message Handler" on page 189 for details.

In addition, the Diagnostics SOAP message handler is not available for all application servers nor is custom instrumentation available to capture SOAP faults or consumer IDs from SOAP payloads, and therefore this feature is not available on all versions of all application servers. For the most recent information on Diagnostics SOAP message handler support refer to the Diagnostics Product Availability Matrix at http://support.openview.hp.com/sc/support_matrices.jsp.

For a Java Probe, you define the limit for the payload size by modifying the **<probe_install_dir>\etc\dispatcher.properties** file. Payloads larger than the specified size are truncated.

For example, the following entry increases the SOAP payload length to 10000 from its default of 5000:

```
max.soap.payload.bytes = 10000
```

Set this property to 0 to disable this feature

Configuring REST Services

You can configure REST style Web services to show up as regular Web Services in the Diagnostics UI. See **<probe_install_dir>\etc\rest.properties**.

Currently only HTTP is supported (no JMS).

Grouping JMS Temporary Queue/Topics

For reporting in Diagnostics, SOAP over JMS temporary queues are grouped into a single node. Diagnostics matches the queue/topic name to a list of regular expressions to find the temporary queue/topic names. The ones that match are replaced with either queue:<probe-id>\TEMPORARY or topic:<probe-id>\TEMPORARY depending on the type.

The list of regular expressions used for this matching is in the **<probe_install_dir>/etc/capture.properties** file. You can customize the list of regular expressions under the property **grouped.temporary.jms.names.**

16

Understanding the .NET Probe Configuration File

You control the configuration of the .NET Probe by modifying the elements and attributes in the .NET Probe configuration file: **<probe_install_dir>/etc/probe_config.xml**.

This chapter includes:

➤ Understanding the .NET Probe Configuration File on page 428

Understanding the .NET Probe Configuration File

The following topics define the elements and attributes that make up the .NET Probe configuration file **<probe_install_dir>/etc/probe_config.xml**. This file is defined in the file **<probe_install_dir>/etc/probe_config.xsd.** Each element in defined by describing its purpose, attributes, and parent and children elements. Remember to check the **<probe_install_dir>/etc/ probe_config.xsd** file if you have any questions about the information presented here. For information on additional .NET probe configuration elements specific to TransactionVision see the *HP TransactionVision Deployment Guide*.

<appdomain> element

Purpose

Builds an appdomain inclusion list for processes that host multiple appdomains.

The value of none means that it applies to all.

Attributes

Attributes	Valid Values	Default	Description
enabled	true false	true	Permits the appdomain to be instrumented.
name	string	none	Name that the .NET AppDomain settings apply to

Elements

Number of Occurrences	zero or more
Parent Elements	process
Child Elements	bufferpool, credentials, diagnosticsserver, mediator, id, ipaddress, logging, lwmd, modes, points, profiler, sample, trim, webserver, symbols, filter, topology

Example

<appdomain enabled="true" name="MSPetShop"/>

<authentication> element

Purpose

List of authenticated user names and passwords.

Attributes

Attributes	Valid Values	Default	Description
username		admin	
password		admin	Passwords must be generated using the passgen utility in the **<probe_install_dir>\bin** directory

Elements

Number of Occurrences	zero to many
Parent Elements	profiler
Child Elements	none

Example

<profiler autheticate="true">
    <authentication username="Test" password="uU8X9zOtl6Twi7TkGAhQ="/>
</profiler>

<bufferpool> element

Purpose

Configures the bufferpool behavior.

Attributes

Attributes	Valid Values	Default	Description
size	number	65536	Size of each buffer
buffers	number	512	Number of buffers in pool
sleep	number	1000	Number of milliseconds between flush checks
expires	number	1000	Number of milliseconds before buffer expires

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	none

Example

<bufferpool size="65536" buffers="512" sleep="1000" expires="1000" />

<captureexceptions> element

Purpose

Enables and controls the stack trace capture for exceptions.

Attributes

Attributes	Valid Values	Default	Description
enabled	true false	true	Enables exception capture.
max_per_request	number	4	Maximum exceptions captured for one server request.
max_stack_size	number	0 (meaning no maximum)	Maximum size of the call stack for a captured exception

Elements

Number of Occurrences	1
Parent Elements	probeconfig
Child Elements	include, exclude

Example

<captureexceptions enabled="true" max_per_request="4">

<consumeridrules> element

Purpose

This is the root element for configuring consumer ID rules

Attributes

Attributes	Valid Values	Default	Description
enabled	true false	false	Enables consumer ID rule evaluation

Elements

Number of Occurrences	1
Parent Elements	probeconfig
Child Elements	httpheaderules, iprules, soaprules

Example

<consumeridrules enabled="false">

<cputime> element

Purpose

Controls the **cputime** setting property.

Attributes

Attributes	Valid Values	Default	Description
mode	none, serverrequest, method	serverrequest	

Elements

Number of Occurrences	1
Parent Elements	probeconfig, process, or appdomain
Child Elements	none

Example

<cputime mode="serverrequest"/>

<credentials> element

Purpose

Supplies credentials that are used to validate for communication with the
Diagnostics Server.

Attributes

Attributes	Valid Values	Default	Description
username		none	Used to validate with the Diagnostics Server
password		none	Used to validate with the Diagnostics Server
authenticate	true, false	true	Enables and disables authentication. This should be disabled for backward compatibility with releases prior to D4

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	none

Example

<credentials username="" password="" authenticate="true"/>

<depth> element

Purpose

Configures depth trimming.

Attributes

Attributes	Valid Values	Default	Description
enabled	true false	true	Enables depth trimming
depth	number	25	Sets the depth for depth trimming

Elements

Number of Occurrences	1
Parent Elements	trim
Child Elements	none

Example

```
<trim>
    <depth enabled="true" depth="25"/>
</trim>
```

\<diagnosticsserver\> element

Purpose

Contains connection and settings information related to the Diagnostics
Server which are used for enterprise mode.

Attributes

Attributes	Valid Values	Default	Description
url	Registrar URL. http:// \<host\>: \<port\>	none	URL to connect to registrar
delay	number	2	Number of seconds to wait before registering
keepalive	number	15	Number of seconds between keepalives
proxy	URL of proxy	none	Registrar connection proxy.
proxyuser	user id for proxy	none	
proxypassword	password for proxy	none	
registeredhost name	string	none	Name of host to register as (external name for firewall traversing)
register_byip	true, false	false	Register using ipaddress instead of hostname

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	none

Example

```
<diagnosticsserver url="http://localhost:2006/commander" delay="2"
keepalive="15" proxy="?" proxyuser="?" proxypassword="?"
registerhostname="?" register_byip="false"/>
```

\<exceptiontype\> element

Purpose

Define an exception type.

Attributes

Attributes	Valid Values	Default	Description
name	string	None	Class name of an exception.

Elements

Number of Occurrences	Zero to many
Parent Elements	include, exclude
Child Elements	None

Example

<exceptiontype name="System.DivideByZeroException"/>

<exclude> element (when parent is captureexceptions)

Purpose

Define a list of exceptions to exclude. (You may want to refer them to the section on this functionality for a more detailed explanation)

Attributes

None

Elements

Number of Occurrences	1
Parent Elements	captureexceptions
Child Elements	exceptiontype

Example

```
<exclude>
    <exceptiontype name="System.DivideByZeroException"/>
</exclude>
```

<exclude> element (when parent is lwmd)

Purpose

Define which collection classes to exclude.

Attributes

None

Elements

Number of Occurrences	Zero to many
Parent Elements	lwmd
Child Elements	None

Example

<exclude>System.Collections.HashTable</exclude>
<exclude>System.Collections.ArrayList</exclude>

<filter> element

Purpose

Filters out certain metrics that would skew the results or not be representative of the processing being monitored.

Attributes

Attributes	Valid Values	Default	Description
firstserverrequest	true, false	false	Enables/disables skipping the collection of metrics for the first time a particular server requests (URL) gets run after application startup.

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	none

Example

<filter firstserverrequest="false"/>

\<httpheaderrule\> element

Purpose

Defines a consumer ID rule for HTTP headers.

Attributes

Attributes	Valid Values	Default	Description
id	string	None	ID of the rule
rule	string	None	A regular expression that is used to match against the URL that the HTTP request is being sent to by the consumer
consumeridfield	string	None	Name of the header to use as the consumer ID

Elements

Number of Occurrences	Zero to many
Parent Elements	httpheaderrules
Child Elements	None

Example

\<httpheaderrule id="httpHeader 1" rule="/Webservice/.* "
consumeridfield="Caller"/\>

<httpheaderrules> element

Purpose

This element contains all of the <httpheaderrule> elements.

Attributes

None

Elements

Number of Occurrences	1
Parent Elements	consmeridrule
Child Elements	httpheaderule

Example

```
<httpheaderrules>
</httpheaderrules>
```

<id> element

Purpose

Provides probe id and probe group id.

Attributes

Attribute	Valid Values	Default	Description
probeid	Letters, digits, underscore, dash, period	$(AppDomain) .NET	Names the probe used by LoadRunner / Performance Center and System Health
probegroup		Default	Defines the grouping for probes used for reporting metrics.

Elements

Number of Occurrences	1 per parent
Parent Elements	probeconfig
Child Elements	none

Example

<id probeid="$(AppDomain).NET" probegroup="Default"/>

<include> element (when parent is captureexceptions)

Purpose

Define a list of exceptions to include.

Attributes

None

Elements

Number of Occurrences	1
Parent Elements	captureexceptions
Child Elements	exceptiontype

Example

```
<include>
    <exceptiontype name="System.DivideByZeroException"/>
</include>
```

<include> element (when parent is lwmd)

Purpose

Define which collections to include to the exclusion of others.

Attributes

None

Elements

Number of Occurrences	Zero to many
Parent Elements	lwmd
Child Elements	None

Example

<include>System.Collections.HashTable</include>
<include>System.Collections.ArrayList</include>

\<instrumentation> element

Purpose

Contains logging configuration for instrumenter.

Attributes

None.

Elements

Number of Occurrences	1 per parent
Parent Elements	probeconfig, process
Child Elements	logging

Example

```
<instrumentation>
    <logging level="off" threadids="true"/>
</instrumentation>
```

\<iprule\> element

Purpose

Defines a consumer ID rule for IP addresses.

Attributes

Attributes	Valid Values	Default	Description
id	string	None	Enables consumer ID rule evaluation
rule	string	None	Define an IP address, or a range of addresses, to be assigned to a consumer ID
consumerid	string	None	The consumer ID to use if there is a match on the rule.

Elements

Number of Occurrences	zero to many
Parent Elements	iprules
Child Elements	none

Example

\<iprule id="IpTest1" rule="43.*.1-20.*" consumerid="HP"/\>

<iprules> element

Purpose

This element contains all of the <iprule> elements.

Attributes

None

Elements

Number of Occurrences	1
Parent Elements	consumeridrules
Child Elements	iprule

Example

```
<iprules>
</iprules>
```

<latency> element

Purpose

Configures latency trimming.

Attributes

Attributes	Valid Values	Default	Description
enabled	true false	true	Enables latency trimming
throttle	true false	true	Enables latency trimming throttling
min	number	2	Minimum latency threshold.
max	number	100	Maximum latency threshold
increment	number	2	Threshold increment
increment threshold	number	75	The percentage of the buffer usage before the throttling should be incremented
decrement threshold	number	50	The percentage of the buffer usage before the throttling should be decremented

Example

```
<trim>
    <latency enabled="true" throttle="true" min="2" max="100" increment="2"
incrementthreshold="75" decrementthreshold="50"/>
</trim>
```

Elements

Number of Occurrences	1
Parent Elements	trim
Child Elements	none

<logdirmgr> element

Purpose

Contains the configuration for the log directory manager. The logdirmgr monitors the log directory to ensure that it does not grow unbounded. The logdirmgr scans the logs periodically as indicated by the scaninterval. If the size has exceeded the size indicated by maxdirsize the logdirmgr deletes the oldest files until the size no longer is greater than the maxdirsize.

Attributes

Attributes	Valid Values	Default	Description
enabled	true false	true	
maxdirsize	number	1024 MB	Largest size that you want to be the limit of the size of the log directory.
scaninterval	number	30m	How often the manager scans the logs to check for growth and size.

Elements

Number of Occurrences	1 per parent
Parent Elements	probeconfig
Child Elements	none

Example

<logdirmgr enabled="true" maxdirsize="1024 MB" scaninterval="30m"/>

<logging> element (when parent is instrumentation)

Purpose

Sets the logging level for the Probe instrumentation processing.

Attributes

Attributes	Valid Values	Default	Description
level	off assert break severe warning info debug points eh sig chi cil classmap ilasm symbol deepmode load all checksum property	"" which is equivalent to "info"	
threadids	true false	true	

Note: Valid Values below 'info' should typically not be used. These are Probe diagnostic settings which may produce extremely large log files.

Elements

Number of Occurrences	zero to many
Parent Elements	instrumentation
Child Elements	none

Example

<instrumentation>
    <logging level=""/ threadids="true">
</instrumentation>

<logging> element (when parent is appdomain, probeconfig, or process)

Purpose

Sets the logging level for the Probe processing for monitoring and reporting application performance.

Attributes

Attributes	Valid Values	Default	Description
level	off severe warning info debug events property webserver http symbols probemetrics registrar threadpool authentication| bufferpool rum bacforsoa vmware exceptions|	"" which is equivalent to "info"	

Note: Valid Values below 'info' should typically not be used. These are Probe diagnostic settings which may produce extremely large log files.

Elements

Number of Occurrences	
Parent Elements	appdomain, probeconfig, process
Child Elements	none

Example

<logging level=""/>

<lwmd> element

Purpose

Configures the light-weight memory diagnostics feature.

Attributes

Attributes	Valid Values	Default	Description
enabled	true false	false	Enables lwmd capturing
sample	string	1m	Sample interval (h-hour/m-minute/ s-second)
autobaseline	string	1h	Auto baseline interval
manualbase line	string	none	Manual baseline time
growth	number	15	Number of collections to growth track
size	number	15	Number of collections to size track

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	exclude, include

Example

<lwmd enabled="false" sample="1m" autobaseline="1h" manualbaseline= "?" growth="15" size="15"/>

<mediator> element

Purpose

Specifies the diagnostics server that is in the Mediator mode to which events are to be sent when in the enterprise mode.

Attributes

Attributes	Valid Values	Default	Description
host	host name	none	Name of mediator
port	number	2612	Mediator port
ssl	true/false	false	When the Diagnostics Server URL starts with http the default is false. When the Diagnostics URL starts with https the default is true.
metricport	number	2006	The port to which the Probe sends the Probe metrics such as heap usage and availability.
block	true/false	false	Block until mediator connection established
ipaddress			local ipaddress to use when connecting to the eventserver.

Attributes	Valid Values	Default	Description
localportstart	number	4000	Beginning of port range to use for tcp event channel connection to the Diagnostics Server in Mediator mode. Used only when ipaddress is specified.
localportend	number	5000	End of port range to use for tcp event channel connection to the Diagnostics Server in Mediator mode. Used only when ipaddress is specified.

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	none

Example

```
<mediator host="localhost" port="2612" ssl="false" metricport="2006"
block="false" ipaddress="16.255.18.99" localportstart="4000"
localportend="5000"/>
```

<modes> element

Purpose

Tells the probe which mode(s) to run in.

Attributes

Attributes	Valid Values	Default	Description
enterprise	true false	Depends on mode chosen in installation. ➤ true if pro is false ➤ false if pro is true	Probe in Enterprise mode (probe is working with Diagnostics Server)
ent	true false	➤ false	This is a short form of the enterprise attribute
ad	true false	➤ false	
am	true false	➤ false	
pro	true false	Depends on mode chosen in installation. ➤ true if enterprise is false ➤ false if enterprise is true	Probe in Profiler mode
tv	true false	false	Enables the capture of TransactionVision events. See the *HP TransactionVision Deployment Guide* for how to configure the required transport settings.
auto	auto	When pro is true, auto is set to auto.	

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	none

Example

<modes enterprise="false" ent="false" ad="false" am="false" pro="true"
auto="auto"/>

<points> element

Purpose

List a capture points file for inclusion in instrumentation.

Attributes

Attributes	Valid Values	Default	Description
file	string	none	Name of instrumentation capture points file

Elements

Number of Occurrences	zero or more
Parent Elements	appdomain, process
Child Elements	none

Example

<points file="ASP.NET.points"/>

<probeconfig> element

Purpose

Provides single containing root element for the probe configuration.

Attributes

None.

Elements

Number of Occurrences	1
Parent Elements	None
Child Elements	appdomain, bufferpool, captureexceptions, consumeridrules, credentials, diagnosticsserver, eventserverhost, id, instrumentation, ipaddress, logging, lwmd, modes, points, process, profiler, sample, soappayload, trim, webserver, topology, vmware

Example

```
<probeconfig>
</probeconfig>
```

<process> element

Purpose

Provide an inclusion filter list of which applies to the probe.

None means none.

Attributes

Attributes	Valid Values	Default	Description
enablealldomains	true false	false	When set to true the enable attribute on all appdomains that are part of the process is overriden so that all will be enabled.
name	string	none	Name of the .NET process that these setting apply to
monitorheap	string	false	Enables the Heap tab and Heap Breakdown processing when the value is set to true.

Elements

Number of Occurrences	zero or more
Parent Elements	probeconfig
Child Elements	appdomain, bufferpool, credentials, diagnosticsserver, mediator, id, instrumentation, ipaddress, logging, lwmd, modes, points, profiler, sample, trim, webserver, filter, symbols, topology

Example

<process enablealldomains="false" name="ASP.NET" monitorheap="false">

<profiler> element

Purpose

Contains settings for the Profiler feature of the probe.

Attributes

Attributes	Valid Values	Default	Description
authenticate	true, false	none	Enables/Disables authentication of incoming Profiler connection requests
register	true, false	false	Tells the probe to register even if it is in Profiler only mode
samples	number	60	Tells the Profiler how many samples to keep for lwmd/heap trending
best	number	1	The number of fastest instance trees to keeps
worst	number	3	The number of slowest instance trees to keep
inactivitytime-out	string	10m	The length of time that the Profiler continues to run after the user has stopped interacting with the Profiler.
disableremoteaccess	true, false	false	Disables remote access to the Profiler, thus not exposing the User/Password, and still be able to telnet/RemoteDestTop into the Probe's machine and run the Profiler locally.

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	authentication

Example

```
<profiler autheticate="true" register="false" samples="60" best="1" worst="3"
inactivitytimeout="10m">
    <authentication username="admin" password="admin"/>
</profiler>
```

<rum> element

Purpose

Controls the settings (enabled/header) for Real User Monitoring.

Attributes

Attributes	Valid Values	Default	Description
enable	true false	true	Enables or disables the RUM Integration feature.
responseheader	string	X-HP-CAM -COLOR	The name of the http header whose value contains the Diagnostics to RUM integration information.
encryptedkey	string		The encrypted key must be generated using the passgen utility in the <probe_install_dir>\bin directory

Elements

Number of Occurrences	1 per parent
Parent Elements	probeconfig
Child Elements	none

Element Example

<rum enabled="true" responseheader="X-HP-CAM-COLOR"
encryptedkey="OBF:3pe941vx43903wre40303xxz3q6r42ob43n93wre3io03xjs4
0h940pc3wir3q233jur3zir3yi03zir3vc03wre3xpi3r8o3olr44na3zor3v6m3vc03zir4
4u03ohb3rdi3xjs3wx03v6m3zor3yc63zor3jqz3q6r3wd740vi40b53xpi3ike3wx04
3gp42ur3q233y3r3zwy3wx0432i42293p9p"/>

To created the encrypted key, use the PassGen utility as follows:

cd <installdir>/bin
PassGen /system encryptionKey

Where **encryptionKey** is a string of alpha-numeric characters with a maximum length of 128 characters. The encryptedkey is shown on stdout.

passgen example:

PassGen /system TheLazyFoxJumpedHigh

Returns:

OBF:3q6r3xxz3y3r3xjs3wx03yc63n0r3lbr3vc03wd745893wre44u0413j3kn93zw
y40vi432i44fr3m453m894493439040pc40303kjd419r44na3wx0451h3wir3v6m3
lfr3mwj3yi03wre3xpi3xxz3y3r3q23

<sample> element

Purpose

Sets the sampling type and rate.

Attributes

Attributes	Valid Values	Default	Description
method	percent, count, period	percent	Sets the sampling method ➤ for percent rate must be 0-100 ➤ for count rate must be >1 ➤ for period rate must be one of standard Diagnostics time strings (3m for 3 minutes, 4s for 4 seconds, and so forth)
rate	number	0	Sets the sampling rate for percent type

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process, ws
Child Elements	none

Example

<sample method="percent" rate="0"/>

<soappayload> element

Purpose

Configures the BAC4SOA SOAP Payload capture on SOAP Faults feature.

The SOAP Payload capture on SOAP Faults feature provides the SOAP payload associated with a SOAP fault. Here the SOAP payload is defined as the entire SOAP Envelope.

Attributes

Attributes	Valid Values	Default	Description
enabled	true false	true	Enables or disables the SOAP Payload capture feature
size	number	5000	This is an optional attribute that specifies the maximum size in characters of any Payload capture. If not present the Default value is used. If present. If present and an error is made in the setting, the Default value is used.

Elements

Number of Occurrences	one per parent
Parent Elements	probeconfig
Child Elements	none

Example

<soappayload enabled="true" maxsize="5000" />

<soaprule> element

Purpose

Defines a consumer ID rule for SOAP headers.

Attributes

Attributes	Valid Values	Default	Description
id	string	None	ID of the rule
rule	string	None	A regular expression that is used to match against the web service name being called by the consumer.
consumeridfield	string	None	The element in the SOAP header to get the value for to use as the consumer ID.

Elements

Number of Occurrences	zero to many
Parent Elements	soaprules
Child Elements	none

Example

<soaprule id="SOAP1" rule="TestService2" consumeridfield="Caller"/>

<soaprules> element

Purpose

This element contains all of the <soaprule> elements.

Attributes

None.

Elements

Number of Occurrences	1
Parent Elements	consumeridrules
Child Elements	soaprules

Example

```
<soaprules>
</soaprules>
```

<symbols> element

Purpose

Limits the number of unique URIs and SQL strings that can been captured to control the amount of memory consumed.

Attributes

Attributes	Valid Values	Default	Description
maxuri	number	1000	Sets the top limit for number of unique URIs that can be captured.
maxuriname	string	Maximum number of unique URIs exceeded	
maxsql	number	1000	Sets the top limit for number of unique URIs that can be captured.
maxsqlname	string	Maximum number of unique SQLs exceeded	

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	none

Example

```
<symbols maxuri="1000" maxuriname="Maximum number of unique URIs
exceeded" maxsql="1000" maxsqlname="Maximum number of unique SQLs
exceeded"/>
```

<topology> element

Purpose

Controls the **topology** setting property.

Attributes

Attributes	Valid Values	Default	Description
enable	true false	true	Enables gathering topology information and passing it to the Diagnostics Server.

Elements

Number of Occurrences	1
Parent Elements	<probeconfig>, <process>, or <appdomain>
Child Elements	none

Example

<topology enable="true">

<trim> element

Purpose

Configures the trimming feature to reduce data volume transferred between the Probe and the Diagnostics Server.

The Profiler user interface ignores all configured trim settings, for example, depth trimming and latency trimming, as the Profiler does not require that any data be sent to the Diagnostics Server.

Attributes

None.

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	depth, latency

Example

<vmware> element

Purpose

Controls the ability to adjust timestamps to be more accurate when running in a VMware environment.

Attributes

Attributes	Valid Values	Default	Description
attempttime-stampadjust-ments	true false	false	Enables time stamp adjustments in VMware environments.

Elements

Number of Occurrences	1
Parent Elements	probeconfig
Child Elements	none

Example

<vmware attempttimestampadjustments="false"/>

<webserver> element

Purpose

Specifies the local Web server properties for communication with the probe.

Attributes

Attributes	Valid Values	Default	Description
start	number	35000	Starting port for webserver
end	number	35100	Ending port for webserver
ipaddress			local ip address to run webserver on.

Example

<webserver start="35000" end="35100" ipaddress="16.255.18.99"/>

<ws> element

Purpose

Controls Web services correlation sampling.

Attributes

None.

Elements

Number of Occurrences	1
Parent Elements	<xvm>
Child Elements	<sample>

Example

<ws>ws>

<xvm> element

Purpose

Controls the cross vm settings.

Attributes

None.

Elements

Number of Occurrences	1
Parent Elements	probeconfig, process, or appdomain
Child Elements	<ws>

Example

17

Advanced .NET Probe Configuration

Instructions are provided for advanced configuration of the .NET Probe. Advanced configuration is intended for experienced users with in-depth knowledge of this product. Use caution when modifying any of the Diagnostics components' properties.

This chapter includes:

Time Synchronization for Probes Running on VMware

.NET probes running in VMware hosts have additional time synchronization requirements. For probes running in a VMware guest, time must be synchronized between the VMware guest and the underlying VMware host. If time is not synchronized properly, the Diagnostics UI may display inaccurate metrics or no metrics at all from a probe running in a VMware guest.

Time should be synchronized according to the recommendations given in the VMware whitepaper on timekeeping (<ins>http://www.vmware.com/pdf/</ins> <ins>vmware_timekeeping.pdf</ins>) in the section "Synchronizing Hosts and Virtual Machines with Real Time." In summary, VMware Tools must be installed in each VMware guest operating system that hosts a Diagnostics probe and the time synchronization option in VMWare Tools should be turned on. Note that this option in VMware Tools will only work if the guest operating system time is initially set earlier than that of the VMware host. For instructions on how to install VMware Tools, refer to the "Basic System Administration" document for VMware ESX Server. In addition, if any non-VMware time synchronization software (such as Network Time Protocol) is used, it should be run in the VMware ESX server service console.

Customizing the Instrumentation for ASP.NET Applications

When the .NET Probe is installed, the **ASP.NET.points** file is created with the standard instrumentation that the Probe applies to all ASP.NET processing on the monitored server.

You must create application-specific instrumentation points to capture performance metrics for the business logic that has been implemented through application-specific classes and methods. The application-specific instrumentation points must be stored in a custom capture points file that can be associated with the application using the attributes in the **<probe_install_dir>/etc/probe_config.xml** file. If the application was auto-detected during the installation or during a rescan of IIS, a custom capture points file was automatically created for the application at the same time.

Note: If you do not know the classes and methods in an application that you may want to monitor, you can use the Reflector tool that was installed with the Probe to analyze the .dll files in the application and discover the classes and methods. See "Discovering the Classes and Methods in an Application" on page 485 for instructions on using Reflector.

To let the .NET Probe know that you want the instrumentation points in a custom capture points file to apply to an application, you must update the **points** attribute of the **appdomain** element in the **probe_config.xml** file.

To associate a custom capture points file with an application:

1 Create a capture points file with the instrumentation for the application specific classes. To create a capture points file, copy an existing capture points file in the **<probe_install_dir>/etc** directory.

Note: If the application was auto-detected during the installation or during a rescan of IIS, a capture points file already exists for the application with some or all of the points file entries commented out.

2 Customize the capture points file by adding instrumentation points so that the Probe captures custom business logic for the applications.

The following example illustrates how to modify the capture points file so that the Probe captures IBuySpy custom code:

```
[IBuySpy Callee]
class = !IBuySpy.*
method = !.*
signature =
scope =
ignoreScope =
layer = Custom.IBuySpy
```

For more information about instrumentation, see Chapter 10, "Custom Instrumentation for .NET Applications."

3 Update the configuration of the Probe in **probe_config.xml** to ensure that the modified capture points file is properly referenced.

Within the ASP.NET <**process**> tag add an <**appdomain**> tag for the application. Include the <**points**> tag with the **file** attribute and the **enabled** attribute.

```
<appdomain name="your app name">, enabled="true"
    <points file="your app.capture points"/>
</appdomain>
```

The example below illustrates this step. A custom capture points file has been created for the MSPetsShop application. The file has been named **MSPetShop.points**. The <**appdomain**> tag for the application, and the capture points file have been added to the ASP.NET <**process**> tag in the **probe_config.xml** file.

```
<?xml version="1.0" encoding="utf-8"?>

<probeconfig>
 <id probeid="" probegroup="Umatilla"/>

 <credentials username="" password=""/>
 <profiler authenticate=""><authentication username="" password=""/></profiler>

 <diagnosticsserver url="http://issaquah:2006"/>
 <mediator host="issaquah" port="2612"/>
 <webserver start="35000" end="35100"/>
 <modes am="true"/>

 <instrumentation><logging level="" threadids="no"/></instrumentation>

 <lwmd enabled="true" sample="1m" autobaseline="1h" growth="10" size="10"/>

 <process name="ASP.NET", enablealldomains="false">
     <logging level=""/>
     <points file="ASP.NET.points"/>
     <appdomain name="MSPetShop", enabled="true">
         <points file="MSPetShop.points"/>
     </appdomain>
 </process>
</probeconfig>
```

4 Restart IIS as instructed in "Restarting IIS" on page 218.

Discovering the Classes and Methods in an Application

If you want to monitor the performance of an application that you are not familiar with, you can use the Reflector automatic discovery tool that is installed with the .NET Probe to find the classes and methods in the application that you want to add to the Probe's instrumentation. The Reflector executable is located at **<probe_install_dir>\bin\reflector.exe**.

To discover classes and methods using Reflector:

1 Locate the installation directory for the application that you want to monitor.

2 Locate the folder in the application installation directory where the .dll files are stored.

3 Open a command prompt and change the directory to the folder where the .dll files for the application are stored.

4 Run the Reflector against all of the .dll files and .exe files in the current directory by executing the following the command at the command prompt:

```
<probe_install_dir>\bin\Reflector.exe
```

You can limit the Reflector to certain .dll and .exe files by adding additional parameters to the command. The following example shows another way to enter the command in the previous example:

```
<probe_install_dir>\bin\Reflector.exe *.dll *.exe
```

This command explicitly tells the Reflector to check all of the .dll and .exe files in the target directory.

To limit the Reflector to specific files, you could enter the following:

```
<probe_install_dir>\bin\Reflector.exe WorkHorse.dll Utility.dll
```

This command explicitly tells the Reflector to check only the two .dll files specified.

The following example shows the commands that you might execute if you have an application called PetShop that has .dll files located in a bin folder:

```
C:\>cd "c:\Program Files\Microsoft\PetShop\Web\bin"

C:\Program Files\Microsoft\PetShop\Web\bin>
C:\MercuryDiagnostics\".NET Probe"\bin\Reflector.exe
```

5 The Reflector displays a report of the assemblies, namespaces, classes, and methods found in the .dll files that you specified.

```
-----------------------------------------------------------------------------
Assemblies:
-----------------------------------------------------------------------------
C:\Program Files\Microsoft\PetShop\web\bin\PetShop.BLL.dll
C:\Program Files\Microsoft\PetShop\web\bin\PetShop.DAL.dll
C:\Program Files\Microsoft\PetShop\web\bin\PetShop.web.dll


-----------------------------------------------------------------------------
Namespaces:
-----------------------------------------------------------------------------
(8  classes)  PetShop.BLL
(6  classes)  PetShop.DALFactory
(17 classes)  PetShop.Web
(11 classes)  PetShop.Web.Controls
(2  classes)  PetShop.Web.ProcessFlow
(1  classes)  PetShop.web.WebComponents


-----------------------------------------------------------------------------
(8 classes)  Namespace: PetShop.BLL
-----------------------------------------------------------------------------|
PetShop.BLL.Account (10 Methods)
    Equals              System.Boolean(System.Object)
    Finalize            System.Void()
    GetAddress          PetShop.Model.AddressInfo(System.String)
    GetHashCode         System.Int32()
    GetType             System.Type()
    Insert              System.Void(PetShop.Model.AccountInfo)
    MemberwiseClone     System.Object()
    SignIn              PetShop.Model.AccountInfo(System.String,System.String)
    ToString            System.String()
    Update              System.Void(PetShop.Model.AccountInfo)

PetShop.BLL.Cart (17 Methods)
    Add                 System.Void(System.String)
    Equals              System.Boolean(System.Object)
    Finalize            System.Void()
    get_Count           System.Int32()
    get_Item            PetShop.Model.CartItemInfo(System.Int32)
    get_Total           System.Decimal()
    GetCartItems        System.Collections.ArrayList()
    GetEnumerator       System.Collections.IEnumerator()
    GetHashCode         System.Int32()
    GetInStock          System.Int32(System.String)
    GetOrderLineItems   System.Collections.ArrayList()
    GetType             System.Type()
    MemberwiseClone     System.Object()
```

Note: You can redirect the output from the Reflector to a file, as shown in the following example:

<probe_install_dir>\bin\Reflector.exe sys*.dll > <report_name>.txt

The output from Reflector is redirected to the file that you specify.

Use the information in the report to customize the instrumentation for the application, as described in "Customizing the Instrumentation for ASP.NET Applications" on page 482.

Configuring Latency Trimming and Throttling

When the .NET Probe determines that it is running out of resources because the Diagnostics Server is not keeping up with the amount of data that the Probe is capturing, it can automatically reduce the number of methods that it captures using a process called *latency trimming*. By default, latency trimming is enabled so that the Probe can adjust its work load as necessary.

When latency trimming is enabled, the .NET Probe trims the number of methods captured by ignoring methods with a total latency below a certain minimum latency threshold. The idea behind trimming is that it is better to miss capturing methods with lower latency that are less likely to be of interest than to allow the Probe to bog down or stop running. Trimming allows the Probe to continue to run so that it can capture the more interesting methods with higher latencies.

Note: Because of threading and buffering, partial information about a method that was trimmed may be transmitted to the Diagnostics Server. When the Diagnostics Server detects that it received only partial information for a method, it issues a warning message. You should ignore these warning messages unless you expected that the information for all methods was to be captured.

Notes:

➤ Latency trimming and throttling are ignored by the Profiler user interface.

➤ The Diagnostics Server may be configured to apply additional trimming of the Probes data which will affect the granularity of the Probe's data shown by the Diagnostics user interface.

Disabling Latency Trimming

By default, trimming is enabled for the .NET Probe. To disable trimming you must change the Probe configuration.

To disable Latency Trimming:

Add the **latency** tag to the **<probe_install_dir>/etc/probe_config.xml** configuration file, as shown in the following example:

```
<trim>
   <latency enabled="false" />
</trim>
```

The attribute of the latency element that turns on latency trimming is **enabled**. Latency trimming is enabled when **enabled** is set to true. When **enabled** attribute is set to false, latency trimming is disabled. The default value for this attribute is true.

For a description of attributes and elements of the **latency** element, see Chapter 16, "Understanding the .NET Probe Configuration File."

Enabling Latency Trimming

By default, trimming is enabled for the .NET Probe. If you have subsequently disabled trimming, you must change the Probe configuration to enable it once more.

To enable Latency Trimming:

Change the value of the enabled attribute of the **latency** element in the **<probe_install_dir>/etc/probe_config.xml** configuration file, as shown in the following example:

```
<trim>
   <latency enabled="true" />
</trim>
```

The attribute of the latency element that turns on latency trimming is **enabled**. Latency trimming is enabled when **enabled** is set to **true**. When **enabled** attribute is set to **false**, latency trimming is disabled. The default value for this attribute is **true**.

For a description of attributes and elements of the **latency** element, see Chapter 16, "Understanding the .NET Probe Configuration File."

Setting Latency Trimming Thresholds

By default, the latency trimming thresholds are set so that those methods with a latency less than 2 ms are trimmed, and those methods with a latency greater than 100 ms are never trimmed.

You can set the minimum trimming threshold by adjusting the value of the **min** attribute. You can set the maximum trimming threshold by adjusting the value of the **max** attribute. These attributes are specified in the **latency** element in the **<probe_install_dir>/etc/probe_config.xml** configuration file.

```
<trim>
   <latency enabled="true" min="50" max="100" />
</trim>
```

The attributes of the latency element that control the trimming thresholds are:

➤ **min**

Sets the minimum latency threshold. When latency trimming is enabled, methods with a latency less than or equal to the value of this attribute are trimmed. If you do not specify a value for this attribute, the default value of 2 ms is used.

The lower the value of the **min** attribute the greater the chance that the performance of the application will be adversely impacted. A lower value means that fewer methods are trimmed because more low-latency methods are captured.

If the information for all methods must be captured, disable latency trimming by setting **latency enabled** equal to false.

➤ **max**

Sets the maximum latency threshold. When latency trimming is enabled, methods with a latency greater than or equal to the value of this attribute are never to be trimmed. The default value for this attribute, if you do not specify a value, is 100ms.

For a description of the attributes and elements of the **latency** element, see Chapter 16, "Understanding the .NET Probe Configuration File."

Configuring Latency Trimming Throttling

Latency trimming is throttled by default. When throttling is enabled, the amount of trimming that the Probe does is automatically adjusted based on the percentage of the Probe resources that are being used up by the Diagnostics Server processing backlog.

Without throttling, the methods that fall below the minimum method latency threshold are always trimmed.

If the percentage resources used by the Probe increases above a set throttling increment threshold, the effective trimming threshold is incremented so that methods with higher latency are trimmed. If the percentage of Probe resources used increases above the threshold again, the effective trimming threshold is incremented once more so that methods with even higher latency are trimmed. If the percentage of Probe resources used drops below the throttling decrement threshold, the effective trimming threshold is decremented so that the methods with lower latencies are captured once more.

The effective trimming threshold cannot be incremented above the maximum method latency threshold, and it cannot be decremented below the minimum method latency threshold.

Below is an example of the **latency** element in the **probe_config.xml** configuration file that includes the throttling attributes:

```
<trim>
    <latency enabled="true" min="50" max="100"
    throttle="true" incrementthreshold="75"
    decrementthreshold="50" increment="2"/>
</trim>
```

The attributes of the **latency** element that control throttling are:

➤ **throttle**

Throttling is enabled when this attribute is set to true. When this attribute is set to false throttling is disabled. The default value for this attribute is true.

➤ **increment**

Sets the amount that the effective trimming threshold is incremented when the percentage of Probe resources used exceeds the **incrementthreshold**. Sets the amount that the effective trimming threshold is decremented when the **decrementthreshold** is crossed. The default value for this attribute is 2 ms.

➤ **incrementthreshold**

When the percentage of Probe resource usage rises to the value of this attribute or higher, throttling is triggered so that the effective trimming threshold is incremented. The default value for this attribute is **75** percent.

➤ **decrementthreshold**

When the percentage of Probe resource usage falls to the value of this attribute or lower, throttling is triggered so that the effective trimming threshold is decremented. The default value for this attribute is **50** percent.

For a description of the attributes and elements of the **latency** element, see Chapter 16, "Understanding the .NET Probe Configuration File".

Configuring Depth Trimming

The .NET Probe can automatically reduce the number of methods that it captures using a process called *depth trimming*. When the Diagnostics Server is not keeping up with the amount of data that the Probe is capturing, the Probe can use depth trimming to help prevent it from running out of resources. By default, depth trimming is enabled.

Note: Depth trimming is ignored by the Profiler user interface.

When depth trimming is enabled, the .NET Probe trims the number of methods captured by ignoring methods that are called at a stack depth that is greater than the maximum stack depth threshold. Those that are called at a stack depth less than or equal to the stack depth threshold are captured. The idea behind trimming is that it is better to miss capturing methods further down in the call stack, that are less likely to be of interest, so that the Probe is able to continue to run and is able to capture the more interesting methods that occur higher in the call stack.

For example, if the stack depth threshold is 3, and the following method calls are made:

```
/login.do calls a() calls b() calls c()
```

where only the /login.do, a, and b methods are captured, and method c is trimmed.

Below is an example of the **depth** element in the **probe_config.xml** configuration file that includes the trimming attributes:

```
<trim>
   <depth enabled="true" depth="10" />
</trim>
```

The attributes of the **depth** element that control trimming are:

➤ **enabled**

Depth trimming is enabled when this attribute is set to true. When this attribute is set to false depth trimming is disabled. The default value for this attribute is true.

➤ **depth**

Sets the threshold that are used for depth trimming. Methods that are called at or below the value of this attribute are trimmed when depth trimming has been enabled. The default value for this attribute is **25**.

Setting **depth** to a lower value can significantly reduce the overhead of capture. For a description of the attributes and elements of the **depth** element, see Chapter 16, "Understanding the .NET Probe Configuration File".

Configuring the .NET Probe for Light-Weight Memory Diagnostics

The .NET Probe has three different levels of Light-Weight Memory Diagnostics (LWMD):

➤ LWMD Off (Do not capture heap or collections metrics)

➤ Capture heap metrics only

➤ Capture heap metrics and collections metrics

When the .NET Probe is installed, the default configuration for the .NET Probe is to have Light-Weight Memory Diagnostics (LWMD) turned off. You can enable LWMD in the probe_config.xml (see "<lwmd> element" on page 457).

If you want to enable LWMD for collection metrics, you must change the Probe's configuration using the instructions below.

Note: Enabling the Probe to capture collections metrics may incur additional overhead on the host for an application.

To enable the capture of collection metrics:

➤ Add a **points** tag for the **Lwmd.points** file to either the **process** tag or to one or more **appdomain** tags in the **probe_config.xml** configuration file.

When you install the .NET Probe, the **Lwmd.points** file is installed in the **<probe_install_dir>/etc/** directory along with the **ASP.NET.points** and **ADO.points** files. The **Lwmd.points** file contains the instrumentation instructions needed to enable the capture of collection metrics.

If you add the points tag to the **process** tag the **probe_config.xml** configuration file should look similar to the following example:

```
<process name="ASP.NET", <enablealldomains="false">
    <points file="ASP.NET.points" />
    <points file="ADO.points" />
    <points file="Lwmd.points"/>
    <appdomain name="your app name", enabled="true">
        <points file="your app.capture points" />
    </appdomain>
</process>
```

If you add the points tag to the **appdomain** tags the **probe_config.xml** configuration file should look similar to the following example:

```
<process name="ASP.NET", <enablealldomains="false">
<points file="ASP.NET.points" />
<points file="ADO.points" />
<appdomain name="your app name", enabled="true">
    <points file="your app.capture points" />
    <points file="Lwmd.points"/>
</appdomain>
</process>
```

To disable LWMD:

➤ Delete the **points** tags for the **Lwmd.points** file from either the **process** tag or from the appropriate **appdomain** tags in the **probe_config.xml** configuration file.

Without the **points** tags in the configuration file, the Probe cannot locate the **Lwmd.points** file and so is missing the instruction that enables LWMD.

Limiting Exception Stack Trace Data

The probe collects exception data for exception throwing server requests and presents the information in the Diagnostics UI. The collected exception data can optionally include a stack trace.

Collecting stack trace data for all exceptions is usually undesirable however, because exception stack traces that are not of interest overload the display as well as the data collection and transfer operations. You can therefore limit the exception types for which stack trace data is collected by the probe. For example, filtering application server-based errors such as **System.Security.Authentication.AuthenticationException** would allow the stack traces to be used for more application-specific errors.

The stack trace data that is collected is controlled in three ways: limiting specific exception types, limiting the number of exceptions for which stack trace data is collected and limiting the size of the stack trace data.

Note: You can disable all stack trace collection by setting **captureexceptions enabled**="false" in the **probe_config.xml** file. By default, stack trace collection is enabled.

Limit Specific Exception Types

The exceptions for which stack trace data is collected is limited by setting the **exclude** and **include** properties in the **probe_config.xml** file as shown in the following example:

```
<exclude>
     <exceptiontype name="System.ArithmeticException"/>
</exclude>
<include>
     <exceptiontype name="System.DivideByZeroException"/>
</include>
```

Subtypes of any exception type specified to be excluded or included are also excluded or included, respectively, unless they are explicitly specified otherwise on the include or exclude list.

The following diagram shows which exception types are included and excluded based on the preceding example:
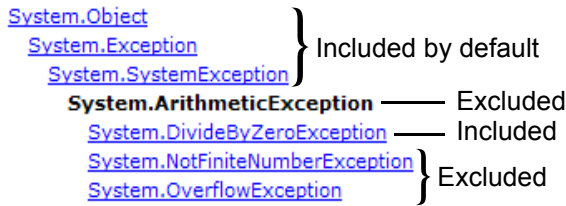
Changes to the **probe-config.xml** file take effect immediately; it is not necessary to restart the application.

Limit the Number of Exceptions per Server Request

By default, the .NET probe collects stack trace data on only the first 4 exceptions it encounters during a server request. If your application has more exceptions for which you want to view stack trace information, you can increase the value of the **max_per_request** property in the **probe_config.xml** file. As with all collected metrics, increased amounts of collected data place a higher load on the Diagnostics Server.

Limit the Size of the Stack Trace

By default, the captured stack trace data can be of any size. You may want to limit the size of the stack trace string to improve the readability of the Exceptions tab. Set the value of the **max_stack_size** property to the maximum stack trace string in the **probe_config.xml** file. As with all collected data, increased amounts of collected data place a higher load on the Diagnostics Server. By default, this property is set to 0 (zero) which means that the stack trace size is not limited.

Example

The following settings enable exception stack traces with a maximum stack trace string size of 2048.

```xml
<captureexceptions enabled="true" max_per_request="4" max_stack_size="2048">
  <exclude>
     <exceptiontype name="System.ArithmeticException"/>
  </exclude>
  <include>
     <exceptiontype name="System.DivideByZeroException"/>
  </include>
</captureexceptions>
```

Disabling Logging

You can disable Probe application logging by changing the **logging level** tag of the ASP.NET process section of the **probe_config.xml** file, as shown in the following example:

```xml
<process name="ASP.NET">
     <logging level="off"/>
</process>
```

You can disable Probe instrumentation logging by changing the **logging level** tag of the instrumentation section, as shown in the following example:

```xml
<instrumentation>
     <logging level="off" />
</instrumentation>
```

Overriding the Default Probe Host Machine Name

The **registered_hostname** property allows you to override the default host machine name that the Probe uses to register itself with the Diagnostics Server in Commander mode. In situations where a firewall or NAT is in place or where your Probe host machine has been configured as a multi-homed device, it may not be possible for the Diagnostics Server in Commander mode to communicate with the Probe unless you override the default host machine name.

To override the default host machine name for a Probe, set the **registered_hostname** attribute, located in the .NET Probe **<registrar>** tag of the **probe_config.xml file**, to an alternate machine name or IP address that allows the Diagnostics Server in Commander mode to communicate with the Probe. The **<registrar>** tag is optional; therefore you may have to enter the entire tag. Enter the port number, 2006, in the example below, as the port that the server is configured to listen on. Below is an example of the override:

```
<registrar url="http://foo:2006/registrar" registered_hostname="bar"/>
```

Notes:

➤ Setting the **registered_hostname** attribute because of a NAT or firewall is only an issue for a test environment where you are using LoadRunner, Performance Center, or Diagnostics Standalone.

➤ However, if you should set the registered_hostname in a production environment where you are using Business Availability Center or Diagnostics Standalone, the name that you specify is shown as the host name in System Health.

Listing the Probes Installed on a Host

When more than one probe has been installed on a single host, you cannot know which port each probe is using since the port that is assigned is based on the one that is available at the time the probe is started. As the applications are started and stopped, the port that is assigned to the probe for a given application is likely to change.

You can determine which probes are installed on a host and the ports that they are using by accessing the following URL:

```
http://<probe_host>:<port>
```

For the port value, enter the port number 35000 or 35001. It does not matter which one you enter.

The list of probes and ports is displayed as shown in the following example:
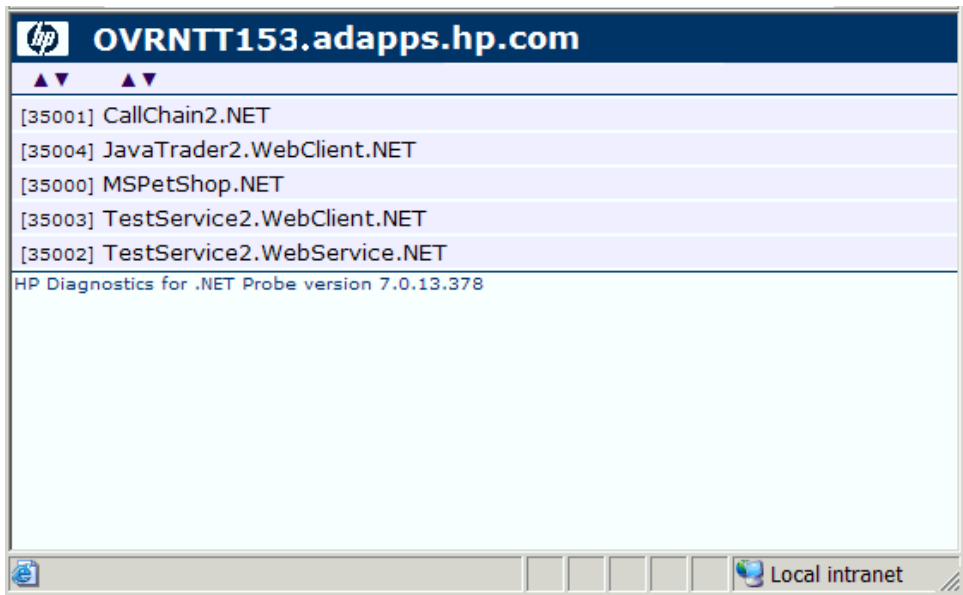
Authentication and Authorization for .NET Profilers in Standalone Mode

When you install the .NET Probe as a Profiler only (not connected to any Diagnostics Server), you manage the authentication and authorization of users of the Profiler in the **<probe_install_dir>/etc/probe_config.xml** file.

Note: If the .NET Probe is configured to work with a Diagnostics Server, the probe (Profiler) authorization and authentication settings are managed from the Diagnostics Server in Commander mode to which this probe is connected. For more information, see "User Authentication and Authorization" on page 593.

When you access the probe from the Diagnostics Server, the default username is admin and the default password is admin.

If the .NET Probe is installed as a profiler only, then by default, users are not required to enter a username and password to access the profiler.

However, you can configure the profiler to require user authentication. If you configure the profiler to require user authentication, you can define new usernames and passwords for accessing the profiler.

To configure the profiler to require user authentication:

➤ Go to the **<probe_install_dir>/etc/probe_config.xml** file and set the value of **profiler authenticate** to **true**.

```
<profiler authenticate="true">
    <authentication username="Test" password="uU8X9zOtl6Twi7TkGAhQ="/>
</profiler>>
```

If you do not set a username and password, the default username is admin and the default password is admin.

To create new usernames and passwords for users of the standalone .NET Diagnostics Profiler:

1 Generate a new username and password using the **PassGen.exe** utility located in the **<probe_install_dir>/bin** directory.

2 In the **probe_install_dir>/etc/probe_config.xml** file, after the **<profiler authenticate="true">** line, enter a username and password for each new user, in the following format:

```
<profiler authenticate="true">
     <authentication username="" password=""/>
</profiler>
```

➤ For **authentication username**, enter the username that you chose.

➤ for **password**, enter the encoded string that was returned by the **PassGen.exe** utility.

Important: If you defined new usernames and passwords to access the profiler, you can no longer use the default username and password (admin, admin). Rather, you have to use one of the new usernames that you defined.

Configuring Consumer IDs

Web service metrics can be grouped by particular consumers of the Web service. The metrics are then aggregated for that consumer and displayed as such in the Services by Consumer ID and Operations by Consumer ID views.

Aggregating the data by consumer ID is useful if you want to determine who is using a particular service and how frequently they are using it. Consumer IDs are also useful for BAC. BAC users can look at the performance of the same application based on consumers to compare their performance characteristics.

Note: For environments in which Diagnostics is integrated with BAC, the consumer ID aggregation feature is supported for BAC version 7.5 or greater only.

Configuring Consumer IDs is optional. By default, the Consumer ID of a Web service being monitored is reported as the IP address of the consumer of the Web service.

There are three ways of defining the consumer ID:

➤ a value that appears in the SOAP header

➤ a value that appears in an HTTP header

➤ to a specific IP address or a range of IP addresses

Basic Procedure for Consumer ID Configuration

The basic procedure to configure consumer IDs is as follows:

1 For each .NET probe for which you want metrics grouped by consumer, update the probe_config.xml file as described in "Rules Syntax and Examples for .NET Probes" on page 505.

2 If you are configuring more than 5 consumer types, update the **max.tracked.ids.per.probe** setting in the **server.properties** file.

About Consumer ID Rules

The assignment of consumer IDs is controlled by consumer ID rules in the **probe_config.xml** file.

Each category of consumer IDs has its own rules: SOAP rules, HTTP header rules, and IP rules. The rules are applied in an order no matter which order the rules are defined. The SOAP header rules are applied first, the HTTP headers rules are applied next, and lastly the IP rules are applied.

All rule types do not need to be used. There may be SOAP rules, no HTTP rules and IP rules. If there is no match on any of these rules then the original IP address is used as the consumer ID.

The SOAP rules allow for the consumer ID to be obtained from an XML element in the SOAP header.

The HTTP header rules allow for the consumer ID to be obtained from a header in the collection of HTTP headers in a HTTP request.

The IP rules allow for the consumer ID to be obtained from the mapping of IP addresses to a consumer ID. The rule is used to define an IP address, or a range of addresses, to be assigned to a consumer ID.

Rules Syntax and Examples for .NET Probes

The rules syntax and examples are specific to how the consumer ID is being defined.

SOAP Header Rules

The SOAP header rules allow for the consumer ID to be obtained from an XML element in the SOAP header. A rule and consumeridfield attribute must both be defined for a SOAP rule element, and an id attribute can also be defined for the user to identify individual rules.

The id can be any name the user would like to use to identify the rule; this attribute is not used by the .NET probe.

The rule is a regular expression that is used to match against the web service name being called by the consumer. If there is a match, then the .NET probe attempts to find a text element for the element defined in the consumeridfield. The element in the consumer ID may be a qualified name, i.e. are composed of a namespace name and the local part, or an unqualified name, which does not have an associated namespace.

If the element is not found in the SOAP header, then this rule is skipped and the probe goes on to the next rule that is defined.

Example:

```
<consumeridrules enabled="true">
  <soaprules>
      <soaprule id="SOAP1" rule="TestService.*" consumeridfield="Caller"/>
  </soaprules>
</consumeridrules>
```

There are some restrictions about how the SOAP header can be structured in XML for the element defined in consumeridfield to be found properly. The element defined in consumeridfield must be an element under <soap:header> or it can be nested one level. The following are examples:

```
<soap:header>
   <namespace:Caller>consumer ID</Caller>
</soap:header>
```

Example SOAP header rule (nested one level):

```
<soap:header>
  <namespace:header1>
     <namespace:Caller>consumer ID</Caller>
  </namespace:header1>
</soap:header>
```

There are some special restrictions about whitespace. In XML whitespace is defined as spaces, carriage returns, line feeds, and tabs. If there is any whitespace in the second example then the user will not be able to use qualified names.

HTTP Header Rules

The HTTP header rules allow for the consumer ID to be obtained from a header in the collection of HTTP headers in a HTTP request. A rule and consumeridfield attribute must both be defined for a HTTP rule element, and an id attribute can also be defined for the user to identify individual rules.

The rule is a regular expression that is used to match against the URL that the HTTP request is being sent to by the consumer. If there is a match, then the .NET probe attempts to find an HTTP header for the header name defined in the consumeridfield. If the header name is not found in the collection of HTTP headers, then this rule is skipped and the probe goes on to the next rule that is defined.

Example httpheader rules:

```
<consumeridrules enabled="true">
  <httpheaderrules>
      <httpheaderrule id="httpHeader 1" rule="/Webservice/.*
consumeridfield="Caller"/>
  </httpheaderrules>
</consumeridrules>
```

IP Address Rules

The IP rules allow for the consumer ID to be obtained from the mapping of IP addresses to a consumer ID. A rule and consumerid attribute must both be defined for an IP rule element, and an id attribute can also be defined for the user to identify individual rules.

The rule is used to define an IP address, or a range of addresses, to be assigned to a consumer ID. This rule may be defined as a single IP address, e.g. 19.225.17.125. The rule may also define a range, e.g. 19.255.17.125,19.255.17.255.

An asterisk may also be used in an octet of an IP address to match against anything in that octet, e.g. 19.255.17.*. A range may be defined in an octet to match a range of values in that octet, e.g. 19.255.17.20-255. Combinations of these may also be used, e.g. 19.*.17.20-255, 20.*.10-55.*. If there is a match, then the .NET probe sets the consumer ID to the consumer ID defined in the rule.

Examples:

```
<consumeridrules enabled="true">
  <iprules>
      <iprule id="IpTest1" rule="18.*.1-20.*" consumerid="Client1"/>
      <iprule id="IpTest2" rule="17.*.*.*" consumerid="Client2"/>
      <iprule id="IpTest3" rule="19.255.17.125,19.255.17.255" consumerid="Client3"/>
  </iprules>
</consumeridrules>
```

Configuring SOAP Fault Data

If a SOAP fault is detected, the SOAP payload is added to the SOAP fault data. In the Diagnostics UI, you can view the payload information as part of the instance tree.

For a .NET probe, you define the limit for the payload size by modifying the **<probe_install_dir>\etc\probe_config.xml** file. For example, the following entry increases the SOAP payload length to 10000 from its default of 5000:

```
<soappayload enabled="true" maxsize="10000"/>
```

Set enabled to "false" to disable this feature.

Part VI

Configuring Communications through Proxies and Firewalls

18

Configuring Diagnostics Server, Java Agent and .NET Agent for HTTP Proxy

Configuration steps are provided for you to enable HTTP proxy communications between the HP Diagnostics components. These configuration instructions are intended for experienced users with in-depth knowledge of HP Diagnostics. Use caution when modifying any configuration settings for the Diagnostics components.

This chapter includes:

➤ Enabling HTTP Proxy Communications for the Diagnostics Servers on page 512

➤ Enabling HTTP Proxy Communications for the Java Probe on page 513

➤ Enabling HTTP Proxy Communications for a .NET Probe on page 513

Enabling HTTP Proxy Communications for the Diagnostics Servers

The following section describes how to configure the Diagnostics Servers in Commander mode and Diagnostics Servers in Mediator mode to communicate with each other through an HTTP proxy.

To configure the Diagnostics Servers for HTTP proxy communications:

1 Set the following properties in **<diagnostics_server_install_dir>/etc/ server.properties**:

➤ Set **proxy.host** to the host name of the proxy server.

➤ Set **proxy.port** to the port of the proxy server.

➤ Set **proxy.protocol** to the protocol to use for the proxy server (http).

➤ Set **proxy.user** to the user used to authenticate the proxy server.

➤ Set **proxy.password** to the password used to authenticate the proxy server.

➤ For a Diagnostics Server in Commander mode that is to run over the proxy, set **commander.url** so that the host name is the real host name and not a localhost.

2 Restart the Diagnostics Server. For instructions, see "Starting and Stopping the Diagnostics Server" on page 54.

Enabling HTTP Proxy Communications for the Java Probe

The following section describes how to configure the Java Probe to communicate with Diagnostics Server in Commander mode through an HTTP proxy:

To configure the Java Probe for HTTP proxy communications:

1 Set the following properties in **<probe_install_dir>/etc/ dispatcher.properties**:

➤ Set **proxy.host** to the host name of the proxy server.

➤ Set **proxy.port** to the port of the proxy server.

➤ Set **proxy.protocol** to the protocol to use for the proxy server (http).

➤ Set **proxy.user** to the user used to authenticate the proxy server.

➤ Set **proxy.password** to the password used to authenticate the proxy server.

2 Restart the instrumented application VM.

Enabling HTTP Proxy Communications for a .NET Probe

The following section describes how to configure the .NET Probe to communicate with the Diagnostics Server in Commander mode through an HTTP proxy:

To configure the .NET Probe for HTTP proxy communications:

1 Set the following **proxy** properties in the **<probe_install_dir>/etc/ probe_config.xml** file to point to the Diagnostics Server host:

➤ Set **uri** to the host for the Diagnostics Server.

➤ Set **proxy** to the proxy url.

➤ Set **proxy.user** to the user used to authenticate the proxy server.

➤ Set **proxy.password** to the password used to authenticate the proxy server.

The following example shows how this would look in the **probe_config.xml** file:

```
<diagnosticsserver url="http://<diagserver_host_name>:2006/registrar/"
proxy="http://proxy:8080" proxyuser= "<username>" proxypassword="<password>"/>
```

2 Set the following **proxy** properties in the **<probe_install_dir>/etc/ metrcis.config** file to configure system metrics to use a proxy:

➤ Set **proxy.uri** to the proxy url

➤ Set **proxy.user** to the user used to authenticate the proxy server.

➤ Set **proxy.password** to the password used to authenticate the proxy server.

3 Restart the instrumented application process.

19

Configuring Diagnostics to Work in a Firewall Environment

Some basic configuration information is provided for you to enable HP Diagnostics to work correctly in an environment where a firewall is present. This additional configuration is required when the firewall separates the probes from the other Diagnostics components or the components of LoadRunner, Performance Center, or Business Availability Center. See the LoadRunner, Performance Center and Business Availability Center documentation for more details.

This chapter includes:

➤ Overview of Configuring Diagnostics for a Firewall on page 516

➤ Collating Offline Analysis Files over a Firewall on page 519

➤ Installing and Configuring the MI Listener on page 520

➤ Configuring the Diagnostics Server in Mediator mode to Work with a Firewall on page 521

➤ Configuring LoadRunner and Performance Center to Work with Diagnostics Firewalls on page 527

Note: The configuration instructions should be used only by experienced users with in-depth knowledge of HP Diagnostics. Use caution when modifying any configuration settings for the Diagnostics components.

Overview of Configuring Diagnostics for a Firewall

Configuring Diagnostics for a firewall differs depending on which HP Software product is part of the Diagnostics integration.

Business Availability Center

The diagram below shows a typical Diagnostics topology where a firewall separates the probe from the other Diagnostics and Business Availability Center components.
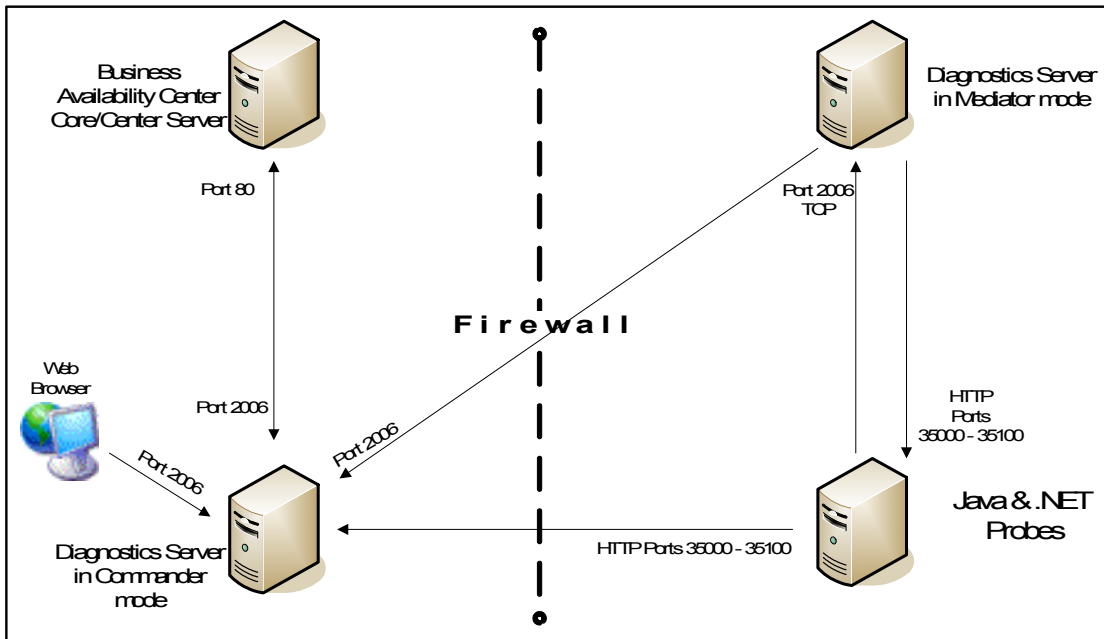
To configure the firewall to enable the communications between the Diagnostics components, open the ports that will allow:

➤ HTTP requests from the Business Availability Center Core/Center server(s) to the Diagnostics Server in Commander mode, on port 2006.

➤ HTTP requests from the Diagnostics Server in Commander mode to Business Availability Center core server on port 80.

➤ HTTP requests from the Diagnostics Server in Mediator mode to ports 35000-35100 of the probe.

➤ HTTP requests from the Web Browser Client machine to the Diagnostics Server in Commander mode on port 2006.

➤ HTTP requests from the Diagnostics Server in Mediator mode to the Diagnostics Server in Commander mode on port 2006.

➤ TCP requests from the probe to the Diagnostics Server in Mediator mode on port 2612.

➤ HTTP requests from the probe to the Diagnostics Server in Mediator mode on port 2006.

➤ HTTP requests from the Diagnostics Server in Commander mode to the probes on ports 35000-35100. The actual ports on which you must allow communications will depend upon the port numbers that you enabled when you configured the probe and the number of instrumented VMs. For information on setting the probe port range, see "Configuring the Probes for Multiple Application Server JVM Instances" on page 387.

LoadRunner and Performance Center

The diagram below shows a typical Diagnostics topology where a firewall separates the probe from the other Diagnostics and LoadRunner components.
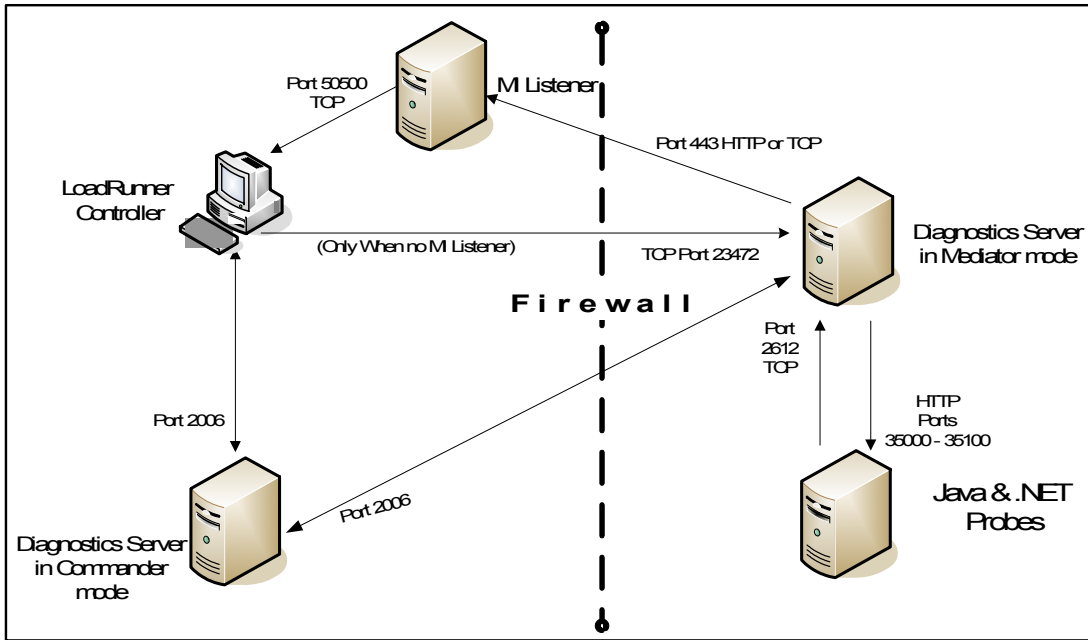
Note: LoadRunner is used in this diagram for illustrative purposes. The same information would apply to Performance Center.

You must configure the firewall to allow the Diagnostics components to communicate with each other.

To configure the firewall to enable the communications between the Diagnostics components, open the ports that will allow:

➤ HTTP requests from the Diagnostics Server in Mediator mode to the Diagnostics Server in Commander mode on ports 2612 and 2006.

➤ TCP requests from the probe to the Diagnostics Server in Mediator mode on port 2612.

➤ HTTP requests from the probe to the Diagnostics Server in Mediator mode on port 2006.

➤ HTTP requests from the Diagnostics Server in Commander mode to the probes on port 35000-35100. The actual ports on which you must allow communications will depend upon the port numbers that you enabled when you configured the probe and the number of instrumented VMs. For information on setting the probe port range, see "Configuring the Probes for Multiple Application Server JVM Instances" on page 387.

Note: In addition to the above topology, if you are using the LoadRunner Analysis Tool to view offline J2EE results, see "Collating Offline Analysis Files over a Firewall" to properly configure the Controller and the Diagnostics Servers in Mediator mode for offline file retrieval.

Collating Offline Analysis Files over a Firewall

During a LoadRunner / Performance Center load test, the Diagnostics Servers that have probes reporting to them generate an offline analysis file on their host machine. The offline analysis files is retrieved by LoadRunner / Performance Center when it collates the results of a load test.

If there is a firewall between the LoadRunner / Performance Center Controller and the Diagnostics Server involved in a load test, you must configure the Controller and the Diagnostics Server to use the MI Listener utility to enable the transfer of the offline analysis file. The MI Listener utility comes with LoadRunner / Performance Center and should be installed on a machine inside your firewall as shown in the following diagram.
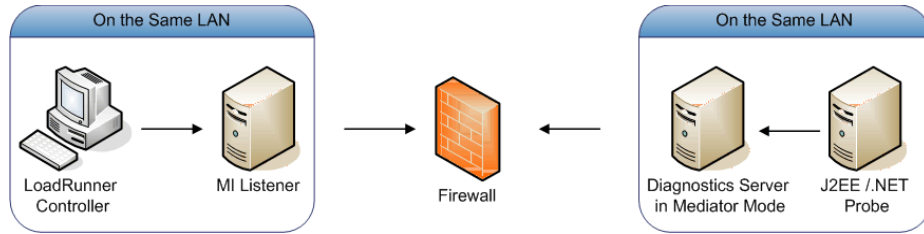
To configure the Controller to access Diagnostics Servers that are behind a firewall see the following sections:

➤ Installing and Configuring the MI Listener.

➤ Configuring the Diagnostics Server in Mediator mode to Work with a Firewall.

➤ Configuring LoadRunner and Performance Center to Work with Diagnostics Firewalls.

Installing and Configuring the MI Listener

The MI Listener component is the same component that is used to serve Load Generators that are outside of a firewall. For more information about how to configure the MI Listener for LoadRunner, refer to the *HP LoadRunner Controller User Guid*e. For more information about how to configure the MI Listener for Performance Center, refer to the *HP Performance Center Administrator's Guide*.

Configuring the Diagnostics Server in Mediator mode to Work with a Firewall

To configure the Diagnostics Server in Mediator mode so that it can work across a firewall, you must complete the following additional configuration steps. If you have not yet installed and configured the Diagnostics Server in Mediator mode you must do so prior to attempting these steps. For instructions on installing the Diagnostics Server in Mediator mode, see Chapter 2, "Installing the Diagnostics Server."

To configure the Diagnostics Server in Mediator mode for a firewall on a Windows machine:

1 Launch the Agent Configuration by running **<diagnostics_server_install_dir>/nanny/windows/bin/AgentsConfig.exe**.

The Agent Configuration dialog box opens:
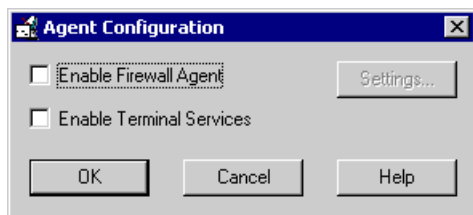
2 Select **Enable Firewall Agent**. The **Settings** button becomes enabled.

3 Click **Settings**. The Agent Configuration process opens the Agent Configuration Settings dialog box.

4 In Value column of the MI Listener Name property, enter the host name or IP address of the machine where the MI Listener was installed.

5 For the **Local Machine Key** property, enter the machine name of the host of the Diagnostics Server in Mediator mode.

Important:

➤ When entering the host name of a Diagnostics component you must use the fully qualified host name, that is, the machine name and the domain name.

➤ Use the System Health Monitor to determine the machine name for the host of the Diagnostics Server in Mediator mode. For more information on the System Health Monitor, see Appendix D, "Using the System Health Monitor."

---

6 Click OK to close the dialog box.
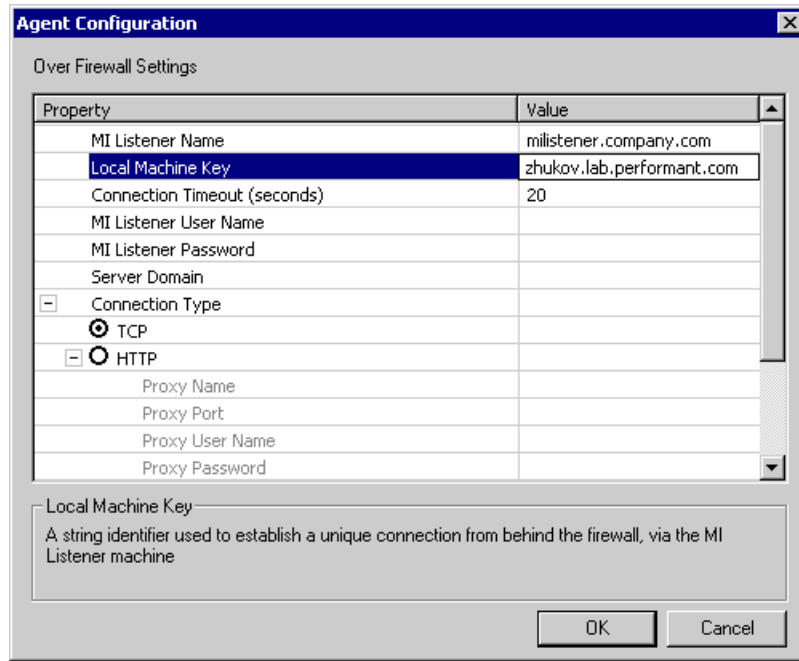
7 Click **OK** again to close the Agent Configuration dialog box.
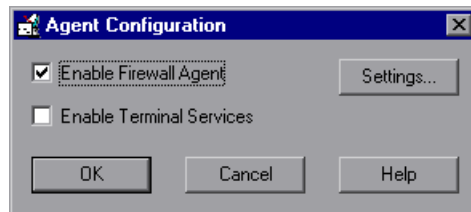
8 The Restart Agent dialog box opens. Click **OK** to restart the Agent.
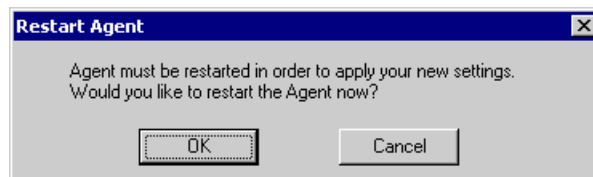
To configure the Diagnostics Server in Mediator mode for a firewall on a UNIX/Linux machine:

1 Modify the **<diagnostics_server_install_dir>/nanny/<platform>/dat/ br_lnch_server.cfg** file.

Change the value of the **FireWallServiceActive** property to 1.

2 Run the following commands to launch the Agent Configuration utility.

For Solaris and Linux, where <platform> is either solaris or linux:

```
export LD_LIBRARY_PATH=.
export M_LROOT=<diagnostics_server_install_dir>/nanny/<platform>
cd $M_LROOT/bin
./agent_config
```

For HP-UX:
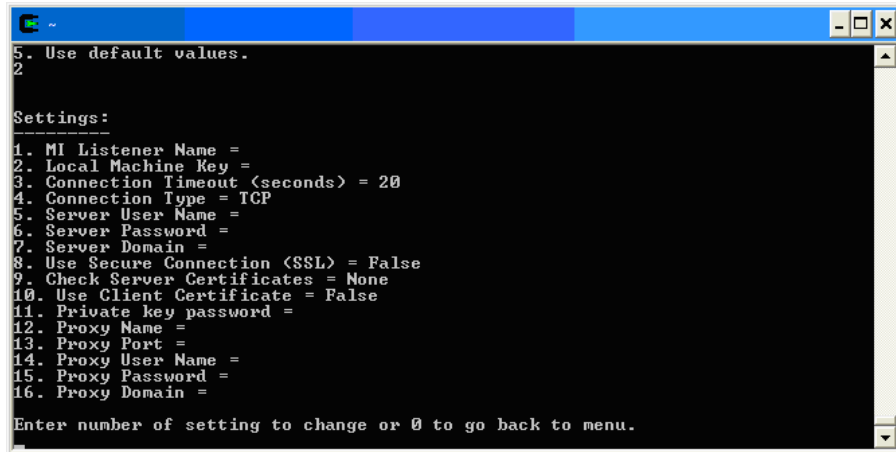
```
export SHLIB_PATH=.
export M_LROOT=<diagnostics_server_install_dir>/nanny/hpux
cd $M_LROOT/bin
./agent_config
```

3 In the Agent Configuration Utility window, press **2**, **Change a Setting**.
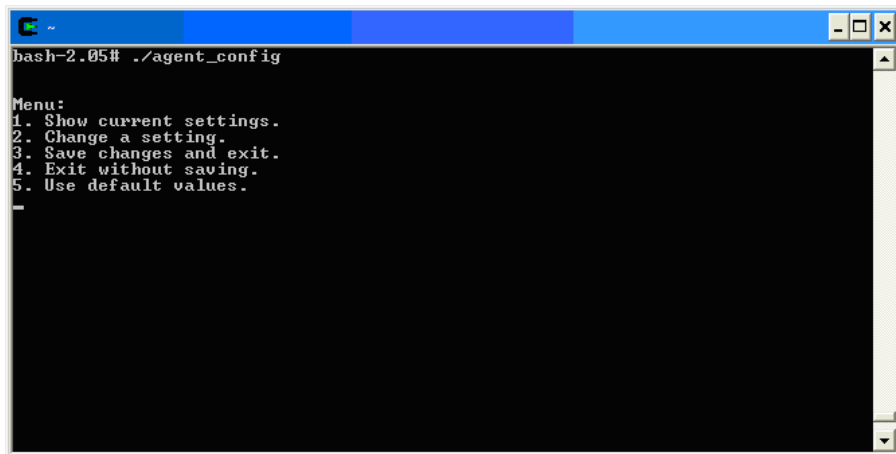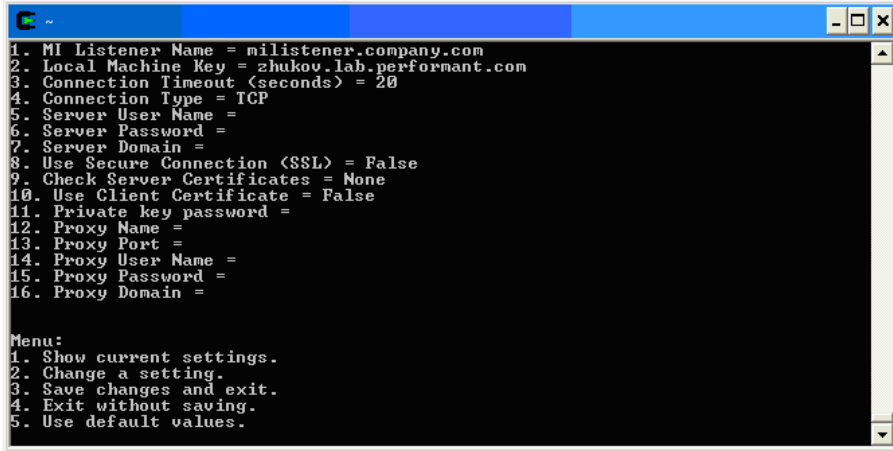
4 A list of settings appears.

Press **1** to select **MI Listener Name**, and enter the machine name or IP address of the MI Listener host.

```
5. Use default values.
2


Settings:
---------
1. MI Listener Name =
2. Local Machine Key =
3. Connection Timeout (seconds) = 20
4. Connection Type = TCP
5. Server User Name =
6. Server Password =
7. Server Domain =
8. Use Secure Connection (SSL) = False
9. Check Server Certificates = None
10. Use Client Certificate = False
11. Private key password =
12. Proxy Name =
13. Proxy Port =
14. Proxy User Name =
15. Proxy Password =
16. Proxy Domain =

Enter number of setting to change or 0 to go back to menu.
```

5 Press **2** to select **Change a Setting** and enter the machine name of the host of the Diagnostics Server in Mediator mode as displayed in the Host: field on the System Health Monitor.

Use the System Health Monitor to determine the machine name for the host of the Diagnostics Server in Mediator mode. For more information on the System Health Monitor, see Appendix D, "Using the System Health Monitor."

6 Press **3**, **Save changes and exit**, to complete the updates.

7 Restart the Diagnostics Server in Mediator mode.

./m_daemon_setup -remove (this stops the server and MI Agent)

./m_daemon_setup -install (this starts the server and MI Agent)

On Linux, if you encounter an error you may need to install the libstdc++.so.5 shared library.

Configuring LoadRunner and Performance Center to Work with Diagnostics Firewalls

After the MI Listener has been installed and your Mediator machines have been configured, you must update the Diagnostics Configuration in LoadRunner / Performance Center so that the application knows to use the MI Listener when it is transferring the offline data from a Mediator that is outside of a firewall.

For Performance Center:

Ensure that you have specified the IP address of the MI Listener machine that is configured to collect application diagnostics data from over the firewall. For details, see the section on MI Listerners in the *HP Performance Center Administrator's Guide*. Also ensure that Diagnostics is enabled in Performance Center. For details, see the section about HP Diagnostics in the *Performance Center User's Guide*.

For LoadRunner:

Ensure that Diagnostics is enabled in LoadRunner. When you configure your load test scenario to work with Diagnostics, ensure that you select the option to work over a firewall and specify the name of the relevant MI Listener server. For details, see the section about HP Diagnostics in the *HP LoadRunner Controller User Guide*.

Part VII

Configuring Diagnostics Metrics Collectors

20

Overview of System and JMX Metrics Collectors

Information is provided about system and JMX metrics capture and how to configure the metric collectors.

This chapter includes:

➤ About Metrics Capture on page 531

➤ Configuring Metric Collector Entries on page 532

About Metrics Capture

The system and JMX metric collectors are installed with the probe. The collectors gather system metrics from the probe host and JMX metrics from the application server. The system and JMX metric collectors are configurable so that you can control which metrics are collected.

Configuring Metric Collector Entries

The system and JMX metric collectors for your probe installation are defined in the metrics configuration file. The properties and entries in the metrics configuration file, **<probe_install_dir>/etc/metrics.config**, allow you to control the metric collectors.

Note: When you update the metrics configuration file, the metric collectors automatically restart so the changes will take effect.

Understanding Metric Collector Entries

Metric Collector entries instruct the metric collectors to gather specific metrics. The parameters on the left hand side of the entry control how the probe gathers the metric from the host or the JVM, and the parameters on the right hand side of the entry define how the collected metrics are processed in Diagnostics and displayed in the user interface.

The entries can have one of the following layouts:

```
<collector_name>/<metric_config>=<metric_id>|<metric_units>|<category_id>
```

or

```
<collector_name>/<metric_config>=
RATE<rate_multiplier>(<metric_id>|<metric_units>|<category_id>)
```

where:

➤ **<collector_name>** indicates the name of the Diagnostics metric collector. The collectors are defined in **metrics.config**.

For system metrics the value of this parameter is system. For JMX metrics the value of this parameter is usually defined as the name of the application server type and the version, such as WebSphere5.

➤ **<metric_config>** identifies the metric that is to be monitored on the host system or on the JVM for the application server. The format of this parameter varies depending on whether you are creating an entry for a system metric or a JMX metric. For information on formatting the **metric_config** property for the system metric collector, see "Capturing Custom System Metrics" on page 540. For information on formatting the **metric_config** property for JMX metrics, see "Capturing JMX Metrics" on page 552.

➤ **RATE(...)** indicates that metric values are converted to a rate (units per second) during sampling.

For example, when the **Rate** parameter is used with the metric **total servlet requests since startup,** the value of the collected metric is converted from a *count of servlet requests* to the *number of servlet requests per second*.

When **Rate** is not used, omit the parenthesis as shown in the first example above.

Note: This parameter should only be used for metrics with non-decreasing values.

➤ **<rate_multiplier>** is an optional parameter that indicates that the rate is to be adjusted by multiplying it by the **<rate_multiplier>**.

For example, when the **Rate** parameter and the **rate_multiplier** are used with the metric **total gc time** (in ms), the value of the metric collected is converted from *the total time for gc* to the *percent time spent in gc*.

➤ **<metric_id>** indicates the name that represents the metric in the UI. The metric_id must be unique in the **metrics.config** file. If the value of the metric_id is the same as one of the default metrics, Diagnostics replaces the metric_id in the entry with a standard name to be used to reference the metric in the UI. If the value of the metric_id is not the same as one of the default metrics, then the metric_id is used as the name of the metric in the UI exactly as shown in the entry.

➤ **<metric_units>** indicates the units of measure in which the metric is reported. This is a required parameter and it must contain one of the following units of measure:

    ➤ microseconds, milliseconds, seconds, minutes, hours, days

    ➤ bytes, kilobytes, megabytes, gigabytes

    ➤ percent, fraction_percent

    ➤ count

    ➤ load

➤ **<category_id>** groups a set of metrics together under the same heading in the tree in the side bar of the Metrics tab in the Java Diagnostics Profiler. This parameter has no impact on the data displayed in the Details Table in the Diagnostics views.

Note: After you have created the metric collector entry, you must add the escape character "\" before each occurrence of a back-slash '\', space ' ', or colon ':'. This is a requirement for Java properties loaded from a file.

Example

The following example shows how to create the metric collector entry for a system metric. To create an entry for a system metric called CPU on a host platform, you would enter the following:

```
system/CPU = CPU|percent
```

where:

- ➤ **system** indicates that the metric is to be collected by the system metric collector

- ➤ the first **CPU** indicates that the metric known as **CPU** on the platform, is being monitored

- ➤ the second **CPU** is the name that is to be used in the UI to label the metric

- ➤ **percent** indicates the units in which the metric is measured on the host, and reported in the UI

Starting Capture for a Metric

To gather information for an additional metric, add an entry for the metric to the appropriate metric collector in the **metrics.config** file using the template described in "Understanding Metric Collector Entries" on page 532.

Modifying a Metric that is Already Being Captured

You can update both the default and the custom metrics entries in the metric collectors in the **metrics.config** file.

Stopping Capture of a Metric

To stop a metric collector from collecting a metric listed in **metrics.config**, you can either delete the metric entry or make the metric entry a comment line by adding a '#' to the beginning.

21

Configuring System Metrics Capture

Information is provided on the process for capturing system metrics and how to configure the system metric collector to capture them.

This chapter includes:

About System Metrics

The system metric collector is installed with the probe. The collector gathers system level metrics, such as CPU usage and memory usage, from the probe's host. The system metric collector is configurable so that you can control which of the system metrics are collected.

Only one instance of the system metric collector is run on a given host, no matter how many instances of the probe have been started on the host. When an instance of the probe is started, it attempts to connect to the UDP port specified in the metrics properties. If a connection is established, the system metric collector instance is started. If a connection cannot be made, then a system metric collector instance has already been started on the host by another instance of the probe and a new instance cannot be started.

Each probe periodically attempts to connect to the port to make sure that a system metric collector is always running. If the probe that started the systems metric collector is stopped, one of the other instances of the probe will start a new instance of the systems metric collector when it finds that the port is available.

System Metrics Captured by Default

The following are the system metrics that the metric collector collects by default for all supported platforms (excluding z/OS):

➤ CPU

➤ MemoryUsage

➤ VirtualMemoryUsage

➤ ContextSwitchesPerSec

➤ DiskBytesPerSec

➤ DiskIOPerSec

➤ NetworkBytesPerSec

➤ NetworkIOPerSec

➤ PageInsPerSec

➤ PageOutsPerSec

You can control which of the default system metrics the system metric collector gathers and you can add other platform specific metrics so that the collector gathers the information for them as well. See "Configuring System Metric Collector" on page 539 for more information. For certain platforms, such as Windows, Solaris, and Linux, you may create custom system metrics which can be gathered by the system metric collector. For details, see "Capturing Custom System Metrics" on page 540.

For information on z/OS system metrics see "Enabling z/OS System Metrics Capture" on page 546.

Configuring System Metric Collector

You can configure the system metrics capture process to run in your environment, and to collect and report the system metrics that are of interest to you, by modifying the properties in the metrics configuration file, **<probe_install_dir>/etc/metrics.config**.

Note: If you update the metrics configuration file, the systems metric collector automatically restarts so that your changes can take effect.

Modifying the Default Port

The default port for the metric collectors is **35000**. This value can be modified using the **system.udp.port** property if the configuration for your probe host requires that another port be used.

To modify the default port:

1 Locate the **system.udp.port** property in **metrics.config**.

2 Change the value of the **system.udp.port** property to the number of the port that you want to be used by the system metric collector. The default port is **35000**.

Note: The port assigned to the system metric collector is not related to the port for the probe's Web server.

For information on configuring the entries in the metric collectors see "Configuring Metric Collector Entries" on page 532.

Disabling System Metrics Collection

To disable the collection of system metrics so that they will not be collected or displayed in the UI, set the value of the **system.udp.port** property to -1.

Capturing Custom System Metrics

The Windows, Solaris, and Linux platforms allow you to create custom system metrics. The custom metrics can be monitored by the system metric collector.

The following sections provide instructions for creating the metrics and updating the entries in the system metric collector so that the custom metrics can be monitored.

Capturing Custom System Metrics on Windows Hosts

Using the features of Windows System Monitor, you can add counters to represent the performance of specific aspects of a system or service. The counters are tracked and reported in the Windows System Monitor, and can be monitored by the Diagnostics system metric collector.

To add counters using the Windows System Monitor:

 1 Start the Windows Performance Monitor:

   a Select **Start** > **Run** from the Start menu.

   b In the **Open** box on the **Run** dialog box type perfmon.

   The **Performance** dialog box opens showing the **System Monitor** graph with a table of the current counters beneath the graph.

 2 Display the Add Counters dialog box:

   Right-click the **System Monitor** graph and select **Add Counters...** from the pop-up menu.

Windows displays the **Add Counters** dialog box:
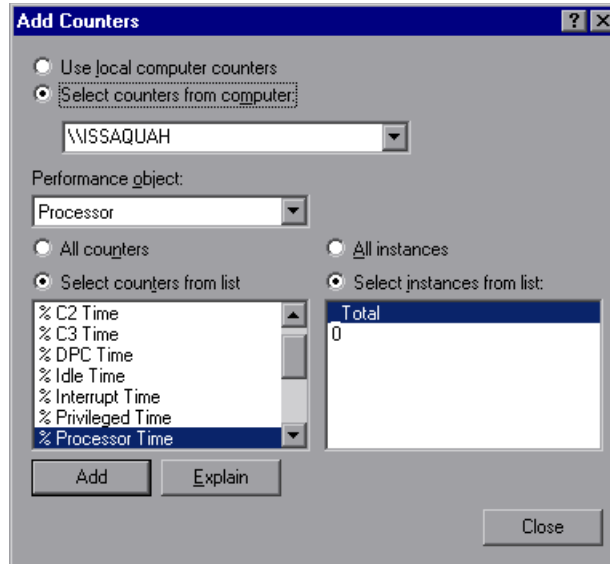
3 Ensure that the host computer is selected from **Select counters from computer** list.

4 In the **Performance object** list, select the object that the counter belongs to.

5 Choose **Select counters from list,** and select a counter from the list of counters that follows.

6 Choose **Select instances from list**, and select an instance from the list of instances that follows.

7 Click **Add**.

Once a counter has been added to the Systems Monitor, the system metric collector can be configured to gather the metrics for the counter. The following instructions will guide you through the steps to create an entry for the **metrics.config** based upon the following template:

```
<collector_name>/<metric_config>= <metric_id>|<metric_units>
```

This template is described in "Understanding Metric Collector Entries" on page 532.

To collect metrics for a Windows System Monitor Counter:

 1 Open **<probe_install_dir>/etc/metrics.config.**

 2 Create the <metric_config> part of the entry using the following template, type the entry for the counter:

```
\<performance_object>(<instance>)\<counter>
```

In the example shown in the preceding screen image:

➤ the selected Performance Object is %Processor

➤ the selected Instance is _Total

➤ the selected Counter is Processor Time

The <metric_config> portion of the entry that would be created for this example would be:

```
\Processor(_Total)\% Processor Time
```

 3 Fill in the rest of the system metric entry template as shown in the following example:

```
system/\Processor(_Total)\% Processor Time = ProcessorTime|percent
```

 4 Format the initial entry by prepending a back-slash '\' before each occurrence of back-slash '\', space ' ', or colon ':' in the initial entry.

Following this step, the initial entry in the previous step becomes:

```
system/\\Processor(_Total)\\%\ Processor\ Time = ProcessorTime|percent
```

This is the correctly formatted entry for **metrics.config** to enable the system metric collector to gather the metrics for a Windows System Monitor counter.

```
system/\\\\RemoteMachine\\Processor(_TOTAL)\\%\ Processor\ Time=
Processor\ Time(Remote Machine)|percent
```

Note: Assuming **perfmon** is setup properly on a remote machine, you can use it to get metrics from remote machines by adding **MachineName** before the Performance object name as shown in the following example:

system/\\\\RemoteMachine\\Processor(_TOTAL)\\%\ Processor\ Time=Processor\ Time(Remote Machine)|percent

---

Capturing Custom System Metrics on Solaris Hosts

The Solaris system metrics that can be monitored by the system metric collector are found using the **kstat** command. Only a subset of the metrics found using the **kstat** command can be monitored by the system metric collector.

To collect metrics for a Solaris system metric:

1 Execute the **ksat** command and identify the metric that you want to monitor.

A Solaris system metric has the following format:

```
module:instance:name:statistic
```

For example:

```
vmem:35:ptms_minor:free
```

2 To cause the metric collector to gather the metrics for an additional system metric, add an entry for the metric to the system metric collector in the **metrics.config** file using the following template:

```
<collector_name>/<metric_config>= <metric_id>|<metric_units>
```

This template is described in "Understanding Metric Collector Entries" on page 532.

Using this template, the example from the previous step would initially appear as follows:

```
system/vmem:35:ptms_minor:free = Virtual Memory (35) Free | count
```

3 Format the initial entry by prepending a back-slash '\' before every back-slash '\', space ' ', or colon ':'.

Following this step the initial entry in the previous step becomes:

```
system/vmem\:35\:ptms_minor\:free = Virtual\ Memory\ (35)\ Free | count
```

This is the correctly formatted entry for **metrics.config** to enable the system metric collector to gather the metrics for a Solaris systems metric.

Capturing Custom System Metrics on Linux Hosts

The Linux system metrics that can be monitored by the system metric collector are found in the /**proc** file system. To configure the system metric collector to gather custom Linux metrics you must scan the/**proc** file system to locate the desired metric, and then create the system metric collector entry for the metric in **metrics.config** based upon the where the metric information is located.

To collect metrics for a Linux system metric:

1 Scan the /**proc** file system to locate the metric that you would like the Diagnostics system metric collector to monitor.

To create the system metrics configuration entry in **metrics.config** for the Linux metric, you must explicitly specify where the value for the system metric is located. The location is specified using the following values:

➤ **File name.** The name of the file where the metric information is located, including the path from the /**proc** directory.

➤ **Line offset.** A count of the number of lines in the file to the line where the system metric is located. The first line is counted as line 0.

➤ **Word offset.** A count of the number of words that the metric value is offset into the line in the file. The first word in the line is counted as line 0. The value at the specified offset must be an unsigned integer.

For example, if you wanted the system metric collector to monitor the SwapFree system metric so that you can see it displayed in the Diagnostics views, you would scan the **/proc** directory to locate the metric, and you would discover that the metric is located in the **meminfo** file. The layout of this file is as follows:

```
MemTotal: 515548 kB
MemFree: 1552 kB
Buffers: 41616 kB
Cached: 152084 kB
SwapCached: 46064 kB
Active: 402720 kB
Inactive: 75328 kB
HighTotal: 0 kB
HighFree: 0 kB
LowTotal: 515548 kB
LowFree: 1552 kB
SwapTotal: 1048568 kB
SwapFree: 779192 kB
Dirty: 4544 kB
Writeback: 0 kB
Mapped: 300056 kB
Slab: 28764 kB
Committed_AS: 801364 kB
PageTables: 3184 kB
VmallocTotal: 499704 kB
VmallocUsed: 2184 kB
VmallocChunk: 497324 kB
HugePages_Total: 0
HugePages_Free: 0
Hugepagesize: 4096 kB
```

The location of the SwapFree metric in this file would lead to the following values:

➤ **File name:** meminfo

➤ **Line offset:** 12

➤ **Word offset:** 1

2 To gather the metrics for an additional system metric, add an entry for the metric to the system metric collector in the **metrics.config** file using the following template:

```
<collector_name>/<line>:<word>:<file>= <metric_id>|<metric_units>
```

This template is a version of the template described in "Understanding Metric Collector Entries" on page 532. The **<metric_config>** property has been replaced with the properties **<line>:<word>:<file>**.

Using this template, the example from the previous step would initially appear as follows:

```
system/12:1:meminfo = Swap Free | kilobytes
```

3 Format the initial entry by prepending a back-slash '\' before every back-slash '\', space ' ', or colon ':'.

Following this step the initial entry in the previous step becomes:

```
system/12\:1\:meminfo = Swap\ Free | kilobytes
```

This is the correctly formatted entry for **metrics.config** to enable the system metric collector to gather the metrics for a Solaris systems metric.

Enabling z/OS System Metrics Capture

The following system metrics can be collected for the z/OS platform:

➤ CPU

➤ DiskIOPerSec

➤ DiskBytesPerSec

System metrics are not captured by default, as this requires some system configuration changes. You must perform the following configuration steps in order to enable capture of z/OS system metrics.

To enable z/OS system metrics capture:

1 Change the permissions for the directory **<probe_install_dir>/bin/** to recursively allow execution. This can be done using a command like the following example:

```
chmod -R 770...
```

2 Change the permissions for the directory **<probe_install_dir>/bin/390-zos/ systemmetrics** to allow execution. This can be done using a command as illustrated in the following example:

```
chmod -R 0+x ...
```

3 Start the RMF Monitor III and ensure that SMF record 70-79 is collecting.

4 Start the RMF Data Buffer on one or more systems in the sysplex.

5 Check the list of system names passed to the ERBDSQRY service.

6 Ensure that the system is collecting SMF record 92 with subtype 5.

22

Configuring JMX Metric Capture

Information is provided on the process for capturing JMX metrics and how to configure metric collectors to capture them.

This chapter includes:

➤ About JMX Metrics on page 549

➤ Configuring WebSphere for JMX Metric Collection on page 550

➤ Configuring JMX Metric Collectors on page 552

➤ Capturing Custom JMX Metrics on page 552

About JMX Metrics

The Java Probe comes with pre-defined JMX metric collectors that access the JMX metrics from the following application servers:

➤ IBM WebSphere

➤ BEA WebLogic

➤ SAP NetWeaver

➤ Oracle AS

The Java Probe can also collect JMX data from any J2EE server that supports the JMX standard.

The Java Probe runs the JMX metric collectors periodically to collect the metrics from the application server. The collected metrics are displayed on the user interfaces in both HP Diagnostics and Java Diagnostics Profiler.

Configuring WebSphere for JMX Metric Collection

You may need to configure the Performance Monitoring Service on the WebSphere server in order to start receiving JMX metrics.

This section includes examples for configuring WebSphere 5.x and 6.x for JMX metrics collection.

To configure WebSphere 5.x server for JMX metrics collection:

 1 Open the WebSphere Administrative Console.

 2 In the Console navigation tree, select **Servers** > **Application Servers**. The consoled displays a table of the application servers.

 3 Click name of the application server that you want to configure from the Application Servers Table. The console displays the **Runtime** and the **Configuration** tabs for the selected application server.

 4 Click the **Configuration** tab.

 5 In the **Configuration** tab:

   ➤ Click **Performance Monitoring Service**.

   ➤ Select the **Startup** check box under **General Properties**.

   ➤ Set the **Initial Specification Level** to **standard**.

   ➤ Click **Save**.

 6 Click **Apply** or **OK**.

 7 If Java 2 Security is enabled on the application server, open the server policy file. **(<WebSphere 5.x Installation Directory>/properties/server.policy)** and add the following security permission to the file, to allow JMX collection:
grant codeBase "file:/<probe_install_dir>/lib/probe-jmx.jar"
{ permission java.security.AllPermission; }

 8 Restart the application server.

The following are the instructions for configuring WebSphere 6.x for JMX metrics collection.

To configure WebSphere 6.x server for JMX metrics collection:

1 Open the WebSphere Administrative Console.

2 In the Console navigation tree, select **Servers > Application Servers**. The consoled displays a table of the application servers.

3 Click name of the application server that you want to configure from the Application Servers Table. The console displays the **Runtime** and the **Configuration** tabs for the selected application server.

4 Click the **Configuration** tab.

5 In the **Configuration** tab:

➤ Under the Performance heading, click **Performance Monitoring Infrastructure (PMI)**.

➤ Under the General Properties heading, select the **Enable Performance Monitoring Infrastructure (PMI)** check box.

➤ Under the **Currently monitored statistic set** heading select **Extended**.

6 Click **Apply** or **OK**.

7 If Java 2 Security is enabled on the application server, open the server policy file. **(<WebSphere 6.x Installation Directory>/work/tools/ibm-6.0/websphere/appserver/profiles/default/properties/server.policy)** and add the following security permissions to the file, to allow JMX collection:
grant codeBase "file:/<probe_install_dir>/lib/probe-jmx.jar"
{ permission java.security.AllPermission; }

grant codeBase "file:/<probe_install_dir>/lib/probe-jmx-was6.jar" {
  permission java.security.AllPermission;
};
Restart the application server.

Configuring JMX Metric Collectors

The JMX metric collectors are configurable so that you can control which of the JMX metrics are collected. The JMX metric collectors are defined in the **<probe_install_dir>/etc/metrics.config** file. Typically a separate collector is defined for each major version of each application server.

For information on configuring the metric collectors see "Configuring Metric Collector Entries" on page 532.

Capturing Custom JMX Metrics

The Java Probe is installed with a number of predefined JMX metric collectors for the application servers listed in "About JMX Metrics" on page 549. You may modify or remove any of the JMX metric entries from the predefined collectors by following the instructions in "Configuring Metric Collector Entries" on page 532. You may also create entries in the existing metric collectors and even create new collectors if there are additional JMX metrics that you would like Diagnostics to monitor.

The following sections provide instructions creating new entries in the JMX metric collectors so that additional JMX metrics can be monitored.

Capturing JMX Metrics

The metric collectors installed with the Java Probe include entries for many of the JMX metrics that are available for each application server. However, there may be other JMX metrics that you would like to monitor or new metrics may be exposed by the application server vendor. The following instructions will guide you through the process of creating the JMX metric entries based upon the following template:

```
<collector_name>/<metric_config>= <metric_id>|<metric_units>
```

This template is described in "Understanding Metric Collector Entries" on page 532.

To capture JMX metrics:

1 Open **<probe_install_dir>/etc/metrics.config.** and locate the JMX metric collector that is appropriate for the application that is being monitored by the Java Probe.

2 The **<collector_name>** parameter is the same as the rest of the entries in the collector. If you were creating an entry for WebLogic, the value of this parameter would be WebLogic.

3 Create the **<metric_config>** parameter using the following template:

```
<MBean object name pattern>.<attribute name>
```

For JMX metrics the **<metric_config>** parameter is a pattern that the collector uses to find a matching MBean. The pattern consists of two components, separated by the '.' character:

➤ **<MBean object name pattern>** is the string representation of the object name of an MBean.

➤ **<attribute_name>** is the name of the attribute that represents the metric.

For example, for a WebLogic application server, the <metric_config> parameter for the throughput of all **Execute Queues** is configured as:

```
*:Type=ExecuteQueueRuntime,*.ServicedRequestTotalCount
```

For an explanation of metric patterns see "Understanding Metric Patterns" on page 554.

4 Fill in the rest of the JMX metric entry template as shown in the following example:

```
WebLogic/*:Type=ExecuteQueueRuntime,*.ServicedRequestTotalCount =
RATE(Execute Queues Requests / sec|count|Execute Queues)
```

5 Format the initial entry by prepending a back-slash '\' before every back-slash '\', space ' ', equals (=), or colon ':'.

Following this step the initial entry in the previous step becomes:

```
WebLogic/*\:Type\=ExecuteQueueRuntime,*.ServicedRequestTotalCount =
RATE(Execute Queues Requests / sec|count|Execute Queues)
```

This is the correctly formatted entry for a JMX metric collector to enable the collector to gather a WebLogic JMX metrics.

Understanding Metric Patterns

For JMX metrics the <metric_config> parameter is a pattern that the collector uses to find a matching MBean. For example:

```
*:Type=ExecuteQueueRuntime,*.ServicedRequestTotalCount
```

In the example above, the object name is ***:Type=ExecuteQueueRuntime,***, which could actually resolves to many MBeans whose names have the **Type** component equal to **ExecuteQueueRuntime**. **ServicedRequestTotalCount** is an attribute name for which metric values will be collected by the JMX metric collector.

Note: Current implementation of the JMX collector only supports attributes that are numeric in type (i.e., long, integer, etc.).

The JMX metric collector first uses MBeanServer's query mechanism to find the matching MBeans for each object name provided in the configuration. For JMX metrics the object names are a pattern that the collector uses to find a matching MBean. For more details around the object names, see http://java.sun.com/j2ee/1.4/docs/api/javax/management/ObjectName.html.

Since MBean object names are patterns that can resolve into multiple
MBeans, the JMX collector will validate all of the attribute names in the
entry against all MBeans that match the pattern, and will aggregate the
attribute values over the set of those matching MBeans. Of course, it is not
always the case that the object name resolves into multiple MBeans. For
example, the following object name resolves to a single MBean (on a
WebLogic application server):

```
*\:Name\=weblogic.kernel.Default,Type\=ExecuteQueueRuntime,
*.ServicedRequestTotalCount
```

Part VIII

Setting Up Integration with Other HP Software Products

23

Setting Up Business Availability Center to Use Diagnostics

Information is provided on setting up HP Business Availability Center to enable integration with HP Diagnostics.

This chapter includes:

About Setting Up Business Availability Center to use Diagnostics

Before using HP Diagnostics with Business Availability Center, you provide Business Availability Center with the information that it needs in order to communicate with the Diagnostics components.

To set up Diagnostics in Business Availability Center:

1 Specify the Diagnostics Server details.

Enter the Diagnostics Server details in Business Availability Center. For more information, see "Specifying the Diagnostics Server Details" on page 561.

2 Assign relevant permissions (optional).

Grant different permissions to different Diagnostics users. (This step is optional.) For more information, see "Assigning Permissions for Diagnostics Users" on page 566.

3 For Windows 2003 only: Change your Internet browser settings.

When your Internet browser is running in a Windows 2003 environment, you have to change your Internet browser settings in order to access the Diagnostics configuration and application pages in Business Availability Center. For more information, see "Accessing the Diagnostics Pages in Windows 2003" on page 567.

4 Enable cookies on the Diagnostics Server host.

Cookies must be enabled to view Diagnostics data in Business Availability Center. This can usually be accomplished by adding the registered Diagnostics Server as a trusted site in the browser configuration.

Specifying the Diagnostics Server Details

To make HP Diagnostics accessible from Business Availability Center, you specify the Diagnostics Server details.

Note: If you are using Windows 2003, you need to configure your Internet browser settings to access the Diagnostics Configuration page. For more details, see "Accessing the Diagnostics Pages in Windows 2003" on page 567.

To specify the Diagnostics Server details in Business Availability Center:

1 Log on to Business Availability Center.

2 Select **Admin** > **Diagnostics** to open the Diagnostics Configuration page.

Note: If you try to access HP Diagnostics (by clicking **Diagnostics** on the Site Map page or by selecting **Applications** > **Diagnostics**) before you have configured the Diagnostics Server, you receive a message instructing you to register the Diagnostics Server. Click the link to open the Diagnostics Configuration page.

---

3 Enter the details for the Diagnostics Server in Commander mode.

➤ **Diagnostics server host name.** Enter the name of the machine that is host to the Diagnostics Server in Commander mode.

Even if the Diagnostics Server is installed on the same system as Business Availability Center, you still need to enter the actual name of the host in the **Diagnostics server host name** box. It is not sufficient to type **localhost** instead of the host name.

If Business Availability Center will be accessed through a fully-qualified domain name then you should also register the Diagnostics Server host with a fully-qualified domain name.

➤ **Diagnostics server port number.** Enter the port number used by the Diagnostics Server in Commander mode. The default port number is **2006**.

➤ **Diagnostics server protocol.** Select the communication protocol through which Business Availability Center connects to Diagnostics - either **HTTP** or **HTTPS**.

Note: If you select **HTTPS** as your communication protocol, additional configuration steps are required. For more information about the steps required, see Chapter , "Enabling HTTPS Between Diagnostics Components."

---

4 After you have entered the Diagnostics Server details and ensured that these details are accurate, click **Submit** to complete the Diagnostics Server configuration process.

If the server name you have entered is incorrect or if the server is unavailable, an error message is displayed.

When you click **Submit**, the Diagnostics Server details are saved in Business Availability Center and the Business Availability Center server details are automatically registered on the Diagnostics Server machine.

The **Registration** tab in the Diagnostics Configuration page opens, displaying the Diagnostics Server details that you just specified and the Business Availability Center server details.

Where necessary, you can manually change the Business Availability Center server details in the **Enter Business Availability Center details** section of the Registration tab.

Note: Verify that the root URL in the **Centers Server URL** box, matches the root URL that you use to access Business Availability Center.

Changing the Diagnostics Server Details

You may want to change the Diagnostics Server details or remove the Diagnostics registration completely.

To remove the Diagnostics registration:

1 Select **Admin** > **Diagnostics**.

2 In the **Registration** tab, click **Remove Diagnostics Registration**.

3 In the message that opens, click **OK** to confirm that you want to remove the Diagnostics registration.

A message is displayed, confirming that you successfully removed the Diagnostics registration.

To register a new Diagnostics Server, select **Admin** > **Diagnostics** and follow the procedure explained in "Specifying the Diagnostics Server Details" on page 561.

Understanding the Diagnostics Configuration Page

You access the Diagnostics Configuration page by selecting **Admin** > **Diagnostics**. The Diagnostics Configuration page consists of the following three tabs, which are described in this section:

➤ Registration Tab

➤ Downloads Tab

➤ System Health Tab

Registration Tab

The Registration tab displays the following details:

➤ The Diagnostics Server details that you registered in Business Availability Center. To change these details, see "Changing the Diagnostics Server Details" on page 564.

➤ The Business Availability Center server details that were automatically registered on the Diagnostics Server machine. You can manually change the Business Availability Center server details in the **Enter BAC server details** section.

For information about registering Diagnostics in Business Availability Center for the first time, see "Specifying the Diagnostics Server Details" on page 561.

Downloads Tab

The Downloads tab provides links to the Diagnostics Probe/Collector installers, enabling you to download the probe or collector for your relevant platform.

If you did not specify the path to the Probe/Collector installers during the installation of the Diagnostics Server, the Downloads tab will not display any components. For more information, see Chapter 2, "Installing the Diagnostics Server."

System Health Tab

Provides you with a map of all the components of your HP Diagnostics deployment and gives you real-time status and health information for each component. For detailed information about the System Health monitor, see Appendix D, "Using the System Health Monitor."

Assigning Permissions for Diagnostics Users

Business Availability Center enables you to apply permissions to users and user groups for specific resources that are defined in the system. There are specific types of permission operations that administrators can grant Diagnostics users.

In the following example of a screen in the Platform Administration page in Business Availability Center, the Diagnostics permission operations are displayed:
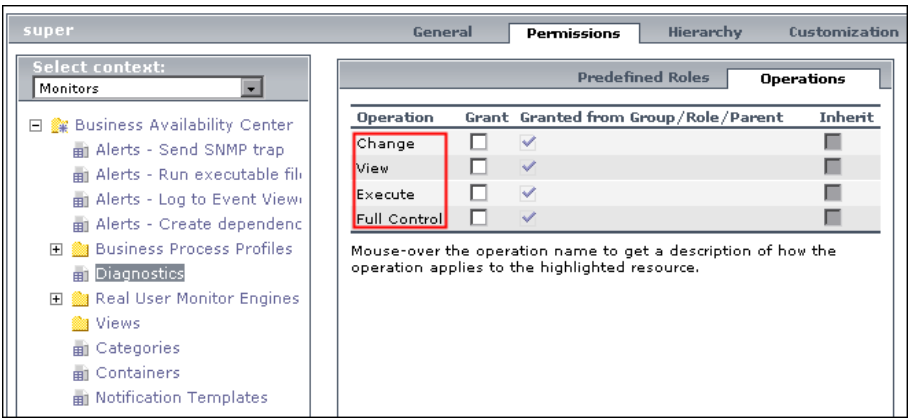
When applying permissions in Business Availability Center, administrators can grant Diagnostics users the following types of permission operations:

➤ **Change**: Enables viewing Diagnostics administration and configuring the Diagnostics settings.

➤ **View**: Enables viewing the Diagnostics application when accessing Diagnostics from the Business Availability Center.

➤ **Execute**: Enables changing the settings in the HP Diagnostics UI, such as setting thresholds.

➤ **Full Control**: Enables performing all operations on Diagnostics, and granting and removing permissions for those operations.

For detailed information about how to assign user permissions in Business Availability Center, refer to *Platform Administration* in the *HP Business Availability Center Documentation Library*.

Accessing the Diagnostics Pages in Windows 2003

When your Internet browser is running in a Windows 2003 environment, you have to change your Internet browser settings in order to access the Diagnostics configuration and application pages in Business Availability Center.

To access the Diagnostics pages in a Windows 2003 environment:

1 In Internet Explorer, select **Tools** > **Internet Options** to open the Internet Options dialog box.

2 In the **Privacy** tab, click **Sites** to open the Per Site Privacy Actions dialog box.

3 In the **Address of Web site** box, enter the name of the Diagnostics Server.

➤ If you entered an IP address when you registered the Diagnostics Server in Business Availability Center, enter the IP address. If you entered a host name in Business Availability Center, enter the host name.

➤ Include the http:// or https:// prefix, and the port number, as illustrated in the following example:

http://<Diagnostics_server_host>:2006/

4 Click **Allow.**

5 Click **OK** to close the Per Site Privacy Actions dialog box.

6 Click **OK** to close the Internet Options dialog box.

Data Samples and Web Service CIs Sent to Business Availability Center

When you integrate Diagnostics with Business Availability Center, Diagnostics monitors your enterprise applications and sends application performance and availability data to Business Availability Center as **Data Samples**. Diagnostics provides the following data samples to Business Availability Center:

➤ ws_perf_aggr_t (SOA Sample)

➤ ws_event_aggr_t (SOA Sample)

➤ appmon_ru_t (Probe Sample)

➤ appmon_vu_t (Transaction (BPM) Sample)

See the *HP Business Availability Center Documentation Library* for more information on Data Samples.

With Diagnostics 7.50 or later, Diagnostics creates CIs for the SOA Sample (ws_perf_aggr_t) and adds these CIs directly into the CMDB. For other types of data samples from Diagnostics, Business Availability Center creates the appropriate CIs.

In rare cases where you want to change the timing of the process that adds these Web services CIs to the uCMDB, a number of properties are provided in the **server.properties** file in Diagnostics.

Also in the rare case when you want to force a synchronization between Diagnostics and Business Availability Center for these Web service CIs, a **synchronize** function is available on the Diagnostics Server. See "Accessing the Diagnostics Server Administration Page" on page 583.

Note: Anytime a Business Availability Center system is upgraded or re-installed, a manual hard sync is needed (or a wait period of 24 hours) before Web services currently shown in Diagnostics are forwarded to Business Availability Center. To do a hard sync, go to the Diagnostics Server Administration page, select **Synchronize** and then select **Hard** for A**ll Customers**.

24

Installing the LoadRunner Diagnostics Add-in

The LoadRunner Diagnostics Add-in makes it possible for you to access the Diagnostics UI from within LoadRunner. Once the LoadRunner Diagnostics Add-in has been installed, you can configure LoadRunner to use the Diagnostics components to gather performance metrics during your load tests, connect to the Diagnostics UI and use the System Health Monitor.

This chapter includes:

➤ Before Installing the LoadRunner Diagnostics Add-in on page 570

➤ Installing the LoadRunner Diagnostics Add-in on page 570

Before Installing the LoadRunner Diagnostics Add-in

Before installing the LoadRunner Diagnostics Add-in, you should have the Diagnostics Server in Commander mode installed and have LoadRunner installed. For instructions on installing LoadRunner, refer to the *HP LoadRunner Installation Guide*.

Diagnostics 8.00 can be integrated with LoadRunner 9.10 or later using the add-in provided with Diagnostics 8.00.

Note: Earlier versions of the LoadRunner add-in will NOT work against a Diagnostics 8.00 server.

Installing the LoadRunner Diagnostics Add-in

The LoadRunner Diagnostics Add-in is installed on the LoadRunner Controller host machine.
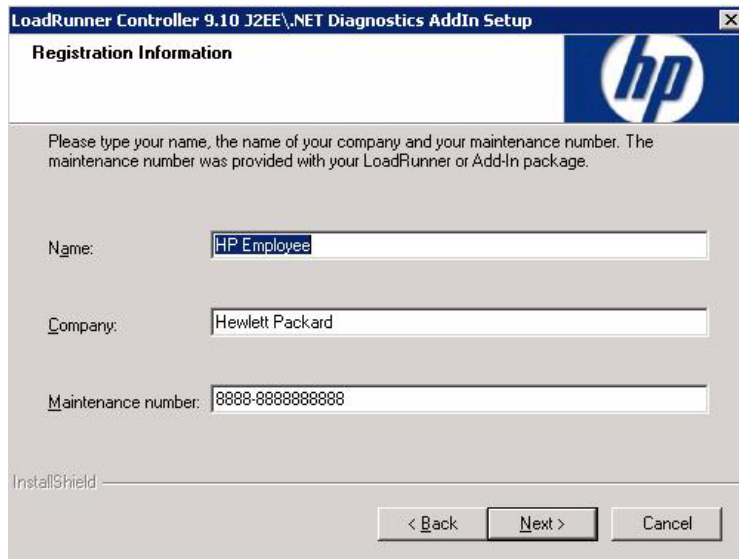
Note: In the Diagnostics 8.00 release, the LoadRunner Diagnostics add-in has been changed to use a small bootstrapper to dynamically download most of the software required from the Diagnostics server when it is first executed. If Diagnostics is updated, the new Diagnostics files should be picked up automatically at the next LoadRunner execution.

To install the LoadRunner Diagnostics Add-in:

1 Close LoadRunner or any LoadRunner related processes (such as the LoadRunner Agent) running on the LoadRunner Controller system.

2 Run **setup.exe** from the LR_AddIn directory of the Diagnostics installation disk. The setup installation program is launched.

 3 The software license agreement is displayed. Read the agreement and click
   Yes to accept it.

 4 The Registration Information dialog box opens.

   In the Registration Information dialog box, type your name, the name of
   your company, and your LoadRunner maintenance number. You can find
   the maintenance number in the maintenance pack shipped with
   LoadRunner.

   Click **Next** to start the installation process. The installation process begins.

 5 When the installation process is complete, the installation wizard displays a
   confirmation message.

   Click **Finish** to complete the installation process.

   In certain cases when related LoadRunner processes are running on your
   computer (such as the LoadRunner Agent), you will be required to restart
   your computer to complete the LoadRunner Add-in installation process.

Note: No uninstall utility is provided for the LoadRunner Diagnostics add-in.

Note: If you are installing the LoadRunner Diagnostics Add-in on a Windows XP machine with service pack 1 and Windows XP Hotfix Q328310 applied, you will receive an Application Error message for **iKernal.exe**. This message is issued because the Windows XP Hotfix Q328310 contains a Win32 API that does not execute as expected by the InstallShield engine. To resolve this problem, see the recommended solutions at the Java Technology Help web site, http://java.com/en/download/help/ikernel.jsp.

Before you can access the Diagnostics UI from within LoadRunner you need to configure LoadRunner and provide the necessary information to enable communication with the Diagnostics components. See Chapter 25, "Setting Up LoadRunner to Use HP Diagnostics" for information on configuring LoadRunner for integration with Diagnostics.

25

Setting Up LoadRunner to Use HP Diagnostics

General information is provided on configuring LoadRunner to enable HP Diagnostics for use in a load test.

This chapter includes:

➤ About Setting Up LoadRunner to Use Diagnostics on page 574

➤ Configuring LoadRunner Scenarios to use HP Diagnostics on page 574

➤ Improve Transfer of Large Offline Analysis Files on page 575

About Setting Up LoadRunner to Use Diagnostics

Before you can access the Diagnostics UI from within LoadRunner, you need to provide LoadRunner with the information it needs to communicate with the Diagnostics components.

To make the Diagnostics UI accessible from LoadRunner, you specify the Diagnostics Server details. You only need to specify these details the first time you use LoadRunner with Diagnostics. See the *HP LoadRunner Controller User Guide* for details about specifying the Diagnostics Server details.

Note: Before specifying the Diagnostics Server details, ensure that the LoadRunner Controller is closed. When the Controller is open, you can view the Diagnostics configuration settings, but you cannot change them.

To specify the Diagnostics Server details in LoadRunner:

Configuring LoadRunner Scenarios to use HP Diagnostics

Each time you want to capture Diagnostics metrics in a load test scenario, you need to configure the Diagnostics parameters for the scenario and select the probes that will be included in the scenario. You configure your scenario for Diagnostics from the LoadRunner Controller. See the HP LoadRunner Controller User Guide for details.

Note: If you have saved a scenario with the Diagnostics settings already configured, you do not need to reconfigure the Diagnostics parameters each time you run that scenario.

Improve Transfer of Large Offline Analysis Files

The offline analysis files (.eve) generated by Diagnostics during a LoadRunner or Performance Center test run can get quite large. At the end of the runs these files are transferred from the Diagnostics servers to the LoadRunner/Performance Center controller for collation and analysis. You can improve the transfer time and load time of the offline analysis files that include Diagnostics data by lowering the resolution of the .eve files.

Use the **bucket.lr.offline.duration** property and the **bucket.lr.offline.sr.duration** properties in the **server.properties** file on the Diagnostics server to increase the aggregation period (for example, from 5s to 15s). These properties allow you to define how many five second trend points are to be aggregated together to produce a single sample for offline analysis.

26

Setting Up Performance Center to Use HP Diagnostics

General information is provided on configuring Performance Center to enable HP Diagnostics for use in a load test.

This chapter includes:

➤ About Setting Up Performance Center to Use HP Diagnostics on page 578

➤ About Configuring Performance Center Load Tests to Use HP Diagnostics on page 579

➤ Managing Performance Center Offline Files on page 580

About Setting Up Performance Center to Use HP Diagnostics

Performance Center and Diagnostics are integrated products that have been designed to work together to provide information to help you understand and improve the performance of your applications.

To make Diagnostics accessible from Performance Center, the following Diagnostics Server details are specified in Performance Center.

➤ **Server Name.** The name of the machine that is host to the Diagnostics Server in Commander mode.

➤ **Port Number.** The port number used by the Diagnostics Server in Commander mode. The default port number is **2006**.

➤ **Login Name.** The user name with which you log on to HP Diagnostics. The default user name is admin.

The user name that you specify should have **view**, **change** and **execute** privileges. For more information about user privileges, see "Understanding User Privileges" on page 596.

➤ **Password.** The password with which you log on to HP Diagnostics. The default password is admin.

➤ **Communication**. The communication protocol with which Performance Center accesses the Diagnostics Server.

If **HTTPS** is the communication protocol, additional configuration steps are required. For more information about the steps required, see Chapter , "Enabling HTTPS Between Diagnostics Components."

Note: You only need to specify these details the first time you use Performance Center with Diagnostics. You provide this information on the Diagnostics page of the Performance Center Administration Site.

About Configuring Performance Center Load Tests to Use HP Diagnostics

Each time you want to capture Diagnostics metrics in a load test, you need to configure the Diagnostics parameters for the load test and select the probes that will be included in the load test.

For complete instructions on how to configure Performance Center to integrate with Diagnostics see the HP Performance Center User's Guide section about HP Diagnostics integration with Performance Center.

If there is a firewall between the Performance Center Controller and the Diagnostics Server involved in a load test, you must configure the Controller and the Diagnostics Server to use the MI Listener utility to enable the transfer of the offline analysis file. Also you need to specify the IP address of the MI Listener machine in Performance Center.

And you must configure the Diagnostics Server in Mediator mode so that it can work across a firewall. See "Configuring Diagnostics to Work in a Firewall Environment" on page 515.

The benefit of enabling the "Monitor server requests" functionality in the integration is that calls into a back-end VM can be captured even in the case where:

➤ the probe is not capturing RMI calls.

➤ RMI calls cannot be captured (perhaps because an unsupported application container is being used).

➤ the application uses some other mechanism for communications between multiple VMs.

Note: If you configure the integration to monitor server requests this functionality imposes an additional overhead on the probe.

To investigate any issues that you have with the connections between the Diagnostics components, use the System Health Monitor accessible from Performance Center.

Managing Performance Center Offline Files

HP Performance Center offline files are kept by default. To manage offline files, you need to configure the Diagnostics Servers in Mediator mode so that they delete these files.

You do this by setting the property **distributor.offlinedelivery.preserveFiles** to true in **<diagnostics_server_install_dir>/etc/server.properties**. When set to true, this property causes the run-specific "offline" files stored in the server's data directory to be retained for the amount of time specified in the **facade.run_delete_delay property** in the server's **webserver.properties** file (default period is 5 days).

During this retention period, the run can be successfully collated. Sometime after the retention period has ended, the associated offline files will be deleted from the system.

Part IX

Appendixes

A

Diagnostics Server Administration Page

Information is provided on how to access the Diagnostics Server Administration page, where you can configure Diagnostics properties. Also information is provided on how to view the Configuration pages, and modify the properties within these pages.

This chapter includes:

➤ Accessing the Diagnostics Server Administration Page on page 583

➤ Configuring Diagnostics Using the Diagnostics Server Configuration Pages on page 585

Accessing the Diagnostics Server Administration Page

You can set the user privileges, configure Diagnostics settings, and open Diagnostics directly from the administration page of the Diagnostics Server.

To use the Diagnostics Server administration page:

1 Open the administration page by navigating to http://<diagnostics_server_host>:2006 in your browser, or by selecting **Start** > **Programs** > **Diagnostics Server** > **Administration.** The port number in the URL, **2006**, is the default port for the Diagnostics Server. If you configured the Diagnostics Server to use an alternative port, use that port number in the URL.

The Diagnostics Server administration page opens in your browser.

2 Click the option for the activity that you want to perform.

➤ **Open Diagnostics.** Opens the Diagnostics Web pages where you can view the performance metrics collected by the probes that are reporting to the Diagnostics Server. The performance metrics are displayed in the standard Diagnostics views.

For more information about opening Diagnostics directly from the Diagnostics Server in Commander mode, see the *HP Diagnostics User's Guide*.

➤ **Configure Diagnostics.** Opens the Components page, which has a link to the Diagnostics Server Configuration page.

For more information about configuring the Diagnostics properties, see "Configuring Diagnostics Using the Diagnostics Server Configuration Pages" on page 585.

➤ **Manage Authorization and Authentication.** Opens the User Administration page where you can add and maintain security information, and user privileges for specific users.

For more information about security and user privileges, see "User Authentication and Authorization" on page 593.

Configuring Diagnostics Using the Diagnostics Server Configuration Pages

In the Diagnostics Server Configuration pages, you set the property values that control how the Diagnostics Server communicates with the other Diagnostics components, and how it processes the data that it receives from the probes.

Note: To ensure that you are entering valid property values, it is recommended that you use the configuration pages to modify the Diagnostics Server properties, rather than editing the property files directly.

Accessing Diagnostics Server Configuration Pages

You access the Diagnostics Server Configuration pages from the Components page.

To access the Diagnostics Server configuration pages:

1 Open the administration page by navigating to http://<diagnostics_server_host>:2006 in your browser, or by selecting **Start** > **Programs** > **HP Diagnostics Server** > **Administration**. The port number in the URL, **2006**, is the default port for the Diagnostics Server. If you configured the Diagnostics Server to use an alternative port, use that port number in the URL.

The Diagnostics Server administration page opens in your browser.

2 Click **Configure Diagnostics**.

3 If you have not already signed into the Diagnostics Server, you are prompted for a user name and password. This must be a valid user name, and must have both **View** and **Change** privileges. For information about valid user names and privileges, see Appendix B, "User Authentication and Authorization."

Notes:

➤ Diagnostics continues to prompt for a user name and password until valid credentials are entered.

➤ If you click **Cancel**, the following error message is displayed in your browser: **Access denied. You must specify a valid user name and password.**

➤ If you entered a valid user name and password, but do not have the proper privileges, the following error message is displayed in your browser: **Access denied. You do not have the required permission to view this screen.**

To log on as a different user to the one you are currently logged on as, you need to close your browser and re-open it.

After you have logged on, the Diagnostics Server Components page opens:
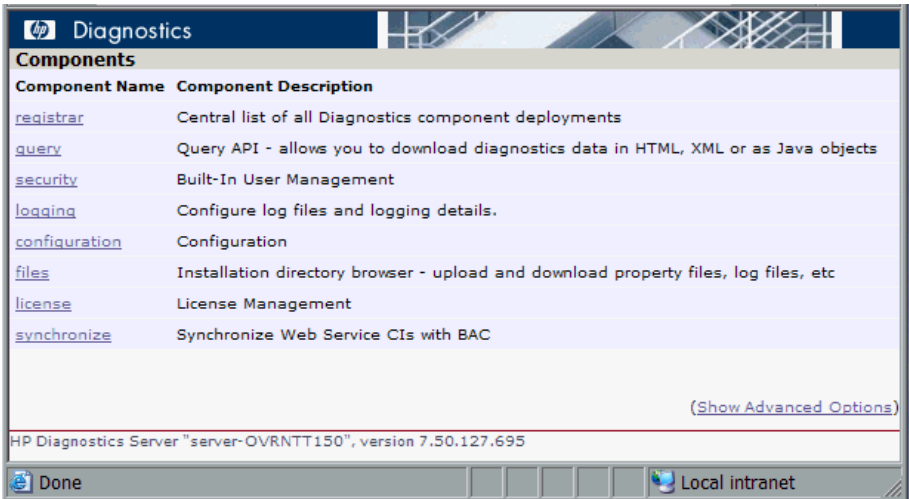
Click the link to the component page:

➤ **Registrar.** Central list of all Diagnostics component deployments.

➤ **Query.** Query API which allows you to download Diagnostics data in HTML, XML or as Java objects. An example is provided in the /contrib directory of the use of the Diagnostics Query API to create a custom dashboard.

➤ **Security.** Built-In User Management.

➤ **Logging.** Configure log files and logging details.

➤ **Configuration.** Configure the Diagnostics Server.

➤ **Files.** Installation directory browser for use in uploading and downloading property files, log files, etc.

➤ **License.** License management.

➤ **Synchronize.** Synchronize Web Service CIs with BAC. You can force a hard sync (perform full synchronization with Business Availability Center) or soft sync (synchronize only new CIs with Business Availability Center).

The components displayed on the Components page are the commonly used components. The more advanced components are hidden by default.

Important: Do not manipulate the advanced options without the guidance of your HP Software Customer Support representative.

To display the advanced options:

➤ At the bottom of the page, click **Show Advanced Options**.

The list of options on the page is updated to include the advanced configuration options, and the link changes to **Hide Advanced Options**.

Additional advanced configuration options are displayed

To hide the advanced options:

➤ At the bottom of the page, click **Hide Advanced Options**.

The list of options on the page is updated so that the advanced configuration options are no longer visible, and the link changes to **Show Advanced Options**.

Accessing the Configuration Component

From the Diagnostics Server Administration page, select Configure Diagnostics and from the Components page select **configuration** to access the Diagnostics Server Configuration page.
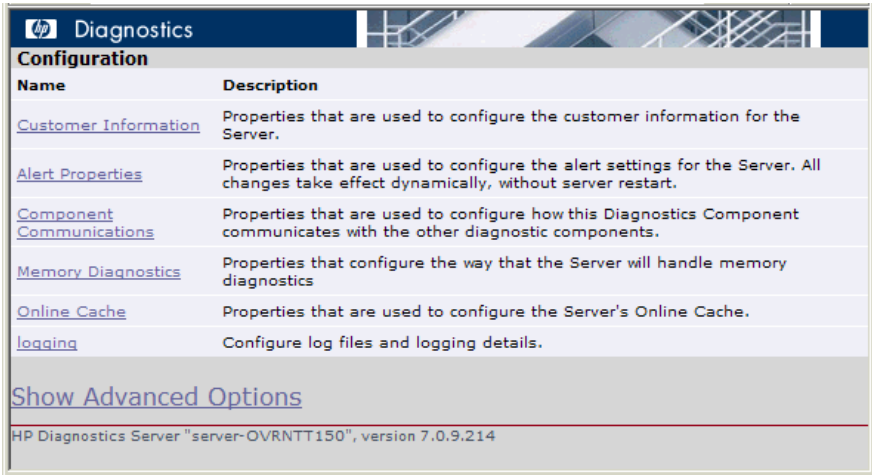
Click the link to the page whose properties you want to update. You can configure:

➤ Customer information

➤ Alert properties

➤ Component Communications

➤ Memory Diagnostics

➤ Online cache

➤ Logging

The configuration options displayed on the Configuration page are the commonly configured options. The more advanced configuration options are hidden by default.

Important: Do not manipulate the advanced options without the guidance of your HP Software Customer Support representative.

To display the advanced options:

➤ At the bottom of the page, click **Show Advanced Options**.

The list of options on the page is updated to include the advanced configuration options, and the link changes to **Hide Advanced Options**.

Additional advanced configuration options are displayed:

To hide the advanced options:

➤ At the bottom of the page, click **Hide Advanced Options**.

The list of options on the page is updated so that the advanced configuration options are no longer visible, and the link changes to **Show Advanced Options**.

Modifying Diagnostics Server Properties

You modify the properties of the Diagnostics Server from the Configuration page.

To modify the Diagnostics Server properties:

1 Access the Diagnostics Server Configuration page, as described in "Accessing Diagnostics Server Configuration Pages" on page 585.

2 Click the link to the properties (for example, **Component Communications**) that you want to update.

3 Review the properties that are displayed and make updates.

4 When you are satisfied with your changes, click **Submit** to save them. Click **Reset All** to reset ALL values back to the default settings or close the dialog if you do not want to submit any changes.

A message appears at the top of the page to indicate that your changes were saved.

Notes:

➤ For most properties that you update, a message is displayed reminding you to restart the Diagnostics Server. The property changes will not take effect until you restart the Diagnostics Server.

If you want make other changes to the Diagnostics Server properties, you should finish making all of your changes before restarting the Diagnostics Server.

Restarting the server will result in a small loss of data (up to 6 minutes). You should therefore schedule restarts at a time that is convenient.

➤ Modifying the logging level details does not require restarting the Diagnostics Server; however, it may take up to a minute for your changes to be applied.

B

User Authentication and Authorization

Information is provided on the Diagnostics authentication and authorization process and describes how to create and maintain user security permissions.

This chapter includes:

About User Authentication and Authorization

User authentication and authorization settings for all the Diagnostics components are configured in the Diagnostics Server in Commander mode.

Authentication is the process of verifying a person's identity. Authorization is the process of verifying that a known person has the authority (permission or privilege) to perform a certain action. Roles are bundles of permissions assigned to a user.

You manage authentication and authorization by creating and editing user names and granting the users privileges so that users are able to perform the functions within the application for which they are responsible.

User permissions and privileges for the Profilers (.NET Diagnostics Profiler or Java Diagnostics Profiler) of the probes connected to a particular Diagnostics Server are also defined in the Diagnostics Server in Commander mode. You can assign users one set of permissions for accessing Profilers in a particular probe group and a different set of permissions for accessing the Diagnostics Server.

You manage users and assign user privileges by selecting Manage Authorization and Authentication on the Diagnostics Server opening dialog to access the Permissions page.
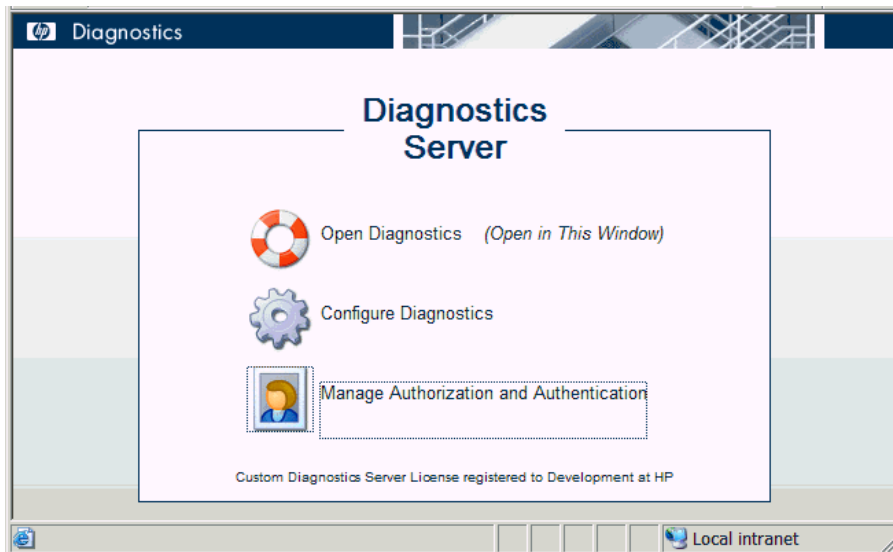
Important:

➤ When you install the probe as a profiler only (not connected to any Diagnostics Server), you manage the authentication and authorization of users of the Profiler in the probe itself.

➤ For information about managing authentication and authorization for the Java Probe installed as a profiler only, see "Authentication and Authorization for Java Diagnostics Profilers in Standalone Mode" on page 409.

➤ For information about managing authentication and authorization for the .NET Probe installed as a profiler only, see "Authentication and Authorization for .NET Profilers in Standalone Mode" on page 502.

Before you can view any Diagnostics data, or make any changes to the Diagnostics configuration or user privileges, you must log on to the Diagnostics Server in Commander mode using a user name that has valid security access with the appropriate privileges.

After logging on to the Diagnostics Server in a particular browser session, the user name remains in effect until the browser session ends. When you are finished using Diagnostics, be sure to close your browser to prevent others from accessing Diagnostics using your privileges.

Understanding User Privileges

The following privilege levels can be assigned to Diagnostics users:

Privilege	Description
View	The user can view Diagnostics data from the UI.
Execute	The user can make changes to the settings on the UI, such as changing thresholds or adding comments. On the Profiler, this privilege gives permission to perform garbage collection and clear the performance data held by the Profiler.
Change	The user can access the **Configure Diagnostics** menu to alter component configuration, and maintain user information. On the profiler, this gives permission to run potentially risky operations, such as taking a heap-dump or changing instrumentation.

Notes:

➤ The privilege levels, **rhttpout** and **system** are for internal purposes only. **rhttpout** is used to grant the user access to the rhttp/out URL for doing remote management of distributed servers.

➤ **system** is an internal permission generally granted only to the **mercury** special user. It is the permission that allows Diagnostics components to talk to one another (for example, the permission required for a probe to register with the Diagnostics Server). You also require **system** permission to view system health.

Each privilege level stands alone. There is no inheritance of privileges from one level to the next. You must grant a user all of the privilege levels that are necessary to perform the functions that they need to perform.

For example, a user must be granted both **View** and **Execute** privileges in order to be able to make changes to thresholds. A user name that has been granted only **Execute** privileges would not be useful, as it would not allow the user to see the UI on which they have permission to make changes.

For information about assigning privileges to users, see "Assigning Privileges Across the Diagnostics Deployment" on page 608.

Understanding Roles

In addition to the user/privilege assignment, it is also possible to assign privileges to roles and assign these roles to users. This makes the management of multiple users easier: when a new user is added to Diagnostics only the user/role assignment has to be performed. This is especially helpful when a user is setup to have different privileges for accessing the Diagnostics Server and the Profiler of a particular probe group.

Consider the following example:

Two development teams (Dev1 and Dev2) that require all permissions (view, execute, change) to the Profiler on the probe system that they own and view permission on the probe system that they don't own. Both teams should have view and execute permissions for the UI.

The following roles have to be created:

Role	Privileges
Enterprise (access to the UI)	[DevUI] = view,execute
Dev1 Probe Group	[Dev1All] = view,execute,change [Dev2View] = view
Dev2 Probe Group	[Dev2All] = view,execute,change [Dev1View] = view

Note that roles need to be enclosed in brackets in order to distinguish them from users. For example, if a new user to the Dev1 team is added to Diagnostics, it would need to be part of the following roles: [DevUI],[Dev1All],[Dev1View].

597

Accessing Diagnostics Using Default User Names

The following default user names have been defined for Diagnostics:

Default User Names	Privileges	Description
user	View	Can only view the data from the UI.
superuser	View, Execute	Can view data, change thresholds, and create alerts and comments from the UI.
admin	View, Change, Execute, System	Can view data, change thresholds, and create alerts and comments from the UI. Can configure components and maintain user information.

You can use these default user names to access Diagnostics functionality.

The passwords for the default user names are the same as the user names. For example, for the user name admin, the password is admin.

You can modify the password or privileges for the default user names to suit your needs. You can also define new user names in order to control user access to Diagnostics.

Important: There are two default users, **mercury** and **bac**, that are used for internal purposes and should never be modified. These users are for internal communication between components.

Understanding the Diagnostics Server Permissions Page

You manage users and assign user privileges in the Permissions page.

This section includes:

➤ "Accessing the Permissions Page" that follows.

➤ "The Permissions Page at a Glance" on page 601

➤ "Enterprise and Application Permissions" on page 602

Accessing the Permissions Page

In the Permissions page, you can create, edit, and delete Diagnostics users and manage the permissions for specific user names.

You can access the Permissions page from the Diagnostics Server Administration page or from the Maintenance page.

To access the Permissions page from the Diagnostics Server Administration page:

1 In your browser, open the Diagnostics Server Administration page by navigating to http://<commanding diagnostics_server_host>:2006/, or by selecting **Start** > **Programs** > **HP Diagnostics Server** > **Administration.**

The port number in the URL, **2006**, is the default port for the Diagnostics Server. If you configured the Diagnostics Server to use an alternative port, use that port number in the URL.

2 In the Diagnostics Server Administration page, click **Manage Authorization and Authentication** to open the Permissions page. This dialog allows you can create, edit, and delete Diagnostics users and manage the permissions for specific user names.

To access the Permissions page from the Diagnostics UI:

1 Click **Maintenance** at the top right-hand the Diagnostics Views window. The Maintenance page opens.

2 Click **security** to open the Permissions page.

When the Permissions page opens, if you have not already signed into the Diagnostics Server, you may be prompted for a user name and password. You must have at least **View** privileges in order to view your privileges, and modify your password. To add or delete users, or update user privileges, you must have both **View** and **Change** privileges.

Notes:

➤ Diagnostics continues to prompt for a user name and password until valid details are entered.

➤ If you click **Cancel**, the following error message is displayed in your browser: **Access denied. You must specify a valid username and password.**

➤ If you entered a valid user name and password, but do not have the proper privileges, the following error message is displayed in your browser: **Access denied. You do not have the required permission to view this screen.**

The Permissions Page at a Glance

The following screen is an example of the Diagnostics Server Permissions page:

The Permissions page is divided into the following three sections:

➤ **Enterprise Diagnostics Permissions.** In this section you manage Diagnostics users and you assign privileges across the whole Diagnostics deployment, including the Diagnostics Servers and probes.

By default, if users are authorized to access a particular Diagnostics Server, they also have the same authorization (and privileges) to access all probes connected to that server.

➤ **Control over probes connected to <commanding_Diagnostics_server>.** In this section you assign privileges for users accessing the probe Profilers. You can assign users one set of permissions for accessing Profilers in a particular probe group and a different set of permissions for accessing the Diagnostics Server.

➤ **Encrypt Internal Diagnostics Passwords.** You can access the EncryptPassword utility to encrypt a password.

Enterprise and Application Permissions

In addition to the enterprise and probe level permissions you set on the Permissions page, you can also set application level permissions. Application permissions are set in the Diagnostics UI in the initial Applications window. See the *HP Diagnostics User's Guide* for details on setting application permissions.

The three groups of permissions are as follows:

➤ Enterprise

➤ **View.** The user can look at performance data in the Diagnostics UI.

➤ **Execute.** The user can change thresholds and add comments and create applications.

➤ **Change.** The user has full administration access to the system (for example, can create users).

➤ Per Probe Group (applied in the Profiler)

➤ **View.** The user can view performance data collected by the Profiler.

➤ **Execute.** The user can run Garbage Collections and clear the performance data held by the Profiler.

➤ **Change.** The user can run operations such as taking a heap-dump or changing instrumentation.

➤ Application

➤ **View.** The user can view applications and edit entity properties (this requires Enterprise permissions set to Change.

➤ **Modify.** The user can delete, rename, modify applications, and can add or remove an entity from an application.

➤ **Edit Screens.** The user can edit using the Application Overview screen.

Note: Permissions are NOT inclusive (Execute does not include View).

Area and Action	Enterprise Permissions			Application Permissions		
	view	execute	change	view	edit	edit screens
Diagnostics UI						
View Diagnostics data in UI	X					
Change custom attributes		X				
Set thresholds in UI		X				
Create/Modify/Delete comments in UI		X				
Create alert rules in UI		X				
Configure Diagnostics page			X			
View system health **Note:** To view system health, you also require **system** enterprise permission.			X			
Manage authorization and authentication for other users			X			
Access maintenance page			X			
Working with incidents	X					
Working with customer views	X					

Area and Action	Enterprise Permissions			Application Permissions		
	view	execute	change	view	edit	edit screens
Profiler UI						
Perform garbage collection in Profiler		X				
Clear performance data in Profiler		X				
View Diagnostics data in Profiler	X					
Perform heap-dump (Memory & Allocation Analysis)			X			
Change configuration			X			
User defined applications						
Create application		X				
Delete application		X			X	
Rename application		X			X	
Change application path		X			X	
Modify application permissions		X			X	
Edit custom application screen	X					X
Add entity to application	X				X	
Remove entity from application	X				X	
Edit entity properties (thresholds, comments, etc)		X		X		
View application	X			X		

Area and Action	Enterprise Permissions			Application Permissions		
	view	execute	change	view	edit	edit screens
Auto discovered applications, transaction applications or Entire Enterprise						
Create, Delete and Change of application path is not allowed						
Modify application permissions		X			X	
Edit application screen	X					X
Add entity to application	X				X	
Remove entity from application	X				X	
Edit entity properties		X		X		
View application	X			X		

Creating, Editing and Deleting Users

Users with both **View** and **Change** privileges can create new users, edit the password for an existing user, or delete users. Users with only **View** privileges can maintain their own password.

To create a new user:

1 Access the Diagnostics Server Permissions page as described in "Accessing the Permissions Page" on page 599.

2 On the Permissions page, click **User Administration** to open the User Administration page.

3 On the User Administration page, click **Create User**.

4 In the **New User Name** box, type a user name for the new user and click **OK**. The new user appears in the list of user names.

Note: Username and password must contain English characters only due to Browser restrictions in handling basic authentication.

---

5 Under **Change Password**, in the **Password** box, type a password for the new user, and confirm it by retyping it in the **Confirm Password** box.

6 In the **Password for <current user>** box, type the password of the user currently logged on.

7 Optionally, assign the roles for this user. Make sure that the roles are enclosed in brackets (e.g. [aRole]). Roles can be separated by comma (e.g. [Role1],[Role2]).

Note: Permissions have to be set up for roles under the Enterprise and/or Per Probe Group dialogs (Assigning privileges across Diagnostics deployment and assigning privileges for probe groups).

---

8 Click **Save Changes**.

By default the new user, has **view** privileges. For information about changing the privileges assigned to the user, see "Assigning Privileges Across the Diagnostics Deployment" on page 608.

To delete a user:

1 Access the Diagnostics Server Permissions page as described in "Accessing the Permissions Page" on page 599.

2 On the Permissions page, click **User Administration** to open the User Administration page.

3 On the User Administration page, in the **Password for <current user>** box, type the password of the user currently logged on.

4 Click the red X (**Delete user)** button corresponding to the user you want to delete.

5 A message box opens asking if you want to delete the selected user.

Click **OK** to delete the user.

To change a user's password if you have View and Change privileges:

1 Access the Diagnostics Server Permissions page as described in "Accessing the Permissions Page" on page 599.

2 On the Permissions page, click **User Administration** to open the User Administration page.

3 On the User Administration page, in the row representing the relevant user, type the new password in the **Password** and **Confirm Password** boxes.

4 In the **Password for <current user>** box, type the password of the user currently logged on.

5 Click **Save Changes** to save all the changes you made to the different user names.

To change your own password if you only have View privileges:

1 Access the Diagnostics Server Permissions page as described in "Accessing the Permissions Page" on page 599.

2 On the Permissions page, click **User Administration** to open the User Administration page.

3 On the User Administration page, type the new password in the **Password** and **Confirm Password** boxes.

4 In the **Old Password** box, type your old password.

5 Click **Save Changes**.

Assigning Privileges Across the Diagnostics Deployment

Users with both **View** and **Change** privileges can grant users privileges across the entire Diagnostics deployment.

Note: For a description of the user privileges that you can assign to Diagnostics users, see "Understanding User Privileges" on page 596.

To assign user privileges across the entire Enterprise:

1 Access the Diagnostics Server Permissions page as described in "Accessing the Permissions Page" on page 599.

2 On the Permissions page, click **Edit Enterprise Permissions** to open the Editing Enterprise Permissions page.

The Editing Enterprise Permissions page is an editable page which allows you to modify user privileges.

3 Locate the name of the user, whose privileges you want to modify.

Important: You add users on the User Administration page, as described in "Creating, Editing and Deleting Users" on page 605.

---

4 Add the privileges to the username as comma separated values.

For example, if you have defined a user by the name of **newuser** and you want to assign this user with view and execute privileges you would locate **newuser** and edit the line so that it appears as follows:

```
newuser = view,execute
```

The Editing Enterprise Permissions page also includes a set of default users. These users are described in "Accessing Diagnostics Using Default User Names" on page 598. You can modify the privileges of these default users.

Assigning Privileges for Probe Groups

Users with both **View** and **Change** privileges can grant users privileges for accessing the probe Profilers belonging to particular probe groups.

By default, if users are authorized to access a particular Diagnostics Server, they also have the same authorization (and privileges) to access all probe Profilers connected to that server.

However, you can assign users a different set of permissions for different probe groups than what they have for the Diagnostics Servers themselves.

Note: For a description of the user privileges that you can assign to Diagnostics users, see "Understanding User Privileges" on page 596.

You can modify user privileges for each probe group individually and you can also modify a Permissions template that defines the user privilege settings for all future probe groups added to your system.

Note: User and permission settings may take up to one minute after the changes are saved to take effect.

For each probe group, there are three default user groups of users with certain privileges. You can choose to comment out these groups or to modify their privileges. The following groups of users are defined by default in all the probe groups:

User Group	Permissions
any_diagnostics_admin	This group refers to any user with administration (change) privileges on the Diagnostics Server. By default, any user who falls into this category and does not have any other predefined permission settings has administration permissions for all probes connected to that server.
any_diagnostics_superuser	This group refers to any user with superuser (execute) privileges on the Diagnostics Server. By default, any user who falls into this category and does not have any other predefined permission settings has execute permissions for all probes connected to that server.
any_diagnostics_user	This group refers to any user with user (view) privileges on the Diagnostics Server. By default, any user who falls into this category and does not have any other predefined permission settings has view permissions for all probes connected to that server.

To assign user privileges for accessing a particular probe group:

1 Access the Diagnostics Server Permissions page as described in "Accessing the Permissions Page" on page 599.

2 In the **Control over probes connected to <commanding_Diagnostics_ server>** section of the permissions page, click **Edit <name of probe group>**.

The Editing Permissions page opens. This is an editable page which allows you to modify user privileges.

3 Enter the username to which you want to assign unique privileges and add the privileges to the username as comma separated values.

For example, if you have defined a user by the name of **newuser** and you want to assign this user with view and execute privileges on this particular probe group, you enter the following line:

```
newuser = view,execute
```

To assign user privileges using the Permissions template:

1 Access the Diagnostics Server Permissions page as described in "Accessing the Permissions Page" on page 599.

2 In the **Control over probes connected to <commanding_Diagnostics_server>** section of the permissions page, click **Edit Permissions Template**.

The Editing Template Permissions page opens. This is an editable page which allows you to modify user privileges.

3 Enter the username to which you want to assign unique privileges and add the privileges to the username as comma separated values.

For example, if you have defined a user by the name of **newuser** and you want to assign this user with view and execute privileges on this particular probe group, you enter the following line:

```
newuser = view,execute
```

You could also modify or comment out one of the user group settings defined in the template.

Important: All future probe groups that are connected to your Diagnostics Server will inherit the user privilege settings from this Permissions template.

Authentication and Authorization for Users of Integrated HP Software Products

Diagnostics can be integrated with other HP Software applications (Business Availability Center, Performance Center, or LoadRunner). This section describes how authentication and authorization works for users of these integrated products and includes the following sections:

➤ "Authentication and Authorization for Business Availability Center Users" on page 612

➤ "Authentication and Authorization for Performance Center and LoadRunner Users" on page 613

Authentication and Authorization for Business Availability Center Users

In Business Availability Center, you can define user permissions for Diagnostics. For more information, see "Assigning Permissions for Diagnostics Users" on page 566.

When an existing or new Business Availability Center user opens Diagnostics from Business Availability Center, their permissions are picked up from the Business Availability Center session and copied into the Diagnostics permissions system (under the SaaS customer, if relevant).

Updates to Business Availability Center user permissions are only picked up when the user opens Diagnostics. (If Diagnostics is already open, changes will not be detected until it is closed and re-opened).

Business Availability Center passwords are never sent to Diagnostics— Diagnostics trusts a successful Business Availability Center login.

If privileges of Business Availability Center users change, the changes are not picked up until that user re-opens Diagnostics.

If a Business Availability Center user is deleted, it is recommended that you manually remove their permissions from Diagnostics. For more information, see "Creating, Editing and Deleting Users" on page 605.

Note: The Diagnostics Server can take up to five minutes to detect permission changes to users.

Authentication and Authorization for Performance Center and LoadRunner Users

When you set up both LoadRunner or Performance Center for integration with Diagnostics, you specify the Diagnostics Server details within LoadRunner / Performance Center. These details include the username and password with which you log on to HP Diagnostics.

When you access Diagnostics from LoadRunner or Performance Center, you are logged into Diagnostics with that same username and password that you specified during the integration setup.

Users accessing Diagnostics from within LoadRunner or Performance Center, will therefore have the privileges that are associated with the username that was specified during the integration setup.

Tracking User Administration Activity

Each time a user enters the Diagnostics Server User Administration page, all activity that takes place is logged in the following log file: **<diagnostics_server_install_dir>\log\useradmin.log**.

The data logged in the file includes the date and time of each action performed, a description of the action, and the name of the user performing the action.

To view the log file:

1 Open the Diagnostics Server administration page in one of the following ways:

➤ By selecting **Start** > **Programs** > **HP Diagnostics Server** > **Administration**.

➤ By navigating to http://<diagnostics_server_host>:2006 in your browser. The port number in the URL, **2006**, is the default port for the Diagnostics Server. If you configured the Diagnostics Server to use an alternative port, use that port number in the URL.

The Diagnostics Server administration page opens.

2 Click **Configure Diagnostics**.

3 If you have not already signed into the Diagnostics Server, you are prompted for a user name and password. This must be a valid user name, and must have both **View** and **Change** privileges. For information about valid user names and privileges, see "Understanding User Privileges" on page 596.

Notes:

➤ Diagnostics continues to prompt for a user name and password until valid credentials are entered.

➤ If you click **Cancel**, the following error message is displayed in your browser: **Access denied. You must specify a valid user name and password.**

➤ If you entered a valid user name and password, but do not have the proper privileges, the following error message is displayed in your browser: **Access denied. You do not have the required permission to view this screen.**

The Diagnostics Server Components page opens.

4 Click **logging**. The logging page opens.

5 Click **View Log Files**. A list of log files appears.

6 Click the **<diagnostics_server_install_dir>\log\useradmin.log** link.

The log file is displayed at the bottom the page.

Configuring Diagnostics to use JAAS

Diagnostics can be configured to use JAAS (Java Authentication and Authorization Service) for authentication of users. If JAAS is enabled, the user name and password entered in the login dialog when the UI is accessed is authenticated by a configured JAAS pluggable authentication module (LoginModule).

Note: JAAS support is only available on the Diagnostics server in commanding mode.

JAAS must be enabled in the **<INSTALL_DIR>/etc/server.properties** file by un-commenting the following two lines:

```
authentication.jaas.config.file=jaas.configuration
authentication.jaas.realm=Diagnostics
```

The **authentication.jaas.config.file** property specifies the configuration file (relative to the etc directory) that defines the LoginModules and **authentication.jaas.realm** specifies the entry that should be used in the configuration file.

Example jaas.configuration:

```
Diagnostics
{
 com.mercury.diagnostics.server.jaas.spi.SiteMinderLoginModule sufficient
       ip="1.2.3.4";

 com.mercury.diagnostics.server.jaas.spi.LDAPLoginModule sufficient
   useSSL="true"
   serverCertificate="etc/ldap.keystore"
   providerURL="ldap://ldap.yourdomain.com:636"
   baseDN="ou=People,o=yourdomain.com";

};
```

For more information on the JAAS configuration file, see the javax.security.auth.login.Configuration documentation.

Note: The users that were created by Diagnostics through the Manage Authorization and Authentication web page are used *first* when authenticating a username and password. Only if that authentication fails will the JAAS authentication be performed.

Diagnostics provides the following LoginModules:

➤ **LDAP.** (com.mercury.diagnostics.server.jaas.spi.LDAPLoginModule) which allows authentication against an LDAP server.

➤ **SiteMinder.** (com.mercury.diagnostics.server.jaas.spi.SiteMinderLoginModule) which allows authentication against a SiteMinder environment.

Notes:

➤ After making any changes to server.properties and/or the jaas.configuration file, you need to re-start the commanding server.

➤ When using a JAAS authentication provider that is also used in other applications (e.g. LDAP), it is recommended to turn on HTTPS for accessing the Diagnostics UI.

➤ When using a JAAS authentication provider, user accounts are maintained by the authenticating source. Ask your administrator (e.g., LDAP for details on user name syntax.

➤ Subsequent authorization of authenticated users privileges is maintained using the permissions page. Roles can also be used if the appropriate LoginModule is configured to use them. In this case, existing roles can be used or new roles can be created.

Configuring LDAP Authentication

To configure LDAP authentication in Diagnostics you must first configure Diagnostics to use JAAS (see "Configuring Diagnostics to use JAAS" on page 615) and then configure the following on the commanding server.

1 Edit the Diagnostics JAAS realm (application) block in **<INSTALL_DIR>/etc/jaas.configuration** with option values specific to your LDAP server. An example configuration is shown below:

```
Diagnostics
{
 com.mercury.diagnostics.server.jaas.spi.LDAPLoginModule required
   providerURL="ldap://ldap.yourdomain.com:636"
   baseDN="ou=People,o=yourdomain.com"
   useSSL="true"
   serverCertificate="etc/ldap.keystore";
};
```

The following is a complete list of the options that can be specified for the LDAP LoginModule in the JAAS configuration file:

Option Name	Description	Required /Optional	Default Value	Example
providerURL	URL to the LDAP server. Used to perform the JNDI lookup.	required		providerURL="ldap:// ldap.yourdomain. com:636"
baseDN	Partial string representation of the DN (Distinguished Name) LDAP entry corresponding to the user to be authenticated. The uidAttribute option will be used to complete the DN. So assuming the baseDN in the Example column and the default uidAttribute, for user "testuser", the resulting DN submitted to LDAP would be "uid=testuser,ou=People,o =hp.com"	required		baseDN="ou=Peo ple,o=yourdomai n.com"
uidAttribute	The LDAP attribute name used in conjunction with baseDN to complete the DN which will uniquely identify the user entry in the LDAP server. The user name submitted during login will become the value for this attribute.	optional	uid	uidAttribute= "employeeNumbe r"
useSSL	Whether or not connecting to the LDAP server requires SSL.	optional	false	useSSL="true"

Option Name	Description	Required /Optional	Default Value	Example
serverCertificate	Applicable when useSSL is set to true. Path to the trust store file containing the LDAP server's certificate. Path can be absolute or relative to the server installation home. Use forward slashes irrespective of platform.	optional	<INSTALL_DIR> /etc/ ldap.keystore	serverCertificate= "etc/ myldap.keystore"
defaultRoles	Comma-delimited list of roles to assign each authenticated user.	optional		defaultRoles= "SuperUsers"
roleAttributes	Comma-delimited list of the user's DN attributes whose values will be used as the user's roles. If defaultRoles is also set, the resulting roles will be the union of defaultRoles and roleAttributes.	optional	roles	roleAttributes= "employeeType, hpJobFunction"
allowAnonymous	Whether or not the LDAP server allows anonymous logins. If set to true and a password is not provided, the login module will attempt to connect to the LDAP server rather than reject the login attempt.	optional	false	allowAnonymous = "true"

Note: After making any changes to server.properties or the jaas.configuration file, you need to re-start the Diagnostics commanding server.

Using Reverse Proxy with SiteMinder JAAS LoginModule

The SiteMinder JAAS LoginModule requires a reverse proxy server in which the SiteMinder web agent is installed. A proxy server simply forwards HTTP/S requests to the Diagnostics server.

Example set-up:
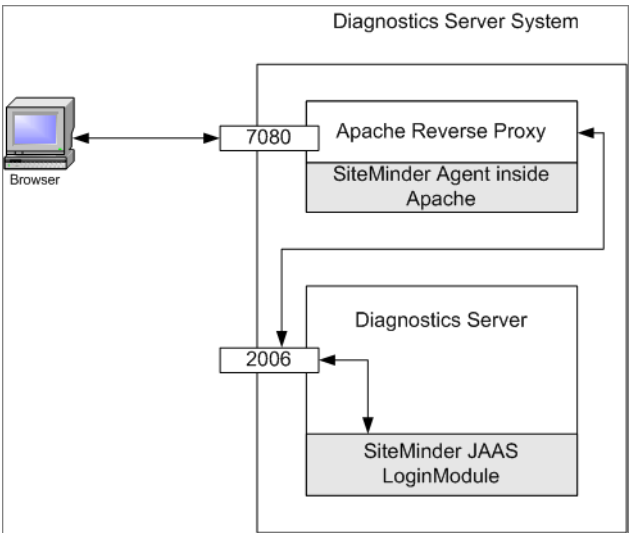
In the above diagram, an Apache web server listening for requests on port 7080 is configured as a reverse proxy. It contains the SiteMinder web agent which performs the authentication and if successful, allows Apache to pass the request through to port 2006 (or whatever Diagnostics Server port is configured) on which the Diagnostics server listens.

Note: It is recommended that the login page is served up by another web server (different web server than the reverse proxy) in order to avoid conflicts with the redirect that the reverse proxy performs and the redirect that the SiteMinder module performs.
Alternatively, with Apache 2.2, the ProxyPass directive can be used to suppress proxying for certain URLs (for example, "ProxyPass /loginpage !". See the Apache 2.2 documentation for more information.

The Diagnostics Server detects requests from SiteMinder via the SiteMinder LoginModule.

Note: To use the SiteMinder JAAS authentication the users must go to the port of the reverse proxy, 7080 in this example, instead of port 2006. If the proxy server is not installed on the same system as the Diagnostics Server, then the computer name for the proxy server must be used in the URL instead of the computer name of the Diagnostics server or localhost.

Example of Apache reverse proxy setup on HP-UX: edit the Apache configuration file **httpd.conf** and add the following properties:

➤ ProxyPass / http://<IP-address of Diagnostics Server>:2006/

(2006 is the default Diagnostics Server port, use the port configured for your Diagnostics Server)

➤ ProxyPassReverse / http://<IP-address of Diagnostics Server>:2006/

(2006 is the default Diagnostics Server port, use the port configured for your Diagnostics Server)

Note: After making any changes to the httpd.conf file, you need to re-start the Apache server (apachectl stop and apachectl start).

You can do the following as an optional step to provide additional security when you are concerned about spoofing:

➤ If the proxy server is not installed on the same system as the Diagnostics server, you can place the Diagnostics Server and the proxy server on the same subnet and configure an ingress filter for the proxy IP-address on the switch/router in order to prevent spoofing of the reverse proxy's IP-address from outside of the subnet.

See the diagram below:
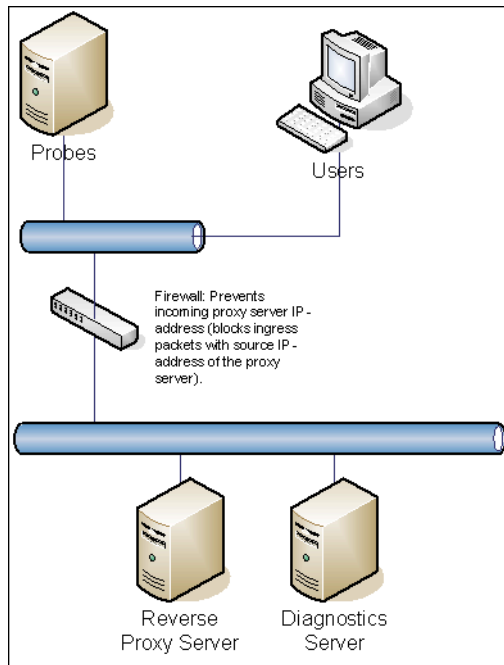
Configuring SiteMinder JAAS Authentication

To configure SiteMinder authentication in Diagnostics you must configure the following on the commanding server:

1 Configure Diagnostics to use JAAS (see "Configuring Diagnostics to use JAAS" on page 615).

2 Edit the **<INSTALL_DIR>/etc/webserver.properties** file.

> **a** Uncomment the **authentication.header.filter.username** property. Set the **authentication.header.filter.username** property to a field in the HTTP request header that should be used to get the username. By default this is set to **SM_UNIVERSALID** which is a field that SiteMinder creates in the HTTP request containing a user ID.

> **b** To use the Diagnostics roles, uncomment the **authentication.header. filter.roles** property (this is an optional step). Set the **authentication.header. filter.roles** property to a field in the HTTP header that should be used to get role information. This field can contain one role or many roles with commas separating them. If defaultRoles is also set, the resulting roles will be the union of defaultRoles and these roles.

3 Edit the Diagnostics JAAS realm (application) block in the **<INSTALL_DIR>/ etc/jaas.configuration** file.

An example is shown below:

```
Diagnostics
{
    com.mercury.diagnostics.server.jaas.spi.SiteMinderLoginModule sufficient
    defaultRoles="Role1,Role2"
     ip="16.228.25.40";
};
```

SiteMinder LoginModule Options

The following is a complete list of the options that can be specified for the SiteMinder LoginModule in the JAAS configuration file:

Option Name	Description	Required /Optional	Default Value	Example
IP	IP address of the reverse proxy server	required		ip="16.228.25.40"
defaultRoles	Comma-delimited list of roles to assign each authenticated user.	optional		defaultRoles= "SuperUsers"

Note: After making any changes to server.properties, webserver.properties or the jaas.configuration file, you need to re-start the Diagnostics commanding server.

Configuring Lightweight Single Sign-On (SSO) Security

Diagnostics is a consumer of lightweight single sign-on (SSO) security.

To avoid having to login again when drilling down from TransactionVision to Diagnostics, set the following property:

lwsso.init.string in **<diagnostics_server_install_dir>/etc/security.properties**

This property must be set to the same value that is specified as the initString attribute of the **crypto** element in the **<TVISION_HOME>/config/ui/ lwssofmconf.xml file**. For example:

```
<crypto cipherType="symmetricBlockCipher"
engineName="AES"
paddingModeName="CBC" keySize="256"
encodingMode="Base64Url" initString="This is a shared secret passphrase" />
```

This section describes:

About Lightweight SSO

The following points summarize the key behaviors of lightweight SSO (LW-SSO).

➤ Confidential InitString parameter in LW-SSO

Lightweight SSO uses Symmetric Encryption to validate and create a token. The initString parameter within the configuration is used for initialization of the secret key. An application creates a token, and each application that uses the same initString parameter validates the token.

   ➤ It is not possible to use LW-SSO without setting the initString parameter.

   ➤ The initString parameter is confidential information and should be treated as such in terms of publishing, transporting, and persistency.

   ➤ The initString should be shared only between applications integrating with each other using LW-SSO.

➤ LW-SSO should be disabled unless it is specifically required.

➤ The weakest authentication framework used by the LW-SSO integrated applications determines the level of authentication security for all of the applications. It is recommended that only applications using strong and secure authentication frameworks issue an LW-SSO token (see configuration section).

➤ Symmetric encryption implication

LW-SSO uses symmetric cryptography for issuing and validating LW-SSO Tokens. Therefore, any application using LW-SSO can issue a token to be trusted by all other applications sharing the same initString.

This potential risk is relevant when an application sharing initString either resides or is accessible in an untrusted location.

➤ User mapping (Synchronization)

LW-SSO framework does not ensure user mapping between the integrated applications. Therefore, the integrated application must monitor user mapping. It is recommended that you share the same user registry (as LDAP/ AD) among all integrated applications.

Failure to map users may cause security breaches and negative application behavior. For example, the same user name may be assigned to different real users in the various applications.

➤ Identity Manager is used for an authentication.

All unprotected resources in the Identity Manager must be configured as nonsecureURLs setting in the LW-SSO configuration.

Using Lightweight Single Sign-On

The following sections describe the guidelines for using lightweight SSO.

➤ The client must access the application with the Fully Qualified Domain Name (FQDN) in the login URL, e.g. http://flood.mercury.global:8080/WebApp

LW - SSO cannot support URLs with an IP Address, e.g. http://16.59.45.143:8080/WebApp

LW - SSO cannot support URLs without a domain, e.g. http://flood:8080/WebApp

➤ Applications can leverage and use LW-SSO capabilities only if integrated within the LW-SSO framework in advance.

➤ Multi domain support limitations:

➤ Multi domain functionality is based on the HTTP referrer. Therefore, LW-SSO supports links from one application to another and does not support typing a URL into a browser window, except when both applications are in the same domain.

➤ First cross domain link using HTTP POST is not supported:

Multi domain functionality does not support the first HTTP POST request to a second application (only the HTTP GET request is supported). For example, if your application has an HTTP link to a second application, an HTTP GET request is supported, but an HTTP FORM request is not supported. All requests after the first can be either HTTP POST or HTTP GET.

➤ LW-SSO Token size limitation:

The size of information that LW-SSO can transfer from one application in one domain to another application on another domain is limited to 15 Groups/Roles/Attributes (note that each item may be an average of 15 characters long).

➤ Linking from Protected (HTTPS) to Non protected (HTTP) in a Multi domain scenario:

Multi domain functionality does not work when linking from a protected (HTTPS) to a non protected (HTTP) page. This is a browser limitation where the referrer header is not sent when linking from a protected to a non-protected resource. For an example, see: http://support.microsoft.com/support/kb/articles/Q178/0/66.ASP

➤ "Third-Party" cookies behavior in Internet Explorer:

In IE6, Microsoft added a module that supports the "Platform for Privacy Preferences(P3P) Project", which means that cookies coming from a "Third Party" domain are by default blocked in the "Internet" security zone. This means that session cookies are considered "Third party" cookies by IE as well, and therefore are blocked, causing LW - SSO to stop working. For details, see: http://support.microsoft.com/kb/323752/en-us. A possible solution is to add the launched application (or a DNS domain subset as *.mydomian.com) to the "Intranet"/"Trusted" zone on your computer (IE Menu -> Tools -> Internet Options -> Security -> Local Intranet -> Sites -> Advanced), which causes the cookies to be accepted

---

Important: Important: The LW-SSO session cookie is only one of the cookies used by the "Third party" application that are blocked.

➤ JAAS Realm in Tomcat is not supported.

➤ Using spaces in Tomcat directories is not supported.

It is not possible to use LW-SSO when a Tomcat installation path (folders) includes spaces (e.g. Program Files) and the LW-SSO configuration file is located in the common\classes Tomcat folder.

➤ Load balancer configuration.

A load balancer deployed with LW-SSO must be configured to use sticky sessions.

➤ Reverse proxy configuration.

LW-SSO can support multiple symmetric reverse proxy virtual nodes. LW-SSO can support only one asymmetric reverse proxy virtual node.

The following reverse proxy configurations are supported by LW-SSO.

Example1:

```
ProxyPass /App1 http://APP_server/App1
ProxyPass /App2 http://APP_server/App2
...
ProxyPass /AppN http://APP_server/App2N
```

Example 2:

```
ProxyPass Node1/App1 http://APP_server/App1
ProxyPass Node1/App2 http://APP_server/App2
...
ProxyPass Node1/AppN http://APP_server/AppN
```

The following reverse proxy configuration is not supported by LW-SSO.

```
ProxyPass Node1/App1 http://APP_server/App1
ProxyPass Node2/App2 http://APP_server/App2
...
ProxyPass NodeN/AppN http://APP_server/AppN
```

Important: Recommended configuration of the LW-SSO Token expiration:

Each application using LW-SSO should configure token expiration. The recommended value is 60 minutes. For an application that does not require a high level of security, it is possible to configure a value of 300 minutes.

Important: All applications participate in an LW - SSO integration must use the same GMT time with a maximum difference of 15 minutes.

Important: Multi domain functionality requires that all applications participating in LW-SSO integration configure the protectedDomains settings, if they are required to integrate with applications in different DNS domains. In addition, they must also add the correct domain in the lwsso element of the configuration.

Important: Normalized URL Parameter functionality: To receive information sent as Normalized URL Parameters from other applications, the host application should configure the correct domain in the lwsso element of the configuration.

Lightweight Single Sign-On System Requirements

Tool	Version
Java:	1.5 and higher
HTTP Servlets API	2.1 and higher
Internet Explorer	6.0 and higher Browser should enable HTTP session cookie and HTTP 302 Redirect functionality
FireFox	2.0 and higher Browser should enable HTTP session cookie and HTTP 302 Redirect functionality

Tool	Version
Tomcat authentications	Standalone Tomcat 5.5.28 Standalone Tomcat 5.5.20
Acegi authentications	Acegi 0.9.0 Acegi 1.0.4
Web Services engines	Axis 1 - 1.4 Axis 2 - 1.2 JAX-WS-RI 2.1.1

C

Enabling HTTPS Between Diagnostics Components

Information is provided on the configuration steps to enable HTTPS communications between the HP Diagnostics components.

This chapter includes:

➤ About Configuring HTTPS Communications on page 632

➤ Enabling Incoming HTTPS Communication for Diagnostics Components on page 633

➤ Enabling Outgoing HTTPS Communication from Diagnostics Components on page 642

➤ Enabling HTTPS Communications for the Business Availability Center Server on page 649

➤ HTTPS Checklist per Diagnostics Component on page 651

Note: The configuration instructions are intended for experienced users with in-depth knowledge of HP Diagnostics. Use caution when modifying any configuration settings for the Diagnostics components.

About Configuring HTTPS Communications

The instructions for configuring each type of component contain details of the following main steps:

➤ Generate a keystore on the component

➤ Export the certificate from the keystore

➤ Obfuscate passwords that provide access to the keystore

➤ Copy the component's certificate to the Diagnostics components that will initiate communication

➤ Configure the component's security properties to enable SSL and provide the passwords necessary for HTTPS communication to take place

---

Note: As you review this information it will be useful to reference the Component Communication Diagram in Appendix F, "Diagnostics Component Configuration and Communication Diagrams."

Enabling Incoming HTTPS Communication for Diagnostics Components

This section includes instructions for configuring the Diagnostics Server, the Java Probe and the Collector to receive incoming HTTPS communications. The HTTPS communications can come from other Diagnostics components, from when the Diagnostics component is accessed using a Web browser, or when the component is accessed by other external applications.

This section includes the following topics:

➤ "Configuring the Diagnostics Server for Incoming HTTPS Connections" on page 633

➤ "Configuring the Java Probe for Incoming HTTPS Connections" on page 636

➤ "Configuring the Collector for Incoming HTTPS Connections" on page 639

Configuring the Diagnostics Server for Incoming HTTPS Connections

Notes:

➤ To avoid issues with DNS and host name resolution, the Commander URL for the Diagnostics Server in Commander mode should be configured as **localhost**. This can be accomplished by setting the commander.url property in **<server_install_dir>/etc/server.properties** to http://localhost:2006 (or the appropriate port number).

➤ When you enable HTTPS communications with a commanding server, you will need to use port 8443 to run the Enterprise UI: **https://commanding_server:8443**

To configure the Diagnostics Server for incoming HTTPS connections:

1 Generate a keystore in the **<diagnostics_server_install_dir>/etc** directory using the following command:

```
<diagnostics_server_install_dir>/_jvm/bin/keytool -genkey -keystore
<diagnostics_server_install_dir>/etc/keystore -storepass <password> -alias SERVER
-keyalg RSA -keypass <password> -dname "CN=<diagnostics_server_hostname>,
OU=Diagnostics, O=Hewlett-Packard, L=Palo Alto, S=CA, C=USA" -validity 3650
```

To use this command example:

➤ Replace **<diagnostics_server_install_dir>** with the path to the installation directory for the Diagnostics Server.

➤ Replace **<diagnostics_server_hostname>** with the machine name for the host of the Diagnostics Server (you should use the fully qualified domain name for the subject (CN) in the certificate).

➤ Replace each occurrence of **<password>** with the same password string. If you want, you may assign different passwords to the **storepass** and the **keypass**.

After you execute this command, a keystore is created in **<diagnostics_server_install_dir>/etc/keystore** with an entry called **SERVER** for the host of the Diagnostics Server.

2 Export the certificate for the SERVER entry in the keystore using the following command.

```
<diagnostics_server_install_dir>/_jvm/bin/keytool -export -keystore
<diagnostics_server_install_dir>/etc/keystore -storepass <password> -alias
SERVER -rfc -file <diagnostics_server_install_dir>/etc/
<server_certificate_name>.cer
```

To use this command:

➤ Replace **<diagnostics_server_install_dir>** with the path to the installation directory for the Diagnostics Server.

➤ Replace **<password>** with the string that you assigned as the **storepass** password when you created the keystore.

➤ Replace **<server_certificate_name>** with the name that you would like to assign to the certificate file. It is recommended that you assign a certificate name that will make it easy to recognize the component for which the certificate was created.

Use **diag_server_commander.cer** or **diag_server_mediator.cer**.

After this command runs, a certificate file with the name assigned in **<server_certificate_name>** is created in the **<diagnostics_server_install_dir>**/ **etc** directory for the Diagnostics Server, for example, **diag_server_commander.cer**.

Note: The certificate file must be imported to the host machines for each of the Diagnostics components that are expected to initiate communications with the Diagnostics Server. The instructions for importing the certificate file to each Diagnostics component are provided below.

---

3 Using the command in the following example, generate an obfuscated version of the **storepass** and the **keypass** passwords that you assigned when you created the keystore.

a Replace **<diagnostics_server_install_dir>** with the path to the installation directory for the Diagnostics Server.

b Replace **<password>** with the string that you assigned as the password when you created the keystore.

```
<diagnostics_server_install_dir>/_jvm/bin/java -cp <diagnostics_server_install_dir>/
lib/ThirdPartyLibs.jar org.mortbay.util.Password <password>
```

The output from the obfuscation is shown in the following example. In this example, the password string was "testpass". The output consists of three lines. The original string that was to be obfuscated and two lines depicting the obfuscated password. Only the line that begins with "OBF" is used to set the properties in the following step of this process.

```
testpass
OBF:1ytc1vu91v2p1y831y7v1v1p1vv11yta
MD5:179ad45c6ce2cb97cf1029e212046e81
```

> **Note:** If you did not use the same password for **keypass** and **storepass**, you must run this command twice to create an obfuscated version for each password.

4 Change the following properties in the file **<diagnostics_server_install_dir>/ etc/security.properties** for the Diagnostics Server in Commander mode.

   a Set **enableSSL=true**.

   b Set **keyStorePassword=<obfuscated_password>**

   c Set **keyPassword=<obfuscated_password>**

   Note: The value entered for **<obfuscated_password>** must include the entire "OBF" line that was output from the command in the previous step. For example:

   keyStorePassword=OBF:1ytc1vu91v2p1y831y7v1v1p1vv11yta

Configuring the Java Probe for Incoming HTTPS Connections

Notes:

➤ Enabling SSL and HTTPS Communications for the Java Probe is supported on SUTs with the Sun, IBM, and JRockit JVMs. However, if you are using a JVM version prior to 1.4, you must download and install the Sun JSSE Optional Package onto the SUT server before you can enable SSL.

Other JSSE implementation, such as IBM's are not supported.

➤ When you enable HTTPS communications with a Java probe system, you will need to use port 45000 to run the Probe Profiler UI: **https:// probe_system:45000**

To configure the Java Probe for incoming HTTPS connections:

1 Generate a keystore in the **<probe_install_dir>/etc** directory using the following command:

```
<probe_install_dir>/_jvm/bin/keytool -genkey -keystore <probe_install_dir>/etc/
keystore -storepass <password> -alias PROBE -keyalg RSA -keypass <password>
-dname "CN=<probe_hostname>, OU=Diagnostics, O=Hewlett-Packard, L=Palo Alto,
S=CA, C=USA" -validity 3650
```

To use this command example:

➤ Replace **<probe_install_dir>** with the path to the installation directory for the Java Probe.

➤ Replace **<probe_hostname>** with the machine name for the host of the Java Probe. This value cannot be the server's IP address. You should use the fully qualified domain name for the subject (CN) in the certificate.

➤ Replace each occurrence of **<password>** with the same password string. If you want, you may assign different passwords to the **storepass** and the **keypass**.

After you execute this command, a keystore is created in **<probe_install_dir>/etc/keystore** with an entry called **PROBE** for the host of the Java Probe.

2 Export the certificate for the PROBE entry in the keystore using the following command.

```
<probe_install_dir>/_jvm/bin/keytool -export -keystore <probe_install_dir>/etc/
keystore -storepass <password> -alias PROBE -rfc -file <probe_install_dir>/etc/
<probe_certificate_name>.cer
```

To use this command:

➤ Replace **<probe_install_dir>** with the path to the installation directory for the Java Probe.

➤ Replace **<password>** with the string that you assigned as the **storepass** password when you created the keystore.

➤ Replace **<probe_certificate_name>** with the name that you would like to assign to the certificate file. It is recommended that you assign a certificate name that will make it easy to recognize the component for which the certificate was created.

Include the type of the probe and the host name for the probe so that it will be easy to recognize the component for which the certificate was created. For example: **Java_probe_<probe_hostname>.**

After this command runs, a certificate file called **Java_probe_<probe_hostname>.cer** is created in the **<probe_install_dir>/etc** directory for the Java Probe.

Note: The certificate file must be imported to the host machines for each of the Diagnostics components that are expected to initiate communications with the Java Probe. The instructions for importing the certificate file to each Diagnostics component are provided below.

---

3 Using the command in the following example, generate an obfuscated version of the **storepass** and the **keypass** passwords that you assigned when you created the keystore.

a Replace **<probe_install_dir>** with the path to the installation directory for the Java Probe.

b Replace **<password>** with the string that you assigned as the password when you created the keystore.

```
<probe_install_dir>/_jvm/bin/java -cp
<probe_install_dir>/lib/ThirdPartyLibs.jar org.mortbay.util.Password <password>
```

The output from the obfuscation is shown in the following example. In this example, the password string was "testpass". The output consists of three lines. The original string that was to be obfuscated and two lines depicting the obfuscated password. Only the line that begins with "OBF" is used to set the properties in the following step of this process.

```
testpass
OBF:1ytc1vu91v2p1y831y7v1v1p1vv11yta
MD5:179ad45c6ce2cb97cf1029e212046e81
```

Note: If you did not use the same password for **keypass** and **storepass**, you must run this command twice to create an obfuscated version for each password.

4 Change the following properties in the file **<probe_install_dir>/etc/security.properties**.

   a Set **enableSSL=true**.

   b Set **keyStorePassword=<obfuscated_password>**

   c Set **keyPassword=<obfuscated_password>**

   Note: The value entered for **<obfuscated_password>** must include the entire "OBF" line that was output from the command in the previous step. For example:

   ```
   keyStorePassword=OBF:1ytc1vu91v2p1y831y7v1v1p1vv11yta
   ```

Configuring the Collector for Incoming HTTPS Connections

This section provides instructions for configuring the Collector to receive incoming HTTPS connections

To configure the Collector for incoming HTTPS connections:

1 Generate a keystore in the **<collector_install_dir>/etc** directory using the following command:

```
<collector_install_dir>/_jvm/bin/keytool -genkey -keystore <collector_install_dir>/etc/
keystore -storepass <password> -alias COLLECTOR -keyalg RSA -keypass
<password> -dname "CN=<collector_hostname>, OU=Diagnostics,
O=Hewlett-Packard, L=Palo Alto, S=CA, C=USA" -validity 3650
```

To use this command example:

➤ Replace **<collector_install_dir>** with the path to the installation directory for the Collector.

➤ Replace <**collector_hostname>** with the machine name for the host of the Collector. This value cannot be the server's IP address. You should use the fully qualified domain name for the subject (CN) in the certificate.

➤ Replace each occurrence of **<password>** with the same password string. If you want, you may assign different passwords to the **storepass** and the **keypass**.

After you execute this command, a keystore is created in **<collector_install_dir>/etc/keystore** with an entry called **COLLECTOR** for the host of the Collector.

2 Export the certificate for the COLLECTOR entry in the keystore using the following command.

```
<collector_install_dir>/_jvm/bin/keytool -export -keystore <collector_install_dir>/etc/
keystore -storepass <password> -alias COLLECTOR -rfc -file
<collector_install_dir>/etc/<collector_certificate_name>.cer
```

To use this command:

➤ Replace **<collector_install_dir>** with the path to the installation directory for the Collector.

➤ Replace **<password>** with the string that you assigned as the **storepass** password when you created the keystore.

➤ Replace **<collector_certificate_name>** with the name that you would like to assign to the certificate file. It is recommended that you assign a certificate name that will make it easy to recognize the component for which the certificate was created.

Include the type of the collector and the host name for the collector so that it will be easy to recognize the component for which the certificate was created. For example: **collector_<collector_hostname>.**

After this command runs, a certificate file called **collector_<collector_hostname>.cer** is created in the **<collector_install_dir>/ etc** directory for the Collector.

Note: The certificate file must be imported to the host machines for each of the Diagnostics components that are expected to initiate communications with the Collector. The instructions for importing the certificate file to each Diagnostics component are provided below.

---

3 Using the command in the following example, generate an obfuscated version of the **storepass** and the **keypass** passwords that you assigned when you created the keystore.

  a Replace **<collector_install_dir>** with the path to the installation directory for the Collector.

  b Replace **<password>** with the string that you assigned as the password when you created the keystore.

```
<collector_install_dir>/_jvm/bin/java -cp
<collector_install_dir>/lib/ThirdPartyLibs.jar org.mortbay.util.Password <password>
```

The output from the obfuscation is shown in the following example. In this example, the password string was testpass. The output consists of three lines. The original string that was to be obfuscated and two lines depicting the obfuscated password. Only the line that begins with "OBF" is used to set the properties in the following step of this process.

```
testpass
OBF:1ytc1vu91v2p1y831y7v1v1p1vv11yta
MD5:179ad45c6ce2cb97cf1029e212046e81
```

Note: If you did not use the same password for **keypass** and **storepass**, you must run this command twice to create an obfuscated version for each password.

4 Change the following properties in the file **<collector_install_dir>/etc/ security.properties**.

   a Set **enableSSL=true**.

   b Set **keyStorePassword=<obfuscated_password>**

   c Set **keyPassword=<obfuscated_password>**

   The value entered for **<obfuscated_password>** must include the entire "OBF" line that was output from the command in the previous step. For example:

   keyStorePassword=OBF:1ytc1vu91v2p1y831y7v1v1p1vv11yta

Enabling Outgoing HTTPS Communication from Diagnostics Components

The following instructions provide you with the steps necessary to configure the Diagnostics components to send outgoing HTTPS communications to other Diagnostics components.

To enable the Diagnostics Server in Commander mode for outgoing communication to the Diagnostics Server in Mediator mode via HTTPS:

1 Copy the certificate file from **<diagnostics_server_install_dir>/etc/ diag_server_mediator.cer** on the Diagnostics Server to **<diagnostics_server_install_dir>/etc/diag_server_mediator.cer** on the Diagnostics Server in Commander mode.

2 Change the value of the **trusted.certificate** property in the file **<diagnostics_server_install_dir>/etc/security.properties** for the Diagnostics Server in Commander mode.

Set **trusted.certificate=diag_server_mediator.cer**. If there are already other certificate files included in the value of this property, add the certificate file to the end of the list separated from the preceding value by a comma.

3 For incoming Diagnostics Server communication, indicate the URL for the Diagnostics Server by updating the following property in the file **<diagnostics_server_inst_dir>/etc/server.properties** on the Diagnostics Server in Commander mode.

Set **commander.url** to **https://<diagserver_commander_hostname>:8443**

To enable the Diagnostics Server in Mediator mode for outgoing communication to the Diagnostics Server in Commander mode via HTTPS:

1 Copy the certificate file from **<diagnostics_server_install_dir>/etc/ diag_server_commander.cer** on the Diagnostics Server in Commander mode to **<diagnostics_server_install_dir>/etc/diag_server_commander.cer** on the Diagnostics Server in Mediator mode.

2 Change the value of the **trusted.certificate** property in the file **<diagnostics_server_install_dir>/etc/security.properties** for the Diagnostics Server in Mediator mode.

Set **trusted.certificate=diag_server_commander.cer**. If there are already other certificate files included in the value of this property, add the certificate file to the end of the list separated from the preceding value by a comma.

3 For incoming Diagnostics Server communication, indicate the URL for the Diagnostics Server by updating the following property in the file **<diagnostics_server_inst_dir>/etc/server.properties** on the Diagnostics Server in Mediator mode.

643

Set **commander.url** to **https://<diagserver_commander_hostname>:8443**

Note: When you enable HTTPS on the server, you will need to use port 8443 in the URL to run the Diagnostics UI.

To enable the Diagnostics Server (in Commander or Mediator mode) for outgoing communications to the probes via HTTPS:

1 Copy the certificate file from **<probe_install_dir>/etc/java_probe_<probe_host>.cer** for each probe to **<diagnostics_server_install_dir>/etc/Java_probe_<probe_host>.cer** on the Diagnostics Server.

2 Change the value of the **trusted.certificate** property in the file **<diagnostics_server_install_dir>/etc/security.properties** for the Diagnostics Server.

Set **trusted.certificate=Java_probe_<probe_host>.cer**. If there are already other certificate files included in the value of this property, add the certificate file to the end of the list separated from the preceding value by a comma.

To enable the Diagnostics Server in Mediator mode for outgoing communications to the collectors via HTTPS:

1 Copy the certificate file from **<collector_install_dir>/etc/collector_<collector_host>.cer** for each probe to **<diagnostics_server_install_dir>/etc/collector_<collector_host>.cer** on the Diagnostics Server.

2 Change the value of the trusted.certificate property in the file **<diagnostics_server_install_dir>/etc/security.properties** for the Diagnostics Server.

Set **trusted.certificate=collector_<collector_host>.cer**. If there are already other certificate files included in the value of this property, add the certificate file to the end of the list separated from the preceding value by a comma.

To enable the Java Probe for outgoing communications to the Diagnostics Server in Mediator mode via HTTPS:

1 Copy the certificate file from **<diagnostics_server_install_dir>/etc/ diag_server_mediator.cer** on the Diagnostics Server in Mediator mode to **<probe_install_dir>/etc/diag_server_mediator.cer** on the Java Probe.

2 Change the value of the **trusted.certificate** property in the file **<probe_install_dir>/etc/security.properties** for the Java Probe.

Set **trusted.certificate=diag_server_mediator.cer**. If there are already other certificate files included in the value of this property, add the certificate file to the end of the list separated from the preceding value by a comma.

3 For incoming Java Probe communication, indicate the URL for the Diagnostics Server in Mediator mode by updating the following property in the file **<probe_inst_dir>/etc/dispatcher.properties**.

Set **registrar.url** to **https://<diagserv_mediatormode_hostname>:8443/ commander/registrar/**

Note: When you enable HTTPS on the server, you will need to use port 8443 in the URL to run the Diagnostics UI.

To enable the server/collector's embedded java probe for outgoing communications to the Diagnostics Server in Mediator mode via HTTPS:

1 Copy the certificate file from **<diagnostics_server_install_dir>/etc/ diag_server_mediator.cer** on the Diagnostics Server in Mediator mode to **<server/collector_install_dir>/probe/etc/diag_server_mediator.cer** on the embedded Java Probe.

2 Change the value of the **trusted.certificate** property in the file **<server/ collector_install_dir>/probe/etc/security.properties** for the embedded Java Probe. Set **trusted.certificate=diag_server_mediator.cer**. If there are already other certificate files included in the value of this property, add the certificate file to the end of the list separated from the preceding value by a comma.

3 For incoming embedded Java Probe communication, indicate the URL for the Diagnostics Server in Mediator mode by updating the following property in the file **< server/collector_install_dir>/probe/etc/ dispatcher.properties**. Set **registrar.url** to **https:// <diagserv_mediatormode_hostname>:8443/commander/registrar/**.

To enable the Collector for outgoing communications to the Diagnostics Server in Mediator mode via HTTPS:

1 Copy the certificate file from **<diagnostics_server_install_dir>/etc/ diag_server_mediator.cer** on the Diagnostics Server in Mediator mode to **<collector_install_dir>/etc/diag_server_mediator.cer** on the Collector.

2 Change the value of the **trusted.certificate** property in the file **<collector_install_dir>/etc/security.properties** for the Collector.

Set **trusted.certificate=diag_server_mediator.cer**. If there are already other certificate files included in the value of this property, add the certificate file to the end of the list separated from the preceding value by a comma.

3 For incoming Collector communication, indicate the URL for the Diagnostics Server in Mediator mode by updating the following property in the file **<collector_install_dir>/etc/collector.properties**.

Set **registrar.url** to **https://<diagserv_mediatormode_hostname>:8443/ commander/registrar/**

Note: When you enable HTTPS on the server, you will need to use port 8443 in the URL to run the Diagnostics UI.

To enable the .NET Probe for outgoing communications to the Diagnostics Server in Commander mode via HTTPS:

1 Copy the certificate for the Diagnostics Server in Mediator mode to the host for the .NET Probe. The certificate was generated when the Diagnostics Server in Mediator mode was configured to receive HTTPS. See "Enabling Incoming HTTPS Communication for Diagnostics Components" on page 633 for instructions to configure the Diagnostics Server in Mediator mode to receive HTTPS. If you followed the instructions in the referenced section, the certificate can be found in **<diagnostics_server_install_dir>/etc/ diag_server_mediator.cer**.

2 On the Windows Taskbar, select **Start** > **Run**.

3 Run the Microsoft Management Console by typing mmc, and then clicking **OK**.

4 On the Microsoft Management Console menu, select **File** > **Add/Remove Snap-in** to display the Add/Remove Snap-in dialog.

5 Click **Add** on the Add/Remove Snap-in dialog.

6 Select **Certificates** from the Available Standalone Snap-in list and click **Add**.

7 In the Certificates Snap-in dialog box select **Computer account**, and click **Next**.

8 In the Select Computer dialog box select **Local Computer: (the computer this console is running on)**, and then click **Finish**.

9 Click **Close** on the Add Standalone Snap-in.

10 Click **OK** on the Add/Remove Snap-in dialog.

11 On the Microsoft Management Console expand the listing for Certificates (Local Computer) in the left pane of the Console Root dialog.

12 Under Certificates (Local Computer), expand Trusted Root Certification Authorities.

13 Under Trusted Root Certification Authorities, right-click Certificates and select **All Tasks** > **Import** to start the Certificate Import Wizard.

14 Click **Next** to move past the Welcome dialog box of the Certificate Import Wizard.

15 Click **Browse** to navigate to the public keystore for the Diagnostics Server in Mediator mode.

    a Select All Files (***.***) in Files of type:

    b Navigate to the directory where the keystore for the Diagnostics Server in Commander mode was copied in step 1 and click Open. This should be **<diagnostics_server_install_dir>/etc/diag_server_mediator.cer**

16 Click **Next** to import the file.

17 Click **Next** to accept the default Certificate Store location of "Trusted Root Certification Authorities."

18 Click **Finish** on Completing the Certificate Import Wizard.

19 Click **OK** on the Certificate Import Wizard confirmation dialog.

20 Select **Certificates** under Trusted Root Certification Authorities to find the certificate you just added (it should be the hostname of the mediator server). Make a note of the value in the **Issued to** column. This value will be used for modifying the probe configuration files.

21 Edit the **<probe_install_dir>/etc/probe_config.xml** and change the **diagnosticsserver url** property to use the HTTPS URL: **<diagnosticsserver url="https://<diagnostics_mediator_server_host>:8443/commander" />**

Change the mediator host and port and add ssl="true": **<mediator host="<diagnostics_mediator_server_host>" port="2612" metricport="8443" ssl="true"/>**

22 Edit **<probe_install_dir>/etc/metrics.config**. Change the metrics.server.uri value to specify the HTTPS URL: **metrics.server.uri = https:// <diagnostics_mediator_server_host>:8443/metricdata/**

Note: For both the **probe_config.xml** and the **metrics.config** files, the **<diagnostics_mediator_server_host>** value must match the name that appears in the certificate. For example, if the hostname in the certificate is fully qualified, the hostname in the configuration files should also be fully qualified.

---

23 Restart IIS. For instructions on restarting IIS see "Restarting IIS" on page 218.

To verify that you have successfully configured the .NET probe for HTTPS communication with the Diagnostics Server in Commander mode:

1 Browse to your .NET application to activate the .NET Probe.

2 Verify that the .NET Probe node is shown in System Health.

Enabling HTTPS Communications for the Business Availability Center Server

The following instructions will guide you through the process of configuring Business Availability Center for HTTPS communication with Diagnostics.

To enable HTTPS communications between the Diagnostics Server in Commander modeand Business Availability Center:

1 Copy the Diagnostics certificate file, **diag_server_commander.cer**, from the Diagnostics Server in Commander mode installation directory, **<diagnostics_server_install_dir>/etc/**, to the Business Availability Center host.

2 Import the copied certificate, **diag_server_commander.cer**, into the Business Availability Center server cacert keystore by running the following command on the Business Availability Center host:

```
<BAC_server_install_dir>/_jvm/bin/keytool -import -file
<copied _diag_certificate_directory>/diag_server_commander.cer -keystore
<BAC_server_install_dir>/jre/lib/security/cacerts -alias SERVER
```

➤ Replace **<BAC_server_install_dir>** with the path to the installation directory for Business Availability Center.

➤ Replace **<copied_diag_certificate_directory>** with the path to the copied Diagnostics certificate file.

Type changeit when you are prompted to enter the keystore password.

Type yes, instead of the default no when you are asked if the certificate should be trusted.

3 Copy the Business Availability Center certificate file,
<BAC_certificate_file.cer>, to the Diagnostics Server host.

4 Import the copied certificate into the Diagnostics Server cacert keystore by
running the following command on the Diagnostics Server host.

```
<diagnostics_server_install_dir>/_jvm/bin/keytool -import -file
<copied_BAC_certificate_directory>/<BAC_certificate_file.cer> -keystore
<diagnostics_server_install_dir>/JRE/lib/security/cacerts
```

> ➤ Replace <**diagnostics_server_install_dir**> with the path to the installation
> directory of the Diagnostics Server in Commander mode.

> ➤ Replace <**copied_BAC_certificate_directory**> with the path to the copied
> Business Availability Center certificate file.

When you are prompted to enter the keystore password, type the string that
you assigned as the **storepass** password when you created the keystore.

Type yes, instead of the default no, when you are asked if the certificate
should be trusted.

5 Point the Business Availability Center server to the HTTPS port on the
Diagnostics Server in Commander mode.

a Open the Diagnostics Administration in Business Availability Center by
selecting **ADMIN > Diagnostics**.

b Click the **Registration** tab.

c Locate the Diagnostics Server details section.

d Provide the following information in the appropriate fields:

> ➤ Enter the host name for the Diagnostics Server in Commander mode
> exactly as it was specified in the **CN** parameter when you created the
> keystore for the Diagnostics Server in Commander mode. You should
> have used the **fully qualified domain name** for the subject (CN) in the
> certificate. See "Enabling Incoming HTTPS Communication for
> Diagnostics Components" on page 633.

> ➤ Enter HTTPS for the protocol

> ➤ Enter 8443 for the Web port of the Diagnostics Server.

e Click **Submit Configuration**.

HTTPS Checklist per Diagnostics Component

The following table summarizes the configuration steps that you must perform to enable HTTPS communications for each Diagnostics component:

Configuration step	Commanding Server	Mediator Server	Java Probe	Collector	.NET Probe
Generate key and export certificate	**Yes**	**Yes**	**Yes**	**Yes**	No
Obfuscate passwords	**Yes**	**Yes**	**Yes**	**Yes**	No
Copy commanding server certificate	No	**Yes**	No	No	No
Copy mediator server certificate	**Yes**	No	**Yes**	**Yes**	**Yes**
Copy Java Probe certificates	**Yes**	**Yes**	No	No	No
Copy Collector certificates	No	**Yes**	No	No	No
Edit security.properties: enablessl=true, keystorepassword, keypassword	**Yes**	**Yes**	**Yes**	**Yes**	No
Edit security.properties: add commanding server certificate to trusted.certificates	No	**Yes**	No	No	No
Edit security.properties: add mediator server certificate to trusted.certificates	**Yes**	No	**Yes**	**Yes**	No
Edit security.properties: add Java probe certificate to trusted.certificates	**Yes**	**Yes**	No	No	No
Edit security.properties: add Collector certificate to trusted.certificates	No	**Yes**	No	No	No
Import mediator server certificate to Trusted Root Authority	No	No	No	No	**Yes**

Configuration step	Commanding Server	Mediator Server	Java Probe	Collector	.NET Probe
Edit server.properties: set commander.url	**Yes**	**Yes**	No	No	No
Edit dispatcher.properties: set registrar.url	No	No	**Yes**	No	No
Edit collector.properties: set registrar.url	No	No	No	**Yes**	No
Edit probe_configuration.xml: set diagnosticsserver url, mediator host, metricport, and ssl	No	No	No	No	**Yes**
Edit metric.config: set metrics.server.uri	No	No	No	No	**Yes**

D

Using the System Health Monitor

You can use the System Health Monitor to review the configuration of the HP Diagnostics components and verify that they are working properly.

This chapter includes:

Introducing the System Health Monitor

The System Health Monitor provides you with a map of all of the components of your HP Diagnostics deployment, and gives you real-time information and health status for each component. At a glance, you can determine which components are experiencing problems, the load on each component, and the amount of data flowing between components.

By drilling down to the System Health Monitor component map, you can reveal the details about the configuration, performance metrics, and processing logs for the components. You can also find troubleshooting tips for handling many of the issues that are revealed in the System Health Monitor component map. In many cases, the System Health Monitor will be your first and only stop when you need to know information about the components in your Diagnostics deployment and the machines that host them.

You can also capture a snapshot of the System Health Monitor information in an .xml file so that you can share it with others who may be able to help you diagnose the issues that you see on the map.

Accessing the System Health Monitor

You can access the System Health Monitor directly from your Web browser, from Performance Center, from the LoadRunner Controller, or from Business Availability Center.

Note: To access the System Health Monitor, you require **system** permissions. For more information about permissions, see "Understanding User Privileges" on page 596.

To access the System Health Monitor from your Web browser:

1 Enter the following URL into your browser:

http://<Diagnostics Server in Commander mode host>:<Diagnostics Server in Commander mode port>/registrar/health

The default port for the Diagnostics Server in Commander mode is 2006. If you have configured the Diagnostics Server to use an alternate port, use that port number in the URL

2 Enter your user name and password. For more information about user names and passwords, see Appendix B, "User Authentication and Authorization."

The System Health Monitor opens in your browser.

Note: You can also access the System Health Monitor from Business Availability Center, Performance Center and LoadRunner.

Troubleshooting Accessing the System Health Monitor

If you are unable to access the System Health Monitor verify that:

➤ the machine name for the **Diagnostics Server** host is correct.

➤ the port number for the **Diagnostics Server** is correct. The default port is **2006**.

➤ the **Diagnostics Server** started successfully, and is running on the host.

Understanding the System Health Monitor

The System Health Monitor is comprised of two panes:

➤ **The graph legend** (in the right pane)

➤ **The System Health Monitor component map** (in the left pane)

When the System Health Monitor opens you see a deployment map of the HP Diagnostics components, accompanied by the graph legend.
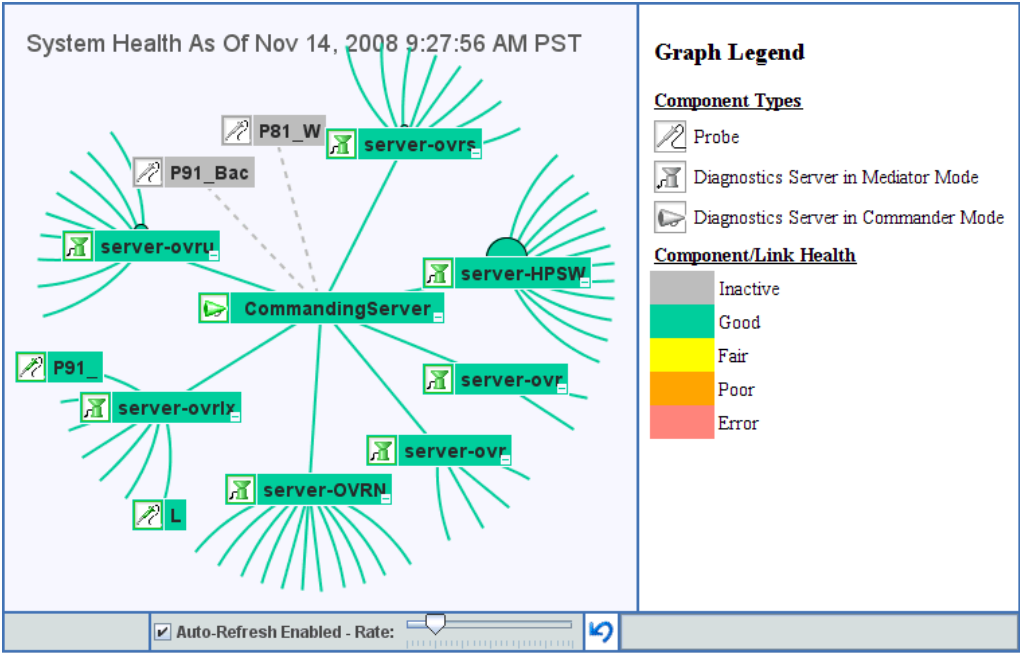
The Graph Legend

The graph legend, to the right of the component map, helps you to interpret the information displayed in the component map.
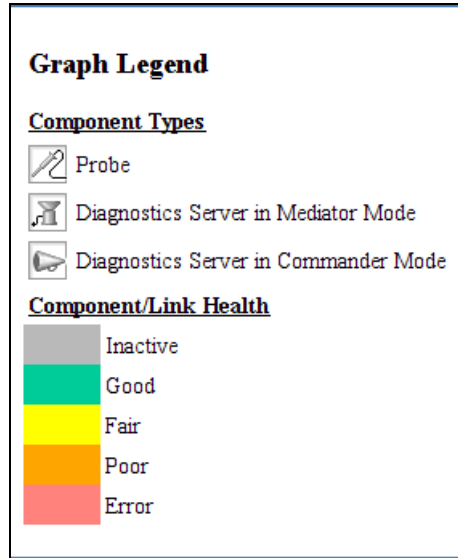
The System Health Monitor Component Map

The System Health Monitor component map displays high-level information about your Diagnostics deployment and how well the components are performing. The System Health Monitor component map displays the following:

➤ Icons that represent each of the Diagnostics components that you have correctly configured as part of your Diagnostics environment.

➤ The color of the component icon that indicates health of the component.

➤ An indicator of the amount of load that the component is processing.

➤ Links between the component icons, which indicate the state of the communication links between components.

   ➤ The color of the link indicates the status of the link.

   ➤ A solid line means data is flowing across the link.

   ➤ A dashed line means there is currently no data flowing.

In the following image of a Diagnostics deployment you can see the following:

One Diagnostics Server is deployed in Commander mode. It is represented in the map by an icon labeled **CommandingServer**.

There are a number of Diagnostics Servers deployed in Mediator mode. These are represented by an icon labeled **server-<host name>**.

Probes that are reporting to each particular server are shown. A probe is represented by an icon labeled **<name of probe>.**

Diagnostics Collectors that are reporting to each particular Server are shown. A Collector is represented by an icon labeled **<name of Collector>.**
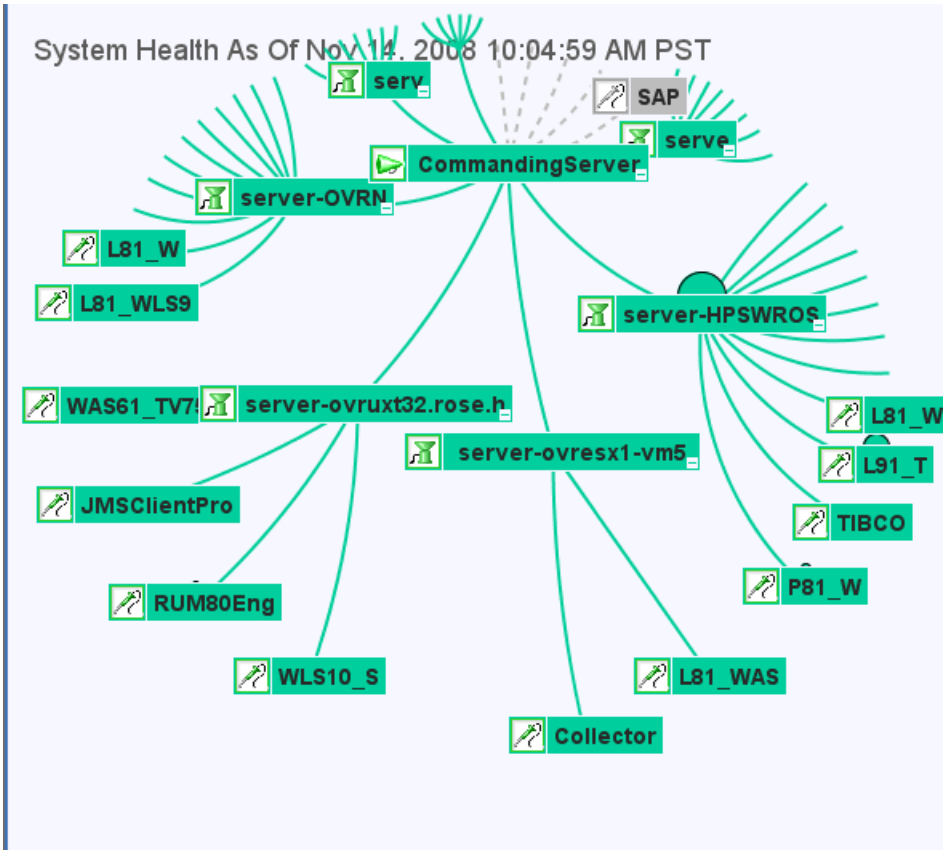
Viewing Component Status and Host Configuration Tooltips

You can view tooltips for each component on the System Health Monitor component map. The tooltips provide you with basic status and host configuration information.

To view component status and host configuration tooltips:

Hold the mouse pointer over the component until a tooltip is displayed.

The examples that follow, show tooltips displaying component status and host configuration information for each different component.

The information in the tooltip for the Diagnostics Server components includes a list of the probes that have been configured to work with the Diagnostics Server.

➤ **Diagnostics Server in Commander mode**
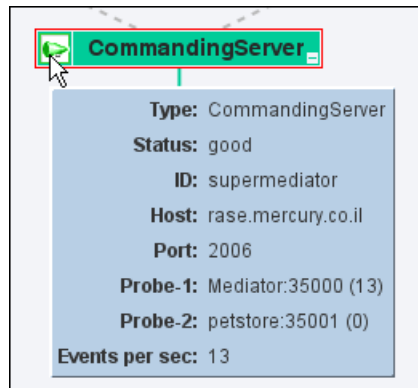
➤ **Diagnostics Server in Mediator mode**

➤ **Diagnostics Probe**

The information in the tooltip for a probe component includes the mode that the probe has been configured to work in (**Mode** field). The tooltip also directs you to the relevant instrumentation points file.

➤ **Diagnostics Collector**

Viewing Detailed Component Information

You can view detailed information about the configuration and performance of each component in the System Health Monitor component map. The information is displayed in the Component Monitor detail table.

To view detailed component information:

Right-click the component and choose the relevant item from the menu.

Alternatively, double-click the component icon and scroll to the relevant section.

You can view the following information:

➤ Component Metric Information: Lists the processing metrics for the selected component.

➤ Component Configuration Information: Lists the component configuration for the selected component.

➤ Component Log information: Lists the log messages for the selected component.

➤ Component Status Information: Provides further information about the performance status of the component, where relevant.

➤ Component Miscellaneous Information: Provides miscellaneous information about the component.

Component Metric Information

The following image shows a section of the Component Metric information for the Diagnostics Server in Commander mode:

Component Monitor Update As Of Dec 7, 2005 2:21:12 PM GMT+02:00	
⊗ Metric	
Events since launch	\<unknown\>
Last update (commander time)	Wed Dec 07 04:22:35 PST 2005
Up time	1 day(s) 0 hour(s) 4 minute(s) 53 second(s)
WDE bytes relayed	166836
WDE bytes relayed since launch	92396096
WDE samples failed	75
WDE samples failed since launch	58870
WDE samples relayed	171
WDE samples relayed since launch	94676

Component Configuration Information

The following image shows the Component Configuration information for the .NET Probe:

Component Monitor Current As Of Nov 14, 2008 10:23:51 AM PST	
⊗ **Configuration**	
Host	OVRNTT209.ovrtest.adapps.hp.com
IP Address	15.8.152.24
Install dir	C:\MercuryDiagnostics\.NET Probe\
Instrumentation	ASP.NET.points,CallChain2_0.points
Lightweight Memory Diagnostics	OFF
Mode	Enterprise,PRO,TV
Port	35003
Probe Group	Sanity_LR_8_1_ovrlxd14
System	OVRNTT209 (Microsoft Windows NT 5.2.3790 Service Pack 1)
VM	.NET 2.0.50727.832
Version	8.0.22.431

Component monitor will auto-refresh with health display...

Component Log information

The following image shows the Component Log information for the Diagnostics Server in Mediator mode:

Component Monitor Update As Of Dec 7, 2005 2:34:02 PM GMT+02:00	
⊗ Log	
Wed Dec 07 04:06:30 PST 2005	WARNING: real user fragment 31289557 timed out
Wed Dec 07 03:09:02 PST 2005	WARNING: real user fragment 32930633 timed out
Wed Dec 07 03:08:49 PST 2005	WARNING: real user fragment 32687513 timed out
Wed Dec 07 01:08:38 PST 2005	WARNING: real user fragment 8834978 timed out
Tue Dec 06 23:08:41 PST 2005	WARNING: real user fragment 12640068 timed out
Tue Dec 06 05:51:44 PST 2005	SEVERE: out of order com.mercury.diagnostics.server.persistence.impl.D
Tue Dec 06 05:51:44 PST 2005	SEVERE: out of order com.mercury.diagnostics.server.persistence.impl.D
Tue Dec 06 05:46:01 PST 2005	SEVERE: out of order com.mercury.diagnostics.server.persistence.impl.D

Component Status Information

Below is an example of Component Status information for the Java Probe. This information is available when the status of the component drops below **Good**.

Component Monitor Update As Of Dec 7, 2005 3:13:20 PM GMT+02:00	
⊗ Status	
Error	Probe reporting communication problems with mediator: myDomain6 alol:2612 [null]

Component Miscellaneous Information

The following image shows the Component Miscellaneous information for the Java Probe:

Component Monitor Update As Of Dec 7, 2005 2:42:42 PM GMT+02:00	
⊗ Miscellaneous	
ProbeProtocolVersion	3

Viewing Troubleshooting Tips

Troubleshooting information is available for each component in the System Health Monitor component map.

To view tips for troubleshooting Diagnostics components:

Right-click the relevant component and choose **View Troubleshooting Tips**. Troubleshooting information for the selected component is displayed.
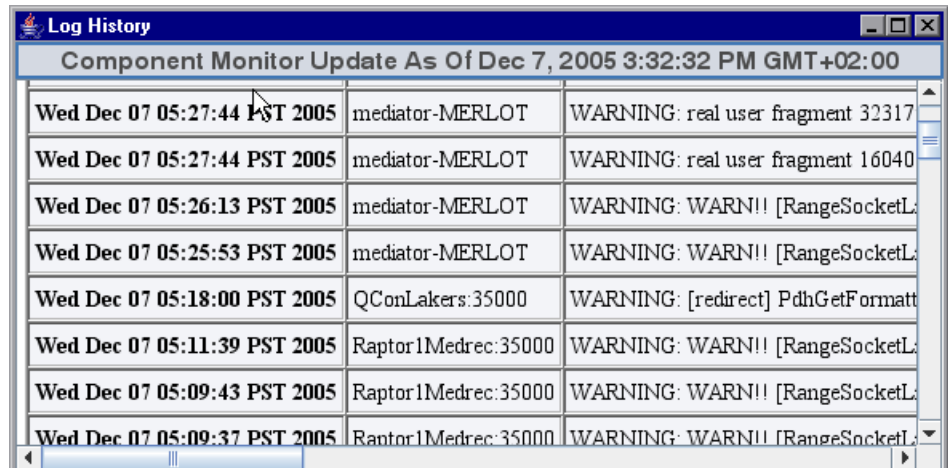
Scan the information displayed for the symptoms that you are investigating, to see if there is a recommended solution documented in the troubleshooting tips.

Viewing Log Information for the Whole System

You can view the log messages for all of the components in the System Health Monitor on one screen.

To view log information for all the components:

Right-click anywhere in the System Health Monitor component map (except on a component icon) and choose **View Log History**. The log for all component activity is displayed.

Log History		_ □ ×
Component Monitor Update As Of Dec 7, 2005 3:32:32 PM GMT+02:00		
Wed Dec 07 05:27:44 PST 2005	mediator-MERLOT	WARNING: real user fragment 32317
Wed Dec 07 05:27:44 PST 2005	mediator-MERLOT	WARNING: real user fragment 16040
Wed Dec 07 05:26:13 PST 2005	mediator-MERLOT	WARNING: WARN!! [RangeSocketL:
Wed Dec 07 05:25:53 PST 2005	mediator-MERLOT	WARNING: WARN!! [RangeSocketL:
Wed Dec 07 05:18:00 PST 2005	QConLakers:35000	WARNING: [redirect] PdhGetFormatt
Wed Dec 07 05:11:39 PST 2005	Raptor1Medrec:35000	WARNING: WARN!! [RangeSocketL:
Wed Dec 07 05:09:43 PST 2005	Raptor1Medrec:35000	WARNING: WARN!! [RangeSocketL:
Wed Dec 07 05:09:37 PST 2005	Raptor1Medrec:35000	WARNING: WARN!! [RangeSocketL:

Customizing the System Health Monitor Display

You can customize the way information is displayed on the System Health Monitor.

Manipulating the Appearance of the System Health Monitor Component Map

You can manipulate the System Health Monitor component map to display any area of the tree, to display expanded branches or collapsed branches, and to change the overall panorama of the image.

To manipulate the System Health Monitor component map display:

➤ Click a component to move it to the center of the map.

➤ Click anywhere in the component map to change the emphasis of the map. The components will center around the point where the mouse was clicked.

➤ Click and drag anywhere in the component map to move and rotate the components around that point.

➤ Expand and collapse the branches of a sub-tree by clicking the expand (+) or collapse (-) symbols, displayed on the lower-right corner of the component icon.

➤ Increase or reduce the gap between each branch by holding down the ALT button on the keyboard and dragging in the component map.

Displaying the Graph Legend

You can choose whether or not to display the graph legend on the System Health Monitor.

To show the graph legend:

➤ If the graph legend is not displayed, right-click anywhere in the System Health Monitor component map (except on a component icon) and choose **Show Graph Legend**.

To hide the graph legend:

➤ If the graph legend is displayed, right-click anywhere in the System Health Monitor component map (except on a component icon) and choose **Hide Graph Legend**.

Displaying Load

You can choose the way in which load is displayed on the System Health Monitor. You can choose between displaying the load indicator as a circle that appears behind the component icon, or as a bar scale that appears below the component icon.

To display load as a circle:

➤ When load is displayed as a bar scale, right-click anywhere in the System Health Monitor component map (except on a component icon) and choose **Show Load Using Circles**.

In the following example, the circle behind the component represents the amount of load.

To display load as a bar scale:

➤ When load is displayed as circles, right-click anywhere in the System Health Monitor component map (except on a component icon) and choose **Show Load Using Scale**.

In the following example, the bar scale below the component represents the amount of load.
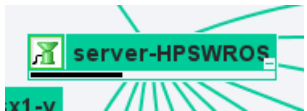
Filtering the System Health Monitor Component Map by Customer

When you are running Diagnostics in an Software as a Service (SaaS) environment with multiple customers, you can filter the information that is displayed in the System Health Monitor so that only the information for a selected customer is displayed in the graph.

By default, the System Health Monitor will display the components for all of the customers.

When more than one customer is represented in the Diagnostics deployment displayed in the System Health Monitor, a list of customers is displayed in the Graph Legend box.
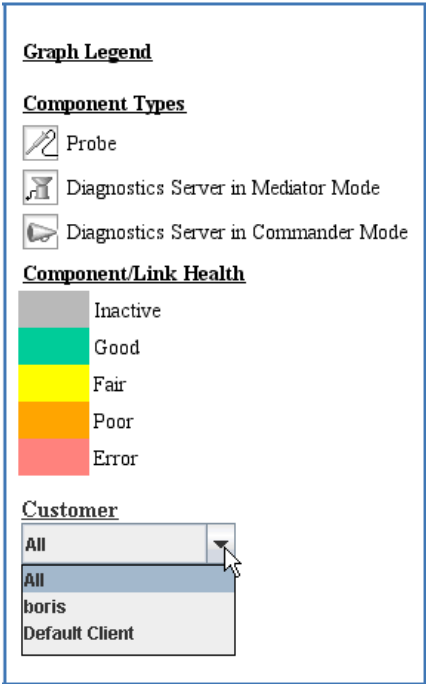
Select the customer whose components you would like to see displayed in the System Health Monitor, or select **All** to display the components for all customers at once.

Controlling the Refresh Rate of the System Health Monitor

By default, the System Health Monitor is configured to automatically refresh the information displayed. You can change the auto-refresh frequency, or completely disable the auto-refresh feature. You can also request a manual refresh at any time.

The controls at the bottom of the System Health Monitor that are used to control the refresh of the information displayed, are shown below.

To disable the automatic refresh feature:

➤ Clear the **Auto-Refresh Enabled** check box.

To enable the automatic refresh feature:

➤ Select the **Auto-Refresh Enabled** check box.

To adjust the auto-refresh rate:

➤ Slide the **Rate** slide control to the right to decrease the auto-refresh frequency and to the left to increase the frequency.

To refresh the display manually:

➤ Click the **Refresh** button located to the right of the slide bar.

Creating and Using System Health Monitor Snapshots

You can take snapshots of the System Health Monitor to share the information with others. You can also import snapshots that others have taken of their System Health Monitors to learn information about their systems.

Exporting a Snapshot of the System Health Monitor

You can export the information displayed on System Health Monitor to an .xml formatted file so that you can share the information with others. This is especially useful when you need help diagnosing a problem. The information in the System Health .xml file can be viewed in .xml format or can be imported into any working copy of the System Health Monitor.

Note: Remember! People who have access to the Diagnostics Server in Commander mode host can view the System Health Monitor directly using their Web browser if you give them the URL.

To export a System Health Monitor snapshot as an .xml file:

1 Enter the following URL in your Web browser:

http://<Diagnostics Server in Commander mode host>:<Diagnostics Server in Commander mode port>/registrar/xml

The default port is **2006**. If you have configured the Diagnostics Server to use an alternate port, use that port number in the URL

2 Using the **Save** menu option in your Web browser, save the Web page to a file. Be sure to give the file a name that will help you remember why you saved the snapshot. For example, to remember that you took a snapshot of a System Health Monitor with a poor performing Diagnostics Server in Mediator mode, you may want to name the file:

sys_health_mediator_yyyymmdd.xml

Importing a Snapshot of a System Health Monitor

If someone gives you an .xml snapshot of a System Health Monitor, you can view it from an existing System Health Monitor.

To import a System Health Monitor snapshot:

1 Copy the System Health snapshot file to a directory on the Diagnostics Server in Commander mode host machine.

2 Enter the following URL in your Web browser:

http://<Diagnostics Server in Commander mode host>:<Diagnostics Server in Commander mode port>/registrar/health?xml=<XML_file>

where <XML_file> is the path to the .xml file on the Diagnostics Server in Commander mode host machine.

For example, if you copied **sys_health_mediator_yyyymmdd.xml** to the hard drive (C:) on the Diagnostics Server in Commander mode host machine, whose name is jake, the URL might look like this:

http://jake:2006/registrar/health?xml=c:\sys_health_mediator_yyyymmdd.xml

The <Diagnostics Server in Commander mode port>, by default, should be 2006. If you have configured the Diagnostics Server to use an alternate port, use that port number in the URL.

When the System Health Monitor is displayed in your Web browser, the information is from the imported .xml file.

E

Diagnostics Data Management

Detailed information is provided on how Diagnostics data is managed and stored.

This chapter includes:

About Diagnostics Data

There are two main types of Diagnostics data. The first consists of the custom views that each user has created. The second consists of the Diagnostics data collected by the probes and aggregated by the Diagnostics Servers. This data is stored in a time-series database on the Diagnostics Servers. Each Diagnostics Server stores the data that is collected by the probes that report to it. In addition, the Diagnostics Server in Commander mode stores the virtual transactions' data both for AD and AM as well as the application metrics. The organization and maintenance of the data files that make up the data base are described in the following discussion.

Custom View Data

Diagnostics users can create and save customized views as described in the chapter, "Customizing Diagnostics Views," in the *HP Diagnostics User's Guide*. Diagnostics stores the customized views as XML files on the host for the Diagnostics Server in Commander mode.

Custom View Data Organization

The user defined custom views are stored as XML files in the **<diagnostics_server_install_dir>/storage/userdata** directory on the host for the Diagnostics Server in Commander mode. The custom view files are relatively small.

Each user that has defined a custom view has their own custom view sub-directory in the **userdata** directory. For example, if the **admin** user created two custom views, Sales Status and Host Status, the two views would be stored as separate .xml files in the **<diagnostics_server_install_dir>/ storage/userdata/Default Client/admin** directory on the Diagnostics Server in Commander mode as shown in the following example.
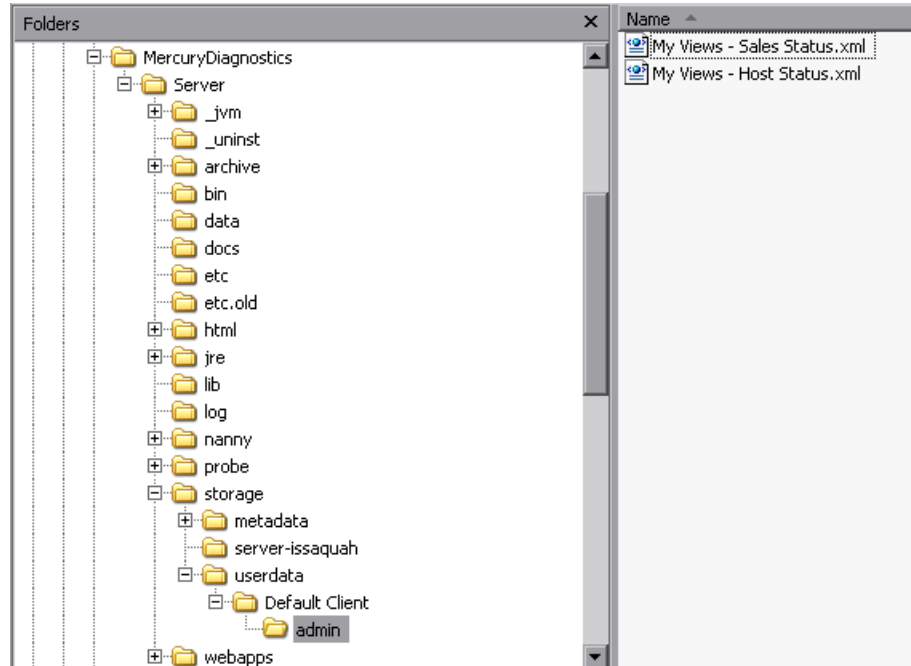
Performance History Data

Diagnostics stores the historical performance data in a time series database on the Diagnostics Server in Mediator mode. If the Diagnostics Server has numerous probes reporting to it, the stored historical performance data can grow to many gigabytes of data. Although the amount of data collected for each application can vary in size, it is recommended that you plan for approximately 3 GB of data for each virtual machine that you are monitoring. For more information, see "Data Storage Size & Data Retention" on page 679.

Performance History Data Organization

The Diagnostics performance history data is in the **<diagnostics_server_install_dir>/archive/mediator-<host_name>/ persistence/<customer_name>_** directory of the Diagnostics Server in Mediator mode where:

➤ **<host_name>** is the name of the host for the Diagnostics Server in Mediator mode

➤ **<customer_name>** is the customer name that you entered when you installed the Diagnostics Server in Mediator mode. The name of this directory is the customer name with an appended underscore.

Notes:

➤ Unless you are an SaaS customer, the customer name should always be Default Client.

➤ The Diagnostics Performance history data for Performance Center or LoadRunner runs is stored in the **../persistence/<customer_name>_<run identifier> directory**.

Performance History Data File Types

The Diagnostics performance history data is stored in five types of files.

Symbol Table Files

The symbol tables contain string-to-integer mappings for small and fast data encoding of the other data files. For example, /login.do might be encoded as 1347854.

The symbol tables are stored in the **<diagnostics_server_install_dir>/ archive/mediator-<host_name>/symboltable** directory.

Per-time-Period Performance Summary Files

In the Diagnostics views, data can be viewed in yearly, quarterly, monthly, weekly, daily, 6-hourly, 1-hour, 20-minute, and 5-minute periods. Each viewable time period is stored in separate *summary* files. For example, over the course of one week, the following summary files would be created:

➤ 1 weekly summary file

➤ 7 daily summary files

➤ 28 six-hourly summary files

➤ 168 hourly summary files

➤ 504 twenty-minute summary files

➤ 2,016 five-minute summary files

Each summary file is named according to the minute (in GMT) at which Diagnostics started storing summary data in it. For example, a summary file that contained data beginning at midnight (GMT) on November 23, 2006 would be named **2006_11_23_0.summary.**

The summary files are stored in sub-directories based upon the duration of the data that they contain. The subdirectories are **Years/1Y, Years/3M, Years/Years, Months/1M, Weeks/7d**, **Days/1d**, **Days/6h**, **Hours/1h**, **Hours/20m**, **Hours/5m**. For example, a summary file for one of the daily time periods would be stored in the sub-directory **<diagnostics_server_install_dir>/archive/mediator-<host_name>/persistence/<customer_name>_ /Days/1d.**

The data from the summary files are used to create the graphs and tables shown in the Diagnostics views.

Per-Major-Time-Period Performance Trend Files

Diagnostics stores the charted trend data for each major time period in *trend* files. The major time periods are years, months, weeks, days, and hours. The trend files are named according to the minute (in GMT) at which the trended data that they contain was captured. For example, a file that contained hourly trend data starting at midnight November 23, 2006 would be named as follows: **Hours/1h/ 2006_11_23_0.trend**.

The trend files are accessed when the user selects an element to be charted on the UI. To retrieve a trend for a minor time period, such as 5 minutes, only a small portion of the data in a major time period trend file is read.

Per-Major-Time-Period Instance Tree Files

The *instance tree* files are similar to the trend files; there is a corresponding dump of the collected instance call trees for each major time period. For example, **Hours/1h/2005_11_23_0.tree**.

Instance tree files are accessed when the user drills down to an instance call tree on the Diagnostics UI.

Data Storage Size & Data Retention

Diagnostics uses data compression and data retention strategies that allow it to optimize its use of disk storage.

Data File Compression

Diagnostics performance data is stored in multiple files that represent different time periods. However, Diagnostics only *writes* new diagnostics data to the files that represent the *current* time period. This means that most of the Diagnostics data files for an application can be stored as read-only files.

Since the uncompressed data files are quite large they are automatically compressed after each time period is complete in order to save disk space. Compressed files have the same file names as the uncompressed file, but with a .zip extension.

Data File Retention

In order to make optimum use of disk space, historical performance data is stored in tiers with the data in each tier retained based upon the diagnostics data retention policy. The data in the tiers with lower resolution data points is kept for longer periods of time to assist with such activities as capacity planning. The data in the tiers with high resolution data points is kept for shorter periods of time to assist with such activities as performance diagnostics. For this retention policy, measurements have shown that Diagnostics uses approximately 3 GB for each probed virtual machine.

The general Diagnostics data retention policy is illustrated in the table below:

Directory	Data is kept for ...	Viewable Periods	Trend Resolution
Years	5 Years	Year, Quarter	1 Day
Months	24 Months	Month	6 Hours
Weeks	52 Weeks	Week	1 Hour
Days	93 Days	Day & 6 Hours	5 minutes
Hours	72 Hours	Hour, 20 minutes & 5 minutes	5 seconds

Note: The above table only applies when the entities don't change and are constantly available for the specified periods of time. See "Data Retention and Purging Policies" that follows.

Data Retention and Purging Policies

The purging mechanism used by the TSDB provides for the following:

➤ Data that is part of a snapshot is not purged since doing so renders the incident useless.

➤ A property can be specified for how much space you want the TSDB to use and it will automatically delete data to maintain a size less than this specified threshold. The **persistence.purging.threshold** property is set in the **<diagnostics_server_install_dir>/etc/server.properties** file.

➤ If a probe gets renamed or no data is received from the probe for 6 months (by default), Diagnostics automatically purges the probe and its data from the database.

The purging interval can be changed in **server.properties** by increasing the number of hours in the **persistency.major.4.total.length** parameter (requires restart of the server). The default value is set to 6 months (4320 hours).

Important: Increasing the purging interval requires more server memory.

➤ Data files that contain data for snapshots will not be purged. Since the TSDB is distributed and the snapshots only reside on the commanding server, a mechanism for determining what time ranges should be preserved is required. When a snapshot is created, the commanding server will add the time range for the incident to the global list of preserved times. Upon snapshot deletion this time range will be removed. In order to allow multiple snapshots at the same time, each snapshot will create a new preservation entry; identical entries will not be merged since the deletion handling would not be possible. The UI will inform the server that a time range needs to be preserved independently of snapshots. This will allow preservation of data for non-snapshot purposes.

When a server starts the purging process it will retrieve the preserved times from the commanding server. Failure to retrieve the list will cancel the purging process and it will be re-run at a later time. If the commanding server hasn't been contacted after 1 week, purging will be run without consideration for the preservation list to prevent unbounded growth on distributed servers in the case that the commanding server has been permanently taken offline.

No data will be deleted until the size of the archive has exceeded the purging threshold (**persistence.purging.threshold** property). After that, the policy defined below will be used to delete files until the archive is less than this threshold. The threshold is set to 5G by default but you can change the value for **persistence.purging.threshold** in the **server.properties** file.

The purging process will scan the data files and identify all candidates for deletion. This process will operate on file sets. Since trend files are the largest consumer of disk space, and they require their summary files, data purging will be done based on trend file. For each trend file that exists, that doesn't have a preserved time range contained within it, a fileset containing all the files that are associated with that trend will be created. This will include the summary and tree file in the major summary that contains the trend file (only majors contain trend files), as well as all the minor summaries that index into that trend file.

Each fileset will have several values associated with it that will be used for determining which ones to purge. The "end time" of a fileset is the time of the last data point contained within the set. The "purge size" is the size of all the files in the summary that can be deleted.

Data File Retention and Trended Data in the Diagnostics Views

When Diagnostics displays trended metrics in the Diagnostics views, it attempts to show approximately sixty data points for each viewable period so that the trend that is presented will be meaningful and easy to understand. To arrive at the data points needed for each viewable period, the Diagnostics Server consolidates the data from the appropriate tier in the data files. For example, when you are looking at trended data for the last hour, one data point per minute is shown in the graph. These data points would have been created by consolidating raw data points for twelve 5-second time periods.

Pre-Installation Data Management Considerations

When preparing to install and configure a large Diagnostics Server, you should consider the following performance tuning recommendations:

➤ For maximum performance, the Diagnostics Server should be installed onto an empty or recently defragmented disk. The archive directory should be stored on that same disk. Alternatively, the archive directory could be mounted on an empty or recently defragmented disk.

Note: It is recommended that the disk used for diagnostics time series database that is stored in the **<diagnostics_server_install_dir>/archive** not be used for other disk activity (i.e. don't mount the **<diagnostics_server_install_dir>/archive** directory on the same disk that is used for your system files, temporary files etc...)

➤ To reduce fragmentation over time and increase system performance a separate disk (or partition) dedicated to the archive directory is recommended.

➤ Intensive background disk processes (such as disk de-fragmentation or virus scans) should be disabled on the disk where the archive directory is stored.

➤ Network file systems such as NFS or Samba should not be used.

Note: The better the raw performance of the disk that you dedicate to the archive directory of the Diagnostics Server, the more load the Diagnostics Server can handle. Ask your system administrator to make sure the disk mounted for the archive directory is a high-performance disk or array.

Backing Up Diagnostics Data

It is recommended that you back up the Diagnostics data regularly so that it can be restored in the case of a disk or system failure.

Backing up Data Remotely

Remote backup is possible by downloading the Diagnostics data files over HTTP to a local directory, and then backing up that directory using your normal backup procedures.

The Diagnostics Server supports the HTTP If-Modified-Since and Request-Range headers ("re-get") to allow standard HTTP mirroring software to download or incrementally update these files. If you choose to use your own HTTP mirroring software, work with your HP support representative to ensure the files are backed up in the proper order to ensure data integrity.

Diagnostics is installed with a remote backup script stored at **<diagnostics_server_install_dir/bin/remote-backup.sh.** The remote backup script is a UNIX script that uses the wget utility (http://www.gnu.org/software/wget/wget.html) to download over HTTP incrementally. On Windows, Cygwin (http://www.cygwin.com/) can be used to run this script.

Note: The backup script supplied with the Diagnostics Server backs up the data in a specific order. Failing to back up files in the correct order causes the restored backup to be unusable. It is therefore recommended to always use the supplied script to create data backups.

The following table lists the **remote-backup.sh** parameters:

Parameter	Description
-h	The host (or IP address) to download from
-o	The directory to store the backup in
-u	The HTTP username to use Default: admin
-p	The HTTP password to use Default: admin
-P	The HTTP port number to use (optional) Default: 2006
-r	The ID of the Diagnostics Server in Mediator mode being read from (for rhttp backups) (optional).

For example, to backup a Diagnostics Server running on the dragonfly machine into the **dragonfly-backup** directory:

```
% mkdir dragonfly-backup
% bin/remote-backup.sh -u admin -p secret -h dragonfly -o dragonfly-backup
```

The data is backed up in the following directories:

Data	Backup Directory
Server configuration	**etc/**
User custom views	**storage/userdata**

Data	Backup Directory
Raw performance history data	**archive/.../persistence/**
Symbol table	**archive/.../symboltable/**

Restoring Data After a Failure

The files in the backup directory are stored in the structure used by the Diagnostics Server.

To restore the time series database from the backup:

1 Install a clean Diagnostics Server. The Diagnostics Server is started automatically after the installation completes.

2 Shut down the Diagnostics Server.

3 Ensure that the Diagnostics Server has been shut down by verifying that there are no java/javaw processes in your process list. On Windows systems, you can use the Task Manager to do this and on UNIX systems, you can use ps.

4 Delete the **<diagnostics_server_install_dir>/archive** directory from Diagnostics Server.

5 Copy the database backup to replace the **<diagnostics_server_install_dir>/ archive**.

6 If the host name for the Diagnostics Server has changed since the backup was taken you must update the directory name that is based upon the Diagnostics Server host name to reflect the new host name.

   Rename **<diagnostics_server_install_dir>/archive/mediator-<host-name>** so that **<host-name>** reflects the new Diagnostics Server host name. For example, if host name in the backup was oldhost and the new host name is newhost you would change **<diagnostics_server_install_dir>/archive/ mediator-oldhost** to **<diagnostics_server_install_dir>/archive/ mediator-newhost**

Index Regeneration

When a restored Diagnostics Server is first started, the indexed data, which was not backed up, must be regenerated. Index regeneration is started automatically in the background and could take several hours to complete. While the indexes are regenerated the Diagnostics Server is able to receive events from probes; but some historical data cannot be displayed in the Diagnostics views until the restoration is complete.

Known Limitation

In HP Diagnostics version 7.50, binary data is written in the native byte order. This means that a Diagnostics data backup from a Big Endian machine cannot be restored and used on a Little Endian machine.

Handling Diagnostics Data when Upgrading Diagnostics

For details on handling Diagnostics data when upgrading, see Appendix G, "Upgrading Diagnostics and Other HP Software Products."

F

Diagnostics Component Configuration and Communication Diagrams

Component and Communication diagrams are provided to assist you as you install and configure the Diagnostics components.

This chapter includes:

➤ Communications with Business Availability Center on page 688

➤ Communications with LoadRunner and Performance Center on page 689

Note: The diagrams are intended to provide a high-level view, not provide an in-depth knowledge of the working of the components.

Communications with Business Availability Center

Web Browser

HTTP port 80

Business
Availability Center
(BAC)

HTTP
Port
2006

Diagnostics Server
(Commander Mode)

Listens for HTTP on port 2006
(Can also be used as a mediator)

Profiler

Optional
direct access
to Profiler

Diagnostics Server(s)
(Mediator Mode)

Listens for HTTP on port 2006
Listens on TCP/IP ports:
443, 2612, 23472

HTTP
Port
2006

HTTP port 35000

TCP/IP port 2612

.NET Probe(s)

Java Probe(s)

HTTP
Port
35000

Diagnostics Collector(s)

PCF (port 1414)

JDBC (port 1521)

JDBC (port 1433)

SAP JCO
(Ports used include:
3200, 3600, and
3300 + system number)

MQ Queue
Manager(s)

Oracle 10g
Instance(s)

SQL Server
Instance (s)

SAP NetWeaver – ABAP
Instance(s)

Communications with LoadRunner and Performance Center

G

Upgrading Diagnostics and Other HP Software Products

Instructions are provided for upgrading your current deployments of Diagnostics and of other HP Software products to use the components included in Diagnostics 8.00.

This chapter includes:

➤ Overview of Product Upgrade Paths on page 691

➤ General Upgrade Recommendations on page 692

➤ Diagnostics Compatibility with Earlier Diagnostics Versions on page 692

➤ Upgrading Diagnostics Components to Diagnostics 8.00 on page 692

➤ Diagnostics Compatibility with Other HP Software Products on page 702

Overview of Product Upgrade Paths

The following instructions help you understand how to upgrade HP Software products that you are currently using so that you can take advantage of the new features of Diagnostics 8.00.

You should consult with HP Software Customer Support if you have any questions about the information provided here.

General Upgrade Recommendations

The following recommendations are generally applicable when upgrading from earlier versions of Diagnostics or from versions of products that did not work with Diagnostics.

➤ Before you install the latest version of a component on the same host that was used for the earlier version of the component, make sure that the host meets the system requirements for the new component.

➤ You should upgrade the Diagnostics Server before upgrading a probe.

➤ You should contact HP Software Customer Support when you need to upgrade from an earlier version of Business Availability Center, or Performance Center.

Diagnostics Compatibility with Earlier Diagnostics Versions

The 8.00 version of the Diagnostics Server is capable of working with the following earlier probe versions:

➤ Java Probe 7.0, 7.50

➤ .NET Probe 7.0, 7.50

➤ Collectors 7.0, 7.50

Upgrading Diagnostics Components to Diagnostics 8.00

The following instructions guide you in the process of upgrading an existing Diagnostics component to Diagnostics 8.00.

This section includes:

➤ "Upgrading a Diagnostics Server" on page 693

➤ "Upgrading a Java Probe" on page 696

➤ "Upgrading a .NET Probe" on page 699

➤ "Upgrading a Diagnostics Collector" on page 700

Upgrading a Diagnostics Server

This section contains instructions for upgrading your Diagnostics server from 7.0/7.50 to 8.00.

Note:

➤ If you update a Diagnostics Server you must upgrade all of the Diagnostics Servers in your deployment. All Diagnostics Servers in your deployment must be running the same Diagnostics version.

➤ If you are an SaaS customer, contact SaaS Support for upgrade instructions.

Important: During the upgrade the keystore is overridden along with the JRE. As a result your trusted certificates will be unavailable after the upgrade.

To upgrade from a 7.0/7.50 Diagnostics Server to an 8.00 Diagnostics Server:

1 Shut down the current Diagnostics Server.

2 Make a backup copy of the current Diagnostics Server directory. By default this is C:\MercuryDiagnostics\Server on Windows and /opt/MercuryDiagnostics/Server on UNIX although a different directory may have been specified when the Server was installed.

Because the upgrade procedure requires you to uninstall the current Diagnostics Server, the backup copy can be used in case you need to start over.

3 Remove the etc.old directory if it already exists (C:\MercuryDiagnostics\Server\etc.old on Windows and /opt/ MercuryDiagnostics/Server/etc.old on UNIX).

4 Uninstall the current Diagnostics Server, but be sure to retain the modified files when prompted. For example, on Windows click **No to All** at the following prompt:
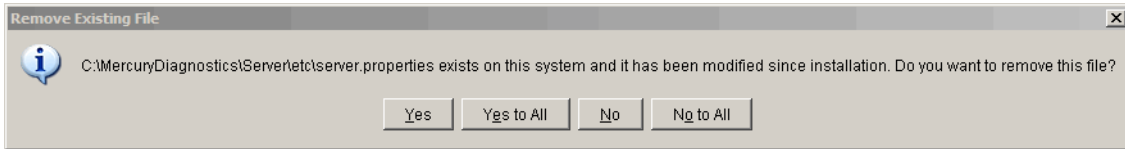
For information about uninstalling the Diagnostics Server, see Chapter 2, "Installing the Diagnostics Server".

5 Install the new Diagnostics Server into the same installation directory that you used for the previous version of the Diagnostics Server. Specify the same host and port for the new Diagnostics Server that was being used by the previous Diagnostics Server.

Reply Yes to the following message:
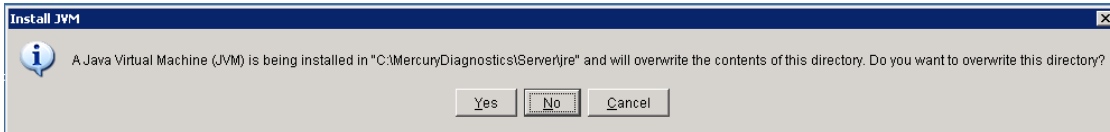
6 If installing on Windows, stop the Diagnostics Server.

On Windows, the Diagnostics Server is started automatically when the installer finishes. On UNIX the server is not automatically started so you do not need to stop it.

7 The installation created an **\etc.old** directory which is the previous Diagnostic Server's data. Compare the **\etc** directory and the**\etc.old** directory so that you can determine the differences between the two. It may be helpful to use a diff/merge tool for this purpose.

Apply any differences that were caused by the customizations that were made to the **\etc.old** directory to the **\etc** directory so that they will not be lost. Here are some common changes:

Property File	Configuration Properties To Be Copied to the New Diagnostics Server
alerting.properties	SNMP and SMTP servers, mail addresses
security.properties	If the system is set up for SSL mode, all parameters should be updated and certificates manually copied to the new /etc folder
server.properties	Timeout/Trimming settings
thresholds.configuration	Update any changes

 8 If the system is integrated with LoadRunner or Performance Center, copy **run_id.xml** from the **\etc.old** directory to the new **\etc** directory to ensure that the Run ID is properly incremented for future runs.

 9 If you are upgrading the Diagnostics Server in Commander mode, copy the **DiagnosticsServer.lic** file from the **\etc.old** directory to the new **\etc** directory.

10 Start the Diagnostics Server.

11 Clear your browser's cache and restart the browser before you attempt to access the Diagnostics UI.

12 You can verify that the upgraded Diagnostics Server is running by checking the version in the System Health graph. Browse to the URL http://<commanding_server>:<port>/registrar/health. Double-click the Diagnostics Server icon. The version under Configuration should be 8.00.x.x if the upgrade was successful and the Diagnostics Server was restarted.

13 Once you are satisfied that the Diagnostic Server has been upgraded successfully, remove the backup copy you created in Step 2.

Note: When you open the custom views that were created in an earlier version of Diagnostics for the first time in Diagnostics 8.00, Diagnostics will upgrade the view for any changes that are necessary because of changes that were made to the functionality of Diagnostics. When Diagnostics changes your custom views a message is displayed to let you know that your custom view has been modified.

Upgrading a Java Probe

Note: You must upgrade the Diagnostics Server before upgrading the probes that are connected to it because Diagnostics Servers are not forward compatible.

To upgrade a 7.0/7.50 Java Probe to 8.00:

Note: The new probe installation will not begin monitoring your applications until you have updated the startup scripts to start the new probe and restarted the applications as described in these instructions.

1 Install the Diagnostics Agent for Java into a different directory than the current probe's installation directory.

In release 7.50 or later the capabilities of the Java Probe and TransactionVision Sensor are combined into the *Diagnostics/TransactionVision Java Agent.* Due to the new structure of the Agent, the same directory as that used prior to release 7.50 cannot be used.

During the installation, be sure to:

➤ configure the Java Agent to be Diagnostics Probe that works with a Diagnostics Server or as a standalone Diagnostics Profiler. The Java Agent can also be configured to be a TransactionVision Sensor if desired.

➤ for the agent name, use the same probe name as used by the previous probe

➤ for the agent group name, use the same group name as used by the previous probe

➤ for the mediator server name and port, use the same information as used by the previous probe

This ensures that the persisted data for your application will match up with the metrics captured by the new probe.

2 The installer creates a **<probe_install_dir>\etc** directory in the new installation directory.

In 7.50 or later releases, the default directory of <probe_install_dir> has changed. The default location is C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent on Windows and /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent on UNIX.

3 Compare the new probe's **\etc** directory and the previous probe's **\etc** directory so that you can determine the differences between the two.

HP recommends that you execute the **Property Scanner** utility which will indicate the differences (properties and points) between two different Java Agent installations. To execute the Property Scanner utility, change the current directory to **<probe_install_dir>/contrib./JASMUtilities/Snapins** and execute the **runPropertyScanner.cmd –console** (**.sh** for Unix) command as follows:

runPropertyScanner –console –diffOnly yes –Source1 ..\..\..\etc –Source2 OtherEtc

Sample Input:

C:\MercuryDiagnostics\JavaAgent8\DiagnosticsAgent\contrib\JASMUtilities\Sna pins>runPropertyScanner -console -diffOnly yes -Source1 C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\etc -Source2 C:\MercuryDiagnostics\JavaAgent8\DiagnosticsAgent\etc

Sample Output:

****** Property dispatcher.properties:stack.trace.method.calls.max PropertyFile=dispatcher.properties Property=stack.trace.method.calls.max

Source1=
Source2=1000

Apply any differences that were caused by the customizations that you made to the previous probe's **\etc** directory to the new probe's **\etc** directory so that they will not be lost. You should look for the following changes:

Property File	Configuration Properties To Be Copied to the New Diagnostics Server
auto_detect.points	See the following step for upgrade instructions.
capture.properties	Depth and latency trimming
inst.properties	Copy custom points that you have created and points that you have modified from the **auto_detect.points** file in the old **\etc** directory to the new **\etc** directory. Be sure to check the points for RMI, LWMD, args_by_class when looking for points you may have modified.
dispatcher.properties	➤ **minimum.sql.latency** - this is a new property that should be set correlated with the Diagnostics Mediator property **server.properties/ sql.latency.trim** ➤ sql.parsing.mode
dynamic.properties	**cpu.timestamp.collection.method**
metrics.config	Verify that any metric that you uncommented in the previous version is also uncommented in the new version so that you can continue to use the metrics that you are used to.
security.properties	If the system is set up for SSL mode, set all properties and copy the certificates from the old property file to the property file.

4 If you are upgrading from 7.0, update the applications startup script to point to the upgraded probe installation. This includes the -javaagent or -Xbootclasspath.

If you are upgrading from 7.50, you may rename the existing JavaAgent directory to JavaAgent_7.5 and the new agent directory to JavaAgent and thus you do not need to update the application startup scripts.

5 At an approved time, shut down the applications that were being monitored by the old probe.

6 Restart the applications to allow the new version of the probe to begin monitoring your applications.

7 Clear your browser's cache and restart the browser before you attempt to access the Java Diagnostics Profiler user interface. Failure to do this may result in a size mismatch error message.

8 You can verify that the upgraded Java Probe is running and properly registered, by checking the version in System Health. Browse to the System Health <u>http://<commanding_server>:<port>/registrar/health</u>, and double-click the probe icon. The version listed under Configuration should be 8.00.x.x, if the upgrade was successful and the Java Probe was started.

9 When all your applications have been migrated over to be version 8.00 and everything is working properly, you can uninstall the earlier version of the Java probe.

Upgrading a .NET Probe

Note: You must upgrade the Diagnostics Server before upgrading the .NET Probes that are connected to it because Diagnostics Servers are not forward-compatible.

Note: The 7.50 or later .NET Agent includes a SOAP Extension Handler which may cause Web applications that use SOAP to restart when the .NET Agent is installed.

The .NET Probe upgrade process differs depending on which version of the probe you are upgrading.

To upgrade a 7.0/7.50 .NET Probe to 8.00:

1 Install the new Diagnostics Probe for .NET.

The upgrade will take effect when the probed applications are restarted.

To force the upgrade to take effect:

a Shut down all applications that are being monitored by the current .NET Probe.

b Restart IIS.

c Restart the applications that were being monitored by the old probe.

2 You can verify that the upgraded .NET Probe is running and properly registered, by checking the version in System Health. Browse to http://<commanding_server>:<port>/registrar/health, and double-click the probe icon. The version listed under Configuration should be 8.00.x.x, if the upgrade was successful and the .NET Probe was started.

Upgrading a Diagnostics Collector

Note: You must upgrade the Diagnostics Server before upgrading the Collectors that are connected to it because Diagnostics Servers are not forward-compatible.

To upgrade a 7.0/7.50 Diagnostics Collector to 8.00:

1 Stop or shut down the Diagnostics Collector that you want to upgrade.

2 Back-up the directory for the current collector installation.

By default this is C:\MercuryDiagnostics\Collector on Windows and /opt/MercuryDiagnostics/Collector on UNIX.

Because the upgrade procedure requires you to uninstall the current Diagnostics Collector, the backup copy can be used in case you need to start over.

3 Uninstall the current Diagnostics Collector, but be sure to retain the modified files when prompted. For example, on Windows:

Remove Existing File	×
(i) C:\MercuryDiagnostics\Collector\etc\collector.properties exists on this system and it has been modified since installation. Do you want to remove this file?	
<u>Y</u>es Y<u>e</u>s to All <u>N</u>o N<u>o</u> to All	
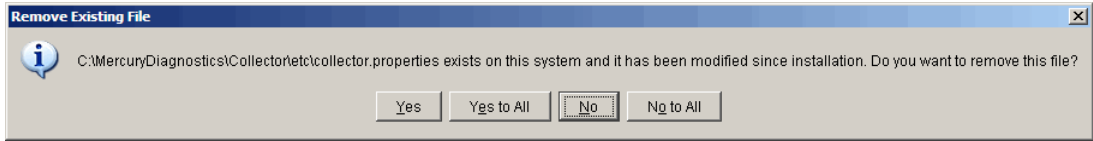
For information about uninstalling the Collector, see "Uninstalling the Diagnostics Collector" on page 99.

4 Install the new Collector into the same installation directory that was used for the old version of the Collector.

Be sure to use the same Collector name and Mediator host in order to ensure that the persisted data for the application will match up with the metrics captured by the new collector.

You can determine the old Collector name by viewing the backed up **<collector_install_dir>\etc\collector.properties** file.

5 If installing on Windows, stop the Collector.

The Collector is started automatically when the installer finishes.

On UNIX the Collector is not automatically started so you do not need to stop it.

6 Compare the new **\etc** directory and the old **\etc** directory to determine the differences between the two. (It may be helpful to use a diff/merge tool for this purpose.)

Apply any differences that were caused by the customizations that you made to the old **\etc** directory to the new **\etc** directory so that they will not be lost.

Property File	**Configuration Properties To Be Copied to the New Diagnostics Server**
mq-config.xml	If the collector is monitoring an MQ environment.
oracle-config.xml	If the collector is monitoring an Oracle environment.

Property File	Configuration Properties To Be Copied to the New Diagnostics Server
sqlserver-config.xml	If the collector is monitoring an SQL Server environment. You must remove the databaseName attribute from the sqlserver-config.xml file. In the 8.00 collector, databaseName is automatically discovered.
r3config.xml	If the collector is monitoring an R3 environment.
security.properties	If the system is set up for SSL mode, set all properties and copy the certificates from the old property file to the property file.

7 Start the Diagnostics Collector.

8 You can verify the upgraded Collector by checking the version in the System Health graph. Browse to http://<commanding_server>:<port>/registrar/health, and double-click the Collector icon. The version listed under Configuration should be 8.00.x.x, if the upgrade was successful and the Collector was started.

9 Once you are satisfied that the Diagnostic Collector has been upgraded successfully, remove the backup copy you created in Step 2.

Diagnostics Compatibility with Other HP Software Products

The following table shows the compatibility of Diagnostics 8.00 components with versions of other HP Software products.

For the most recent information on version compatibility refer to the Diagnostics Product Availability Matrix at http://support.openview.hp.com/sc/support_matrices.jsp.

HP Software Product	Version Compatible with Diagnostics 8.00
LoadRunner	9.10, 9.50
Performance Center	9.10, 9.50

HP Software Product	Version Compatible with Diagnostics 8.00
Business Availability Center	7.0, 7.50, 8.00
TransactionVision	8.00

H

Troubleshooting HP Diagnostics

Troubleshooting tips are provided for problems that may occur when using HP Diagnostics.

This chapter includes:

➤ Component Installation Interrupted on a Solaris Machine on page 705

➤ Java Probe Fails to Operate Properly on page 706

➤ Java Probe Throttling Seems Excessive on page 706

➤ Error During WAS Startup with Diagnostics Profiler for Java on page 707

➤ Missing Server-Side Transactions on page 708

➤ Emergency Reserve Buffers Exhausted Error on page 708

➤ Java Agent Support Collector on page 709

Component Installation Interrupted on a Solaris Machine

If a component installer on a Solaris machine is interrupted before it has finished installing the component, there is no option for automatically uninstalling or reinstalling the component. You must manually clean up the partial installation of the component before you can start the installation again.

To manually clean up after an interrupted installation:

1 Clean the installation directory.

2 Delete **~/vpd.properties** and **~/vpd.patches**.

3 Delete the Solaris directories: **/var/sadm/pkg/IS*** and **/var/sadm/pkg/MERQ**.

Java Probe Fails to Operate Properly

If the Java Probe does not operate properly, check whether the **ClassLoader.class** file located in the folder **<probe_install_dir>\classes\boot\java\lang** was created during the installation process.

If the file was not created, run the JRE Instrumenter to create it. See Chapter 6, "Running the JRE Instrumenter."

Java Probe Throttling Seems Excessive

By reviewing the probe logs, you can determine when throttling is being started and stopped. If throttling is being engaged more than you would like you should check to make sure that the probe thread configuration is synchronized with the number of threads that the application server has been configured to allow. If the application server has more threads than the probe can handle, throttling is engaged sooner and may not be disengaged.

For information on tuning the probe to reduce throttling see "Controlling Probe Throttling" on page 398.

Error During WAS Startup with Diagnostics Profiler for Java

Symptoms:

Class Loader errors occur when starting WAS with the Diagnostics Profiler for Java.

Reason:

Additional classes need to be excluded from the instrumentation.

Solution:

1 Open the property file, **<probe_install_dir>\etc\inst.properties**

2 Update the **classes.to.exclude** property to exclude **!com\.ibm\..*** by appending the class to the end of the existing values.

```
classes.to.exclude=!iaik\.security\..*,!c8e\..*,!org\.jboss\.net\.protocol\.file\.Handler,!
org\.jboss\.net\.protocol\.file\.FileURLConnection,!.*ByCGLIB.*,!com\.ibm\..*
```

Missing Server-Side Transactions

Symptoms:

The server requests for each Probe are displayed in Diagnostics but the BPM transactions that are associated with the server requests are not displayed.

There are two symptoms to look for in the **server.log** file:

"**not dropping at least one transaction that timed out**" – this indicates that a transaction has not received any data for a period of time (10m by default) and has not received the ELT. This warning is issued infrequently, and only when the transaction times out…after this warning you should see the transaction data in the UI. For more information on ELT see "Reducing Diagnostics Server Memory Usage" on page 370.

"**Late data received for time period that was already persisted. Adjusting data by…**" – this indicates that the server received an ELT unreasonably late, but before the transaction timed out. The data will be reported, but not at the same time that BAC or SaaS reported it.

Reason:

If you do not see either of the log messages listed above and there is no transaction data the most likely cause is the BPM is not running the scripts.

Solution:

1 Verify that BPM is running in Business Availability Center or SaaS and that the BPM is running.

2 Verify the state of the profile in the BPM Console.

Emergency Reserve Buffers Exhausted Error

Symptoms:

Some Diagnostics data loss is occurring and the following error appears in the probe log file:

"Emergency reserve buffers exhausted. Application Threads will begin dropping events."

Reason:

The log entry indicates that the application load is too high, the number of application threads is too large, or that the application is excessively instrumented.

Solution:

If the application uses a large number of threads, the following modification to **capture.properties** can be tested as a solution.

➤ Double the number of buffers in these properties:

maximum.private.buffer.count

gentle.reserve.buffer.count

hard.reserve.buffer.count

➤ Reduce the size of the buffers by 50% in this property:

maximum.buffer.size

Java Agent Support Collector

The **runSupportSnapshot** utility creates a .zip file containing the entire set of files relevant to troubleshooting one or more instances of the Java agent in a Diagnostics or TransactionVision deployment environment.

The .zip file contains the following:

➤ Files from the <Diagnostics_probe_install_dir>\etc directory

➤ Files from the <Diagnostics_probe_install_dir>\log directory

➤ Files from the <TransactionVision_sensor_install_dir>\config directory

➤ Files from the <TransactionVision_sensor_install_dir>\logs directory

➤ Property Scanner report, which compares two agent directories and reports differences between property files, points files, and XML files (TransactionVision Sensors only).

➤ Probe or Sensor instance information, including property settings. For agents running in 1.5 JVMs, environment variables, stack dumps, and class loader information is also included.

To run runSupportSnapshot:

1 Navigate to **<Diagnostics_probe_install_dir>\contrib\JASMUtilities\Snapins**.

2 Execute **.\runSupportSnapshot.cmd -console** on Windows, or **./runSupportSnapshot.sh -console** on UNIX or Linux.

3 A .zip file is created.

I

Using UNIX Commands

When running an installation on a UNIX platform, you can usually follow the instructions that appear on the screen. The on-screen instructions can be confusing at times.

If something is unclear, use the following guidelines:

➤ To select an option from a list of options, type the number corresponding to the option and press **Enter**. Then type 0 and press **Enter** again to confirm your choice.

➤ When selecting multiple options, for each selection type the corresponding number and press **Enter**. Only when you have finished selecting all the options that you want to select, type 0 and press **Enter** again to confirm your choices.

➤ If you have selected an option and want to clear it, retype the corresponding number, or type the number of another option, and press **Enter**. Then type 0 and press **Enter** again to confirm your choice.

➤ When entering information at a prompt:

➤ To accept a default value that is displayed at the prompt, press **Enter**.

➤ Type the information, and press **Enter** to continue.

➤ To continue with the next step of an installation, type 1 to select **Next**, and press **Enter**.

➤ To go back to previous prompts to make changes, type 2 to select **Previous**, and press **Enter**.

➤ To cancel an installation, type 3 to select **Cancel**, and press **Enter**.

➤ To redisplay a prompt, type 4 to select **Redisplay**, and press **Enter**.

J

Using Regular Expressions

When you specify the instrumentation definitions for each probe instance in the capture points file, you can use regular expressions for most of the arguments in a point.

A regular expression is a string that specifies a complex search phrase. By using special characters, such as a period (.), asterisk (*), caret (^), and brackets ([ ]), you can define the conditions of a search.

Note: Regular expressions in Diagnostics must be prefaced with an exclamation point.

By default, Diagnostics treats all characters in a regular expression literally, except for the period (.), hyphen (-), asterisk (*), caret (^), brackets ([ ]), parentheses (()), dollar sign ($), vertical line (|), plus sign (+), question mark (?), and backslash (\). When one of these special characters is preceded by a backslash (\), Diagnostics treats it as a literal character.

This chapter includes:

➤ Common Regular Expression Operators on page 714
➤ Combining Regular Expression Operators on page 721

Common Regular Expression Operators

This section describes some of the more common operators that can be used to create regular expressions.

Note: For a complete list and explanation of supported regular expression characters, refer to the Regular Expressions section in the Microsoft VBScript documentation.

Operator	Used for
(\ )	Rendering Special Characters Literal
	Creating Special Characters out of Literal Characters
(.)	Matching Any Single Character
([*xy*])	Matching Any Single Character in a List
([^*xy*])	Matching Any Single Character Not in a List
([*x-y*])	Matching Any Single Character within a Range
(*)	Matching Zero or More Specific Characters
(+)	Matching One or More Specific Characters
(?)	Matching Zero or One Specific Character
(( ))	Grouping Regular Expressions
(	)	Matching One of Several Regular Expressions
(^)	Matching the Beginning of a Line
($)	Matching the End of a Line
(\w)	Matching Any AlphaNumeric Character Including the Underscore
(\W)	Matching Any Non-AlphaNumeric Character

Using the Backslash Character

A backslash (\) can serve two purposes. It can be used in conjunction with a special character to indicate that the next character be treated as a literal character. For example, \. would be treated as period (.) instead of a wildcard (see "Matching Any Single Character" on page 716).

Alternatively, if the backslash (\) is used in conjunction with some characters that would otherwise be treated as literal characters, such as the letters n, t, w, or d, the combination indicates a special character. For example, \n stands for the newline character.

For example:

➤ w matches the character w

➤ \w is a special character that matches any word character including underscore

➤ \\ matches the literal character \

➤ \(matches the literal character (

For example, if you were looking for a file called:

filename.ext

the period would be mistaken as an indication of a regular expression. To indicate that the period is not part of a regular expression, you would enter it as follows:

filename\.ext

Note: If a backslash character is used before a character that has no special meaning, the backslash is ignored. For example, \z matches z.

Matching Any Single Character

A period (.) instructs Diagnostics to search for any single character (except for \n). For example:

welcome.

matches **welcomes**, **welcomed**, or **welcome** followed by a space or any other single character. A series of periods indicates the same number of unspecified characters.

To match any single character including \n, enter:

(.|\n)

For more information on the **( )** regular expression characters, see "Grouping Regular Expressions" on page 718. For more information on the | regular expression character, see "Matching One of Several Regular Expressions" on page 719.

Matching Any Single Character in a List

Square brackets instruct Diagnostics to search for any single character within a list of characters. For example, to search for the date 1967, 1968, or 1969, enter:

196[789]

Matching Any Single Character Not in a List

When a caret (^) is the first character inside square brackets, it instructs Diagnostics to match any character in the list except for the ones specified in the string. For example:

[^ab]

matches any character except **a** or **b**.

Note: The caret has this special meaning only when it is the first character displayed within the brackets.

Matching Any Single Character within a Range

To match a single character within a range, you can use square brackets ([ ]) with the hyphen (-) character. For example, to match any year in the 1960s, enter:

196[0-9]

A hyphen does not signify a range if it is displayed as the first or last character within brackets, or after a caret (^).

For example, [-a-z] matches a hyphen or any lowercase letter.

Note: Within brackets, the characters ".", "*", "[" and "\" are literal. For example, **[.*]** matches **.** or *****. If the right bracket is the first character in the range, it is also literal.

Matching Zero or More Specific Characters

An asterisk (*) instructs Diagnostics to match zero or more occurrences of the preceding character. For example:

ca*r

matches **car**, **caaaaaar**, and **cr**.

Matching One or More Specific Characters

A plus sign (+) instructs Diagnostics to match one or more occurrences of the preceding character. For example:

ca+r

matches **car** and **caaaaaar**, but not **cr**.

Matching Zero or One Specific Character

A question mark (?) instructs Diagnostics to match zero or one occurrences of the preceding character. For example:

ca?r

matches **car** and **cr**, but nothing else.

Grouping Regular Expressions

Parentheses (()) instruct Diagnostics to treat the contained sequence as a unit, just as in mathematics and programming languages.

Using groups is especially useful for delimiting the argument(s) to an alternation operator (|) or a repetition operator (* , + , ? , { }).

Matching One of Several Regular Expressions

A vertical line (|) instructs Diagnostics to match one of a choice of expressions. For example:

foo|bar

causes Diagnostics to match either **foo** or **bar**.

fo(o|b)ar

causes Diagnostics to match either **fooar** or **fobar**.

Matching the Beginning of a Line

A caret (^) instructs Diagnostics to match the expression only at the start of a line, or after a newline character.

For example:

book

matches **book** within the lines **book**, **my book**, and **book list**, while

^book

matches **book** only in the **lines book** and **book list**.

Matching the End of a Line

A dollar sign ($) instructs Diagnostics to match the expression only at the end of a line, or before a newline character. For example:

book

matches **book** within the lines **my book**, and **book list**, while a string that is followed by ($), matches only lines ending in that string. For example:

book$

matches **book** only in the line **my book**.

Matching Any AlphaNumeric Character Including the Underscore

\w instructs Diagnostics to match any alphanumeric character and the underscore (A-Z, a-z, 0-9, _).

For example:

\w* causes Diagnostics to match zero or more occurrences of the alphanumeric characters—**A-Z**, **a-z**, **0-9**, and the underscore (_). It matches **Ab**, **r9Cj**, or **12_uYLgeu_435**.

For example:

\w{3} causes Diagnostics to match 3 occurrences of the alphanumeric characters—**A-Z**, **a-z**, **0-9**, and the underscore (_). It matches **Ab4**, **r9_**, or **z_M**.

Matching Any Non-AlphaNumeric Character

\W instructs Diagnostics to match any character other than alphanumeric characters and underscores.

For example:

\W matches **&**, *****, **^**, **%**, **$**, and **#** .

Combining Regular Expression Operators

You can combine regular expression operators in a single expression to achieve the exact search criteria you need.

For example, you can combine the '.' and '*' characters to find zero or more occurrences of any character (except \n).

For example,

start.*

matches **start**, **started**, **starting**, **starter**, and so forth.

You can use a combination of brackets and an asterisk to limit the search to a combination of non-numeric characters. For example:

[a-zA-Z]*

To match any number between 0 and 1200, you need to match numbers with 1 digit, 2 digits, 3 digits, or 4 digits between 1000-1200.

The regular expression below matches any number between 0 and 1200.

([0-9]?[0-9]?[0-9]|1[01][0-9][0-9]|1200)

K

Multi-Lingual User Interface Support

The Diagnostics user interface (UI) can be viewed in multiple languages in your Web browser. This applies in a Windows environment, when Diagnostics is integrated with Business Availability Center or running in standalone mode (no integration).

If Diagnostics is integrated with LoadRunner or Performance Center, the display language of the UI is determined by the client locale setting (defined in the Regional Settings of your operating system)

This appendix explains how to view the Diagnostics user interface in a specific language. The Diagnostics UI can be viewed in the following languages in your Web browser:

Language	Language preference in Web browser
English	English
Simplified Chinese	Chinese (China) [zh-cn], Chinese (Singapore) [zh-sg]
Korean	Korean [ko]
Japanese	Japanese [ja]

You use the language preference option in your browser to select how you view Diagnostics. The language preference chosen affects only the user's local machine and not the Diagnostics Server or any other user accessing the same Diagnostics Server.

To view the Diagnostics UI in a specific language:

1 Install the appropriate language's fonts on your local machine if they have not yet been installed. If you choose a language in your Web browser whose fonts have not been installed, the Diagnostics user interface uses the default language of your local machine.

Assume, for example, that the default language on your local machine is English and the Web browser is configured to use Japanese. If Japanese fonts are not installed on the local machine, the Diagnostics user interface is displayed in English.

2 If you are using Internet Explorer, configure the Web browser on your local machine to select the language in which you want to view the Diagnostics user interface. For details, see http://support.microsoft.com/kb/306872/en-us.

Continue with step 4.

3 If you are using FireFox, configure the Web browser on your local machine as follows:

a Select **Tools** > **Options** > **Advanced**. Click **Edit Languages**. The Language dialog box opens.

b Highlight the language in which you want to view Diagnostics.

If the language you want is not listed in the dialog box, expand the **Select language to add** list, select the language, and click **Add**.

c Click **Move Up** to move the selected language to the first row.

d Click **OK** to save the settings. Click **OK** to close the Language dialog box.

4 Close your existing browser and open Diagnostics in a new browser. The Diagnostics user interface is displayed in the selected language.

L

Data Exporting

The metric data collected by Diagnostics can be archived directly to a third-party database where it can be retained as needed for legal purposes or it can be formatted into reports as supported by the database.

This data export is accomplished by using XPath-like queries to pull the desired metrics from the Diagnostics Time Series database (TSDB), which is the repository for all persistent Diagnostics data. For information about the TSDB, see "Diagnostics Data Management" on page 673.

This chapter includes:

Task 1: Prepare the target database

The target database for the exported data can be an SQL Server 2005, SQL Server 2008 or Oracle 10g database to which the commanding Diagnostics Server has access.

The data export performed by the Diagnostics server automatically creates the schema and tables in the target database. The target database should have at least 1 GB of space available. During the first few export operations, you should monitor the size of the database to see if more space is needed.

To connect to the target database, you need to specify the login credentials for a user that has read/write privileges to the database and has table definition privileges.

Task 2: Determine which metrics you want to export

You can control which metrics are exported in different ways. You can specify to get all metrics for a particular entity type. You can exclude specific metrics from that grouping or you can specify to include only specific metrics.

Metrics are grouped by the entity type to which they apply as well as other criteria. The following entity type groupings are the most commonly used:

➤ /probegroup/probe

Metrics for all probes across all probe groups.

➤ /probegroup/probe/fragment

Metrics for all server requests across all probe groups and probes.

➤ /probegroup/index[name='rollup_fragment']/fragment

Metrics for server requests that are rolled up by probe across all probe groups.

➤ /probegroup/probe/index[name='services']/service

Metrics for Web services (excluding operation) across all probe groups and probes.

➤ /index [equals(name,'apps')]/app/app_metrics

Metrics for a particular application.

➤ /probegroup/probe[equals(probeType, 'Oracle')]

Metrics for all Oracle collectors.

➤ /probegroup/probe[equals(probeType, 'SqlServer')]

Metrics for all SqlServer collectors.

➤ /host

Metrics for all hosts (various system metrics).

➤ /txn

Metrics for all BPM transactions.

Note: The data export operation exports metric data only, that is counts, latencies, and averages. No instance data or status data is exported.

The following tables lists the metrics and the categories to which the belong:

Category	Metric
Classes	Classes Currently Loaded Classes Loaded Classes Unloaded
Dynamic Caching	Caching Current Cache Size Caching Max Cache Size
EJB	EJB Activates EJB Activation Time EJB Committed Transactions / sec EJB Concurrent Active Methods EJB Concurrent Live Beans EJB Create Time EJB Creates EJB Drain Size EJB Drains From Pool EJB Frees EJB Gets Found EJB Gets From Pool EJB Instantiates EJB Load Time EJB Loads EJB Passivates EJB Passivation Time EJB Passive Beans EJB Pools Size EJB Ready Beans EJB Remote Time EJB Removes EJB Response Time EJB Returns Discarded EJB Returns To Pool EJB Rolled Back Transactions / sec EJB Store Time EJB Stores

Category	Metric
EJB (Continued)	EJB Timed Out Transactions / sec EJB Total Method Calls EJB-Cache Access / sec EJB-Cache Beans Cached EJB-Cache Get Failures / sec EJB-Pool Access / sec EJB-Pool Available Beans EJB-Pool Beans in Use EJB-Pool Current Waiters EJB-Pool Get Failures / sec EJB-Pool Get Timeouts / sec
Execute Queues	Execute Queues Idle Threads Execute Queues Pending Requests Execute Queues Requests / sec Execute Queues Total Threads
GC	GC Collections/sec GC Time Spent in Collections
Http Status	5xx-6xx
J2C Connections	J2C Connection Handles J2C Connection Released J2C Connections Allocated J2C Connections Closed J2C Connections Created
JDBC	JDBC Connections Created/sec JDBC Create Connection Delay JDBC Current Capacity JDBC Execute Statement JDBC Leaked Connections JDBC Reconnect Failures JDBC Requests Waiting for Connection JDBC Statement Cache Accesses / sec JDBC Statement Cache Hits / sec JDBC Statement Cache Size JDBC Total Connections Opened JDBC Wait Seconds High

For descriptions of these metric, see Appendix A in the *Diagnostics User's Guide*.

You specify the group or individual metric to export as described in Task 4.

Task 3: Determine the frequency and the recovery period

Each export operation has a specified frequency which controls how often it occurs and therefore the granularity of the returned metrics. The recommended frequency is 1h (hourly) which means that every hour the export operation is run. Other options for frequency are: 5m and 1d.

Note: The data export operation can be run as frequently as desired however the data export operation affects the Diagnostics Server performance. The higher the frequency, the greater the load on the server.

You can also specify a frequency recovery period. The recovery period is used only when the commanding Diagnostics Server is shut down or becomes otherwise unavailable. This value tells the commanding Diagnostics Server how far back to go to resume running the data export operations when it resumes operation.

The frequency recovery period formula is:

(current time) - (recovery-periods * frequency)

For example, assume a commanding Diagnostics Server was not active for 24 hours. A data export operation with an hourly frequency has missed a minimum of 23 executions. By default, the data export operations would start querying at the time the outage occurred (24 hours in the past). The metrics for the hourly data have already been aggregated into larger buckets and therefore the returned metrics are not meaningful.

However, if the recovery periods is specified as 6h, the hourly task would go back 6 hours in time (instead of 24) to start it's querying against the TSDB. These metrics are meaningful.

Set the <frequency> and <recovery-periods> elements as described in Task 4.

Task 4: Modify the data export configuration file

The queries that export the Diagnostics data are defined in
<diagnostics_server_install_dir>/etc/data-export-config.xml file of the
Diagnostic Server running in Commander mode.

Follow these steps to set up this file:

1 Make a backup copy of the <diagnostics_server_install_dir>/etc/
data-export-config.xml file if desired.

2 Open the <diagnostics_server_install_dir>/etc/data-export-config.xml file
for editing.

3 Locate the <enabled> element and set it to true:

```
<enabled>true</enabled>
```

This element is used to turn on or off the data export operation. You should
disable the data export operation when it is not needed to avoid
unnecessary system overhead. By default the data export operation is
disabled.

4 Locate the <customer name> element and set it to the customer name:

Unless you are a SaaS customer, the customer name should always be
Default Client.

```
<customer name='Default Client">
```

5 Locate the <db-target> element and enter the driver name, connection URL,
user name and password (encrypted or plain text) for the target database.

For example, for SQL Server with an encrypted password:

```
<db-target>
   <driver>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver>
   <connection-url>jdbc:sqlserver://testapps.hp.com:1433;databaseName=DIAG</
connection-url>
   <username>sa</username>
   <encrypted-password>OBF:1ym51y0s1uo71z0f1unr1y0y1ym9</encrypted-password>
   <batchsize>200</batchsize>
</db-target>
```

For example, for Oracle with an unencrypted password:

```
<db-target>
   <driver>oracle.jdbc.driver.OracleDriver</driver>
   <connection-url>jdbc:oracle:thin:@testapps.hp.com:1521:ORCL</connection-url>
   <username>diagfan</username>
   <password>tiger2</password>
   <connection-property  name="oracle.jdbc.defaultNChar" value="true"/>
</db-target>
```

For Oracle databases, the **oracle.jdbc.defaultNChar** property must be set to true when UTF8/UTF15 character support is required.

To encrypt a database password, use the Diagnostics password encryptor. See "Password Obfuscation" on page 94.

For the <batchsize> element, specify the batch size in units used for optimal JDBC PreparedStatement execution. By default, this is set to 100. Large implementations with large payloads require adjustments to the default.

6 For each set of metrics that you want to export, specify the following:

<id>: A name which identifies the query being defined. Must be unique to this data-export-config.xml file.

<frequency>: A string value that specifies how often to run the query. Options are: 1h, 5m, and 1d.

<recovery-periods>: specifies how far back in time to start querying after an outage occurs.

<entity-path>: One of the entity path groupings described in Task 2.

<init-query-time> or <init-query-periods>: A time in the past at which the query starts. <init-query-time> is a time value specified in standard XSD time format. <init-query-periods> is an integer that is multiplied by the frequency to determine the query time. If omitted, the query runs at the next frequency boundary.

For example, the following entry creates a query that runs every hour and returns all of the metrics for all probes in all probe groups. If an outage occurs, only recover back 2 hours from current time:

```
<query id="Probes" frequency="1h" recovery-periods="2">
   <entity-path>/probegroup/probe</entity-path>
</query>
```

The following entry creates a query that runs every hour and returns every hour's rollup fragment latency metrics for all probe groups:

```
<query id="Aggregate-SRs" frequency="1h" recovery-periods="2">
   <entity-path>/probegroup/index[name='rollup_fragment']/fragment</entity-path>
</query>
```

7 Optionally, each query can have an include or exclude filter applied to the query specified in the <entity-path> element. The include filter elements must be specified first before the exclude filter elements.

For either filter, specify:

<name>: A regular expression to match against the metric name to filter. A value of "" matches all metrics.

<category>: A regular expression to match against the category name to filter. A value of "" matches all categories.

<order>: For multiple include or exclude filters, the order in which to process the filters.

For example, the following entry returns metrics for Database metrics only:

```
<query id="Probes" frequency="5m" recovery-periods="2">
   <entity-path>/probegroup/probe</entity-path>
   <metric-include-filter order="1" name="" category="Database" />
   <metric-exclude-filter order="1" category="" />
</query>
```

The following example returns all metrics for EJBs excluding EJB-Poll metrics.

```
<query id="EJBStats" frequency="5m" recovery-periods="2">
 <entity-path>/probegroup/probe</entity-path>
 <metric-include-filter order="1" name="" category="EJB" />
 <metric-exclude-filter order="1" name="EJB-Pool" />
</query>
```

For more information about regular expressions; see Appendix J, "Using Regular Expressions".

8 Optionally, specify the data retention rules for the extracted data by specifying the <purge> element. This prevents the database from running out of storage.

For example, the following entry causes data that is over 24 hours old (retention="1d") to be deleted from the target database. The purge operation is initiated every hour (frequency="1h") and any needed purge operations use 1hr increments (purgeInterval="1h") to purge the data, thus reducing overall load on the system:

```
<purge id="Default.Client.Purger" frequency="1h" retention="1d" purgeInterval="1h"/>
```

9 Save the changes to the **data-export-config.xml** file.

Task 5: Monitor the data export operation

Assuming that the corresponding commanding Diagnostics Server is started, the entries in the saved configuration file take effect immediately. You do not need to restart the commanding Diagnostics Server.

The **data-export-config.xml** is parsed and verified as follows:

1 Each [db-target] specified in the XML is verified by connecting and verifying the database tables are available. If they are not available they are created with the following data relationships:
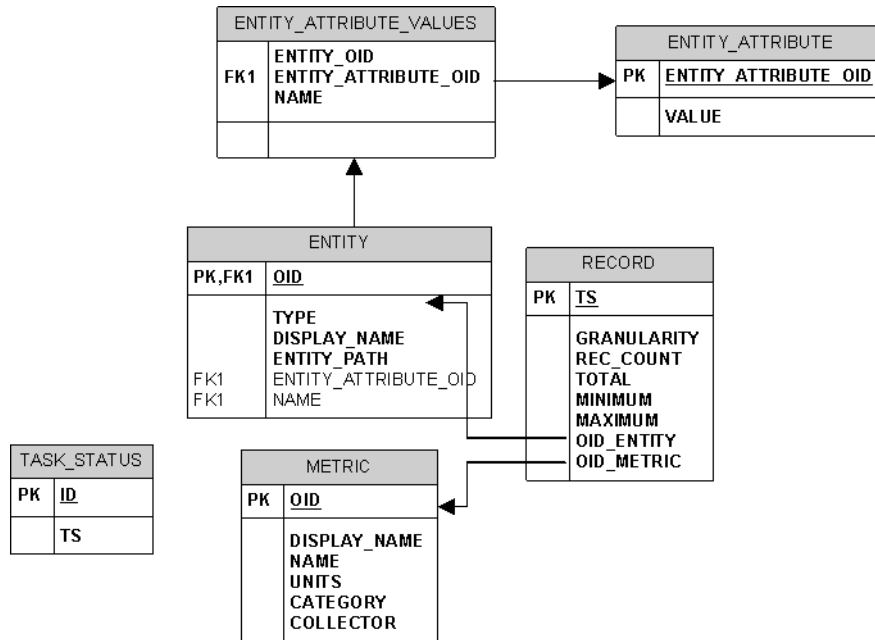
2 The Diagnostics Server schedules when to run each query based on the <frequency> specified. For example, a daily report (frequency = 1d) is run once a day after the last hour of the day has been aggregated into that daily summary. This means, in particular, that the queries are automatically aligned at the existing granularity boundaries.

3 When the scheduled query executes, the results of the query are stored in the database tables as follows:

➤ Entity descriptions, such as probe, fragment, or host are stored in the ENTITY table and the unique key is a MD5 hash of the TSDB entity key and the distributed source value to make it unique within a federated environment.

735

➤ Metrics descriptions are stored in the METRIC table and the unique key is a MD5 hash of the metric data name, metric data collector name and the distributed source to make it unique within a federated environment.

➤ The metric values are stored in the RECORD table and have the foreign key values pointing to the ENTITY and METRIC table unique keys. This is done to reduce overall size of data storage as the descriptions of both entities and metrics would be duplicated on every RECORD table row.

➤ The 2 additional tables, ENTITY_ATTRIBUTE_VALUES and ENTITY_ATTRIBUTE, serve as lookup tables to further describe the ENTITY dimension.

Task 6: Verify the results

You can use the database tools of your target database to verify the expected results. For example, the following image shows the results stored in the METRIC table of a SQL Server database. This set of metrics is returned based on the query statement shown:

```
<query id="Probes" frequency="1h" recovery-periods="2">
  <entity-path>/probegroup/probe</entity-path>
</query>
```
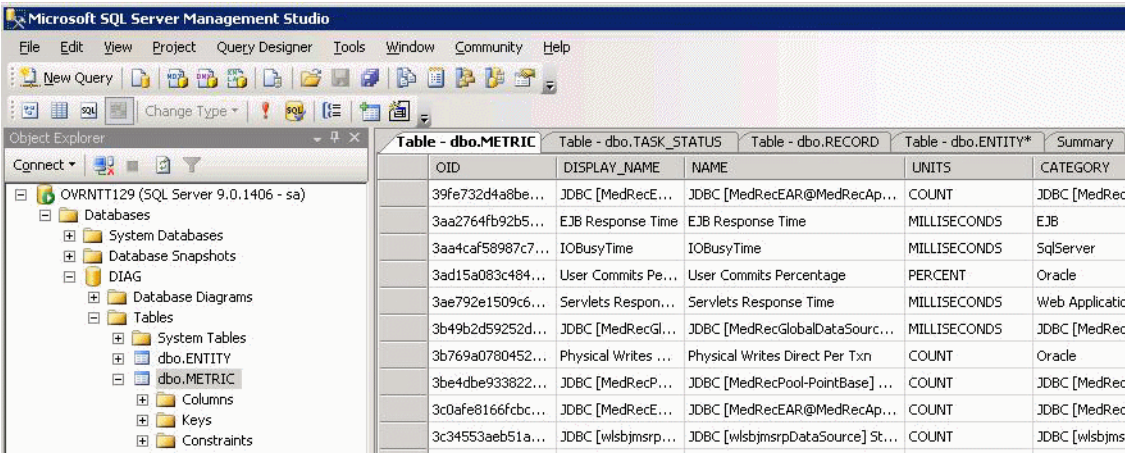
Task 7: Select the data from the target database

Once the exported data is stored in the target database, you can query it as desired. However, a pivot manipulation is required to get the data into a useful reporting format. The pivot uses the foreign key references to combine the dimension table data into a flatten row with the description data joined to the fact table.

Sample SQL scripts, queries and reports are included in <server_install_dir>/contrib/dataexport/ as follows:

➤ **sql_server_sample_view.sql.** SQL Server views used to denormalize and pivot exported data into a more friendly reporting format.

➤ **oracle_server_sample_view.sql**. Oracle DB Server views used to denormalize and pivot exported data into a more friendly reporting format.

➤ **oracle_view_query_samples.sql.** Various examples of querying the Oracle views.

➤ **sql_server_reports directory**. A directory containing SQL Server Reports project with various sample reports. Using the SQL Server Reporting tool, open the sql_server_reports directory and then open the file "Diagnostic Fragments.sln."

Index

Index