

HP Diagnostics

for the Windows® , Unix and Linux operating systems

Software Version: 9.00

Installation and Configuration Guide

Document Release Date: July 2010

Software Release Date: July 2010



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

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

© 2004 - 2010 Hewlett-Packard Development Company, L.P.

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

The title page of this document 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

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 online support 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 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 register for an HP Passport ID, go to:

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

To find more information about access levels, go to:

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

Table of Contents

Welcome to This Guide	17
How This Guide Is Organized	17
HP Diagnostics Online Documentation.....	19
Additional Online Resources.....	20
Documentation Updates	21

PART I: INTRODUCTION

Chapter 1: Preparing to Install HP Diagnostics	25
HP Diagnostics Components and Data Flow	26
Supported Application Servers and Environments	28
System Requirements for the Diagnostics Components.....	29
Information Required for Installation	37
Pre-installation Considerations.....	44
Recommended Order of Installation.....	45
Licensing HP Diagnostics	46
Upgrading from Earlier Versions of Diagnostics.....	47

PART II: INSTALLATION OF THE DIAGNOSTICS SERVER AND COLLECTORS

Chapter 2: Installing the Diagnostics Server	51
Installing the Diagnostics Server on Windows and UNIX.....	52
Silent Installation of the Diagnostics Server	61
Starting and Stopping the Diagnostics Server.....	63
Verifying the Diagnostics Server Installation	65
Licensing the Diagnostics Server in Commander Mode.....	66
Configuring the Diagnostics Server	66
Determining the Version of the Diagnostics Server	66
Uninstalling the Diagnostics Server.....	67
Manual Installation of OM Agent and IAPA Components.....	68
Manual Uninstall of OM Agent and IAPA Components	70

Chapter 3: Licensing HP Diagnostics	73
About HP Diagnostics Licensing	74
Types of Licenses	74
Licensing the Diagnostics Server in Commander Mode.....	75
View License Information	78
Licensing the Other Diagnostics Components.....	79
Chapter 4: Installing Diagnostics Collectors.....	81
About Installing the Diagnostics Collector.....	82
Installing the Diagnostics Collector on a Windows Machine.....	82
Installing the Diagnostics Collector on a UNIX Machine.....	90
Silent Installation of the Diagnostics Collector.....	98
Installing the Diagnostics Collector Using the Generic Installer	99
Configuring the Active System Property Files	100
Configuration for SAP NetWeaver-ABAP.....	100
Configuration for Oracle.....	103
Configuration for MQ.....	105
Configuration for SQL Server.....	108
Password Obfuscation	112
Verifying the Diagnostics Collector Installation	114
Starting and Stopping the Diagnostics Collector.....	115
Determining the Version of the Diagnostics Collector	117
Uninstalling the Diagnostics Collector.....	117

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

Chapter 5: Installing Java Agents.....	121
About the Java Agent Installer	122
Installing and Configuring the Java Agent on Windows	123
Installing and Configuring the Java Agent on UNIX	139
Installing the Java Agent on a z/OS Mainframe	154
Installing the Java Agent Using the Generic Installer	156
Silent Installation of the Java Agent	158
About Configuring the Application Server	160
Verifying the Java Agent Installation.....	161
Determining the Version of the Java Agent.....	162
Uninstalling the Java Agent.....	163
About Advanced Java Agent Configuration.....	163
About Custom Instrumentation for Java Applications.....	163
Chapter 6: Running the JRE Instrumenter	165
About the JRE Instrumenter	166
Running the JRE Instrumenter.....	169

Chapter 7: Configuring Application Servers for the Java Agent	179
About Configuring the Application Server Startup Scripts	180
Configuring WebSphere Application Servers.....	181
Configuring WebLogic Application Servers.....	195
Configuring Oracle Application Servers.....	201
Configuring the JBoss Application Server	210
Configuring Tomcat 5.x/6.x When Running as a Windows Service	213
Configuring the SAP NetWeaver Application Server	214
Configuring a Generic Application Server	216
Adjusting the Heap Size for the Java Agent in the Application Startup Script	218
Configuring TIBCO ActiveMatrix/Business Works	219
Configuring the SOAP Message Handler.....	219
Configuring the Discovery of a New J2EE Server for CI Population	223
Chapter 8: Installing .NET Agents.....	225
About the .NET Agent Installer	226
Points Files and the probe_config.xml File	226
Discovering ASP.NET Applications	228
Automatic Configuration for Discovered ASP.NET Applications	229
Discovery of IIS Metadata for CI Population of IIS Deployed ASP.NET Applications	229
Non ASP.NET Applications	232
Installing the .NET Agent.....	232
Verifying the Connection with the Diagnostics Server	249
About Custom Configuration and Instrumentation for the .NET Agent	250
Enabling and Disabling Standard Instrumentation for Applications	251
Restarting IIS.....	253
Verifying the .NET Agent Installation.....	254
Determining the Version of the .NET Agent.....	255
Uninstalling the .NET Agent	256
Enabling and Disabling the Diagnostics Agent for .NET	256
Disabling Logging.....	257

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

Chapter 9: Custom Instrumentation for Java Applications261
About Instrumentation and Capture Points Files262
Coding Points in the Capture Points File264
Defining Points With Code Snippets273
Controlling Class Map Capture288
Instrumentation Examples.....289
Understanding the Overhead of Custom Instrumentation305
Instrumentation Control on a Per Layer Basis.....306
Advanced Instrumentation307
Maintaining Instrumentation from the Java Profiler UI322
Default Layers Defined for Typical Java Classes and Methods334

Chapter 10: Custom Instrumentation for .NET Applications337
About Instrumentation and Capture Points Files338
Locating the .NET Capture Points Files339
Coding Points in the Capture Points File340
Instrumentation Examples.....344
Understanding the Overhead of Custom Instrumentation369
Default Layers for Typical .NET Applications370

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

Chapter 11: Advanced Diagnostics Server Configuration	373
Synchronizing Time Between Diagnostics Components.....	374
Configuring the Diagnostics Server for a Large Installation.....	378
Overriding the Default Diagnostics Server Host Name.....	383
Changing the Default Diagnostics Server Port	383
Configuring the Diagnostics Server for Multi-Homed Environments.....	384
Reducing Diagnostics Server Memory Usage	388
Configuring Fragment Name Based Trimming.....	388
Preparing a High Availability Diagnostics Server.....	390
Configuring Diagnostics for HP ServiceGuard (HA solution).....	391
Diagnostics Server Assignments (LoadRunner/Performance Center Runs)	393
Configuring the Diagnostics Server for LoadRunner Offline Analysis File Size.....	394
Configuring Business Service Management Samples Queue Size and Web Services CI Frequency.....	397
Configuring Diagnostics Using the Diagnostics Server Configuration Pages.....	398
Optimizing the Diagnostics Server in Production to Handle More Probes.....	398

Chapter 12: Advanced Java Agent and Application	
Server Configuration	401
Advanced Configuration	402
Controlling Which HP Software Products the Agent can Work With	403
Configuring the Agent for Multiple Application Server JVM Instances	406
Disabling the Java Diagnostics Profiler	412
Specifying Probe Properties as Java System Properties.....	413
Controlling Probe Logging.....	414
Setting the Probe's Host Machine Name	415
Controlling Automatic Method Trimming on the Agent.....	417
Configuring an Agent for a Proxy Server	418
Configuring Reverse HTTP for an Agent in SaaS	419
Time Synchronization for Probes Running on VMware.....	421
Limiting Exception Tree Data	422
Diagnostics Probe Administration Page.....	425
Authentication and Authorization for Java Diagnostics Profilers in Standalone Mode	428
Configuring Collection of CPU Time Metrics.....	431
Configuring Consumer IDs.....	434
Configuring SOAP Fault Payload Data	444
Configuring REST Services	445
Customizing Grouping JMS Temporary Queue/Topics	445
Configuring Display of Application Name for Server Requests.....	445
Maintaining Probe Settings from the Java Profiler UI	446
Chapter 13: Understanding the .NET Agent Configuration File	455
Understanding the .NET Agent Configuration File	455

Chapter 14: Advanced .NET Agent Configuration	517
Time Synchronization for .NET Agents Running on VMware	518
Customizing the Instrumentation for ASP.NET Applications	518
Discovering the Classes and Methods in an Application	522
Controlling Which HP Software Products the Agent can Work With	524
Configuring Support for MSMQ Based Communication	526
Configuring Latency Trimming and Throttling	527
Configuring Depth Trimming.....	531
Configuring the .NET Agent for Light-Weight Memory Diagnostics	533
Limiting Exception Stack Trace Data	536
Disabling Logging.....	538
Overriding the Default Probe Host Machine Name.....	539
Listing the Probes Running on a Host	540
Authentication and Authorization for .NET Profilers in Standalone Mode	541
Configuring Consumer IDs	543
Configuring SOAP Fault Data.....	548
Collecting Additional Probe Metrics or Modifying Probe Metrics ...	549

PART VI: CONFIGURING COMMUNICATIONS THROUGH PROXIES AND FIREWALLS

Chapter 15: Configuring Diagnostics Servers and Agents for HTTP Proxy	555
Enabling HTTP Proxy Communications for the Diagnostics Servers.....	556
Enabling HTTP Proxy Communications for the Java Agent.....	557
Enabling HTTP Proxy Communications for a .NET Agent	557
Chapter 16: Configuring Diagnostics to Work in a Firewall Environment.....	559
Overview of Configuring Diagnostics for a Firewall.....	560
Collating Offline Analysis Files over a Firewall	563
Installing and Configuring the MI Listener	564
Configuring the Diagnostics mediator server to Work with a Firewall	565
Configuring LoadRunner and Performance Center to Work with Diagnostics Firewalls	571

PART VII: CONFIGURING DIAGNOSTICS METRICS COLLECTORS

Chapter 17: .NET System Metrics Agent - Systems Metrics Capture **575**
About the .NET System Metrics Agent 575
System Metrics Captured by Default..... 576
Configuring .NET System Metrics Capture 577
Adding System Metrics Using the Windows Performance Monitor. 580
Default Entries in the .NET Agent metrics.config File 582
Keywords in the metrics.config File 583

Chapter 18: Java Agent Metrics Collectors **587**
About Metrics Capture 587
What Metrics are Being Collected by the Java Agent 588
Understanding Metric Collector Entries 590
About Collecting Additional Probe Metrics 592
Modifying Probe Metrics Already Being Captured 592
Stopping Capture of a Metric 592

Chapter 19: Java Agent - System Metrics Capture **593**
About System Metrics 593
System Metrics Captured by Default..... 594
Configuring the System Metrics Collector..... 595
Capturing Additional Custom System Metrics 597
Enabling z/OS System Metrics Capture..... 603

Chapter 20: Java Agent - JMX Metrics Capture **605**
About JMX Metrics 605
Configuring WebSphere for JMX Metric Collection..... 606
Configuring JMX Metric Collectors 606
Capturing Additional Custom JMX Metrics 606

PART VIII: SETTING UP INTEGRATION WITH OTHER HP SOFTWARE PRODUCTS

Chapter 21: Setting Up the Integration Between Business Service Management and Diagnostics.....	615
About Setting Up the Integration Between Business Service Management and Diagnostics	617
Registering the Diagnostics Server in Business Service Management 9.0	618
Changing the Diagnostics Server Details	624
Understanding the Diagnostics Admin Page	624
Assigning Permissions for Diagnostics Users in Business Service Management	626
Accessing the Diagnostics Pages in Windows 2003	627
Accessing the Diagnostics Application from Business Service Management	628
Data Samples Sent to Business Service Management.....	628
Diagnostics Populates CIs and Models in Business Service Management	629
Synchronize CIs Between Diagnostics and Business Service Management	629
Diagnostics Provides KPI/Hi Coloring to Business Service Management	630
Chapter 22: Installing the LoadRunner Diagnostics Add-in.....	633
Before Installing the LoadRunner Diagnostics Add-in	634
Installing the LoadRunner Diagnostics Add-in.....	634
Chapter 23: Setting Up LoadRunner and Diagnostics Integration ..	637
About Setting Up LoadRunner to Use Diagnostics	638
Configuring LoadRunner Scenarios to use HP Diagnostics	638
Selecting Probe Metrics to Include in the Offline Analysis File.....	639
Improving Transfer of Large Offline Analysis Files.....	642
Chapter 24: Setting Up Performance Center to Use Diagnostics	643
About Setting Up Performance Center to Use Diagnostics.....	644
Configuring Performance Center Load Tests to Use Diagnostics	645
Managing Performance Center Offline Files.....	646

PART IX: APPENDIXES

Appendix A: Diagnostics Administration UI	649
Accessing the Diagnostics Administration UI.....	649
Using the Diagnostics Administration UI.....	652

Appendix B: User Authentication and Authorization	657
About User Authentication and Authorization	658
Understanding User Privileges	659
Understanding Roles	660
Accessing Diagnostics Using Default User Names	661
Understanding the Diagnostics Server Permissions Page	662
Creating, Editing and Deleting Users.....	669
Assigning Privileges Across the Diagnostics Deployment	672
Assigning Privileges for Probe Groups	673
Authentication and Authorization for Users of Integrated HP Software Products.....	676
Tracking User Administration Activity	677
List of Active Users	679
Configuring Diagnostics to use JAAS	679
Appendix C: Enabling HTTPS Between Components	689
About Configuring HTTPS Communications	689
HTTPS Checklist per Diagnostics Component.....	690
Enabling Incoming HTTPS Communication for Diagnostics Components.....	692
Enabling Outgoing HTTPS Communication from Diagnostics Components.....	702
Enabling HTTPS Communications for the Business Service Management Server	708
Appendix D: Using the System Health Monitor	711
Introducing the System Health Monitor	712
Accessing the System Health Monitor	712
System Health Monitor Views.....	713
Understanding the System Health Graph.....	714
Viewing Component Status and Host Configuration Tooltips	717
Viewing Detailed Component Information	719
Viewing Troubleshooting Tips	723
Viewing Log Information for the Whole System.....	723
Customizing the System Health Monitor Display	724
Filtering the System Health Monitor Component Map by Customer.....	726
Controlling the Refresh Rate of the System Health Monitor	727
Creating and Using System Health Monitor Snapshots	728

Appendix E: Diagnostics Data Management	731
About Diagnostics Data.....	732
Custom View Data.....	732
Performance History Data	733
Data Retention.....	739
Pre-Installation Data Management Considerations.....	744
Backing Up Diagnostics Data	745
Handling Diagnostics Data when Upgrading Diagnostics.....	749
Appendix F: Diagnostics Technical Diagrams	751
Communications with Business Service Management	752
Communications with LoadRunner and Performance Center	753
.NET Probe Aggregator Data Flow	754
Appendix G: Upgrade and Patch Install Instructions	755
General Recommendations	756
Diagnostics Compatibility with Earlier Diagnostics Versions	756
Upgrade or Patch Install Instructions for Diagnostics Components	756
Diagnostics Compatibility with Other HP Software Products	767
Appendix H: Troubleshooting HP Diagnostics	769
Component Installation Interrupted on a Solaris Machine	770
Java Agent Fails to Operate Properly	770
Error During WAS Startup with Diagnostics Profiler for Java.....	771
Missing Server-Side Transactions	772
Event Capture Buffer Full Warning.....	772
Java Agent Support Collector	773
OM Agent Troubleshooting	774
Event Based Health Indicator Status Troubleshooting Flow.....	776
Appendix I: General Reference Information	781
Using UNIX Commands	781
Using Regular Expressions.....	782
Multi-Lingual User Interface Support	790
Appendix J: Data Exporting	793
Task 1: Prepare the target database	793
Task 2: Determine which metrics you want to export	794
Task 3: Determine the frequency and the recovery period	798
Task 4: Modify the data export configuration file	799
Task 5: Monitor the data export operation.....	802
Task 6: Verify the results	804
Task 7: Select the data from the target database	805
Sample Queries	805

Index.....809

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 gives an overview of the integrations 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

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 Agents.

Part VI Configuring Communications through Proxies and Firewalls

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

Part VII Configuring Diagnostics Metrics Collectors

Describes metrics capture and how to configure the metric collectors for the .NET Agent and the Java Agent.

Part VIII Setting Up Integration with Other HP Software Products

Gives an overview of how to set up LoadRunner, Performance Center and Business Service Management for integration with HP Diagnostics.

Part IX Appendixes

Describes administrative tasks for the Diagnostics Administrator such as:

- Using the Admin UI to configure and manage Diagnostics
- Setting up users, permissions, authorization and authentication
- Enabling HTTPS secure communications between components
- Using System Health UI
- How data is persisted and managed as well as backup and recovery
- Technical diagrams
- Upgrade instructions
- How to use the Data Export feature
- Troubleshooting and additional reference information

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 help, or from the Windows Start menu (**Start > Programs > HP Diagnostics Server > User Guide**), or 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, or from the **Documentation** directory on the Diagnostics installation disk, or from the Diagnostics Server installation directory, or from the Windows Start menu (**Start > Programs > HP Diagnostics Server > Install Guide**),
- ▶ **Readme.** Provides last-minute technical and troubleshooting information about HP Diagnostics. The file is located in the HP Diagnostics installation disk root directory. There is also a Patch Install Instructions document with details for installing a patch release.
- ▶ **Diagnostics Java Agent Guide.** Describes how to install, configure, and use the Diagnostics Java Agent and the Diagnostics Profiler for Java. You access this guide on the agent system, or from the Profiler UI Help link or in the Documentation directory on the HP Diagnostics installation disk.
- ▶ **Diagnostics .NET Agent Guide.** Describes how to install, configure, and use the Diagnostics .NET Agent and Diagnostics Profiler for .NET. You access this guide on the agent system, or from the Profiler UI Help link or in the Documentation directory on the HP Diagnostics installation disk.

Note: The information in the Diagnostics Agent guides is based on information in the **Diagnostics Installation and Configuration Guide** and the **Diagnostics 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/hpsupport.

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>

Documentation Updates

HP Software is continually updating its product documentation with new information.

To check for recent updates, or to verify that you are using the most recent edition of a document, go to the HP Software Product Manuals Web site (<http://h20230.www2.hp.com/selfsolve/manuals>).

Welcome to This Guide

Part I

Introduction

This section includes:

- ▶ Preparing to Install HP Diagnostics

1

Preparing to Install HP Diagnostics

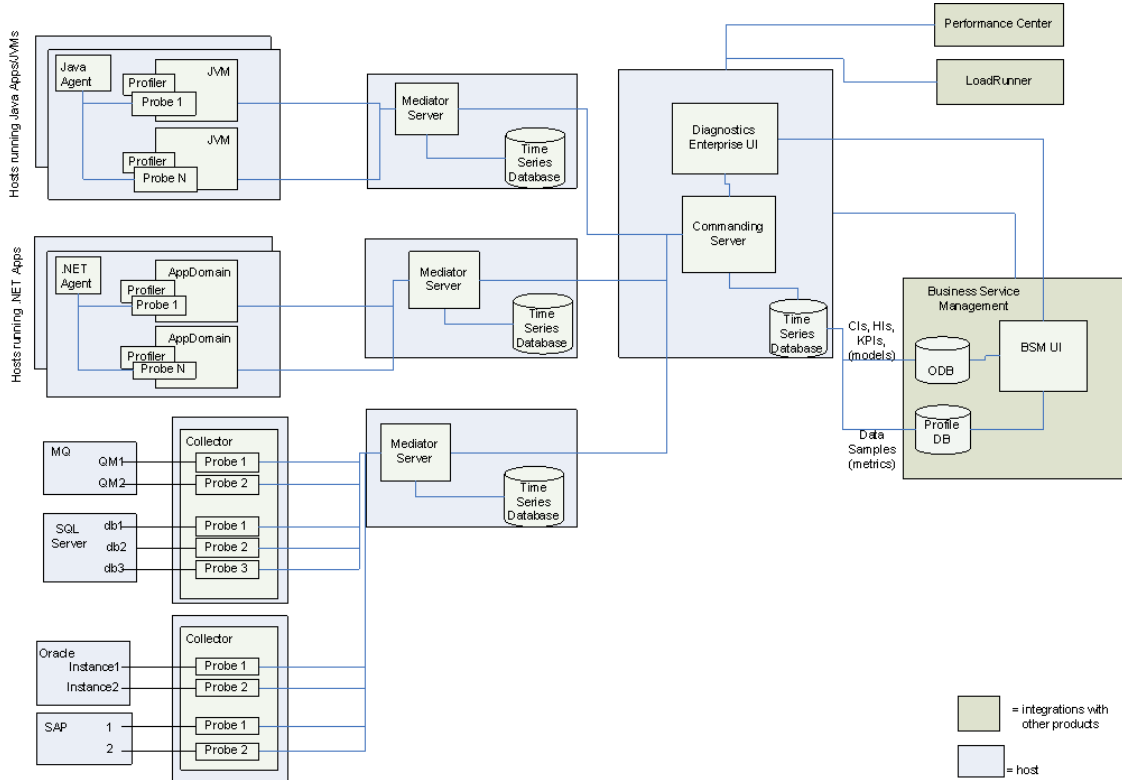
Before you install HP Diagnostics, read the following information and instructions that will help you plan and prepare for the installation and configuration of the Diagnostics components.

This chapter includes:

- HP Diagnostics Components and Data Flow on page 26
- Supported Application Servers and Environments on page 28
- System Requirements for the Diagnostics Components on page 29
- Information Required for Installation on page 37
- Pre-installation Considerations on page 44
- Recommended Order of Installation on page 45
- Licensing HP Diagnostics on page 46
- Upgrading from Earlier Versions of Diagnostics on page 47

HP Diagnostics Components and Data Flow

The following diagram illustrates the data flow among Diagnostics components and integrations with other HP Software products.



HP Diagnostics consists of the following components:

- ▶ **Diagnostics Agents.** Responsible for capturing events from your application such as method invocations, the beginning and end of business transactions and server requests and then aggregating performance metrics to be sent to a Diagnostics Server. The Diagnostics Agent software is installed on the systems to be monitored. With the Java Agent you instrument the application server(s) for monitoring. With the .NET Agent you instrument the application domains for monitoring. Each instrumented application server or application domain results in an agent instance represented by a probe entity. You control the data collection settings for these probe entities using a number of different configuration files in the agent installation folder.

Note: The functionality of the Java Agent is provided by the HP Diagnostics/ TransactionVision Java Agent. The functionality of the .NET Agent is provided by the HP Diagnostics/TransactionVision .NET Agent.

- ▶ **Diagnostics Collectors.** Responsible for collecting 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 monitored instance is represented as a probe entity in the Diagnostics UI.
- ▶ **Diagnostics Servers.** Responsible for working with the agents, collectors and with other HP Software products to capture, process, and present the performance metrics for your applications.

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

A Diagnostics deployment may consist of one or many Diagnostics servers. If there is only one Diagnostics server in your deployment, it is configured as the Diagnostics commanding server and must perform both the commander and mediator roles. If there is more than one Diagnostics server in a deployment, one must be configured as the Diagnostics commanding server, with all the rest running as (distributed) mediators.

In a typical deployment there is a Diagnostics commanding server connected to a one or more servers running as mediators. Each Diagnostics Mediator Server is configured to receive data from systems where the agents and collectors are installed. The Diagnostics Mediator Server then filters and aggregates the events it receives. This information is sent to the Diagnostics commanding server, which displays the processed metrics in customizable views.

The Diagnostics commanding server 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. This server keeps track of the location and status of the other Diagnostics components, and is the communication hub between the other components. And 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.

- ▶ **User Interfaces.** When you are monitoring your application in real time, you access the Diagnostics UI from Business Service Management 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 agent system or through the Diagnostics UI.

Diagnostics has integrations with the following other HP Software products:

- ▶ Business Service Management
- ▶ LoadRunner
- ▶ Performance Center

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, see the Diagnostics Product Availability Matrix at http://support.openview.hp.com/sc/support_matrices.jsp.

Note: Diagnostics monitors only those methods that are invoked by the application server and are instrumented by the agent. 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. See the deployment diagram in the previous section to understand the component hosts described in this section.

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

This section includes the following:

- “Supported Environments for the Diagnostics Components” on page 30
- “Requirements for the Diagnostics UI Host” on page 30
- “Requirements for the Diagnostics Server Host” on page 30

- “Scalability Information” on page 32
- “Requirements for the Diagnostics Agent for Java Host” on page 34
- “Requirements for the Java Diagnostics Profiler User Interface Host” on page 35
- “Requirements for the Diagnostics Agent for .NET Host” on page 36
- “Requirements for the Diagnostics Collector Host” on page 36

Supported Environments for the Diagnostics Components

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

Diagnostics Servers and Diagnostics Collectors use the Java 1.6 JVM.

Requirements for the Diagnostics UI Host

The user interface for Diagnostics is presented in a web browser using a Java applet that requires JRE 1.6, minimally to be installed on the machine that displays the UI. Supported browsers include Microsoft Internet Explorer 7, Internet Explorer 8 and Mozilla Firefox 3.5.

Requirements for the Diagnostics Server Host

The system configuration requirements for the host of a Diagnostics Server depend upon the number of probes and mediating servers that are reporting to it. When a server is designated as the Diagnostics commanding server, each mediating server 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 that are 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 need and the actual number of supported probes are affected by several characteristics of the monitored environment including 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 does not apply to them.

The following table lists the desired system requirements for the host of a Diagnostics server receiving data from 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				
<ul style="list-style-type: none"> ➤ 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 receiving data from .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 Information

The following scalability numbers are derived from the following reference hardware configuration:

Platform:	Windows
Operating System:	Windows Server 2008, 64-bit
CPU:	Intel Xeon 5160 @ 3.00Ghz (quad core)
Memory:	8 GB
Disk I/O:	Smart Array P400i, 2SCSI drives in RAID 0 (136 GB) [130 MB/S, sequential read and write]
Java Heap:	5.9 GB (-Xmx6096m); 64-bit JVM

Disk Space:	2-4 GB per probe (overall disk space can be adjusted by changing retention intervals)
Network:	1 GBps

Note: 64-bit OS and a 64-bit JVM is highly recommended because this combination allows better scaling of memory (32-bit JVMs on Windows have 1.2 GB limit). 64-bit JVMs require more memory than their 32-bit counterpart due to doubling of references.

Scalability numbers for the previously referenced 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 378.

Notes:

- ▶ For environments with many probes, better performance can be achieved by having two or more instances of the Diagnostics 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 commanding server, see “Pre-Installation Data Management Considerations” on page 744.
 - ▶ For information on how to optimize a Diagnostics server to handle more probes, see “Optimizing the Diagnostics Server in Production to Handle More Probes” on page 398.
-

Requirements for the Diagnostics Agent for Java Host

The overhead that the Diagnostics Java Agent imposes on the system being monitored is extremely low. The following are the recommendations for memory and disk space that support the agent’s processing:

Platform:	All Platforms
Memory:	50MB Additional RAM
Free Hard Disk Space:	200MB free disk space is required for the initial Java probe install. More space might be required during runtime due to the creation of logfiles and classmap. For large applications, it is recommended to have an additional 200MB available per probe for logfiles and classmap data.

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 startup script.

For information on setting the max heap for the Java Agent, see “Adjusting the Heap Size for the Java Agent in the Application Startup Script” on page 218.

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 1.6 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 Agent 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 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 Internet Explorer 6 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 supported systems that can interact with the host machines of the SAP NetWeaver-ABAP, Oracle, or SQL Server application from which it is collecting data.

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

Information Required for Installation

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

The tables in the following sections can help you gather the information required during the installation of the Diagnostics components.

Note: When you are installing Business Service Management 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
<p>For a Diagnostics commanding server, the location of the HP Diagnostics license that was generated for the machine that will host the server</p>	<p>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.</p>	
<p>For a Diagnostics mediator server, the URL for the Diagnostics commanding server</p>	<p>After the Diagnostics commanding server has been installed, the URL is available from the System Health Monitor. (See Appendix D, “Using the System Health Monitor.”)</p>	
<p>Will the Diagnostics Server be used in a SaaS environment or in the Business Service Management environment?</p>		

Java Agent

► HP Software Product and Diagnostics Server Information

Information Required	Where to find it	Value
Name of other HP Software product(s) that will use the Diagnostics data collected	Choose according to product license. ► Performance Center / LoadRunner ► Business Service Management	
Diagnostics Server Name	Fully qualified host name or IP address of the host of the Diagnostics Server. System Health Monitor. (See Appendix D, “Using the System Health Monitor.”) This is not required for using the Java Diagnostics Profiler in a standalone mode.	If there is only one Diagnostics Server in the deployment where the agent will run, this is the Diagnostics commanding server. In a distributed environment with a commanding server and mediating servers, this is the Diagnostics mediator server that is to receive the events from the agent.
Diagnostics Server Port	System Health Monitor. (See Appendix D, “Using the System Health Monitor.”) This is not required for using the Java Diagnostics Profiler in a standalone mode.	Default value: 2006

► Instrumented Application Server and Agent Information

Information Required	Where to find it	Value
<p>Java agent name</p>	<p>A <i>unique</i> string; Created by user.</p> <p>The agent name is assigned as the default probe entity name.</p> <p>The name of the agent should indicate the application you plan to monitor and the type of probe instrumentation, to help you distinguish between the different applications and types of probes.</p> <p>There can be multiple probes using a single Java Agent configuration. In this case you can later configure unique probe names for each monitored application.</p>	<p>For example: WebLogic_MedRec_java</p>
<p>Java agent group</p>	<p>This is user-defined at the time that the agent is installed.</p> <p>The agent group name you enter is used as the probe group name.</p> <p>Probe groups are logical groupings of probes that report to the same Diagnostics Server.</p>	<p>Default value: DefaultProbeGroup</p>
<p>Type of application server that will be instrumented for monitoring</p>	<p>The host system administrator.</p>	

Information Required	Where to find it	Value
Application Server configuration properties	The host system administrator. The details vary according to the application server you are monitoring.	
Location of the JRE executable	The host system administrator. Depends on the type of application server you plan to monitor. See “Running the JRE Instrumenter” on page 165.	

.NET Agent

► Diagnostics Server Information

Information Required	Where to find it	Value
<p>Diagnostics Server Name</p>	<p>Fully qualified host name or IP address of the host of the Diagnostics Server. System Health Monitor. (See Appendix D, “Using the System Health Monitor.”)</p> <p>This is not required for using the .NET Diagnostics Profiler in a standalone mode.</p>	<p>If there is only one Diagnostics Server in the deployment where the agent will run, this is the Diagnostics commanding server.</p> <p>In a distributed environment with a commanding server and mediating servers, this is the Diagnostics mediator server that is to receive the events from the agent.</p>
<p>Diagnostics Server Port</p>	<p>System Health Monitor.</p> <p>This is not required for using the .NET Diagnostics Profiler in a standalone mode.</p>	<p>Default value: 2612</p>

➤ **Agent and Port Information**

Information Required	Where to find it	Value
agent group	<p>This is user defined at the time that the agent is installed.</p> <p>The agent group name you enter is used as the probe group name</p> <p>Probe groups are logical groupings of probes that report to the same Diagnostics Server.</p>	<p>Default value: Default</p>
Web Port Min	<p>System Administrator.</p> <p>The lowest port number in a range of ports on the agent system that can be assigned to the probe.</p>	<p>Default value: 35000</p>
Web Port Max	<p>System Administrator.</p> <p>The highest port number in a range of ports on the agent system that can be assigned to the probe.</p>	<p>Default value: 35100</p>

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.

Diagnostics Server

- ▶ The performance metrics for HP Diagnostics cannot be displayed until the Diagnostics commanding server 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, your screen resolution should be at least 1024x768.

Diagnostics Java Agent

- ▶ The Java Agent must be installed on the same system as the Java application under test.
- ▶ The Diagnostics Profiler for Java operates in an unlicensed mode with load restrictions until it is able to connect to a Diagnostics commanding server that is properly licensed. For more information on obtaining a license and other licensing issues, see Chapter 3, “Licensing HP Diagnostics.”

Diagnostics .NET Agent

- ▶ The .NET Agent must be installed on the same system as the .NET application under test.
- ▶ The Diagnostics Profiler for .NET operates in an unlicensed mode with load restrictions until it is able to connect to a Diagnostics commanding server that is properly licensed. For more information on obtaining a license and other licensing issues, see Chapter 3, “Licensing HP Diagnostics.”

LoadRunner and Performance Center Host Machines

- If LoadRunner is already installed, make sure 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.

Recommended Order of Installation

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

Note: The following order of the installation is recommended for the products and components. Deviating from it could increase the complexity of the installation process and produce unpredictable results.

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

Recommended order of installation:

1 Check the system requirements and installation considerations.

See “System Requirements for the Diagnostics Components” on page 29.

2 Install the Diagnostics Server.

For more information, see Chapter 2, “Installing the Diagnostics Server” and Chapter 3, “Licensing HP Diagnostics.”

3 Install the Diagnostics Agents and Collectors.

For a Java environment, see Chapter 5, “Installing Java Agents.”

For a .NET environment, see Chapter 8, “Installing .NET Agents.”

For Oracle Database, SAP NetWeaver-ABAP, SQL Server Database and MQ environments, see Chapter 4, “Installing Diagnostics Collectors.”

- 4 For Java agents, instrument the application servers for monitoring by Diagnostics (this results in agent instances which are represented as probe entities in Diagnostics).**

For more information, see Chapter 6, “Running the JRE Instrumenter.”

For more information, see Chapter 7, “Configuring Application Servers for the Java Agent.”

- 5 Customize the instrumentation and control the data collection settings using a number of different configuration files in the agent installation folder.**

For more information, see the sections on “Custom Instrumentation for Monitoring Java and .NET applications” and “Advanced Configuration of the Diagnostics Server and the Java and .NET Agents”

- 6 If HP Diagnostics is integrated with LoadRunner, Performance Center, or Business Service Management, it must be set up to use HP Diagnostics.**

For Business Service Management, see Chapter 21, “Setting Up the Integration Between Business Service Management and Diagnostics.”

For LoadRunner integration, install the LoadRunner Diagnostics Add-in (see Chapter 22, “Installing the LoadRunner Diagnostics Add-in”) and set up LoadRunner to use Diagnostics (see Chapter 23, “Setting Up LoadRunner and Diagnostics Integration”).

For Performance Center, see Chapter 24, “Setting Up Performance Center to Use 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 commanding server. The license is a node-locking license that unlocks the Diagnostics views displayed by the Diagnostics commanding server and removes the load restriction from the Profiler views that are accessed from the Diagnostics agent system. 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 in an environment where a previous version of the product was installed, or where other HP Software products need to be upgraded so that the features of Diagnostics can be accessed, follow the instructions in Appendix G, “Upgrade and Patch Install Instructions.” 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

This section includes:

- ▶ Installing the Diagnostics Server
- ▶ Licensing HP Diagnostics
- ▶ Installing Diagnostics Collectors

2

Installing the Diagnostics Server

This section explains how to install the Diagnostics Server on Windows and UNIX machines.

This chapter includes:

- ▶ Installing the Diagnostics Server on Windows and UNIX on page 52
- ▶ Silent Installation of the Diagnostics Server on page 61
- ▶ Starting and Stopping the Diagnostics Server on page 63
- ▶ Verifying the Diagnostics Server Installation on page 65
- ▶ Licensing the Diagnostics Server in Commander Mode on page 66
- ▶ Configuring the Diagnostics Server on page 66
- ▶ Determining the Version of the Diagnostics Server on page 66
- ▶ Uninstalling the Diagnostics Server on page 67
- ▶ Manual Installation of OM Agent and IAPA Components on page 68
- ▶ Manual Uninstall of OM Agent and IAPA Components on page 70

Installing the Diagnostics Server on Windows and UNIX

The following sections provide detailed instructions for installing the Diagnostics Server and 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, “Upgrade and Patch Install Instructions.”

Important: The following instructions assume an understanding of UNIX console screens and commands. For more information about working with UNIX screens and commands, see “Using UNIX Commands” on page 781.

This section includes:

- ▶ “Launching the Diagnostics Server Installer” on page 52
- ▶ “Running the Installation” on page 54

Launching the Diagnostics Server Installer

Depending on your environment, launch either the Windows installer or the UNIX installer. See also “Silent Installation of the Diagnostics Server” on page 61.

Note: Allow approximately 400MB of free space in the temp directory.

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 installs the 32-bit Windows version of the Diagnostics Server. To install the 64-bit version of the Diagnostics Server, **Browse the DVD** to locate the **Diagnostics_Servers** directory and double-click the **HPDiagServer_<release number>_win64.exe** file.

Note: To launch the installer from a different location, copy the executable file **HPDiagServer_<release number>_win32.exe** (32-bit) or **HPDiagServer_<release number>_win64.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 it.

Continue with “Running the Installation” on page 54.

To launch the UNIX installer:

- 1** From the **<HP Diagnostics Installation Disk>/Diagnostics_Servers** directory, copy the installer **HPDiagServer_<release number>_<platform>.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 **HPDiagServer_<release number>_<platform>.bin** filename at the UNIX command prompt; for example:

```
./HPDiagServer_9.00_sol.bin
```

- ▶ To run the installer in console mode enter the installer **HPDiagServer_<release number>_<platform>.bin** filename with the **-console** option, at the UNIX command prompt; for example:

```
./HPDiagServer_9.00_sol.bin -console
```

Continue with “Running the Installation” on page 54.

Running the Installation

After you launch the installer, the software license agreement opens.

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.

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), click **Browse** to navigate to another directory.

Select **Next** to continue.

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

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

The Diagnostics deployment you are setting up can 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 commanding and mediating 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 you are currently installing is to be configured in Commander mode, select **Commander Mode**. Otherwise, select **Mediator Mode**.

Select **Next** to continue.

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

- ▶ To install the Diagnostics command server, continue with “Installing the Diagnostics Server in Commander Mode” on page 56.
 - ▶ To install the Diagnostics mediator server, continue with “Installing the Diagnostics Server in Mediator Mode” on page 59.
-

Installing the Diagnostics Server in Commander Mode

If you are installing the Diagnostics command server, continue as follows:

1 Select a time synchronization method.

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 Service Management server.** If the Diagnostics Server is to work in a Business Service Management environment, select this option to synchronize with the Business Service Management server.
- ▶ **Synchronize with system time.** Select this option if the Diagnostics Server is to work in an environment other than Business Service Management and there is no access to an NTP server.

Select **Next** to continue.

2 Select optional configurations for the Diagnostics Server.

Indicate the other products that the Diagnostics Server may interact with so that the installer can perform additional configuration steps.

This Server is to be used in an HP Software-as-a-Service (SaaS) environment. Check this box if the Diagnostics commanding server will be used in a SaaS environment.

This Server is to be used with HP Business Service Management (BSM). Check this box if the Diagnostics commanding server will be used with Business Service Management. If integrating with Business Service Management 9.00 or later, checking this option means additional OM agent and IAPA components are installed for use in sending Health Indicators to Business Service Management. You will be prompted towards the end of the installation to confirm if you want to install these components.

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

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

- 3 Provide the path to the directory on the HP Diagnostics Installation DVD where the Diagnostics installers are located (Business Service Management environment only).**
-

Note: You must 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 Service Management, you must 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.

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

Note: You can 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 Service Management to access.

4 Review the pre-installation summary information.

The installation settings you selected are displayed. Review the information for accuracy.

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

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**.

- a** If you selected that the Diagnostics commanding server was to be used with Business Service Management 9.0 you will see an additional dialog box.

OM Agent and IAPA component installations.

When integrating Diagnostics with Business Service Management **9.0**, the OM Agent and IAPA components must be installed on the Diagnostics Commander Server. These components are used by Diagnostics to send Health Indicator status events to the Business Service Management Gateway Server.

Confirm whether you want to install the OM agent (formerly OVO agent) and IAPA components.

If you are integrating with an earlier version of Business Service Management you do not need to install these components.

Check the box to install these components. Errors are reported in the `<Diagnostics_install_dir>/server/log.txt` file. See “OM Agent Troubleshooting” on page 774 if you have any problems with the installation.

If the OM agent is already installed on the system then these installers will update if they are an older version.

On Windows systems these components take a while to install. So you can leave the box unchecked to skip their installation.

The installer bits are always laid down so you can install the components later when needed from <Diagnostics_install_dir>/server/setup/ovo-agent and /ovoiapa. See “Starting and Stopping the Diagnostics Server” on page 63 for details.

Additional configuration steps are required in Business Service Management after installing these components. See “Registering the Diagnostics Server in Business Service Management 9.0” on page 618 for more information.

5 Close the installation wizard.

When the installation is complete, 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 does 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 63.

6 Upload the Diagnostics license file.

To view performance metrics, upload a valid HP Diagnostics license file after you install the Diagnostics command server.

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 mediator server, continue the installation as follows:

1 Provide the location of the Diagnostics command server.

Provide the information that enables the Diagnostics mediator server to connect to the Diagnostics command server.

- a** Enter the host name for the Diagnostics command server.
- b** Enter the port for the Diagnostics command server.

The default port for the Diagnostics command server is **2006**. If you changed the port since the Diagnostics Server was installed, 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 383.

- 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.

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 Service Management.

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

Note: If you selected only the **Business Service Management** 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.

4 Review the pre-installation summary.

The installation settings you selected are displayed. Review the information for accuracy.

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 is complete, 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 does 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 63.

Silent Installation of the Diagnostics Server

A *silent installation* 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 install step.

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 must 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.

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.

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

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

When performing a silent installation you can specify the following two additional options after the response file name.

- You can create a log file by specifying the **is:log <logfilepath>** option.
- You can change the temp directory to a user-specified directory by specifying the **is:tempdir <tempDirPath>** option.

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 Make sure 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>
```

In the example, <platform> is solaris, linux, or hpux. If the **M_LROOT** environment variable is not defined as the root directory, the following error is displayed:

```
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; for example:

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

On Linux, if you encounter an error you might 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; for 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:

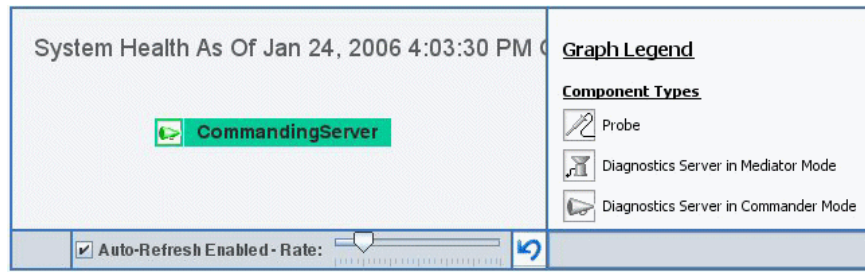
Terminate the process using a utility such as `kill`.

Verifying the Diagnostics Server Installation

To verify that the Diagnostics Server was installed correctly and 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 command server 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 command server is represented in the System Health Monitor component map by an icon labeled **Commanding Server**.

When a Diagnostics Server is deployed in Mediator mode, it is represented by a single icon labeled **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 is installed, 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 63.

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 install the Diagnostics Server, you must provide a valid license file. For instructions on requesting a license file and uploading it, 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 could 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 11, “Advanced Diagnostics Server Configuration.”

Determining the Version of the Diagnostics Server

When you request support, you must 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 displays the version of the Diagnostics Server.

Uninstalling the Diagnostics Server

The following section contains instructions for uninstalling the Diagnostics Server.

Important: Note that the OM agent is not uninstalled with the Diagnostics Server in case it is used by other products. If you want to uninstall the OM agent and IAPA components they must be uninstalled before you uninstall the server because the uninstaller for these components is under the server directory.

To uninstall the Diagnostics Server From a Windows Machine:

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

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

- 2** During the uninstallation process, a message asks if you want to remove specific files. Do the following:
 - ▶ 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, back up the property files located in the **etc** directory to a new location.

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

If you did not back up these files, select **No** or **No to All**.

To uninstall the Diagnostics Server From a UNIX Machine:

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

- 1 Stop the Diagnostics Server. For instructions, see “Starting and Stopping the Diagnostics Server” on page 63.
- 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
```

Manual Installation of OM Agent and IAPA Components

The installer for the Diagnostics commanding server includes installation of the OM agent and IAPA components used for sending Health Indicator status events to Business Service Management 9.00 or later. (If you are integrating with an earlier version of Business Service Management you do not need to install these components).

The OM Agent and IAPA component installation on Windows can take a while. So you can choose to skip installing these components during the Diagnostics Server installation and install the components manually on the Diagnostics commanding server at a later time as described below.

See “OM Agent Troubleshooting” on page 774 if you have any problems with the installation.

Note: The OM agent installer and IAPA installer bits are laid down during the Diagnostics server installation whether it is a commanding server or mediating server and even if you elect to install these components later.

To manually install OM agent on a Windows systems:

- 1** For Windows systems, change directory to `<Diagnostics_install_dir>/server/setup/ovo-agent/<platform>` where `<platform>` is either **win32** or **win64**.
- 2** From the command line in this directory execute
`cscript.exe opc_inst.vbs`

To manually install OM agent on Linux or Solaris systems:

- 1** For Linux or Solaris systems, change directory to `<Diagnostics_install_dir>/server/setup/ovo-agent/<platform>` where `<platform>` is either **Linux32** or **Linux64** or **solaris**.
- 2** As root user, from the command line in this directory execute
`./opc_inst`

To manually install the IAPA component on Windows systems:

- 1** For Windows systems, change directory to `<Diagnostics_install_dir>/server/setup/ovo-iapa/<platform>` where `<platform>` is either **win32** or **win64**.
- 2** From the command line in the win32 directory execute
`cscript.exe <install_dir>/sever/bin/install_ovo_iapa.vbs /i HPOprIAPA-09.00.111-WinNT4.0-release.msi`

Or from the command line in the win64 directory execute

`cscript.exe <install_dir>/sever/bin/install_ovo_iapa.vbs /i HPOprIAPA-09.00.111-Win5.2_64-release.msi`

To manually install the IAPA component on Linux or Solaris systems:

- 1 For Linux or Solaris systems, change directory to `<Diagnostics_install_dir>/server/setup/ovo-iapa/<platform>` where `<platform>` is either **Linux32** or **Linux64** or **solaris**.
- 2 As root user, from the command line in the Linux32 directory execute
rpm -ivh HPOprIAPA-09.00.111-Linux2.6-release.rpm
Or as root user, from the command line in the Linux32 directory execute
rpm -ivh HPOprIAPA-09.00.111-Linux2.6_64-release.rpm
Or as root user, from the command line in the solaris directory execute
**pkgadd -a ./noask_pkgadd -d
HPOprIAPA-09.00.111-SunOS5.10-release.sparc HPOprIAPA**

To complete the OM agent installation you must also do the following:

- 1 To complete the OM agent configuration you must complete the steps to register Diagnostics with Business Service Management. See “Registering the Diagnostics Server in Business Service Management 9.0” on page 618 for details relating to the OM agent.

Manual Uninstall of OM Agent and IAPA Components

The OM agent and IAPA components are not uninstalled when you uninstall the Diagnostics Server. If you want to uninstall the OM agent and IAPA components they must be uninstalled before you uninstall the server because the uninstaller for these components is under the server directory. And the components must be uninstalled in this order: first the IAPA component and then the OM agent.

To manually uninstall the IAPA component on Windows systems:

- 1 For Windows systems, change directory to `<Diagnostics_install_dir>/server/setup/ovo-iapa/<platform>` where `<platform>` is either **win32** or **win64**.
- 2 From the command line in the win32 directory execute

```
cscript.exe <install_dir>\sever\bin\install_ovo_iapa.vbs /x  
HPOprIAPA-09.00.111-WinNT4.0-release.msi uninstall.log
```

Or from the command line in the win64 directory execute

```
cscript.exe <install_dir>\sever\bin\install_ovo_iapa.vbs /x  
HPOprIAPA-09.00.111-Win5.2_64-release.msi uninstall.log
```

To manually uninstall the IAPA component on Linux or Solaris systems:

- 1** For Linux or Solaris systems, change directory to `<Diagnostics_install_dir>/server/setup/ovo-iapa/<platform>` where `<platform>` is either **Linux32** or **Linux64** or **solaris**.
- 2** As root user, from the command line in the Linux32 or Linux 64 directory execute

```
rpm -e HPOprIAPA
```

Or as root user, from the command line in the solaris directory execute

```
pkgadd HPOprIAPA
```

To manually uninstall OM agent on a Windows systems:

- 1** For Windows systems, change directory to `<Diagnostics_install_dir>/server/setup/ovo-agent/<platform>` where `<platform>` is either **win32** or **win64**.
- 2** From the command line in this directory execute

```
cscript.exe opc_inst.vbs -r
```

To manually uninstall OM agent on Linux or Solaris systems:

- 1** For Linux or Solaris systems, change directory to `<Diagnostics_install_dir>/server/setup/ovo-agent/<platform>` where `<platform>` is either **Linux32** or **Linux64** or **solaris**.
- 2** As root user, from the command line in this directory execute

```
./opc_inst -r
```


3

Licensing HP Diagnostics

HP Diagnostics requires you to upload valid licenses onto the Diagnostics command server.

This chapter includes:

- ▶ About HP Diagnostics Licensing on page 74
- ▶ Types of Licenses on page 74
- ▶ Licensing the Diagnostics Server in Commander Mode on page 75
- ▶ View License Information on page 78
- ▶ Licensing the Other Diagnostics Components on page 79

About HP Diagnostics Licensing

Diagnostics is licensed using a file that you upload to the Diagnostics command server. 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 Java and .NET Agents and Diagnostics mediator server first connect with the Diagnostics command server they are licensed based on the license installed on the Diagnostics command server.

Types of Licenses

At installation you are given a 30-day trial (Instant-On) license. With the Instant-On license you can install Diagnostics components, 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 is 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 will not 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 provide a valid license file.

Licensing the Diagnostics Server in Commander Mode

Obtain your Diagnostics license from your 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 command server by accessing the main Diagnostics Server UI page and selecting **Configure Diagnostics > Licensing**.

The License Management page opens providing the following:

- ▶ The MAC address 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 Application Server instance totals in your monitored environment, as well as, the number of .NET processes, Java probes and collector instances.

Diagnostics

License Management

System Information

Attribute	Value
MAC Address:	00:16:35:3B:6D:D1, 00:16:35:3B:6D:D2
<input type="button" value="Refresh"/>	

License Information

Attribute	Value
License:	Diagnostics Server License
Version:	1
Type:	Perpetual
Registered To:	Hewlett Packard
Issue Date:	Friday, July 27, 2007
Expiration:	Never
Number of application server and collector instances (JVMs and .NET processes):	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:

License information based on currently connected probes

Attribute	Value
Application Server Instances:	53
Number of .NET processes:	13
Number of Java probes:	35
Number of old .NET probes:	1
Number of unknown probes:	0
Number of collector instances:	5
<input type="button" value="Refresh"/>	
Details	

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

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 to 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 command server. The name of the file must end with the extension **.lic**.

In the License Management page you can see information on the **License information based on currently connected probes**. 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 or later 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.

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.

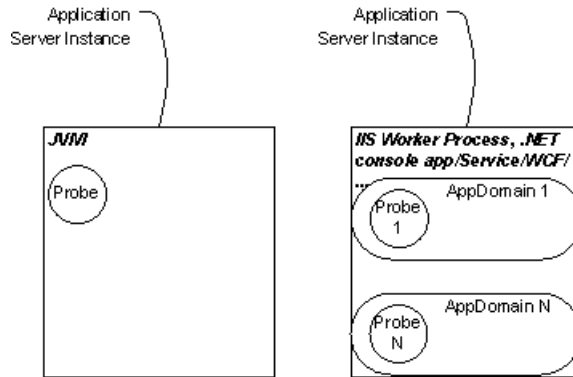
Type the path to the location where the license file was stored or click **Browse** to navigate to the license file location. Click **Upload** to apply the license file to the Diagnostics Server.

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.

View License Information

Information on your current licenses is reported in the License Management page. You can see the type of license, expiration date, if any, and the maximum number of application server (JVMs and .NET processes) and collector instances covered in the license.

You can also view license information based on currently connected probes. This information includes the number of application server instances, .NET processes, JVM instances and collector instances currently being monitored by Diagnostics.



- ▶ **Application Server Instance.** The physical process in which one or more probes are deployed or instances monitored by an SAP, Oracle, SQL or MQ collector.
- ▶ **.NET processes.** Any processes (application domains) instrumented for monitoring by one or more .NET probes. For example, IIS worker process or .NET console application/service/WCF. In the license report you may see the number of Old .NET probes which are probes versioned prior to 8.00.

- ▶ **JVM instances (Java probes).** Monitored java or javaw processes or any other processes embedding the JVM. This is equivalent to a Java probe.
- ▶ **Collector instances.** Includes the following:
 - ▶ Oracle - An instance in the (executed) Oracle software (Oracle processes) and the memory they use (SGA). A SID identifies an instance. Instances configured for monitoring with a <oracleInstance> entry in oracle-config.xml are counted.
 - ▶ SQL Server - Instances apply primarily to the database engine and its supporting components. Instances configured for monitoring with a <sqlserverInstance> entry in sqlserver-config.xml are counted.
 - ▶ MQ - Instances configured for monitoring with a <mqInstance> entry in mq-config.xml are counted.
 - ▶ SAP/ABAP - Each discovered Dialog instance (SAP ABAP probes) is counted.

Licensing the Other Diagnostics Components

The Diagnostics servers running as mediators and the Diagnostics agents do not have independent licenses. Their license is based on the license of the Diagnostics command server. The first time these components connect to a licensed Diagnostics command server, the Diagnostics Agents and Diagnostics mediator server are automatically licensed.

When you install the Java or .NET Agent, the Diagnostics Profiler is automatically installed. You view the Diagnostics Profiler in the context of a probe entity. The Diagnostics Profiler is an independent UI that can be accessed either directly on the system where the agent is installed or through the HP Diagnostics UI.

The Diagnostics Profiler operates in an unlicensed mode with load restrictions until a probe is able to connect to a Diagnostics command server that is properly licensed. In unlicensed mode, the Profiler is limited to capturing data from five concurrent threads.

4

Installing Diagnostics Collectors

You can install Diagnostics Collector on Windows and UNIX machines.

This chapter includes:

- About Installing the Diagnostics Collector on page 82
- Installing the Diagnostics Collector on a Windows Machine on page 82
- Installing the Diagnostics Collector on a UNIX Machine on page 90
- Silent Installation of the Diagnostics Collector on page 98
- Installing the Diagnostics Collector Using the Generic Installer on page 99
- Configuring the Active System Property Files on page 100
- Configuration for SAP NetWeaver–ABAP on page 100
- Configuration for Oracle on page 103
- Configuration for MQ on page 105
- Configuration for SQL Server on page 108
- Password Obfuscation on page 112
- Verifying the Diagnostics Collector Installation on page 114
- Starting and Stopping the Diagnostics Collector on page 115
- Determining the Version of the Diagnostics Collector on page 117
- Uninstalling the Diagnostics Collector on page 117

About Installing the Diagnostics Collector

The Diagnostics Collector gathers data from remote systems. You can configure the Collector to collect performance data from the following types of active systems:

- SAP NetWeaver–ABAP
- 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. Each monitored instance is represented by a probe entity. Multiple probes can be configured for each Collector.

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 36.

Launching the Installation

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

Note: Allow approximately 400MB of free space in the temp directory.

Note: If there is a pre-existing installation of the Collector on the host machine, see Appendix G, “Upgrade and Patch Install Instructions.”

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 **HPDiagCollector_<release number>.win.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 84.

To launch the Installer from the Business Service Management Diagnostics downloads page:

Note: The Collector installer is available in Business Service Management if you put it into the required directory for Business Service Management to access.

You can enable the Collector installer download from Business Service Management during the installation of the Diagnostic Server by providing the path to the Diagnostics Agent and Collector installers, or you can manually copy the `\Diagnostics_Installers\CollectorSetupWin_<version>.exe` file from the installation disk to the `<diag_server_install_dir>/html/opal/downloads` folder of the Diagnostics Server installation directory. See Step 3 on page 57 of Chapter 2, “Installing the Diagnostics Server.”

- 1** In **Business Service Management**, select **Admin > Diagnostics** from the top menu 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 launch 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.

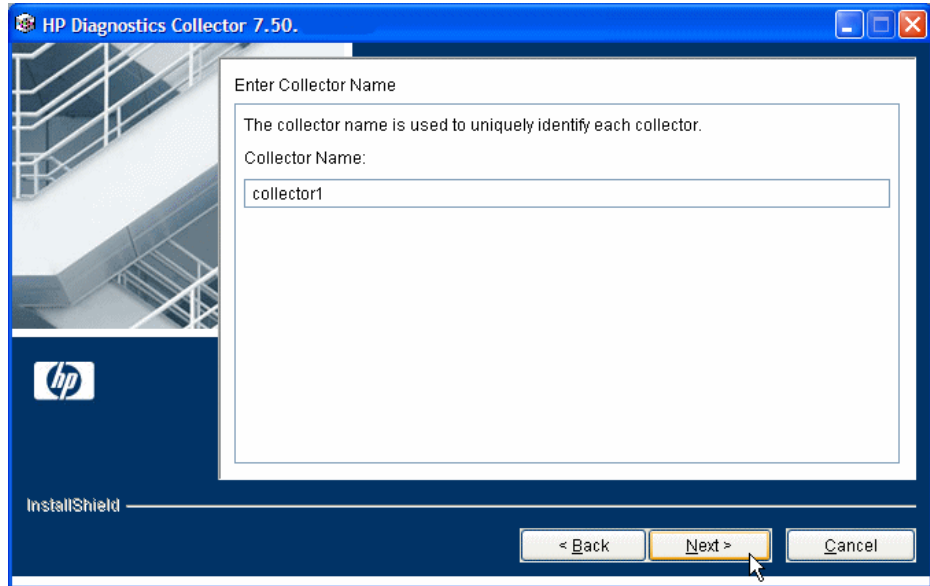
2 Specify the location to install the Collector.

In the **Installation Directory Name** box, type the name of the directory where you want to install the Collector. Or 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 you want to upgrade, cancel this installation and follow the upgrade procedure for Collectors as described in Appendix G, “Upgrade and Patch Install Instructions.”

Click **Next** to continue.

3 Assign a unique name to the Collector.

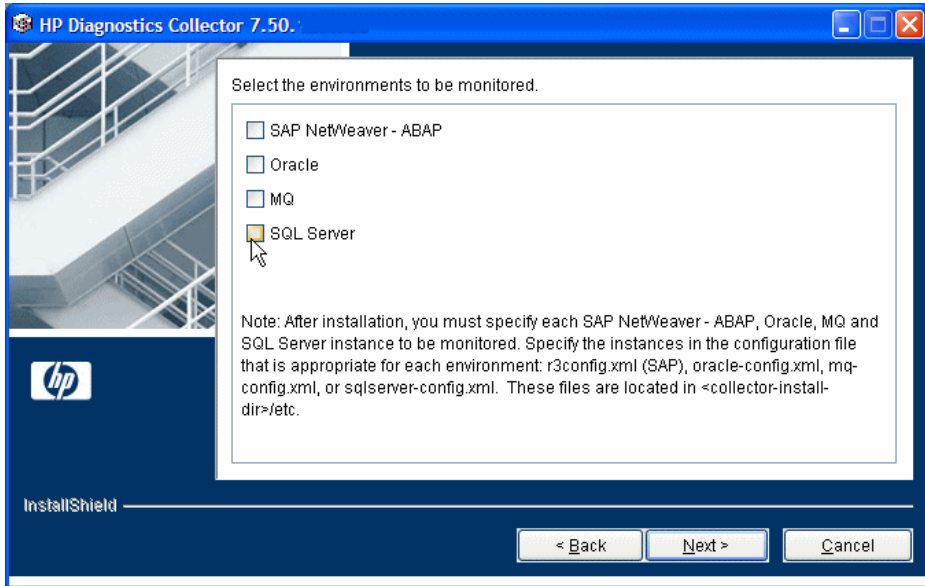


Assign a name to the Collector that uniquely identifies this specific Collector in the System Health Monitor. Use the System Health Monitor to verify the Collector installation and configuration. For more information, see “Verifying the Diagnostics Collector Installation” on page 114.

You can use -, _ and all alphanumeric characters in the name.

Click **Next** to continue.

4 Select the environment to monitor.



Select the options that apply to this Collector. You can select one or more options.

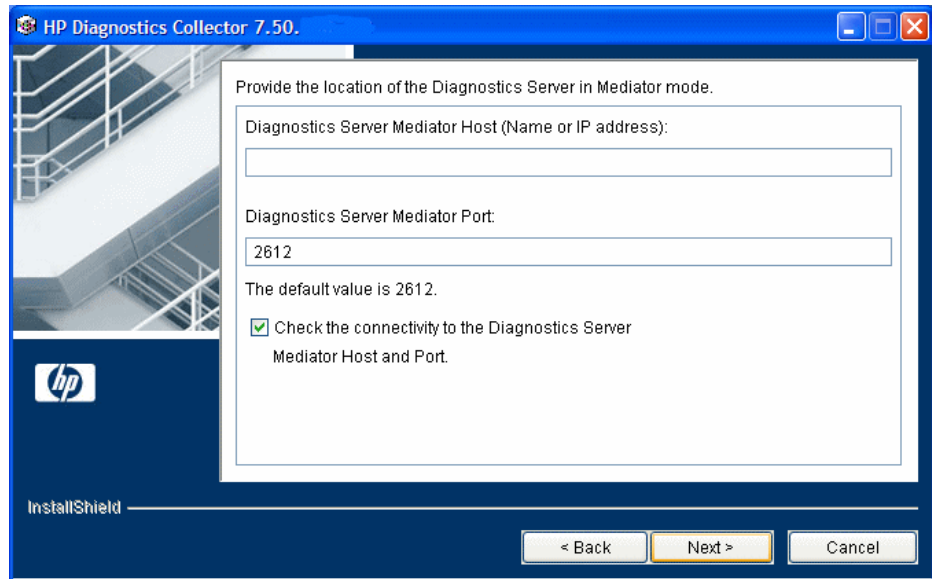
- To collect data in an SAP NetWeaver–ABAP environment, select **SAP NetWeaver–ABAP**.
- To collect data on an Oracle 10g database server, select **Oracle**.
- To collect data in an MQ series environment, select **MQ**.
- To collect data on an SQL Server database, select **SQL Server**.

Important: After installation, 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 100.

Click **Next** to continue.

5 Provide information about the Diagnostics mediator server.

Provide the details that enables communication with the Diagnostics mediator server.



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 mediator server 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 mediator server.

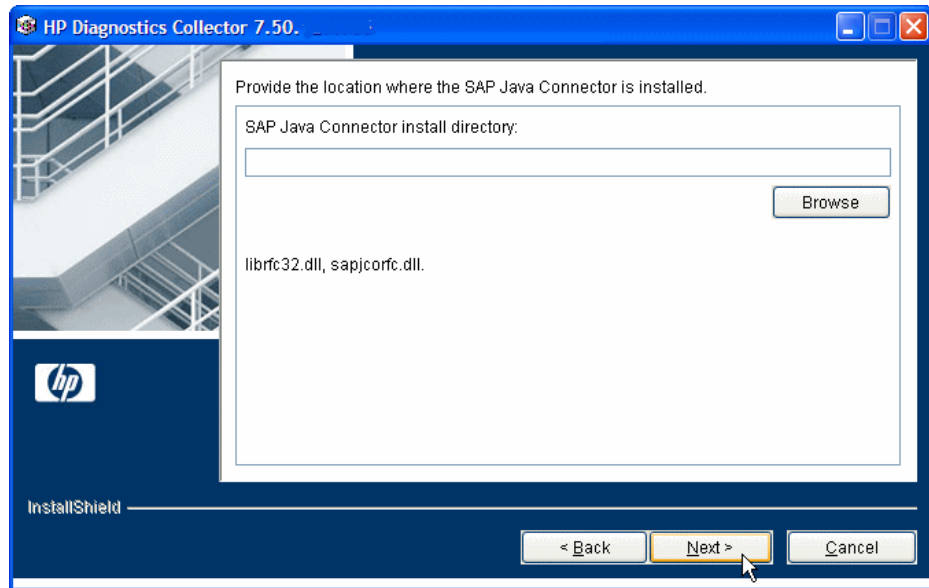
Note: You must specify the fully qualified host name. In a mixed OS environment, where UNIX is one of the systems, this is essential for proper network routing.

- 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 changed the port since the Diagnostics Server was installed, specify that port number instead of the default.
- c** 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 encountered connectivity problems, you will see the results of the connectivity check, which the installer provides. 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 you selected are displayed in a read-only window. Review the information for accuracy.

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

To begin installation, click **Next**.

8 Close the installation wizard.

When the installation completes, a message is displayed confirming that the Collector is 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 to be monitored. For each of these active systems, you must 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 100.

10 Verify that the 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 114.

In the Diagnostics UI, each collector instance is represented as a probe entity 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 the steps necessary to install the Collector in UNIX environments, using either a graphical installation or a console mode installation. For instructions on using the generic Unix installer for platforms others than Linux and Solaris see “Installing the Diagnostics Collector Using the Generic Installer” on page 99

The installation screens 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 82.

Launching the Installation

You can launch the installer directly from the Diagnostics installation disk or from the Diagnostics Downloads page in Business Service Management.

To launch the installation from the installation disk:

- 1** From the <HP Diagnostics Installation Disk>/Diagnostics_Installers directory, copy the installer **HPDiagCollector_<release number>_<platform>.bin** to the machine where the Collector is to be installed; for example **HPDiagCollector_9.00_linux.bin**.
- 2** Continue with “Running the Installation” on page 92.

To launch the Installer from the Business Service Management Diagnostics downloads page:

- 1** From the top menu in Business Service Management, 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 Service Management only if you provided the path to the Agent installers directory when you installed the Diagnostics Server in Commander mode. See “Installing the Diagnostics Server in Commander Mode” on page 56.

Continue with “Running the Installation” on page 92.

Running the Installation

After you copy 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, “Upgrade and Patch Install Instructions.”
 - ▶ The following instructions assume an understanding of UNIX console screens and commands. For more information about UNIX screens and commands, see “Using UNIX Commands” on page 781.
-

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 graphical mode, enter the <installer> executable at the UNIX command prompt, where <installer> is, for example:

```
HPDiagCollector_<release number>_sol.bin
```

```
HPDiagCollector_<release number>_linux.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 82.

- ▶ To run the installer in console mode, enter <installer> -console at the UNIX command prompt.

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.

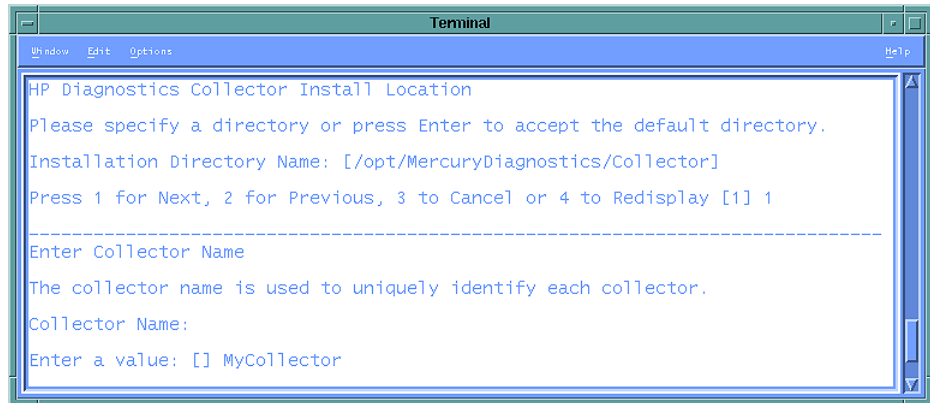
Select **Next** to continue with the installation.

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.



```

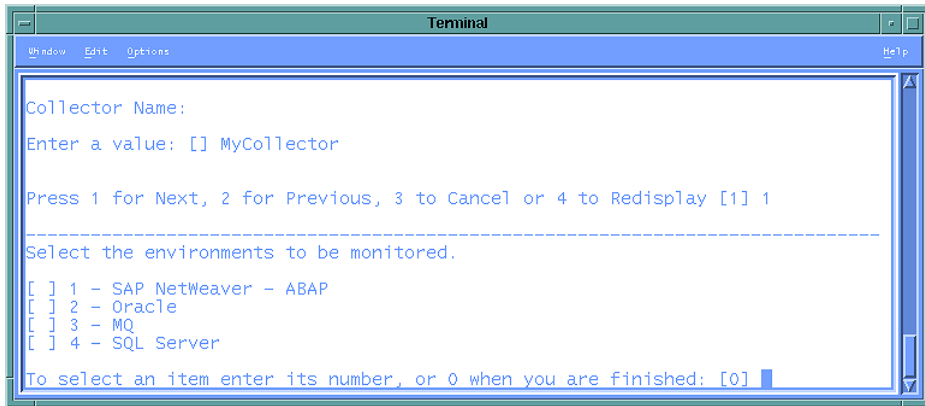
Terminal
Window Edit Options Help
HP Diagnostics Collector Install Location
Please specify a directory or press Enter to accept the default directory.
Installation Directory Name: [/opt/MercuryDiagnostics/Collector]
Press 1 for Next, 2 for Previous, 3 to Cancel or 4 to Redisplay [1] 1
-----
Enter Collector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

Assign a name to the Collector that uniquely identifies 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 114.

You can use -, _ and all alphanumeric characters in the name.

Select **Next** to continue.

5 Specify the active systems that the Collector will monitor.



Select the active systems that apply to this Collector.

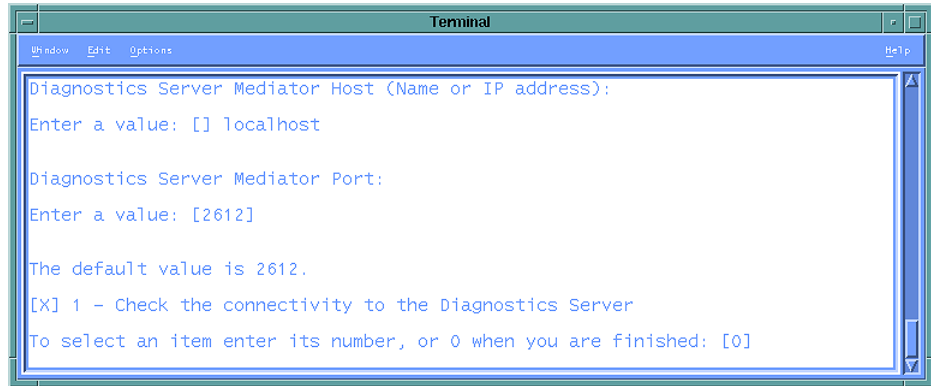
- To collect data in an SAP NetWeaver–ABAP environment, select **SAP NetWeaver–ABAP**.
- To collect data on an Oracle 10g database server, select **Oracle**.
- To collect data in an MQ series environment, select **MQ**.
- To collect data on an SQL Server database, select **SQL Server**.

Important: After installation, you must 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 100.

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

6 Provide the details for the Diagnostics mediator server.

Provide the information that enables communication with the Diagnostics mediator server.



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 mediator server that is to receive the events from the Collector.

- a** Enter the host name or IP address of the host of the Diagnostics mediator server.

Note: You must 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 changed the port since the Diagnostics Server was installed, specify that port number instead of the default.

- 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

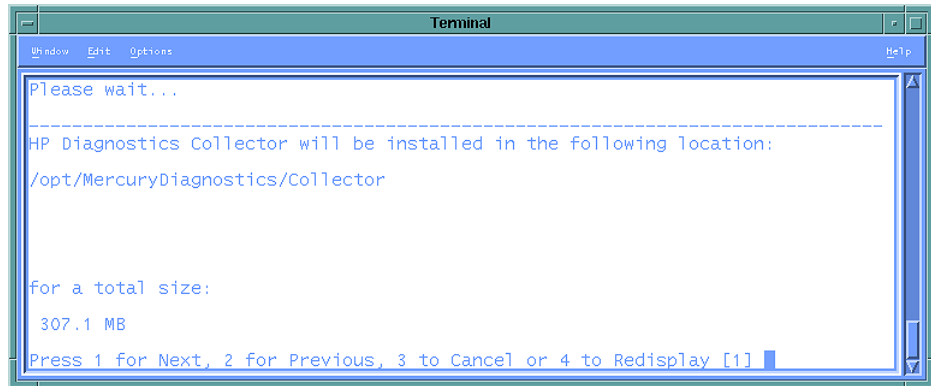
Notes:

- 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 selected are displayed. Review the information for accuracy.



```

Terminal
-----
Please wait...

-----
HP Diagnostics Collector will be installed in the following location:
/opt/MercuryDiagnostics/Collector

for a total size:
307.1 MB

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

```

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 completes, 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 to be monitored. For each of these active systems, you must 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 100.

11 Verify the Collector installation.

Using the System Health Monitor, verify that the Collector is running as it should be. For details, see “Verifying the Diagnostics Collector Installation” on page 114.

Silent Installation of the Diagnostics Collector

A *silent installation* 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 install step.

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 must 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.

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.

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

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

When performing a silent installation you can specify two additional options.

- You can create a log file by specifying the **is:log <logfilepath>** option after the response file name.
- You can change the temp directory to a user-specified directory by specifying the **is:tempdir <tempDirPath>** option after the response file name.

Installing the Diagnostics Collector Using the Generic Installer

The installers for the Diagnostics Collector support installing the Collector on Windows, Linux and Solaris systems. A generic Unix installer is provided on the installation disk to allow you to install the Collector on other platforms such as HP-UX and AIX.

To install the Diagnostics Collector using the generic installer:

- 1** Locate **HPDiagCollector_<release version>_unix.zip** from the Diagnostics Installers folder on the HP Diagnostics installation disk.
- 2** Unzip the file on the system where you want the Collector installed.
- 3** Then for each of the active systems you want to monitor, you must 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 100.

Configuring the Active System Property Files

When you install the Collector, you are asked to indicate the active systems 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 is viewed as a probe entity on the instance of the active system.

Configuration for SAP NetWeaver–ABAP

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 might be direct or might require connection through the SAP Message Server. For each SAP NetWeaver–ABAP probe entity, you must know what connection option is used.

You configure the Collector to collect data for each instance of an active SAP NetWeaver–ABAP system to be monitored. You configure SAP NetWeaver–ABAP for monitoring in the `<collector_install_dir>\etc\r3config.xml` file. The layout, elements, and attributes of the xml file are described in `<collector_install_dir>\etc\r3config.xsd`.

To configure SAP NetWeaver–ABAP monitoring:

- 1 Open `<collector_install_dir>\etc\r3config.xml`.
- 2 If you are defining an SAP NetWeaver–ABAP probe entity 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 entity 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 save the file.

Property	Description	Value
r3system name	A logical name for the probe group under which this SAP NetWeaver–ABAP probe entity appears in the Diagnostics UI	User-defined.
systemId	The ID of the SAP NetWeaver–ABAP system. Consists of 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	<p>The name of the user connecting to the SAP NetWeaver–ABAP system</p> <p>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 is to install the Collector on a machine that is time-synched with the ABAP host and then disable time-synching in the Collector by setting property <code>timesynch.interval.secs = 0</code> (in <code><collector_install_dir>\etc\r3.properties</code>).</p>	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 <code>EncryptPassword.jsp</code> utility (see “Password Obfuscation” on page 112) to encrypt the password.
messageServerHost (Message Server connection only)	The name of the SAP Message Server host machine	Obtainable from the SAP system administrator.
r3Name (Message Server connection only)	Consists of 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.

Configuration for Oracle

You configure the Collector to collect data for each instance of an active Oracle system to be monitored. You configure Oracle monitoring in the `<collector_install_dir>\etc\oracle-config.xml` file. The layout, elements, and attributes of the xml file are described in `<collector_install_dir>\etc\oracle-config.xsd`.

To configure Oracle monitoring:

- 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 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 <code>EncryptPassword.jsp</code> utility (see “Password Obfuscation” on page 112) to encrypt the password.

Properties	Description	Value
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 is used.
probeGroupName	The logical name of the probe group under which the probe entity 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

Configuration for MQ

You configure the Collector to collect data for each instance of an active MQ system to be monitored. You configure MQ monitoring in the `<collector_install_dir>\etc\mq-config.xml` file. The layout, elements, and attributes of the xml file are described in `<collector_install_dir>\etc\mq-config.xsd`.

The MQ probe 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 configure MQ monitoring:

- 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 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	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 entity 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 the `<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
<code>collect.predefined.queues</code>	<code>MQQDT_PREDEFINED</code>
<code>collect.permanent.dynamic.queue</code>	<code>MQQDT_PERMANENT_DYNAMIC</code>
<code>collect.temporary.dynamic.queues</code>	<code>MQQDT_TEMPORARY_DYNAMIC</code>
<code>collect.shared.dynamic.queues</code>	<code>MQQDT_SHARED_DYNAMIC</code>

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

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

Configuration for SQL Server

You configure the Collector to collect data for each instance of an active SQL Server system to be monitored. You configure SQL monitoring in the `<collector_install_dir>\etc\sqlserver-config.xml` file. The layout, elements, and attributes of the xml file are described in `<collector_install_dir>\etc\sqlserver-config.xsd`.

To configure SQL Server monitoring:

- 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 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 . To exclude some of these databases from collection (for example, system databases), specify a comma-separated list in the exclude.db.list property in the <code><collector_install_dir>\etc\sqlserver.properties</code> file.	Default value: Default Obtainable from the SQL Server administrator.

Properties	Description	Value
<p>integratedSecurity</p>	<p>If set to true, no username/ password should be specified. The JDBC driver searches the local computer credential cache for credentials that have been provided at the computer or network logon.</p> <p>When the collector is run from the service HP Diagnostics Collector, the Windows user credentials used to connect to SQL Server must be set as the logon property for the service. To do this, run the Windows Services Manager (services.msc from the run dialog, or My Computer > Manage > Services and Applications > Services). Open the Properties dialog for service HP Diagnostics Collector, select the Log On tab, and set Log on as: to the user granted access to your SQL Server instance. This must be the domain account. Restart the service.</p> <p>When using Windows authentication, a domain account (not a local one) needs to be used for making the connection to the SQL Server instance.</p> <p>If set to false, the username and password must be supplied. If this is not specified, its default value is false.</p>	<p>Default value: false</p>

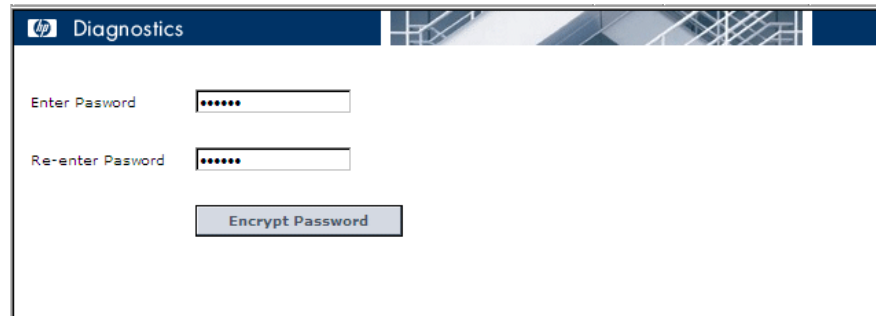
Properties	Description	Value
userId	<p>The ID of the user connecting to the SQL Server database</p> <p>Note: The user needs at least VIEW SERVER STATE to collect performance metrics.</p> <p>Create the user as follows for VIEW SERVER STATE:</p> <ul style="list-style-type: none"> ➤ CREATE LOGIN diag WITH PASSWORD = '<pwd>'; ➤ USE master; ➤ GRANT VIEW SERVER STATE TO diag; ➤ GO <p>From the SQL Server Management Studio GUI, you can right-click on the Instance name and select the properties.</p>	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.
encrypted-password	The password (encrypted) of the user connecting to the SQL Server database.	Use the EncryptPassword.jsp utility (see “Password Obfuscation” on page 112) to encrypt the password.

Properties	Description	Value
probeName	<p>The logical name to represent this Oracle instance in the Diagnostics UI. This name must be unique.</p> <p>When you have <i>n</i> databases in your instance, you actually have <i>n+1</i> probes: an extra probe for the totals of the instance that includes metrics such as wait events.</p> <p>The extra probe is shown in the UI as probeName. The probes for each database are shown as probeName_databaseName.</p>	<p>User-defined.</p> <p>If this value is not defined, the same value given for instanceName is used.</p>
probeGroupName	<p>The logical name of the probe group under which this probe entity appears in the Diagnostics UI.</p> <p>This can be an existing probe group or you can define a new one.</p>	<p>User-defined; for example:</p> <p>Existing: Default</p> <p>New: SQL Server</p>

Password Obfuscation

Create an obfuscated password using the web application included with Diagnostics: **http://<host name>:2006/security/EncryptPassword.jsp**. Replace <host name> 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.

The screenshot shows a web interface titled "Diagnostics" with the HP logo. It contains two text input fields: "Enter Password" and "Re-enter Password", both filled with six dots. Below these fields is a button labeled "Encrypt Password".

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

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

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 affected section of these properties files:

```
#####  
# 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:1c431jg81hv41k1d11161wu81z0d1pyl1wmt1n6h1y  
m71n511wnd1pw11z0h1wu61kxw1jyl1hse1jd21c2z
```

Verifying the Diagnostics Collector Installation

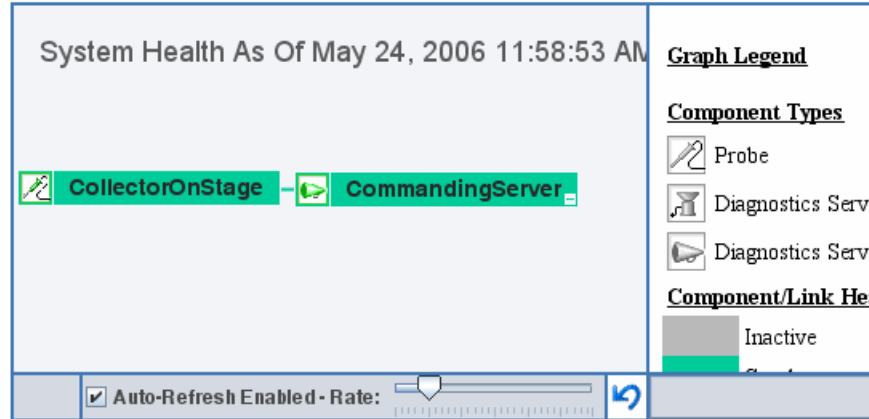
The Collector starts running automatically when the installation is complete. You can verify the Collector installation using the System Health Monitor. For detailed instructions on using the System Health Monitor, see Appendix D, “Using the System Health Monitor.”

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

- ▶ The Diagnostics command server is successfully installed.
- ▶ Where relevant, additional Diagnostics Servers in Mediator mode are successfully installed and communicating with the Diagnostics command server.

- ▶ The Collector is successfully installed and connected 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 command server or a Diagnostics mediator server. The Collector is represented by an icon labeled **<name of Collector>**. The name of the collector is assigned during the installation procedure.



When the Collector is stopped, the Collector node in the System Health monitor is gray. When the Collector starts, 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 Make sure 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 see 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
```

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 using 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 or 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**.

Or 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

This section includes:

- Installing Java Agents
- Running the JRE Instrumenter
- Configuring Application Servers for the Java Agent
- Installing .NET Agents

5

Installing Java Agents

This section describes how to install a Java Agent on Windows machines, UNIX machines, and z/OS mainframe machines. It also describes how to use a generic installer to install the Java Agent on other platforms.

This chapter includes:

- ▶ About the Java Agent Installer on page 122
- ▶ Installing and Configuring the Java Agent on Windows on page 123
- ▶ Installing and Configuring the Java Agent on UNIX on page 139
- ▶ Installing the Java Agent on a z/OS Mainframe on page 154
- ▶ Installing the Java Agent Using the Generic Installer on page 156
- ▶ Silent Installation of the Java Agent on page 158
- ▶ About Configuring the Application Server on page 160
- ▶ Verifying the Java Agent Installation on page 161
- ▶ Determining the Version of the Java Agent on page 162
- ▶ Uninstalling the Java Agent on page 163
- ▶ About Advanced Java Agent Configuration on page 163
- ▶ About Custom Instrumentation for Java Applications on page 163

About the Java Agent Installer

The HP Diagnostics/TransactionVision Java Agent installer installs a Java Agent to collect data for either Diagnostics or TransactionVision or both. TransactionVision provides data to the Transaction Management application in Business Service Management. See the *TransactionVision Deployment Guide* in the Business Service Management Documentation Library for more information about installing the Java Agent for TransactionVision.

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

Before you can use a Java Agent to monitor an application in HP Diagnostics, you must:

- ▶ Install the Java Agent and complete the setup module.
- ▶ Instrument the JRE (see Chapter 6, “Running the JRE Instrumenter”).
- ▶ Configure the application server startup scripts to work with the Java Agent (see Chapter 7, “Configuring Application Servers for the Java Agent”).

Note: Allow approximately 400MB of free space in the temp directory.

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 Agent for Java Host” on page 34.

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” on page 123
- “Installing and Configuring the Java Agent on UNIX” on page 139
- “Installing the Java Agent on a z/OS Mainframe” on page 154
- “Installing the Java Agent Using the Generic Installer” on page 156
- “Silent Installation of the Java Agent” on page 158

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 “Upgrade and Patch Install Instructions” on page 755 for important instructions on how to upgrade the agent systems.

This section includes:

- “Launching the Java Agent Installer on Windows” on page 123
- “Running the Installation” on page 125
- “Configuring the Java Agent as a Profiler Only” on page 126
- “Configuring the Java Agent to Work with a Diagnostics Server” on page 131

Launching the Java Agent Installer on Windows

You can install the Java Agent from the Diagnostics installation disk or copy the executable installation file to another location and run it, or select to install the Java Agent from the Diagnostics Downloads page in Business Service Management.

When installing just the Profiler trial software, you launch the installer from the HP Software Web site.

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 Java** to launch the installer.
- 3** Continue with “Running the Installation” on page 125.

To launch the installer from another location:

- 1** From the <HP Diagnostics Installation Disk>\Diagnostics_Installers directory, copy the executable file **HPDiagTVJavaAgt_<release number>.win.exe** to the machine where the Java agent is to be installed.
- 2** Continue with “Running the Installation” on page 125.

To launch the installer from the HP Software Trial Software Download site:

- 1** Go to the HP Software web site’s Download Center.
- 2** In the **Quick Search** section, in the **Products** list, click **Diagnostics** and choose the appropriate link.
- 3** Follow the download instructions on the web site.

Continue with “Running the Installation” on page 142.

To launch the installer from the Business Service Management Diagnostics downloads page:

- 1** In Business Service Management 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 Service Management only if they are placed into a directory that Business Service Management can access. You can enable this during the installation of the Diagnostic Server, or you can copy the Java Agent installers manually from the installation disk to the required location.

Continue with “Running the Installation” on page 125.

Running the Installation

After you launch 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 “Upgrade and Patch Install Instructions” on page 755 for important instructions on how to upgrade the agent systems.

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 starts.

Configuring the Java Agent as a Profiler Only

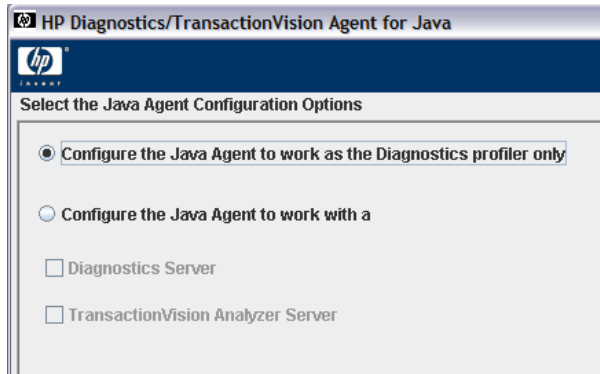
The Java Agent can be configured as a standalone agent that works with the Diagnostics Profiler or as an agent that works with a Diagnostics Server. When the agent is configured as a Profiler only, you can reconfigure the agent to work with the Diagnostics Server.

To configure the agent as a Diagnostics Profiler only, 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 Configure the Java Agent to work as the Diagnostics Profiler only option.

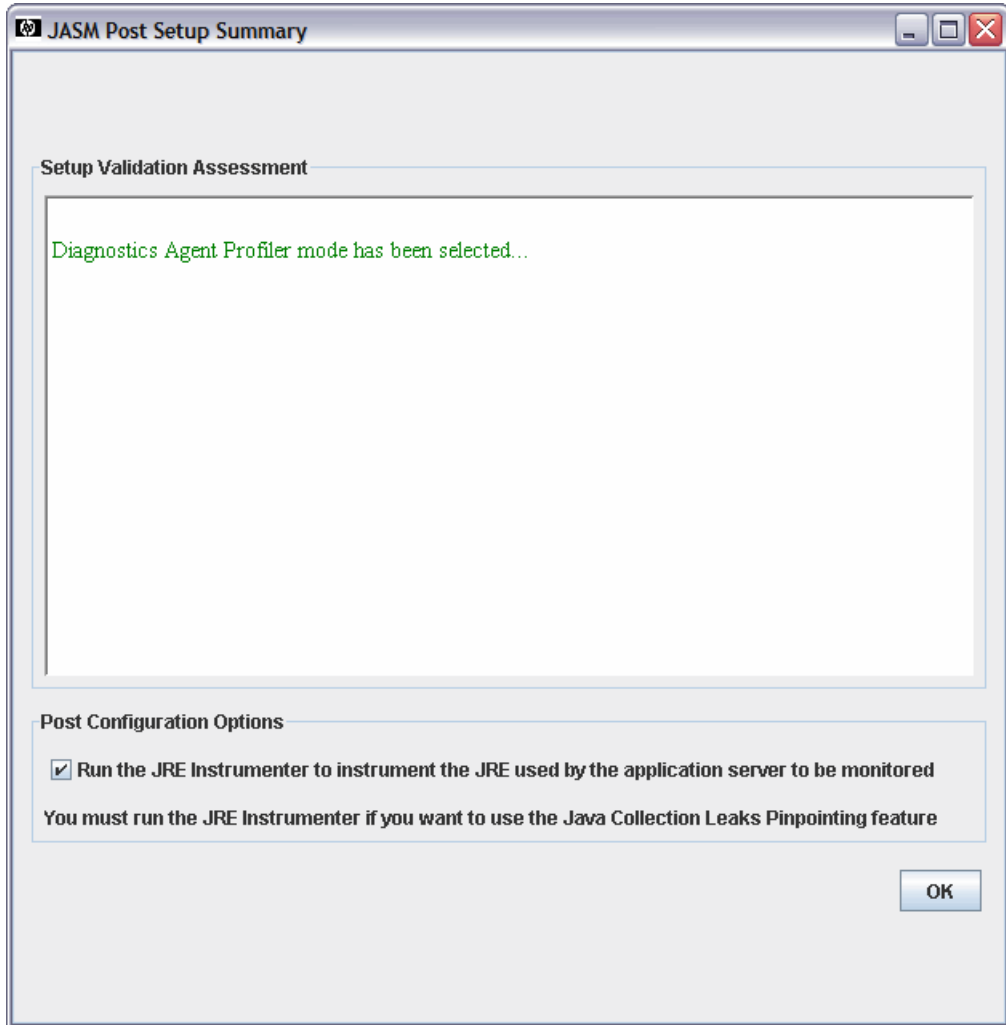


There are no other configuration options.

Click **Finish** to complete the configuration.

The Post Setup Summary dialog is displayed.

2 Review the Post Setup Summary and 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**.

Regardless of JRE version, to enable all Diagnostics capabilities, to maximize the instrumentation coverage and to optimize monitored application performance, you should run the JRE Instrumenter. After you instrument the JRE version used by your application, copy the command line options presented by the JRE Instrumenter to the Java command in your application startup script. For information about running the instrumenter, see Chapter 6, “Running the JRE Instrumenter.”

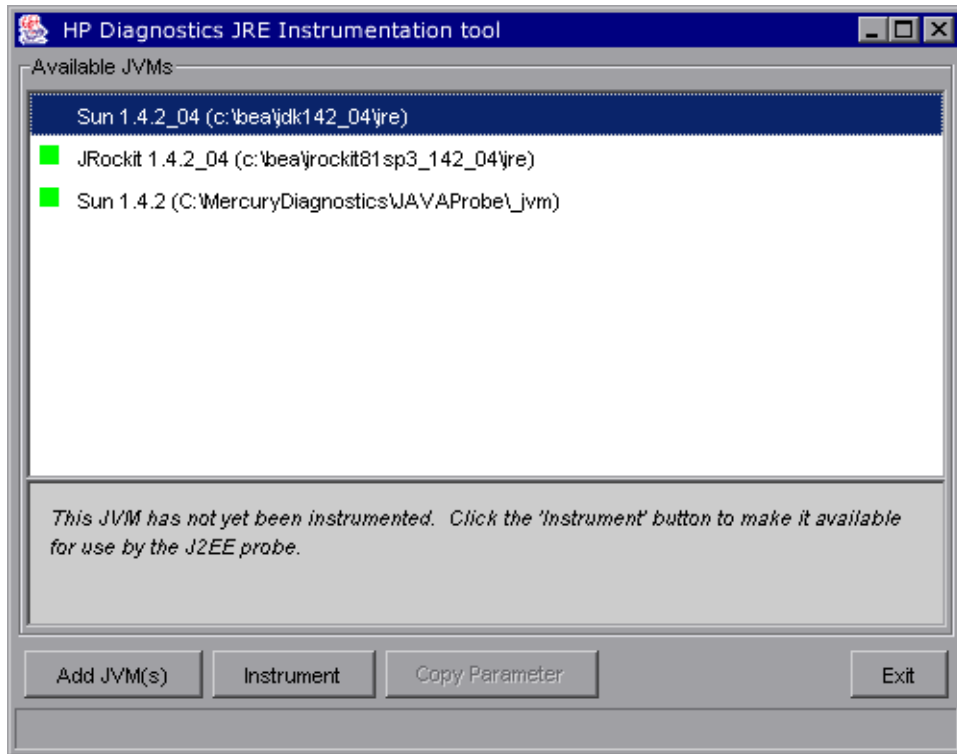
To run the JRE Instrumenter later, clear the check box and click **OK**. The JRE Instrumenter window closes and the Java Agent installation and setup is complete.

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. For more information, see “Running the JRE Instrumenter for WebSphere IDE” on page 192.

3 Instrument the JVM and copy the startup parameters.

You can continue to run the JRE Instrumenter from the Java Agent Setup program by checking the post configuration option “Run the JRE Instrumenter to instrument the JRE used by the application server to be monitored” and clicking OK. The JRE Instrumenter window is then displayed.

- 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 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 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.

After you install the agent, configure it as a Java Agent, and instrument the JRE, do the following:

- 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 Servers for the Java Agent.”

- Perform additional probe configurations.

Determine which advanced configurations of the probe apply to your environment. See Chapter 12, “Advanced Java Agent and Application Server Configuration.”

Configuring the Java Agent to Work with a Diagnostics Server

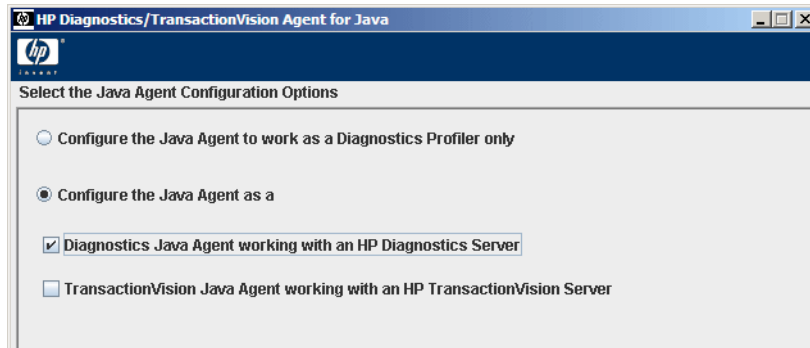
The Java Agent can be configured as a standalone Java Agent that works with the Diagnostics Profiler or as a Java Agent that works with a Diagnostics Server.

To configure the Java Agent as a Java Agent reporting to a Diagnostics Server, 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 configure 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 identifies the agent within HP Diagnostics. You can use -, _ and all alphanumeric characters in the name. The agent name is assigned as the default probe entity name. If you have a single agent installed on a system and plan to monitor multiple application servers or application domains you can later configure unique probe names for each monitored application.

When assigning a name to an agent, choose a name that will help you recognize the application being monitored (for example WebLogic, WebSphere) and the type of probe (for example Java, .NET, Oracle).

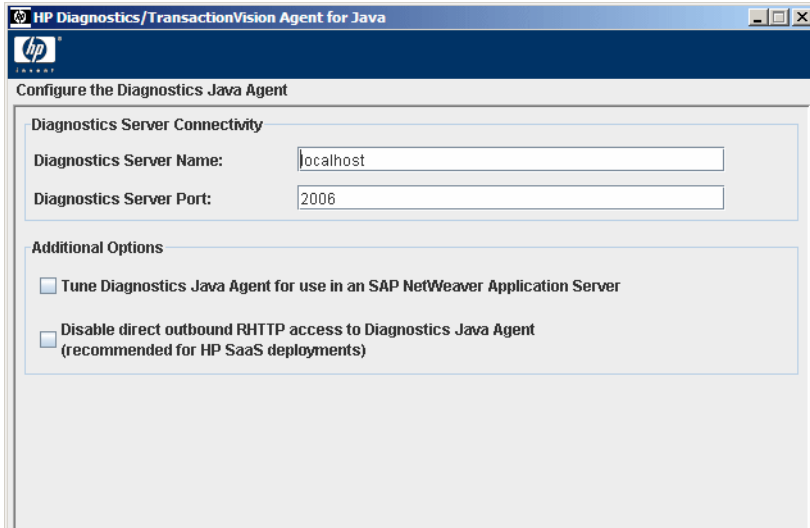
- For the Java Agent group name, enter a name for an existing group or a new group to be created. The agent group name is case-sensitive. The agent group name is used as the probe group name.

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 can assign all of the probes for a particular enterprise application to a probe group so that you can monitor both the performance at the group level and the performance based on individual probe entities.

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.

Commanding Server. If there is only one Diagnostics Server in the Diagnostics deployment where the agent will run, enter the Diagnostics Server (in Commander Mode) host name and event port information here.

Mediating Server. In a distributed environment with a commanding server and mediating servers, enter the information for the Diagnostics Server (in Mediator Mode) 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 changed the port since the Diagnostics Server was installed, you should specify that port number here instead of the default.

- c** 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** To allow this agent to work in an HP SaaS environment, set the **Disable direct outbound RHTTP access to Diagnostics Java Agent** check box.

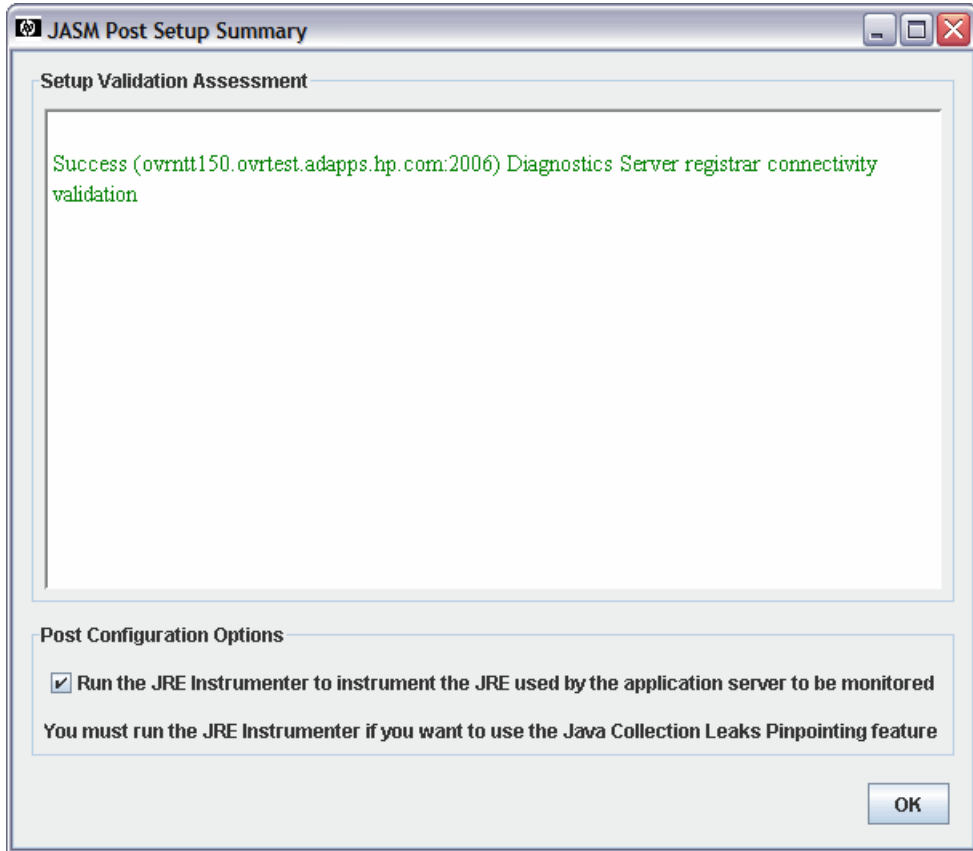
The Diagnostics Server uses rhttp to query metrics from the Java Agent. You should disable the probe's outgoing rhttp to prevent HP Software-as-a-Service (SaaS) from accessing the data directly from the probe.

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

The configuration process begins. A progress bar indicates how the configuration is proceeding.

The connectivity to the Diagnostics Server is tested. If any connectivity problems are encountered, the Set Up Program displays the results of the connectivity check.

4 Review the Post Setup Summary and 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**.

Regardless of JRE version, to enable all Diagnostics capabilities, to maximize the instrumentation coverage and to optimize monitored application performance, you should run the JRE Instrumenter. After you instrument the JRE version used by your application, copy the command line options presented by the JRE Instrumenter to the Java command in your application startup script. For information about running the instrumenter, see Chapter 6, “Running the JRE Instrumenter.”

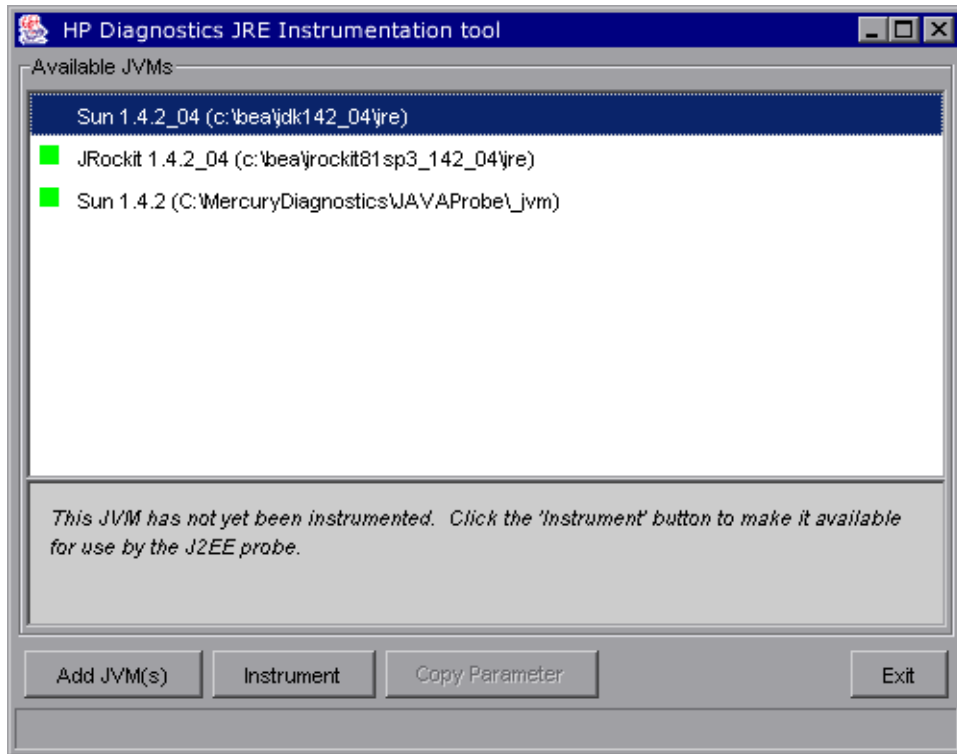
To run the JRE Instrumenter later, clear the check box and click **OK**. The JRE Instrumenter window closes and the Java Agent installation and setup is complete.

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. For more information, see “Running the JRE Instrumenter for WebSphere IDE” on page 192.

5 Instrument the JVM and copy the startup parameters.

You can continue to run the JRE Instrumenter from the Java Agent Setup program by checking the post configuration option “Run the JRE Instrumenter to instrument the JRE used by the application server to be monitored” and clicking OK. The JRE Instrumenter window is then displayed.

- 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 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 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.

6 Proceed with post-installation tasks.

Once you install the agent, configure it as a Java Agent, and instrument the JRE, perform the following 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 Servers for the Java Agent.”

- ▶ 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 161.

- ▶ Perform additional probe configurations.

Determine which advanced configurations apply to your environment. See Chapter 12, “Advanced Java Agent and Application Server Configuration.”

Installing and Configuring the Java Agent on UNIX

Java Agent installers are provided for several UNIX platforms. The following instructions describe the steps for installing 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 123 section because 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 could further restrict access to this directory; for example:

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

You might not be able to use the regular UNIX installers. In this case, use the Generic installer as described in “Installing the Java Agent Using the Generic Installer” on page 156.

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

Note: If there is a pre-existing installation of the Java Agent on the host machine, see “Upgrade and Patch Install Instructions” on page 755 for important instructions on how to upgrade the agent systems.

This section includes:

- “Setting File Permissions (UNIX Only)” on page 141
- “Launching the Java Agent Installer on UNIX” on page 141
- “Running the Installation” on page 142
- “Configuring the Java Agent as a Profiler Only” on page 144
- “Configuring the Java Agent to Work with a Diagnostics Server” on page 148

Setting File Permissions (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 might want to further restrict access to this directory; for example:

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

Launching the Java Agent Installer on UNIX

Launch the installer from the HP Software web site when installing the Profiler trial software. You could also install the Java Agent from the Diagnostics product installation disk or from the Diagnostics Downloads page in Business Service Management, when you have the full Diagnostics product.

To launch the Installer from the product installation disk:

- 1** From the <HP Diagnostics InstallationDisk>/Diagnostics_Installers directory, copy the appropriate installer **HPDiagTVJavaAgt_<release number>_<platform>.bin** to the machine where the Java Agent is to be installed.
- 2** Continue with "Running the Installation" on page 142.

To launch the installer from the HP Software Trial Software Download site:

- 1** Go to the HP Software web site's Download Center.
- 2** In the **Quick Search** section, in the **Products** list, click **Diagnostics** and choose the appropriate link.
- 3** Follow the download instructions on the web site.

Continue with "Running the Installation" on page 142.

To launch the installer from the Business Service Management Diagnostics downloads page:

- 1** In Business Service Management 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 Service Management only if enabled during the installation of the Diagnostic Server. See Step 3 on page 57 of Chapter 2, "Installing the Diagnostics Server."

Continue with "Running the Installation" on page 142.

Running the Installation

After you copy the installer to the machine where the agent is to be installed, you are ready to run the installation.

Note: If there is a pre-existing installation of the Java Agent on the host machine, see "Upgrade and Patch Install Instructions" on page 755 for important instructions on how to upgrade the agent systems.

Note: The following instructions assume an understanding of UNIX console screens and commands. For more information about UNIX screens and commands, see “Using UNIX Commands” on page 781.

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 following steps.

- ▶ To run the installer in 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 123.

2 Accept the end user license agreement.

The end user software license agreement is displayed.

Read the agreement. 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 can take a few minutes.

The Java Agent Setup Module launches.

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 agent is configured as a Profiler only, you can reconfigure the agent to work with the Diagnostics Server.

To configure the Java Agent as a Diagnostics Profiler, 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:

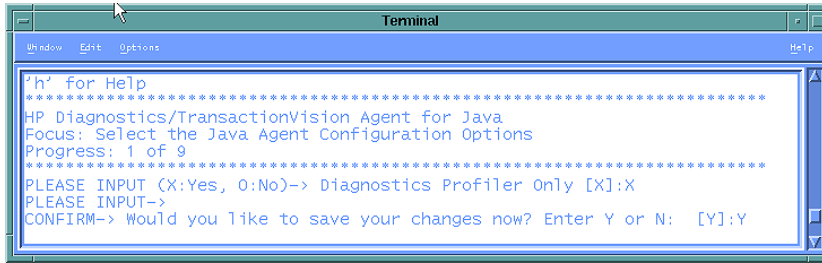
- 1** Enter **0** to select the **Diagnostics Profiler Only** option. There are no other configuration options.

```

Terminal
Window Edit Options Help
# ./setupModule.sh
*****
INFORMATION-> [Welcome]: Welcome to the Java Agent Setup Module (JASM)
INFORMATION-> [Init]: Initializing...
INFORMATION-> [Init]: Initializing...complete
INFORMATION-> [Options]:
At any input prompt you may enter:
'Enter' for default value,
'-' for Prev Step,
'+' for Next Step,
'c' to Cancel,
'f' to Finish, or
'h' for Help
*****
HP Diagnostics/TransactionVision Agent for Java
Focus: Select the Java Agent Configuration Options
Progress: 1 of 9
*****
PLEASE INPUT (X:Yes, 0:No)-> Diagnostics Profiler Only [0]:

```

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

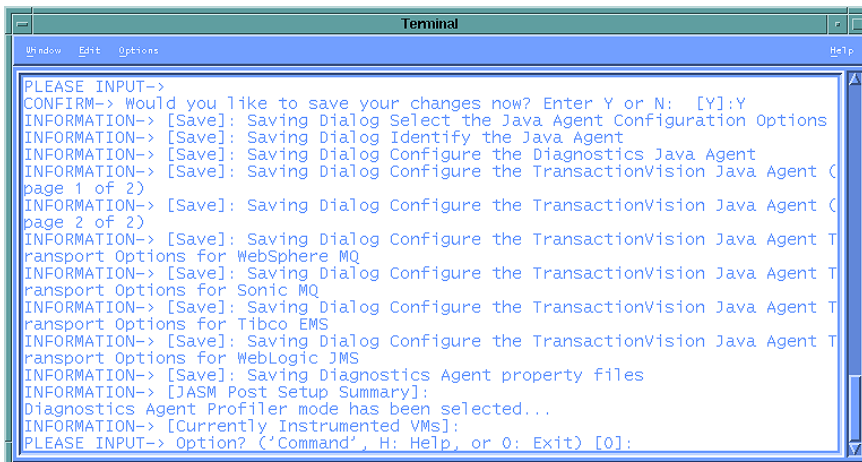


```

'h' for Help
*****
HP Diagnostics/TransactionVision Agent for Java
Focus: Select the Java Agent Configuration Options
Progress: 1 of 9
*****
PLEASE INPUT (X:Yes, 0:No)-> Diagnostics Profiler Only [X]:X
PLEASE INPUT->
CONFIRM-> Would you like to save your changes now? Enter Y or N: [Y]:Y

```

3 Run the JRE Instrumenter.



```

PLEASE INPUT->
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 T
ransport Options for WebSphere MQ
INFORMATION-> [Save]: Saving Dialog Configure the TransactionVision Java Agent T
ransport Options for Sonic MQ
INFORMATION-> [Save]: Saving Dialog Configure the TransactionVision Java Agent T
ransport Options for WebLogic JMS
INFORMATION-> [Save]: Saving Diagnostics Agent property files
INFORMATION-> [JASM Post Setup Summary]:
Diagnostics Agent Profiler mode has been selected...
INFORMATION-> [Currently Instrumented VMs]:
PLEASE INPUT-> Option? ('Command', H: Help, or 0: Exit) [0]:

```

Enter an instrumentation command:

- To instrument the JVM in a specific directory and copy the classloader information to `<probe_install_dir>/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 installed WebLogic 8.1 (using Sun JDK) in `/opt/bea`, enter:

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

- To instrument the JVM in a specific directory (most users will never need to use this option), enter:

```
jreinstrumenter.sh -i <jvm_directory>
```

- To list JVMs below a specific directory, enter:

```
jreinstrumenter.sh -a <directory>.
```

- To list all known JVMs, enter:

```
jreinstrumenter.sh -l
```

Regardless of JRE version, to enable all Diagnostics capabilities, to maximize the instrumentation coverage and to optimize monitored application performance, you should run the JRE Instrumenter. After you instrument the JRE version used by your application, copy the command line options presented by the JRE Instrumenter to the Java command in your application startup script. For information about running the instrumenter, see Chapter 6, “Running the JRE Instrumenter.”

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. For more information, see “Running the JRE Instrumenter for WebSphere IDE” on page 192.

4 Enter 0 (zero) to complete the configuration.

Once you install the agent, configure it as a Java Agent, and instrument the JRE, 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 Servers for the Java Agent.”

- Perform additional probe configurations.

Determine which advanced configurations of the probe apply to your environment. See Chapter 12, “Advanced Java Agent and Application Server Configuration.”

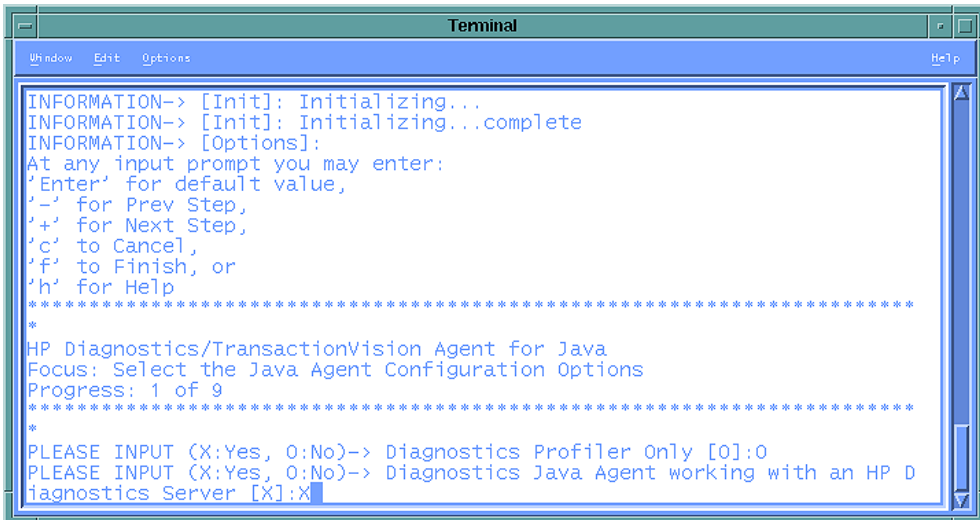
Configuring the Java Agent to Work with a Diagnostics Server

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 with a Diagnostics Server, use the Java Agent Setup Module. 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.**

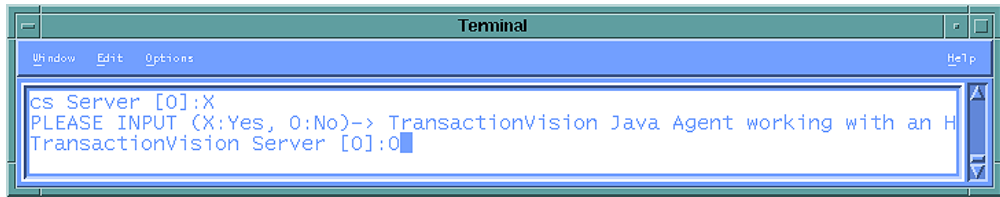


```

Terminal
-----
INFORMATION-> [Init]: Initializing...
INFORMATION-> [Init]: Initializing...complete
INFORMATION-> [Options]:
At any input prompt you may enter:
'Enter' for default value,
'-' for Prev Step,
'+' for Next Step,
'c' to Cancel,
'f' to Finish, or
'h' for Help
*****
*
HP Diagnostics/TransactionVision Agent for Java
Focus: Select the Java Agent Configuration Options
Progress: 1 of 9
*****
*
PLEASE INPUT (X:Yes, 0:No)-> Diagnostics Profiler Only [0]:0
PLEASE INPUT (X:Yes, 0:No)-> Diagnostics Java Agent working with an HP D
iagnostics Server [X]:X

```

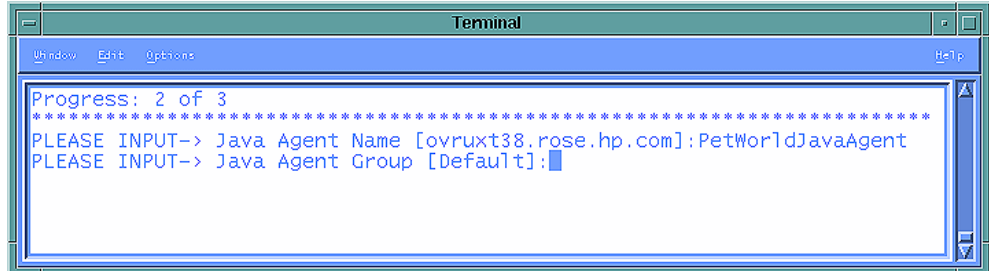
2 Specify whether the agent should also be configured to run as a TransactionVision sensor.



- ▶ Choose O to specify that the Agent should not be configured as a TransactionVision sensor.
- ▶ Choose X to specify that the Agent should be configured as a TransactionVision sensor.

You can configure the Java Agent as a TransactionVision Sensor or a Diagnostics Java Agent. If you configure the Java Agent to be a TransactionVision Sensor, the Java Agent Setup Module will show prompts that are not described in this manual. For more information, 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 identifies the agent within HP Diagnostics. You can use -, _ and all alphanumeric characters for the name. The agent name is assigned to be the probe name.

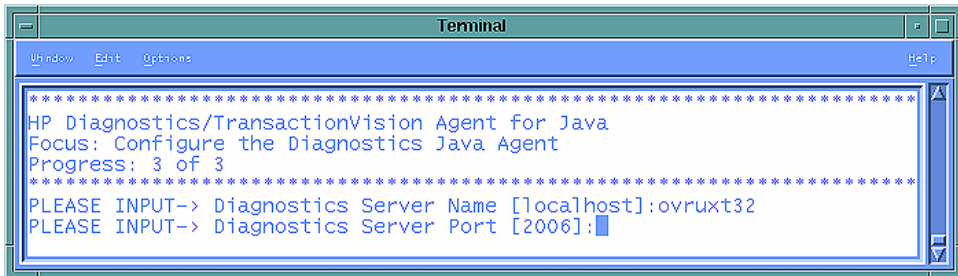
When assigning a name to an agent, choose a name that will help you recognize the application that the agent is monitoring, and the type of probe—in this case, a PetWorld application agent.

- ▶ For the Java Agent group name, enter a name for an existing group or a new group to be created. The agent group name is case-sensitive. The agent group name is assigned to be 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 can assign all of the probes for a particular enterprise application to a probe group so that you could monitor both the performance of the group and the performance of the individual probes.

4 Enter the configuration information for the Diagnostics Server.



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

Commanding Server. If there is only one Diagnostics Server in the Diagnostics deployment where the agent will run, enter the Diagnostics Server (in Commander Mode) host name and event port information here.

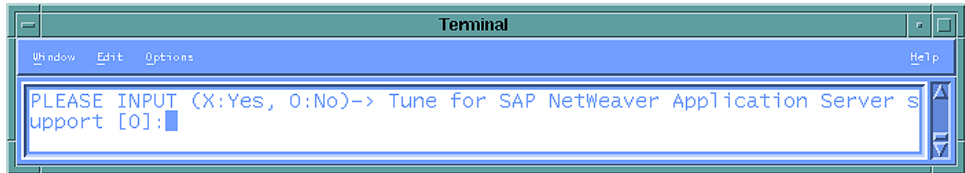
Mediating Server. In a distributed environment with a commanding server and mediating servers, enter the information for the Diagnostics Server (in Mediator Mode) that is to receive the events from the agent.

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 changed the port since the Diagnostics Server was installed, specify that port number here instead of the default.

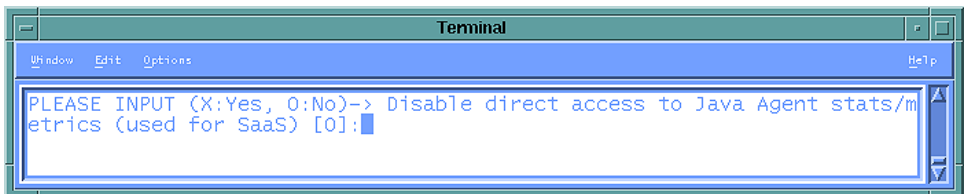
5 Specify whether to tune the agent for running in a SAP NetWeaver Application Server.



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

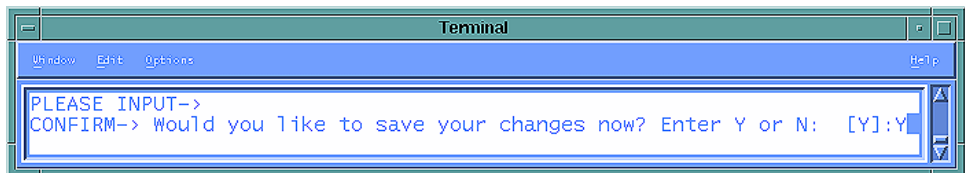
6 Specify whether the agent will be running in an HP SaaS environment.

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 agent.



- 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
-----
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/beat10/jdk150_04/jre)
PLEASE INPUT-> Option? ('Command', H: Help, or 0: Exit) [0]:

```

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. See “Running the JRE Instrumenter for WebSphere IDE” on page 192.

Regardless of JRE version, to enable all Diagnostics capabilities, to maximize the instrumentation coverage and to optimize monitored application performance, you should run the JRE Instrumenter. After you instrument the JRE version used by your application, copy the command line options presented by the JRE Instrumenter to the Java command in your application startup script. For information about running the instrumenter, see Chapter 6, “Running the JRE Instrumenter.”

Enter an instrumentation command as follows:

- To instrument the JVM in a specific directory and copy the classloader information to `<probe_install_dir>/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 installed WebLogic 8.1 (using Sun JDK) in `/opt/bea`, enter:

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

- To instrument the JVM in a specific directory (most users will never need to use this option), enter:

```
jreinstrumenter.sh -i <jvm_directory>
```

- To list JVMs below a specific directory, enter:

```
jreinstrumenter.sh -a <directory>
```

- To list all known JVMs, enter:

```
jreinstrumenter.sh -l
```

9 Enter 0 (zero) to complete the setup.

Once you install the agent, configure it as a Java Agent, and instrument the JRE, you must 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 Servers for the Java Agent.”

11 Verify the agent installation.

The agent installation is verified by using the System Health Monitor as described in “Verifying the Java Agent Installation” on page 161.

12 Perform additional agent configurations.

Determine which advanced configurations of the agent apply to your environment. See Chapter 12, “Advanced Java Agent 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 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 603.

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

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

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

- 1 Upload **HPDiagTVJavaAgt_<release number>_zos.tgz** 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 **HPDiagTVJavaAgt_<release number>_zos.tgz** using gzip as shown in the following example:

```
gzip -d HPDiagTVJavaAgt_9.00_zos.tgz
```

This command creates the unzipped file, **HPDiagTVJavaAgt_<release number>_zos.tar**.

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

```
tar -xvf HPDiagTVJavaAgt_9.00_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> -i <jvm_directory>
```

In this instance:

- ▶ <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 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 163.
 - 7 After you install the agent and instrument the JRE, you must manually modify the startup script for the application server so that a probe is started together with the monitored application. For detailed instructions, see Chapter 7, “Configuring Application Servers for the Java Agent.”
 - 8 Verify the agent installation as described in “Verifying the Java Agent Installation” on page 161.
 - 9 To collect z/OS system metrics, see “Enabling z/OS System Metrics Capture” on page 603.

Installing Java Agents on Multiple z/OS Machines

If you plan to install Java Agents on more than one z/OS machine, you might 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

The installers for the Java Agent were built to support installing the agent on all of the platforms for which the component was certified. However, the agent might work on other platforms that are not yet certified. A generic installer is 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, run the generic installer and manually configure the agent as a Java Probe so that it can communicate with the other Diagnostics components and monitor the processing of your application.

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

- 1** Locate **HPDiagTVJavaAgt_<release number>_unix.tgz** from the **Diagnostics_Installers** folder on the HP Diagnostics installation disk.
- 2** Unzip **HPDiagTVJavaAgt_<release number>_unix.tgz** using gzip as shown in the following example:

```
gzip -d HPDiagTVJavaAgt_9.00_unix.tgz
```

When this command completes, the unzipped file is called **HPDiagTVJavaAgt_<release number>_unix.tar**.

- 3** To unpack the tar file, run the following tar command:

```
tar -xf HPDiagTVJavaAgt_9.00_unix.tar
```

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

- 4** Configure the agent to enable the agent to work the other Diagnostics components as described in “About Advanced Java Agent Configuration” on page 163.
- 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 165.
 - b** After you install the agent and instrument the JRE, you must 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 Servers for the Java Agent.”
- 6** Verify the agent installation as described in “Verifying the Java Agent Installation” on page 161.

Silent Installation of the Java Agent

A *silent installation* 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 install step.

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 must 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.

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  
<JASMRspFileName>
```

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 Java agent install response file.

First set an environment variable and then run the installer with the following **-silent** command-line option:

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

On UNIX systems, 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 Java agent setup module response file.

Unset the environment variable and then run the Java Agent Setup Module with the following **-silent** command-line option:

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

On UNIX systems, use quotes when specifying the environment variable.

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

To specify two additional options after the response file name when performing a silent installation:

- ▶ You can create a log file by specifying the **is:log <logfilepath>** option.
- ▶ You can change the temp directory to a user-specified directory by specifying the **is:tempdir <tempDirPath>** option.

About Configuring the Application Server

To allow the agent to monitor an application, you need to instrument the JRE, manually modify the application server startup script, and optionally configure SOAP message handlers for some application servers.

Instrumenting the JRE

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

If the JRE is 1.5 or later, you can skip instrumenting the JRE although this is not recommended.

Regardless of JRE version, to enable all Diagnostics capabilities, to maximize the instrumentation coverage and to optimize monitored application performance, you should run the JRE Instrumenter. After you instrument the JRE version used by your application, copy the command line options presented by the JRE Instrumenter to the Java command in your application startup script. For information about running the instrumenter, see Chapter 6, “Running the JRE Instrumenter.”

The JRE Instrumenter runs the instrumentation. The JRE Instrumenter can be run at the end of the agent installation to prepare the application for instrumentation.

If you skip running the JRE Instrumenter during the agent installation, you can run it manually before you can monitor the application. For more information, see Chapter 6, “Running the JRE Instrumenter.”

Modifying the Application Startup Script

After you install the agent and instrument the JRE, you must manually modify the startup script for the application server so that a 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 Servers for the Java Agent.”

Configuring SOAP Message Handlers

The Diagnostics SOAP message handler is required 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 were 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 “Loading the Diagnostics SOAP Message Handler” on page 221.

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 Agent for Multiple Application Server JVM Instances” on page 406.

Verifying the Java Agent Installation

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

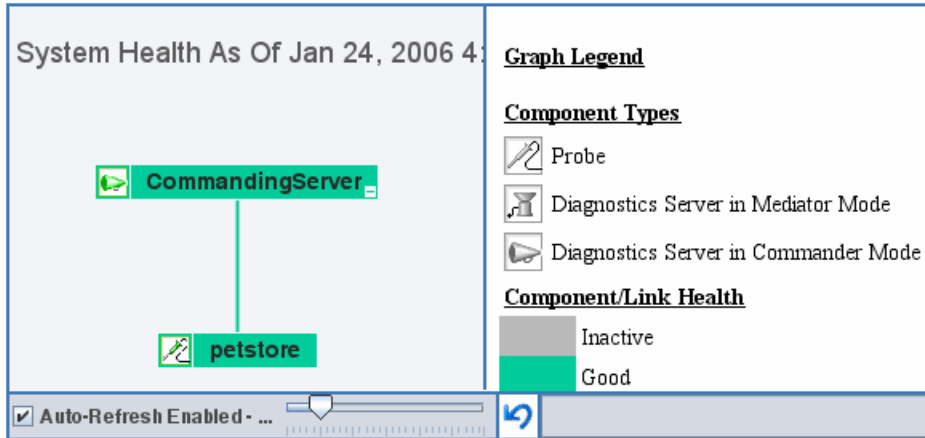
The agent does not register with the Diagnostics Server until a probe is started. The probe is started when the instrumented application server is started. Therefore, you cannot verify the installation of the agent using the System Health Monitor until you configure the Application Server startup scripts to start a probe instance.

If you followed the recommended installation sequence, after installing the Java Agent you can verify the following:

- The Diagnostics command server is successfully installed.
- Where relevant, additional Diagnostics Servers are successfully installed and communicating with the Diagnostics Server (CommandingServer).

- ▶ The Java Agent is successfully installed and probe instances are communicating with the Diagnostics Server.

Depending on your deployment, the new probe instance is shown on the System Health Monitor as a child of either the Diagnostics command server or the Diagnostics mediator server.



The probe instance is colored gray in the System Health Monitor when the monitored application is closed. When the application starts, the probe is colored green.

For detailed instructions on using 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 you have a question about.

You can find the version of the Java Agent in one of the following ways:

- ▶ Locate the version file `<Probe_install_dir>\version.txt`. The file contains the four-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 instance on System Health Monitor.

Uninstalling the Java Agent

To uninstall the Java Agent:

- ▶ On a Windows machine, choose Start > All Programs > HP Java Agent > Uninstaller.
Or run **uninstaller.exe**, which is located in the `<probe_install_dir>_uninst` directory.
- ▶ On a Linux or Solaris 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 Agent.

About Advanced Java Agent Configuration

You should review the topics in Chapter 12, “Advanced Java Agent 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

You can create custom instrumentation points to handle unique situations in your application environment. For general information on custom instrumentation, review the topics in Chapter 9, “Custom Instrumentation for Java Applications.”

6

Running the JRE Instrumenter

This chapter explains how to run the JRE Instrumenter to allow the Java Agent to monitor an application.

This chapter includes:

- ▶ About the JRE Instrumenter on page 166
- ▶ Running the JRE Instrumenter on page 169

About the JRE Instrumenter

Diagnostics' JRE Instrumenter is used to prepare (instrument) the JRE used by your application to make monitoring by Diagnostics possible. JRE Instrumenter does not modify the installed JRE in any way, but rather places copies of instrumented classes somewhere under the `<probe_install_dir>/classes` directory. You can use the JRE Instrumenter to instrument multiple JREs if they are installed on your system.

The JRE Instrumenter can be run at the end of the Java Agent installation. Or if you skip running the JRE Instrumenter during the agent installation, you can run it manually before you can monitor the application.

Regardless of JRE version, to enable all Diagnostics capabilities, to maximize the instrumentation coverage and to optimize monitored application performance, you should run the JRE Instrumenter. After you instrument the JRE version used by your application, copy the command line options presented by the JRE Instrumenter to the Java command in your application startup script.

The JRE Instrumenter instruments some standard Java classes for the JVM the application is using. It also provides you with the JVM parameters that must be used when the application server is started so that the application server uses the instrumented classes.

You must run the JRE Instrumenter again if the JDK (java.exe executable) used by the application server changes so that the Java Agent can continue to monitor the processing.

Notes:

- ▶ To instrument IBM's 1.4.2 J9 JRE, you must instrument the correct classes and add the -Xj9 option on the application's command line. The correct classes are located in the <java dir>\jre\lib\jclSC14 directory; for example:

```
jreinstrumenter.sh -i \usr\java14_64\jre\lib\jclSC14
```

- ▶ The IBM 1.5.0 JVM supports the -javaagent option. However, using the option may sometimes cause increased CPU utilization by the JVM, which can mislead you to see it as Java agent overhead. Moreover, this option makes the JVM ignore the -agentpath option which is used by Diagnostics agents to load native libraries needed for the Diagnostics Java Profiler Heap Walker functionality. To avoid performance issues or to use Heap Walker, you need to use the JRE instrumenter and pre-instrument this JVM as documented in this section.
-

Note: If you've configured the Java Agent to use one probe 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. See "Configuring the Agent for Multiple Application Server JVM Instances" on page 406.

JRE Instrumenter Processing

The JRE Instrumenter performs the following functions:

- ▶ Identifies JVMs that are available to be instrumented.
- ▶ Searches for additional JVMs in directories you specify.
- ▶ Instruments the JVMs you specify and provides the parameter you must add to the startup script for the JVM to point to the location of the instrumented classes.

The Instrumenter puts the instrumented classes in different places according to how it is executed.

- ▶ When the Instrumenter is run from the Java Agent installer, the Instrumenter places the instrumented classes 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 classes 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 classes 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 177.

Running the JRE Instrumenter

The following explains how to run the JRE Instrumenter in a Windows environment and in a UNIX environment in console mode.

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

This section includes:

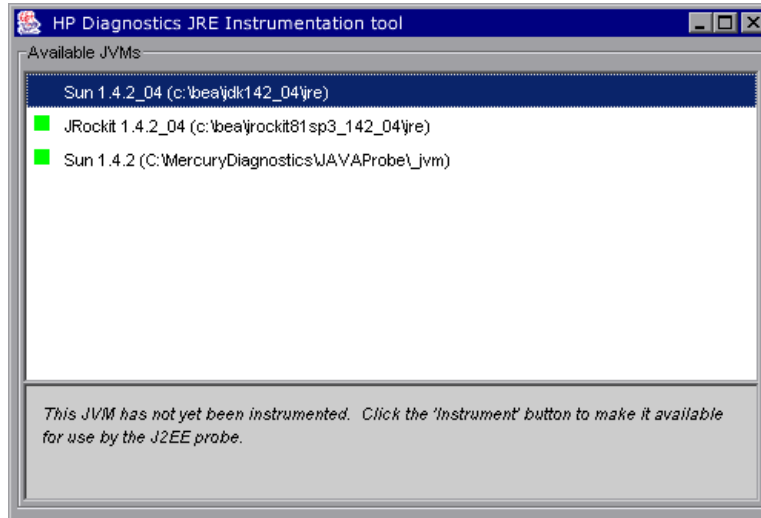
- “Running the JRE Instrumenter on a Windows Machine” on page 169
- “Running the JRE Instrumenter on a UNIX Machine” on page 175

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 were instrumented are listed with a green square preceding the name of the JVM.

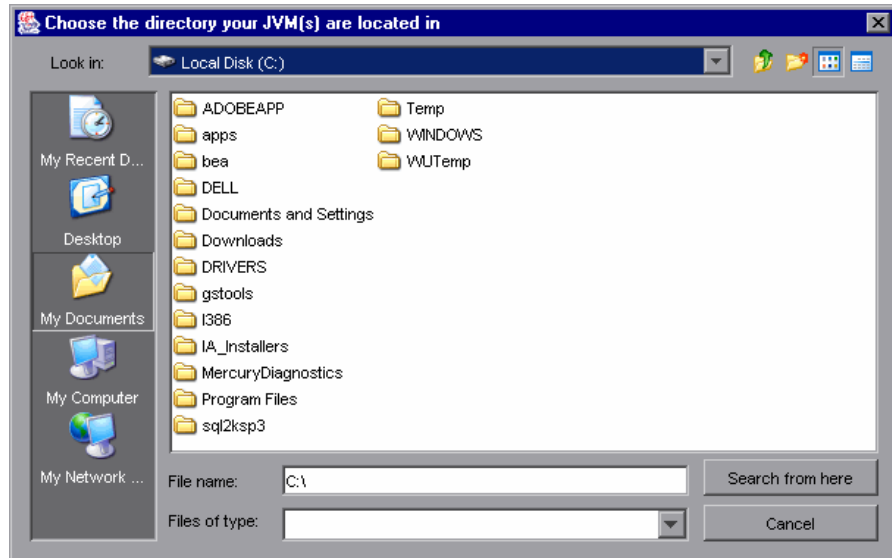
From the JRE Instrumentation Tool dialog you can do the following tasks:

- ▶ If the JVM you want to instrument is not listed in the Available JVMs list in the dialog, add JVMs to the list as described in “Adding JVMs to the Available JVMs List” on page 171.
- ▶ If the JVM you want to instrument is listed but is not yet instrumented, instrument the JVM as described in “Instrumenting a Selected JVM” on page 174.
- ▶ If the JVM you want to instrument is listed and instrumented, 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 174.

- When you finish using the JRE Instrumenter, click **Exit** to close the JRE Instrumentation Tool.

Adding JVMs to the Available JVMs List

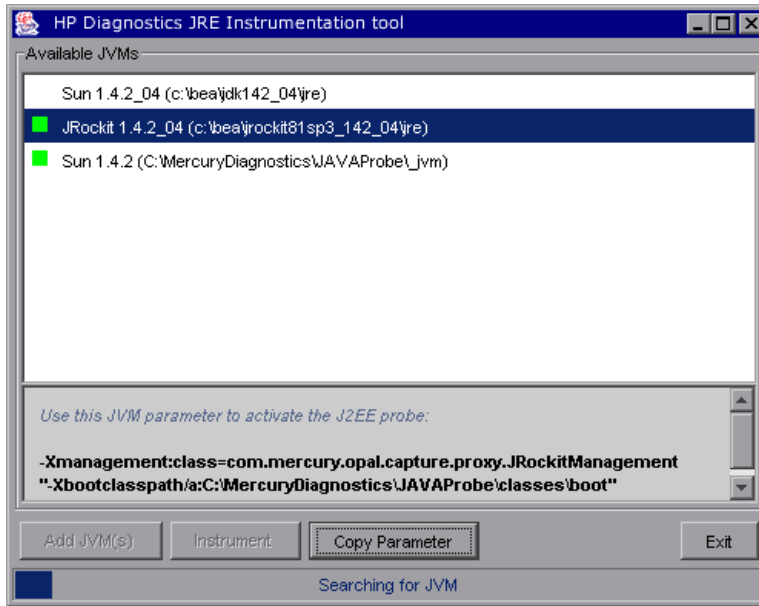
- 1 In the JRE Instrumentation Tool dialog, click **Add JVM(s)** to search for other JVMs and add them to the Available JVMs list. The Instrumenter displays the Choose the Directory dialog box.



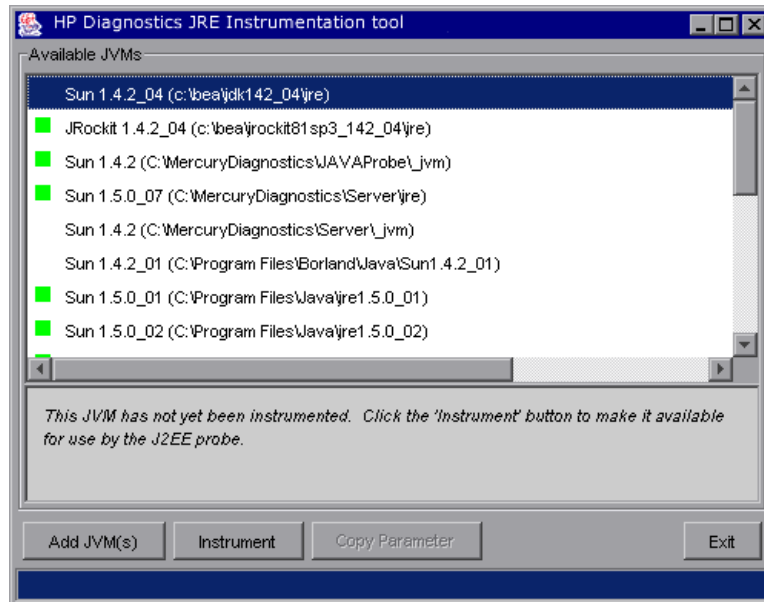
- 2 Navigate to the directory location where you would like the Instrumenter to begin searching for JVMs using the standard Windows Explorer type navigation controls.
- 3 Select the file where you would like to begin the search so that its name appears in the **File name** box.
- 4 Click **Search from 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.

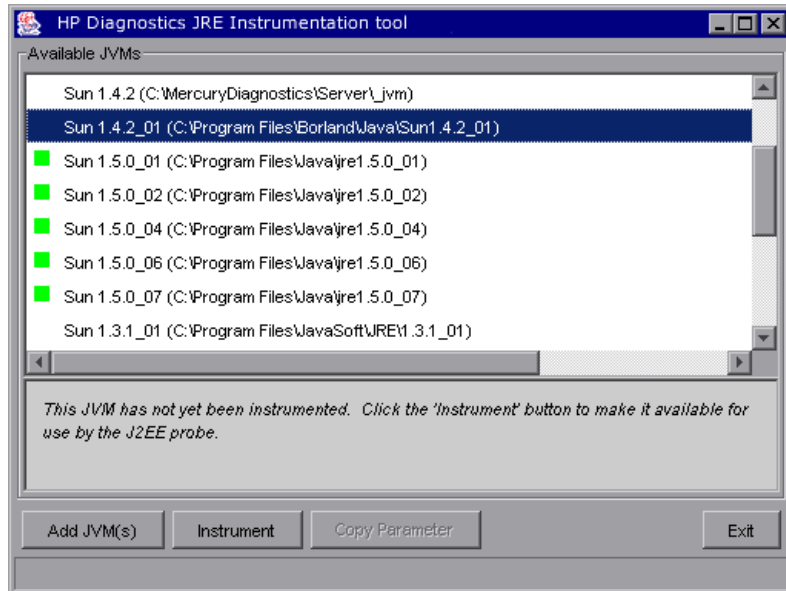


When the Instrumenter completes the search, it enables the command buttons on the dialog as shown below. If the selected row is a JVM that was instrumented, the Instrument button is disabled.



Instrumenting a Selected JVM

Select a JVM that is not instrumented from the **Available JVMs** list and click **Instrument**.



The JRE Instrumenter instruments some of the classes for the selected JVM and places the instrumented classes 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 you must include in the startup script for the application server to cause your application to use the instrumented classes. 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 the steps necessary to run the JRE Instrumenter using either a graphical mode installation or a console mode installation.

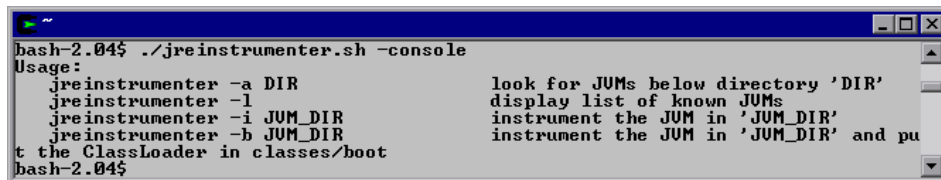
The JRE Instrumenter screens that are displayed in a graphical mode are the same as those documented for the Windows installer in “Running the JRE Instrumenter on a Windows Machine” on page 169.

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:



```
bash-2.04$ ./jreinstrumenter.sh -console
Usage:
  jreinstrumenter -a DIR          look for JUMs below directory 'DIR'
  jreinstrumenter -l             display list of known JUMs
  jreinstrumenter -i JUM_DIR     instrument the JUM in 'JUM_DIR'
  jreinstrumenter -b JUM_DIR     instrument the JUM in 'JUM_DIR' and pu
t the ClassLoader in classes/boot
bash-2.04$
```

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

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

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

Instrumenter Function	Documentation Section
<code>jreinstrumenter -l</code>	“Displaying the List of Instrumented JVMs” on page 176
<code>jreinstrumenter -a DIR</code>	“Adding JVMs to the Available JVMs List” on page 176
<code>jreinstrumenter -i JVM_DIR</code> <code>jreinstrumenter -b JVM_DIR</code>	“Instrumenting a Listed JVM” on page 177

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 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 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 completes 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 JRE

```
./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 classes for the selected JVM and places the instrumented classes 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 JRE

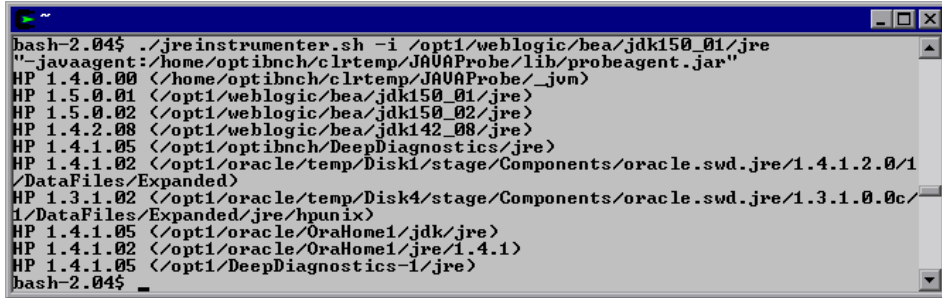
```
./jreinstrumenter.sh -b JVM_DIR
```

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 classes for the selected JVM and places the instrumented classes in a folder under the `<probe_install_dir> /classes/boot` directory.

Use this command only if you are monitoring a single JVM with the Probe and you do not want to use the more explicit path generated with the `-i` command option.

When the Instrumenter finishes 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:



```

bash-2.04$ ./jre instrumenter.sh -i /opt1/weblogic/boa/jdk150_01/jre
"-javaagent:/home/optibnch/clrtemp/JAUAProbe/lib/probeagent.jar"
HP 1.4.0.00 </home/optibnch/clrtemp/JAUAProbe/_jvm>
HP 1.5.0.01 </opt1/weblogic/boa/jdk150_01/jre>
HP 1.5.0.02 </opt1/weblogic/boa/jdk150_02/jre>
HP 1.4.2.08 </opt1/weblogic/boa/jdk142_08/jre>
HP 1.4.1.05 </opt1/optibnch/DeepDiagnostics/jre>
HP 1.4.1.02 </opt1/oracle/temp/Disk1/stage/Components/oracle.swd.jre/1.4.1.2.0/1
/DataFiles/Expanded>
HP 1.3.1.02 </opt1/oracle/temp/Disk4/stage/Components/oracle.swd.jre/1.3.1.0.0c/
1/DataFiles/Expanded/jre/hpunix>
HP 1.4.1.05 </opt1/oracle/OraHome1/jdk/jre>
HP 1.4.1.02 </opt1/oracle/OraHome1/jre/1.4.1>
HP 1.4.1.05 </opt1/DeepDiagnostics-1/jre>
bash-2.04$

```

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

When the JRE Instrumenter instruments a JVM, it also creates the JVM parameter you must include in the startup script for the application server in order to cause your application to use the instrumented classes. When the Instrumenter finishes 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 Servers for the Java Agent

This section describes how to configure the application servers to allow the Java Agent to monitor the application.

This chapter includes:

- ▶ About Configuring the Application Server Startup Scripts on page 180
- ▶ Configuring WebSphere Application Servers on page 181
- ▶ Configuring WebLogic Application Servers on page 195
- ▶ Configuring Oracle Application Servers on page 201
- ▶ Configuring the JBoss Application Server on page 210
- ▶ Configuring Tomcat 5.x/6.x When Running as a Windows Service on page 213
- ▶ Configuring the SAP NetWeaver Application Server on page 214
- ▶ Configuring a Generic Application Server on page 216
- ▶ Adjusting the Heap Size for the Java Agent in the Application Startup Script on page 218
- ▶ Configuring TIBCO ActiveMatrix/Business Works on page 219
- ▶ Configuring the SOAP Message Handler on page 219
- ▶ Configuring the Discovery of a New J2EE Server for CI Population on page 223

About Configuring the Application Server Startup Scripts

After you run the JRE Instrumenter for the Java Agent, you must modify the startup script for the application you are going to monitor. You manually update the startup script so that the probe that is to monitor the application starts when the application starts.

The following sections provide instructions for updating different types of application server startup scripts 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, see 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. The generic procedure described in “Configuring a Generic Application Server” on page 216 provide the information 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 following sections, follow the procedure in “Configuring a Generic Application Server” on page 216.

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 Agent for Multiple Application Server JVM Instances” on page 406.

Notes:

- `<probe_install_dir>` indicates the directory where the Java Agent was installed.
 - When modifying the `-Xbootclasspath` parameter, 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 enables you to add classpath elements, define runtime variables (`-D` variables), and configure the bootclasspath for WebSphere. Use the Administrative Console to add the `Xbootclasspath` property and any additional arguments that are needed for the JVM command line.

Important: If Diagnostics is not able to identify your application server as a WebSphere server, enable PMI and add the Jar files to the `server.policy` file. See “Configuring WebSphere for JMX Metric Collection” on page 192.

The appearance of the Console can differ for different versions of WebSphere. Changes are implemented differently in each version of WebSphere. As a result, the following examples might not correspond exactly to your WebSphere version but do provide the information needed 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.

This section includes examples for the following:

- “WebSphere 5.x and 6.0” on page 182
- “WebSphere 6.1/7.0” on page 190
- “Running the JRE Instrumenter for WebSphere IDE” on page 192
- “Configuring WebSphere for JMX Metric Collection” on page 192

WebSphere 5.x and 6.0

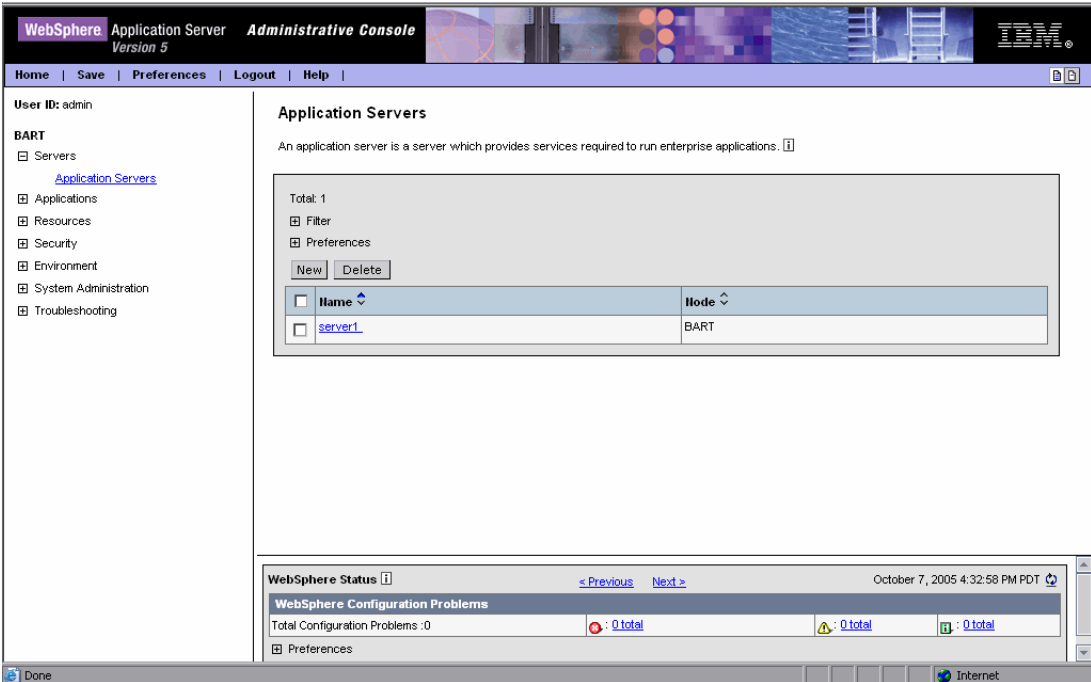
To configure a WebSphere 5.x and 6.0 application server:

- 1** Use your Web browser to access the WebSphere Application Server Administrative Console for the application server instance to be monitored by the probe:

```
http://<App_Server_Host>:9090/admin
```

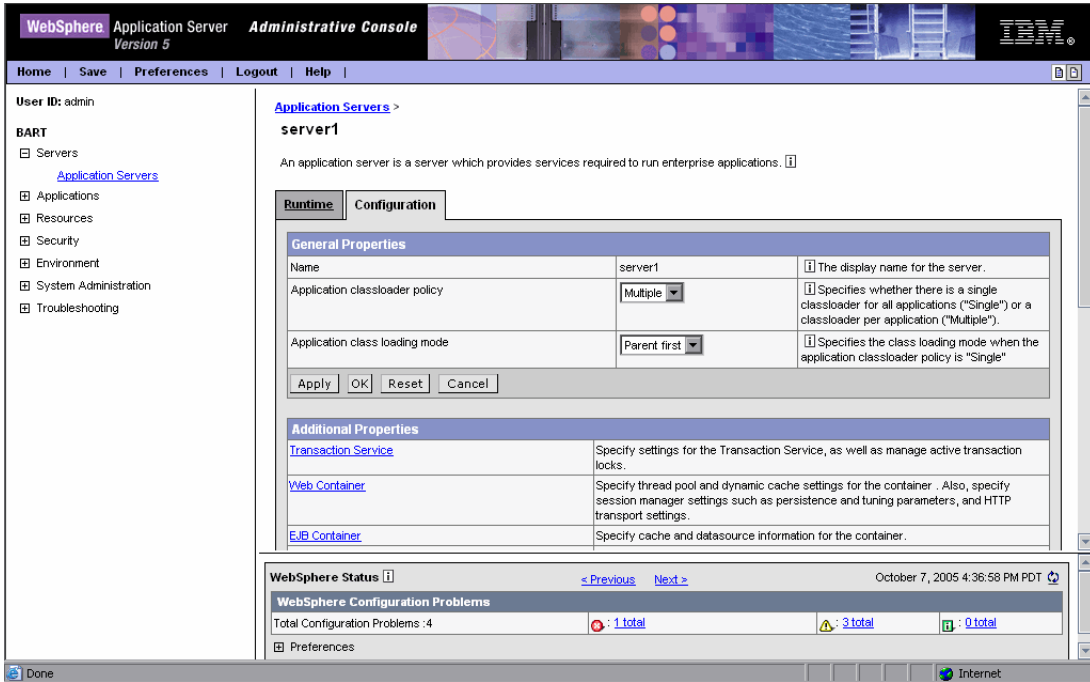
Replace **<App_Server_Host>** with the machine name for the application server host.

The Websphere Application Server Administrative Console opens.

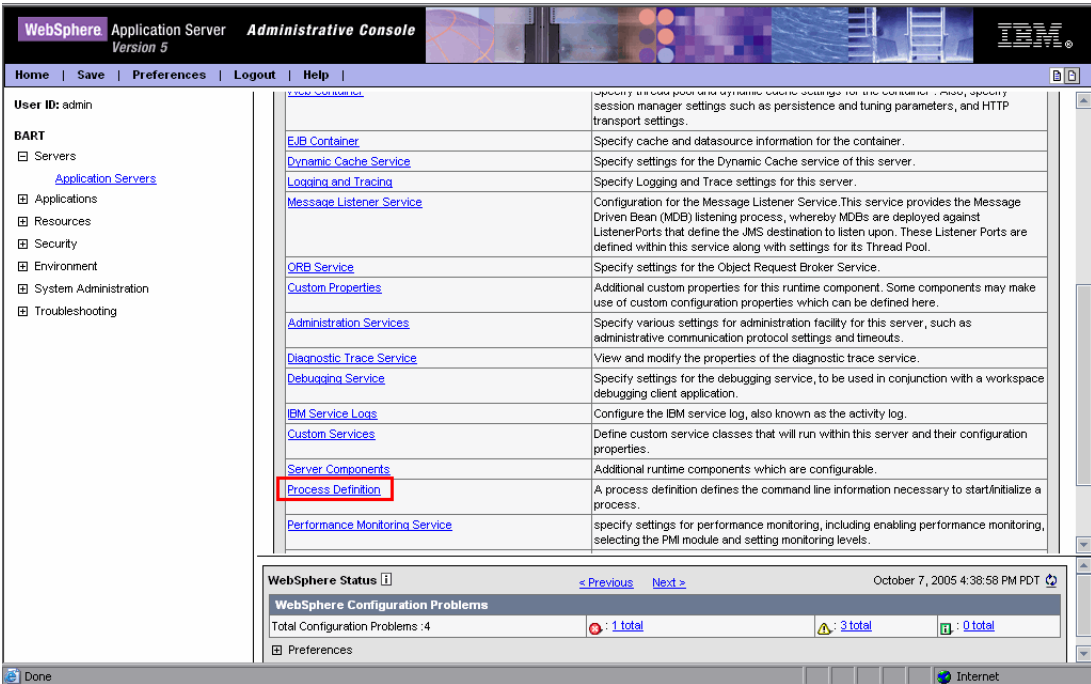


- 2 In the left panel, select **Servers > Application Servers**.
- 3 From the list of application servers in the right panel, select the name of the server that you want to configure so that it will be monitored by the probe.

The Configuration tab for the selected application server is displayed.

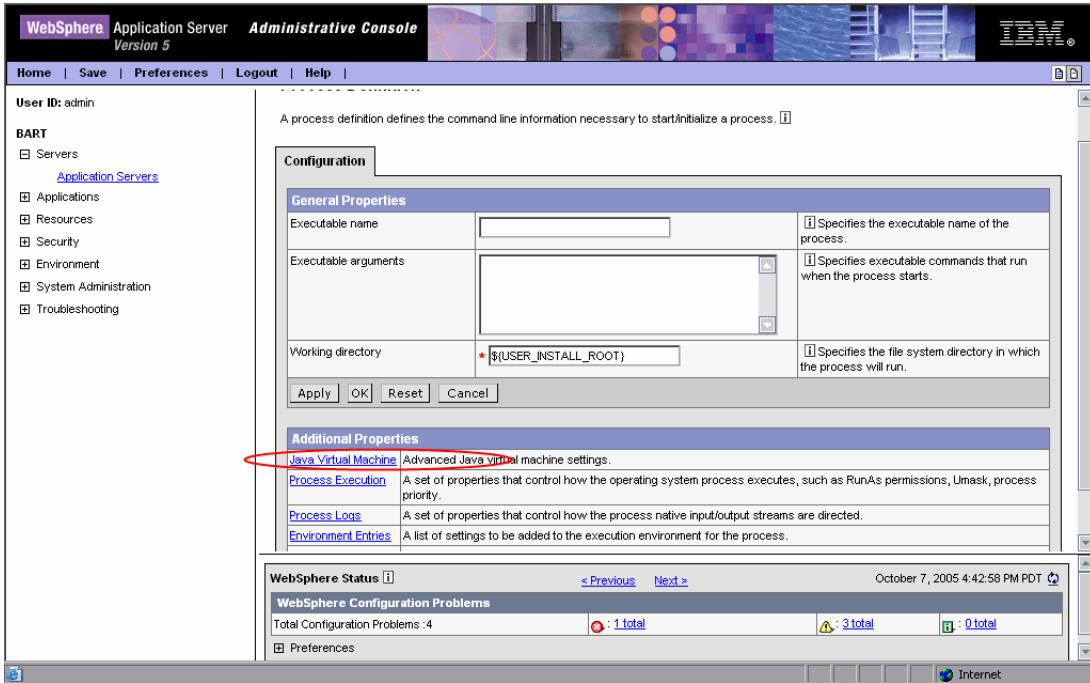


4 Scroll down in the Configuration tab and, in the **General Properties** column, look for the **Process Definition** property.



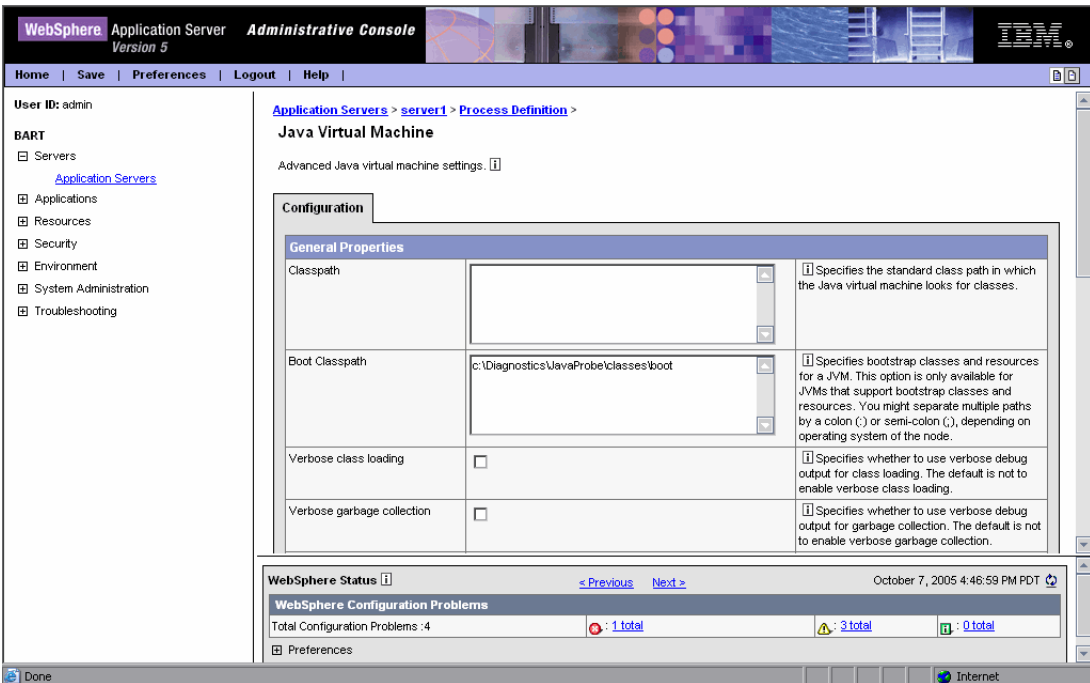
5 Click **Process Definition**.

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.

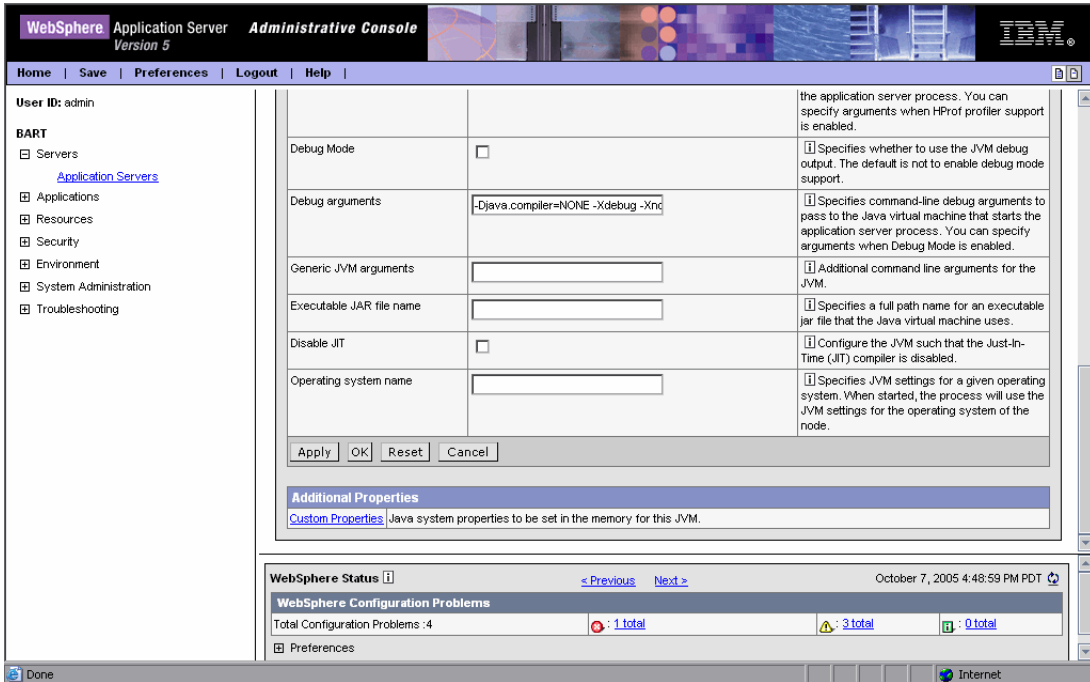


9 In the **Boot Classpath** box, type the path to the boot directory for the Java Agent as follows:

```
<probe_install_dir>\classes\IBM\1.4.2_06;<probe_install_dir>\classes\boot
```

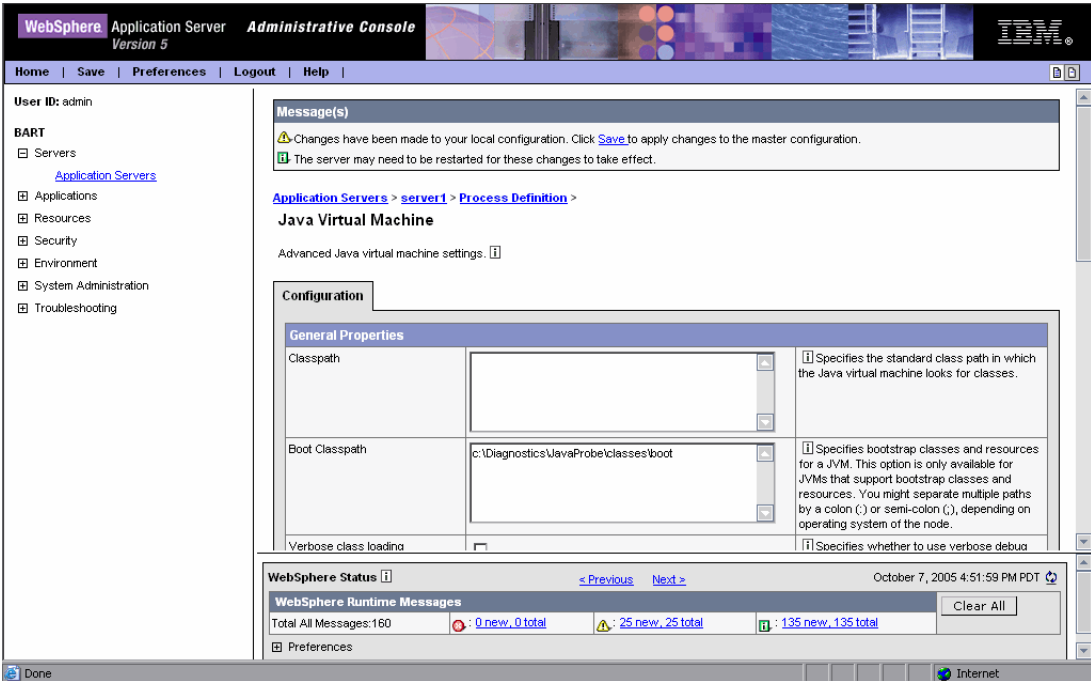
In this instance, **<probe_install_dir>** is the path to the location where the Java Agent was installed.

- 10 Scroll to the bottom of the Configuration tab until the command buttons are visible.



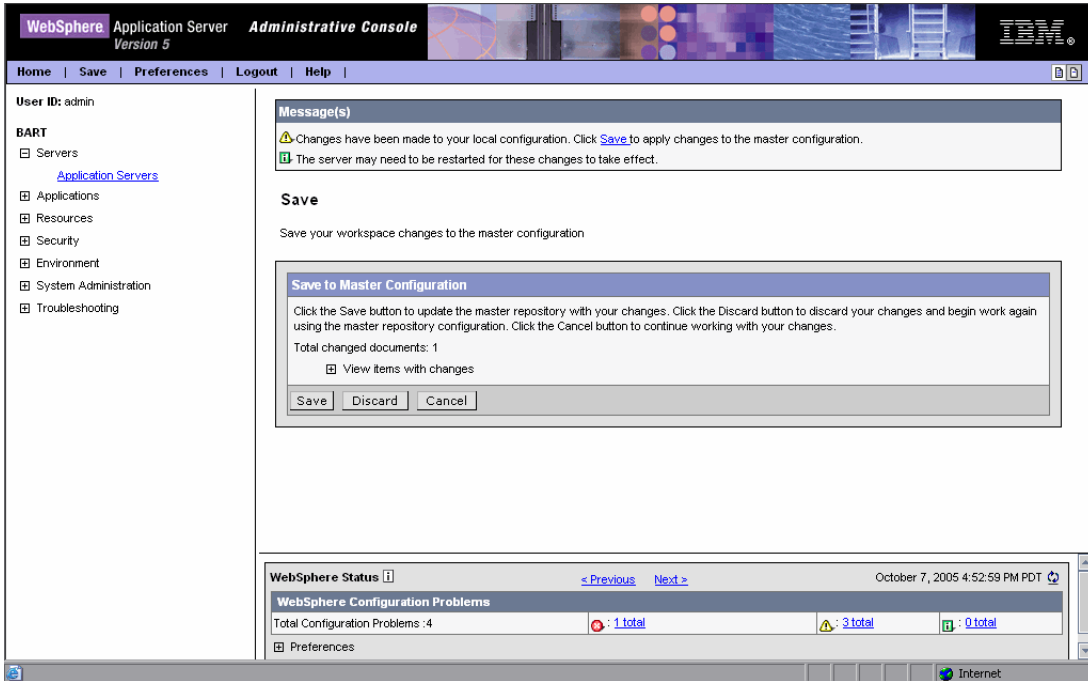
Click **Apply**.

11 A message confirms that your changes were 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, 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 165.

WebSphere 6.1/7.0

To configure a WebSphere 6.1/7.0 application server:

- 1** Open the Websphere Application Server Administrative Console.
- 2** Navigate to the Java Virtual Machine page; for example, on 6.1, navigate to: **Application servers > Application server instance name (server1) > Process Definition > Java Virtual Machine page**

For example, on 7.0, navigate to:

Server > Server Types > WebSphere Application Servers > Application server instance name

Select Process Definition under Java and Process Management > Java Virtual Machine.

- 3** For WebSphere 7.0, in the Classpath box, add:

```
<websphere7.0_install_dir>/AppServer/lib/j2ee.jar
```

- 4** On the Java Virtual Machine page, in the **Generic JVM Arguments** box, enter the JVM parameter (an example is shown below):

```
-Xbootclasspath/p:<probe_install_dir>/classes/IBM/1.5.0/  
instr.jre:<probe_install_dir>/classes/boot  
-Xshareclasses:none
```

In this example, **<probe_install_dir>** is the path to the location where the Java Agent was installed.

- 5** Apply and save your changes.
- 6** Restart the WebSphere application server. You do not need to restart 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, the Java Agent was not started correctly.

Running the JRE Instrumenter for WebSphere IDE

If you are using WebSphere IDE, you must run the JRE Instrumenter manually to make sure the correct Java executable for the WSAD IDE was instrumented.

The WSAD IDE has 10 different java.exe executables to choose from. You must instrument the one that is used to run the IDE.

To instrument the correct java.exe:

- 1 Determine the version of WebSphere you are using.
- 2 Determine the location of the appropriate java.exe. See the following table.
- 3 Run the JRE Instrumenter as described in “Running the JRE Instrumenter” on page 165.

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

Configuring WebSphere for JMX Metric Collection

You might need to configure the Performance Monitoring Service (PMI) on the WebSphere server to start receiving JMX metrics.

Important: If Diagnostics is not able to identify your application server as a WebSphere server, you must enable PMI and add the Jar files to the server.policy file.

This section includes examples for configuring WebSphere 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 console displays a table of the application servers.
- 3 Click the name of the application server 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 enable JMX collection:

```
grant codeBase "file:/<probe_install_dir>/lib/probe-jmx.jar"  
{ permission java.security.AllPermission; }
```
- 8 Restart the application server.

To configure WebSphere 6.x/7.0 server for JMX metrics collection:

- 1 Open the WebSphere Administrative Console.
- 2 In the Console navigation tree, select **Servers > Application Servers**. The console displays a table of the application servers.
- 3 Click the name of the application server 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 or <WebSphere 7.0 Installation Directory>/AppServer/profiles/<your_profile_name>/properties/server.policy) and add the following security permissions to enable 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 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 you plan to update before 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 the **Copy Parameter** button. This copies 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"
```

In this instance, `<probe_install_dir>` is the path to the directory where the Java Agent is installed.

- 3 Locate the startup script used to start WebLogic for your domain. This file is typically located in a path similar to the following 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 would look like the following:

```
D:\bea\weblogic81\samples\domains\medrec\startMedRecServer.cmd
```

- 4 Create a backup copy of the startup script before 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 following is an example of a WebLogic 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\weblogic81/lib/weblogic.policy" weblogic.Server
```

Note: The startup script examples are shown with line breaks. The actual scripts do not have line breaks. 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" -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, 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 165.

To configure a WebLogic 8.1 application server for the JRockit JVM:

- 1 Run the JRE Instrumenter and add the JRockit JVM that WebLogic is using.
- 2 Once the JVM is added, click on the **Copy Parameter** button. This copies the required parameters into the clipboard.
- 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 before making any changes to the script. You could 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
```

The following is an 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 follow 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. 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 required parameters:

```
"%JAVA_HOME%\bin\java" %JAVA_VM% %MEM_ARGS% %JAVA_OPTIONS%
-Xmanagement:class=com.mercury.opal.capture.proxy.JRocketManagement
-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, 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 165.

WebLogic 9.x and 10.0

To configure a WebLogic 9.0 and 10.0 application server:

- 1 Run the JRE Instrumenter and add the JVM that WebLogic 9.x or 10.0 is using.
- 2 Once the JVM is added, click the **Copy Parameter** button. This copies the JVM parameter into the clipboard; for example:

```
JAVA_OPTIONS="-javaagent:<probe_install_dir>\lib\probeagent.jar"
```

In this instance, **<probe_install_dir>** is the path to the directory where the Java Agent 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 the following:

```
D:\bea\wlserver_10.0\samples\domains\medrec\bin\startWebLogic.cmd
```

- 4 Create a backup copy of the startup script before 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.

Note: The startup script examples are shown with line breaks. The actual scripts do not have line breaks. 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 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, 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 165.

Configuring Oracle Application Servers

This section provides instructions for configuring the following Oracle application servers:

- “Configuring Oracle 9i” on page 202
- “Configuring Oracle 10.1.3” on page 205
- “Configuring Oracle 10.1.2” on page 206

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. Therefore, 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 before 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 before 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
```

In this instance, `<probe_install_dir>` is the path to the directory where the Java Agent was installed.

Note: When modifying the **-Xbootclasspath** parameter, use quotes if there are spaces in the path you specify.

The following image is an example of an Oracle 9iAS startup script before adding the **Xbootclasspath** parameter:

```

- <ias-instance xmlns="http://www.oracle.com/ias-instance">
- <notification-server>
  <port local="6100" remote="6200" request="6003" />
  <log-file path="/opt/oracle/ora9ias/opmn/logs/ons.log" level="3" />
</notification-server>
- <process-manager>
- <ohs gid="HTTP Server" maxRetry="3">
  <start-mode mode="ssl" />
</ohs>
- <oc4j instanceName="home" numProcs="1" maxRetry="3">
  <config-file path="/opt/oracle/ora9ias/j2ee/home/config/server.xml" />
  <oc4j-option value="-properties" />
  <port ajp="3000-3100" rmi="3101-3200" jms="3201-3300" />
  <environment>
    <prop name="LD_LIBRARY_PATH" value="/opt/oracle/ora9ias/lib" />
  </environment>
</oc4j>
- <oc4j instanceName="OC4J_Demos" gid="OC4J_Demos">
  <config-file path="/opt/oracle/ora9ias/j2ee/OC4J_Demos/config/server.xml" />
  <java-option value="-Xmx512M" />
  <oc4j-option value="-userthreads -properties" />
  <port ajp="3001-3100" rmi="3101-3200" jms="3201-3300" />
  <environment>
    <prop name="%LIB_PATH_ENV%" value="%LIB_PATH_VALUE%" />
  </environment>
</oc4j>
- <custom gid="dcm-daemon" numProcs="1" noGidWildcard="true">
  <start path="/opt/oracle/ora9ias/dcm/bin/dcmctl daemon -logdir /opt/oracle/ora9ias/dcm/logs/daemon_logs" />
  <stop path="/opt/oracle/ora9ias/dcm/bin/dcmctl shutdowndaemon" />
</custom>
  <log-file path="/opt/oracle/ora9ias/opmn/logs/lpm.log" level="3" />
</process-manager>
</ias-instance>

```

The following image is an example of an Oracle 9iAS startup script after adding the **Xbootclasspath** parameter:

```

- <ias-instance xmlns="http://www.oracle.com/ias-instance">
- <notification-server>
  <port local="6100" remote="6200" request="6003" />
  <log-file path="/opt/oracle/ora9ias/opmn/logs/ons.log" level="3" />
</notification-server>
- <process-manager>
  <ohs gid="HTTP Server" maxRetry="3">
    <start-mode mode="ssl" />
  </ohs>
- <oc4j instanceName="home" numProcs="1" maxRetry="3">
  <config-file path="/opt/oracle/ora9ias/j2ee/home/config/server.xml" />
  <java-option value="-Xmx512m -Xbootclasspath/p:C:\Program
Files\MercuryInteractive\common\JavaProbe\classes\boot" />
  <oc4j-option value="-properties" />
  <port ajp="3000-3100" mmi="3101-3200" jms="3201-3300" />
  <environment />
</oc4j>
- <oc4j instanceName="OC4J_Demos" gid="OC4J_Demos">
  <config-file path="/opt/oracle/ora9ias/j2ee/OC4J_Demos/config/server.xml" />
  <oc4j-option value="-userthreads -properties" />
  <port ajp="3001-3100" mmi="3101-3200" jms="3201-3300" />
- <environment>
  <prop name="%LIB_PATH_ENV%" value="%LIB_PATH_VALUE%" />
</environment>
</oc4j>
- <custom gid="dcm-daemon" numProcs="1" noGidWildcard="true">
  <start path="/opt/oracle/ora9ias/dcm/bin/dcmctl daemon -logdir /opt/oracle/ora9ias/dcm/logs/daemon_logs" />
  <stop path="/opt/oracle/ora9ias/dcm/bin/dcmctl shutdowndaemon" />
</custom>
  <log-file path="/opt/oracle/ora9ias/opmn/logs/ipm.log" level="3" />
</process-manager>
</ias-instance>

```

- 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, 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 165.

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 before making any changes.
- 3** Open the `opmn.xml` file to be edited using your editor.
- 4** Add the JRE Instrumenter provided parameter to the *java-option value*.
- 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, the Java Agent 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,

The screenshot displays the Oracle Enterprise Manager 10g Application Server Control Console. The page title is "Application Server: 102_w2k3.ros59631st.ovrtest.adapps.hp.com". The navigation menu includes "Home", "J2EE Applications", "Ports", "Infrastructure", and "Backup/Recovery". The page was refreshed on Aug 7, 2007 9:37:42 AM.

General section shows the server status as "Up". Other details include Host: ros59631st.ovrtest.adapps.hp.com, Version: 10.1.2.0.2, Installation Type: J2EE and Web Cache, Oracle Home: C:\OraHome_1. There are "Stop All" and "Restart All" buttons.

CPU Usage pie chart shows: Application Server (0%), Idle (99%), and Other (1%).

Memory Usage pie chart shows: Application Server (13% 262MB), Free (58% 1,181MB), and Other (29% 603MB).

System Components section includes buttons for "Start", "Stop", "Restart", and "Delete OC4J Instance". Below is a table of components:

Select	Name	Status	Start Time	CPU Usage (%)	Memory Usage (MB)
<input checked="" type="checkbox"/>	home	↑	Aug 2, 2007 10:41:38 AM	0.16	51.19
<input type="checkbox"/>	HTTP_Server	↑	Aug 2, 2007 8:07:55 AM	0.03	50.96
<input type="checkbox"/>	Management	↑	Aug 2, 2007 8:08:17 AM	0.00	159.96

A TIP note states: "This table contains only the enabled components of the application server. Only components that have the checkbox enabled can be started or stopped."

Related Links include "Process Management" and "All Metrics".

- 2 Double-click the **home** System Component.
- 3 On the **OC4J: home** page, select **Administration**.

4 On the Administration page, select **Server Properties**.

The screenshot shows the Oracle Enterprise Manager 10g Application Server Control interface. At the top, the page title is "ORACLE Enterprise Manager 10g Application Server Control". Below the title, there are navigation links for "Logs", "Topology", "Preferences", and "Help". The current page is "Administration", as indicated by the breadcrumb "Home Applications Administration" and the selected tab. The main content area is titled "OC4J: home" and includes a "Page Refreshed Aug 7, 2007 9:42:38 AM" notification. Under the "Instance Properties" section, the "Server Properties" link is highlighted with a red box. Other links in this section include "Website Properties", "JSP Container Properties", "Replication Properties", and "Advanced Properties". Below this is the "Application Defaults" section with links for "Data Sources", "Security", "JMS Providers", and "Global Web Module". The "Related Links" section contains a link for "ADF Business Components". On the right side, an "OC4J Inheritance" tooltip provides information about the hierarchical parent-child relationship of OC4J applications. At the bottom of the page, there are copyright notices and additional navigation links for "Logs", "Topology", "Preferences", and "Help".

- In the Server Properties window, under **Command Line Options**, add the **Xbootclasspath** property to the **Java Options** box.

ORACLE Enterprise Manager 10g
Application Server Control

Application Server 102_w2k3_ros596311st_ovrtest_adapps_hp.com > OC4J_home >

Server Properties Page Refreshed Aug 7, 2007 8:22:45 AM

General

Name: **home**
 Server Root: C:\OraHome_1\j2ee\home\config
 Configuration File: C:\OraHome_1\j2ee\home\config\server.xml
 Default Application Name: **default**
 Default Application Path: **application.xml**

Default Web Module Properties: global-web-application.xml
 Application Directory: ./applications
 Deployment Directory: ./application-deployments

Multiple VM Configuration

TIP If OC4J is running, newly added OC4J Clusters and associated processes will be automatically started.

Clusters(OC4J)

Cluster(OC4J) Name	Number of Processes	Related Links
default_island	1	Virtual Machine Metrics

[Add Another Row](#)

Ports

TIP Be sure that the port ranges specified below are large enough to accommodate the total number of processes in the Clusters (OC4J) table.

RMI Ports: 12401-12500
 JMS Ports: 12601-12700
 ΔJP Ports: 12501-12600

RMI-IIOP Ports

IIOP Ports:
 IIOP SSL (Server only):
 IIOP SSL (Server and Client):

Command Line Options

Java Executable:
 OC4J Options:
 Java Options:

Related Links: [Tracing Properties](#)

Note: In Oracle 10.1.2 it is required to add a (^) prior to the switch or 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
```

In this instance, `<probe_install_dir>` is the path to the directory where the Java Agent 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 212.

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 apply exactly for each situation. Therefore, 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 plan to update before making the changes.

This section provides instructions for configuring the following JBoss application servers:

- ▶ “JBoss Versions Earlier Than 4.0.5” on page 211
- ▶ “JBoss Version 4.0.5 and Later” on page 212

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 before 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 following is an example of the **Xbootclasspath** parameter:

```
-Xbootclasspath/p:<probe_install_dir>\classes\Sun\1.4.2_04;  
<probe_install_dir>\classes\boot
```

In this instance, **<probe_install_dir>** is the path to the directory where the Java Agent was installed.

Note: When modifying the **-Xbootclasspath** parameter, 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. 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, 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 165.

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 before making any changes to the script.

- 3 Open the startup script to be edited using your editor.
- 4 Add the JRE Instrumenter provided parameter:
- 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, the probe was not started correctly.

Note: When you add the `-agentpath:<probe_install_dir>\lib\x86-windows\jvmti.dll` for the heap dump, you must 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 Tomcat 5.x/6.x When Running as a Windows Service

The following describes how to configure Tomcat 5.x/6.x running as a web service for the Java Agent:

- 1 Tomcat 5.x/6.x is installed in service mode and the Tomcat service is started.
- 2 From the Windows Task bar, right-click on the Apache Tomcat service icon and then select Configure.
- 3 In the Apache Tomcat Properties dialog box, select the Java tab.
- 4 Add the JRE Instrumenter provided Java parameters to the Java options section.
- 5 Restart the Tomcat service.

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 is located in the **usr\sap\j2e\jc00\j2ee\configtool** directory.
- 4 Add the **-Xbootclasspath** 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:

event_buffer.size = 10000

event_buffer.flush.level = 1000

- 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, 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 can configure the application server using an alternative, site-specific method. The generic procedure might 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 JRE Instrumenter provided options to the Java command line that starts the application server, for example:

```
-Xbootclasspath/p:<probe_install_dir>\classes\Sun\1.4.2_04;  
<probe_install_dir>\classes\boot
```

In this instance, **<probe_install_dir>** is the path to the directory where the Java Agent was installed.

This connects the probe to the application.

The 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. 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, you did not run the JRE Instrumenter or did not enter the **Xbootclasspath** parameter correctly.

Note: You can also configure the JVM process definitions of the application server by going into the Administrative Console. This does not apply to WebLogic servers.

For details on running the JRE Instrumenter, see “Running the JRE Instrumenter” on page 165.

Adjusting the Heap Size for the Java Agent in the Application Startup Script

The size of the heap can impact the performance of the Java Agent.

The default value for the heap size is 64 MB. The heap size is set in the application’s startup script using the following VM argument:

```
-Xmx<size>
```

You can increase the heap size by updating the value specified in the **-Xmx<size>** option. See your JVM documentation for more help on setting this parameter.

Configuring TIBCO ActiveMatrix/Business Works

The following instructions describe how to configure TIBCO ActiveMatrix and Business Works so that the applications can be monitored by the probe.

For TIBCO ActiveMatrix Service Bus, locate the `.tra` file and set up the Diagnostics agent JVM parameters as shown in the example below. Note the use of slash (`/`).

```
java.extended.properties=-javaagent:C:/diag/lib/probeagent.jar  
-Dprobe.id=tibco_node1
```

For TIBCO Business Works, you set up the Diagnostics agent JVM parameters in different files depending on how you deployed the application.

- Update **bwengine.tra** with the JVM parameters (see the example above) if deploying the application using TIBCO Administrator.
- Update **designer.tra** with the JVM parameters (see the example above) if deploying the application using TIBCO Designer.

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 are 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 “Loading the Diagnostics SOAP Message Handler” on page 221.

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. Therefore, this feature is not available on all versions of all application servers. For the most recent information on Diagnostics SOAP message handler support, see the Diagnostics Product Availability Matrix at http://support.openview.hp.com/sc/support_matrices.jsp.

This section includes the following:

- ▶ “Disabling the SOAP Message Handler” on page 220
- ▶ “Loading the Diagnostics SOAP Message Handler” on page 221
 - ▶ “WebSphere 5.1 JAX-RPC” on page 221
 - ▶ “Oracle 10g JAX-RPC” on page 222

Disabling the SOAP Message Handler

By default, the SOAP message handler is enabled. You can disable the handler as follows:

In the `<probe_install_dir>/etc/inst.properties` file edit the `details.conditional.properties` property to include `mercury.enable.autoLoadSOAPHandler = false`.

If the SOAP message handler is disabled, you must 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 JAX-RPC, follow these steps:

Note: For WebSphere 6.1 JAX-WS web services, Diagnostics handlers are not supported. You must recompile the application with the Diagnostics SOAP handler classes.

- 1 Locate the Web service deployment descriptor (**webservices.xml**) for the application. The directory path should look similar to the following:

```
<install_root>\config\cell\<Server>\applications\  
<WebServiceEAR>\deployments\<WebServiceName>\  
<WebServiceJAR|WARName>\WEB-INF
```

Here is an 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.

Here is an 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 JAX-RPC, follow these steps.

- 1 Locate the Web service deployment descriptor (**webservices.xml**) for the application. The directory path should look similar to the following:

```
<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.

Configuring the Discovery of a New J2EE Server for CI Population

The agent provides data to populate the J2EE Application Server and J2EE Application Domain CIs in Business Service Management.

The probe automatically populates CIs for well known J2EE servers such as JBoss and WebLogic.

You can also configure application server discovery to populate CIs for other J2EE servers. Application server name can be directly specified or configured to be discovered by JMX or be discovered by a point/code snippet.

You configure application server discovery in the probe `etc/metrics.config` file as described below.

The class `AppServerDiscoveryCollector` is located in the `<probe_install_dir>/lib/probe-jmx.jar` file and you can write your own collector class to do both application server discovery and metrics collection.

The following is the configuration for application server discovery for a generic application server. Note the collector name is case sensitive and should be different from any collector name in the `metrics.config` file.

```
<user-defined-collector-name>.class.name =
com.mercury.diagnostics.capture.metrics.jmx.AppServerDiscoveryCollector
<user-defined-collector-name>.class.path = probe-jmx.jar
<user-defined-collector-name>.app_server.configure.discovery = true
<user-defined-collector-name>.app_server.type = <user-defined-type>
<user-defined-collector-name>.app_server.server_name =
<user-defined-server-name>
<user-defined-collector-name>.app_server.domain_name =
<user-defined-domain-name>
```

And then you should add the following Java system property definition in the `app-server/javaprobe` startup script or Java command line.

```
-Dapp_server.discovery.collector=<user-defined-collector-name>
```

Every 15 minutes the probe refreshes the collectors (including the `AppServerDiscoveryCollector`) and makes the discovery based on any new configuration.

For the advanced user who knows how to use JMX to discover the new application server name and J2EE domain name, you may add the following configuration in the probe `etc/metrics.config` file.

```
<user-defined-jmx-collector-name>.class.name =  
com.mercury.diagnostics.capture.metrics.jmx.JMXCollector  
<user-defined-jmx-collector-name>.class.path = probe-jmx.jar  
<user-defined-jmx-collector-name>.depends.on.class =  
javax.management.MBeanServer  
<user-defined-jmx-collector-name>.app_server.configure.discovery = true  
<user-defined-jmx-collector-name>.app_server.type = <user-defined-type>  
<user-defined-jmx-collector-name>.app_server.server_name =  
<user-defined-server-name>  
<user-defined-jmx-collector-name>.app_server.server_name.discovery.by.jmx =  
<jmx-ObjectName>.<jmx-AttributeName>  
<user-defined-jmx-collector-name>.app_server.domain_name =  
<user-defined-domain-name>  
<user-defined-jmx-collector-name>.app_server.domain_name.discovery.by.jmx =  
<jmx-ObjectName-1>.<jmx-AttributeName-1>@<jmx-ObjectName-2>.<jmx-AttributeNa  
me-2>
```


8

Installing .NET Agents

This section describes how to install a .NET Agent and gives you information about the setup and configuration that happens during the installation.

This chapter includes:

- About the .NET Agent Installer on page 226
- Points Files and the probe_config.xml File on page 226
- Discovering ASP.NET Applications on page 228
- Automatic Configuration for Discovered ASP.NET Applications on page 229
- Discovery of IIS Metadata for CI Population of IIS Deployed ASP.NET Applications on page 229
- Non ASP.NET Applications on page 232
- Installing the .NET Agent on page 232
- Verifying the Connection with the Diagnostics Server on page 249
- About Custom Configuration and Instrumentation for the .NET Agent on page 250
- Enabling and Disabling Standard Instrumentation for Applications on page 251
- Restarting IIS on page 253
- Verifying the .NET Agent Installation on page 254
- Determining the Version of the .NET Agent on page 255
- Uninstalling the .NET Agent on page 256
- Enabling and Disabling the Diagnostics Agent for .NET on page 256

- ▶ Disabling Logging on page 257

About the .NET Agent Installer

The HP Diagnostics/TransactionVision .NET Agent installer installs a .NET Agent to collect data for either Diagnostics or TransactionVision or both. TransactionVision provides data to the Transaction Management application in Business Service Management. See the *TransactionVision Deployment Guide* in the Business Service Management Documentation Library for more information about installing the .NET Agent for TransactionVision.

The .NET Agent is installed on the machine hosting the application you want to monitor.

Points Files and the probe_config.xml File

During the .NET Agent installation, the following setup and configuration are 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 228.
- ▶ **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 “Automatic Configuration for Discovered ASP.NET Applications” on page 229.
 - ▶ Default Asp.Net.points, Ado.points and WCF.points files are installed and enabled providing standard instrumentation to enable you to start monitoring ASP.NET applications. See Chapter 1, “Preparing to Install HP Diagnostics” for .NET and .NET WCF requirements and limitations. The points files control the workload the agent captures for the application. (To generate .NET Remoting Events, you also need Remoting.points and must set up the application for instrumentation.)

- ▶ Separate instrumentation points files are created for each detected application. Instrumentation points files are created for each IIS installed ASP.NET application domain detected (<applicationDomain>.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 they are not automatically 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, webservicess.points (not used for WCF web services, use WCF.points file instead).

You can enable those points files by adding a reference to them in the **probe_config.xml** file. See “Enabling and Disabling Standard Instrumentation for Applications” on page 251.
- ▶ A default set of .NET application layers is configured. See “Default Layers for Typical .NET Applications” on page 370 for more information.

After the .NET Agent installer runs, do the following:

- ▶ **Optional custom instrumentation.** You can manually customize the instrumentation points files to capture application specific methods. See Chapter 10, “Custom Instrumentation for .NET Applications.”
- ▶ **Optional configuration.** You can modify the agent configuration in the **probe_config.xml** file. See Chapter 13, “Understanding the .NET Agent Configuration File” and Chapter 14, “Advanced .NET Agent Configuration.”
- ▶ **Restart IIS.** 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 253.
- ▶ **Begin data collection.** After you restart IIS and a URL in the application is 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 you might want to instrument. After you install the .NET Agent, you can request that the agent rescan 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 is when an ASP.NET application is installed as a Web directory instead of a virtual directory.

Discovering ASP.NET Applications After Installation

You can request a rescan of the IIS configuration if you modified an existing ASP.NET application deployment or installed new ASP.NET applications.

To request that the agent rescan the IIS configuration and update the `probe_config.xml` file, select **Start > HP Diagnostics .NET Probe > Rescan ASP.NET Applications**.

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 13, “Understanding the .NET Agent Configuration File” for details.

Note: 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 257.

Discovery of IIS Metadata for CI Population of IIS Deployed ASP.NET Applications

With Diagnostics 9.0 or later, Diagnostics populates CIs and model relationships in the Business Service Management 9.0 Operational Database (ODB) for application infrastructure elements and business transactions.

For CI population the .NET Agent installer automatically discovers the IIS configuration metadata for ASP.NET applications that are deployed under IIS versions 6.x or greater. The discovered IIS configuration metadata is written to the `iis_discovery_data.xml` file which is located in the `<probe_install_dir>/etc` directory. After you have installed the .NET Agent, you can request that the agent re-scan your IIS configuration to update for any additions or changes.

Note: This information is for integrating with Business Service Management 9.0

Runtime Population CIs for IIS Deployed ASP.NET Applications

At runtime the .NET Agent queries the `iis_discovery_data.xml` file for IIS configuration metadata associated with the instrumented appdomain. If the associated metadata is found, the agent forwards the data to its Diagnostic Server which populates the Business Service Management ODB CITs for .NET Application. See Chapter 21, “Setting Up the Integration Between Business Service Management and Diagnostics” for a discussion of the integration with the Business Service Management 9.0 ODB model for .NET Applications.

Discovery of IIS Metadata of IIS Deployed ASP.NET Applications During Installation

The .NET Agent installer discovers IIS deployed ASP.NET applications on the machine when the agent is installed. The .NET Agent installer discovers applications by querying the WMI (WMEB) Provider for the IIS configuration metadata. The pertinent metadata is written to the `iis_discovery_data.xml` file.

Discovery of IIS Metadata of IIS Deployed ASP.NET Applications After Installation

You must request a re-scan of the IIS configuration metadata 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 write a new `iis_discovery_data.xml` file, run **Start > HP Diagnostics .NET Probe > Rescan ASP.NET Applications** shortcut. Note that the new `iis_discovery_data.xml` file is not intended for editing by the user; any such user edits will be overwritten by executing this shortcut.

Privilege Requirements for Discovery of IIS Deployed ASP.NET Applications

The user must have Administrator privileges on the machine that the .NET Agent is installed on, otherwise the WMI queries will fail and the `iis_discovery_data.xml` file will not be created.

Debugging the Discovery of IIS Deployed ASP.NET Applications

If the `iis_discovery_data.xml` file is not created or there is any reason to suspect that some of its metadata may be inaccurate, you can enable the creation of a detailed debug file to examine the results of the WMI queries. To enable the creation of a detailed debug file, change last parameter of the Target Property for the **Start > HP Diagnostics .NET Probe > Rescan ASP.NET Applications** shortcut from "false" to "true". When the Rescan ASP.NET Applications shortcut is executed, an `<probe_install_dir>/log/AutoDetect.log` is created. Note that you should have Administrator privileges when executing this shortcut. You can send the `AutoDetect.log` to HP Support for analysis.

Non ASP.NET Applications

The .NET Agent installation automatically discovers your ASP.NET applications, creates settings for the applications in the **probe_config.xml**, and creates template points file for them. For each non-ASP.NET application—for example, NT Service, console application, UI client—you must 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 want to monitor.

The following is an example of a **probe_config.xml** setting for an application called SimpleConsoleHost.exe:

```
<process name=="SimpleConsoleHost.points">
  <points file="SimpleConsoleHost.points"/>
  <logging level=" "/>
</process>
```

The following is an 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.

Note: If there is a pre-existing installation of the .NET Agent on the host machine, see “Upgrade and Patch Install Instructions” on page 755 for important instructions on how to upgrade the agent systems.

This section includes:

- “Preparing for the Installation” on page 233
- “Probe Aggregator Service” on page 235
- “Launching the .NET Agent Installer” on page 236
- “Running the Installation” on page 237
- “Installing the Agent as a Profiler Only” on page 241
- “Installing the Agent to Work with a Diagnostics Server” on page 243

Preparing for the Installation

You must install the .NET Agent on the host machine of the application you want to monitor. The default configuration of the .NET Agent is set to monitor the application effectively, with very little impact on the performance of the application.

For instructions on advanced .NET Agent configuration, see Chapter 14, “Advanced .NET Agent Configuration.”

Framework Requirements for the .NET Agent Host

The .NET Agent (version 9.x) requires .NET Framework 2.0 or later. The .NET Framework must be installed on the machine before you run the .NET Agent installation.

.NET Framework	2.0 or later
----------------	--------------

Important: If you must support .NET Framework 1.1, you will need to use an earlier version of the .NET Agent (8.x).

WCF Requirements and Limitations: Monitoring .NET Windows Communication Foundation (WCF) services requires .NET Framework 3.0 SP1 or greater. Only the following bindings are supported:

- BasicHttpBinding
- WSHttpBinding
- NetTcpBinding

If your application uses a binding that is not supported, the .NET probe only creates a generic server request for each WCF method. It will not be a web Service and there will be no XVM correlation.

System 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 support the agent's processing:

Platform	All Supported Platforms
Memory	60 MB Additional RAM
Free Hard Disk Space	200 MB Additional Space

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 Internet Explorer 6 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 .NET Agent installation. The default port range is 35000 - 35100.

Probe Aggregator Service

The Probe Aggregator Service can optionally be installed as part of the .NET Agent installation. It runs as a Windows Service, **HP Probe Aggregator**.

The Probe Aggregator Service aggregates probe data to 5 second intervals before sending the performance data to the Diagnostics mediating server. This is useful when the volume of data collected based on instrumentation of multiple applications is high and networking traffic would be too great if not aggregated. See “.NET Probe Aggregator Data Flow” on page 754 for a technical diagram of the data processing.

The basic .NET Agent installation, without the Probe Aggregator Service, results in performance data being sent to the Diagnostics mediating server as method starts and stops occur.

There are performance trade-offs to using the Probe Aggregator Service. So you must assess the requirements in your environment. For example, consider using the probe aggregator when you have two or more .NET probe instances running on the same system. Actual network overhead is dependent on the applications being monitored, so you need to determine if the potential savings in network bandwidth and mediator load offsets the increased memory usage on the application system.

When you install the .NET Agent with the Probe Aggregator Service, this service runs automatically and waits for connections from the .NET probes. Standard configuration of the probe aggregator is done during the .NET Agent installation. The `<probe_install_dir>\ProbeAggregator\etc\probeaggregator.properties` file is used to set configuration parameters for the Probe Aggregator (for example, setting the SQL trending threshold).

If you decide, post installation, to install the Probe Aggregator Service you can run the .NET Agent installation again, selecting the **Change** button. Then select the check box for installing the **Probe Aggregator Service**.

Performing a remove or uninstalling the .NET Agent also removes the Probe Aggregator Service. See “Uninstalling the .NET Agent” on page 256.

See “Enabling and Disabling the Diagnostics Agent for .NET” on page 256 for how to disable and enable the Probe Aggregator Service.

Launching the .NET Agent Installer

Launch the installer from the HP Software Web site when installing the Profiler trial software. You can also install the Java Agent from the Diagnostics installation disk, another location or from the Diagnostics Downloads page in Business Service Management, when you have the full Diagnostics product.

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 installs the 32-bit Windows version of the .NET agent. To install the 64-bit version of the .NET agent, **Browse the DVD** to locate the **Diagnostics_Installers** directory and double-click the **HPDiagTV.NetAgt_<release number>_win64.msi** file.
- 3 Continue with “Running the Installation” on page 237.

To launch the Installer from another location:

- 1 From the <HP Diagnostics Installation Disk>\Diagnostics _Installers directory, copy the file **HPDiagTV.NetAgt_<release number>_win32.msi** (for 32-bit Windows) or **HPDiagTV.NetAgt_<release number>_win64.msi** (for 64-bit Windows) to the new location and double-click the file.
- 2 Continue with “Running the Installation” on page 237.

To launch the installer from the HP Software Trial Software Download Web site:

- 1 Go to the HP Software Web site’s Download Center.

2 In the **Quick Search** section, in the **Products** list, click **Diagnostics** and click **Search**.

3 Under **Software Trial**, select the appropriate link.

4 Follow the download instructions on the web site.

Continue with “Running the Installation” on page 237.

To launch the Installer from the Business Service Management Diagnostics downloads page:

1 In **Business Service Management**, 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 Service Management if put into the required directory for Business Service Management to access. You can enable this during the installation of the Diagnostic Server, or you can copy the .NET agent installers manually from the Diagnostics installation disk to the required location.

Continue with “Running the Installation” on page 237.

Running the Installation

After you launch the installer, you are ready to begin the main installation procedure.

Note: If there is a pre-existing installation of the .NET Agent on the host machine, see “Upgrade and Patch Install Instructions” on page 755 for important instructions on how to upgrade the agent systems.

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.

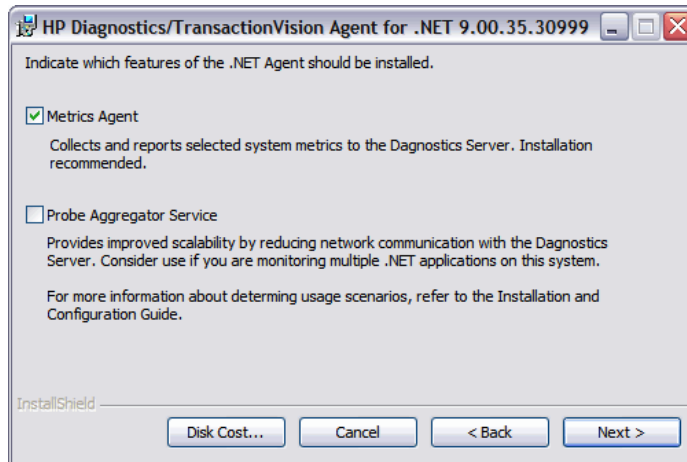
- 2 Provide the location where you want the Agent installed.

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 you want to install.



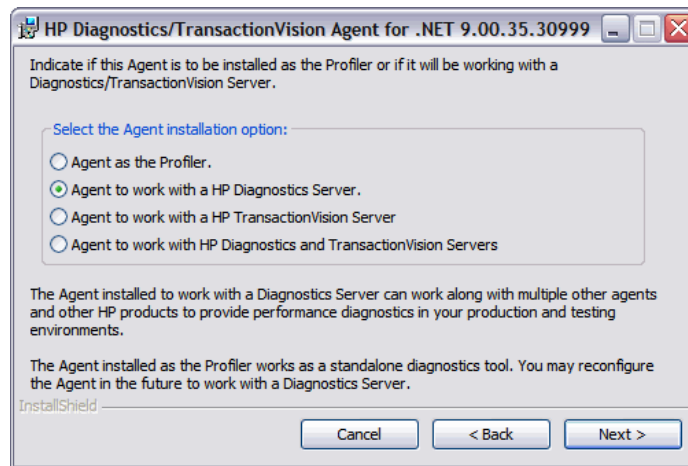
Metrics Agent. It is recommended that you install the Metrics Agent which is checked by default. See Chapter 17, “.NET System Metrics Agent - Systems Metrics Capture” for more information. But if you do NOT want to capture system metrics on the host machine you can uncheck the **Metrics Agent** box.

Probe Aggregator. Optionally you can select to install the Probe Aggregator Service. This service aggregates .NET Agent data to 5 second intervals before sending the performance data to the Diagnostics mediating server. This can improve scalability by reducing network communications with the server but the aggregator can also increase system overhead. See “Probe Aggregator Service” on page 235 for more information on the performance tradeoffs to installing the Probe Aggregator.

Disk Cost. 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 with a Diagnostics Server, 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.

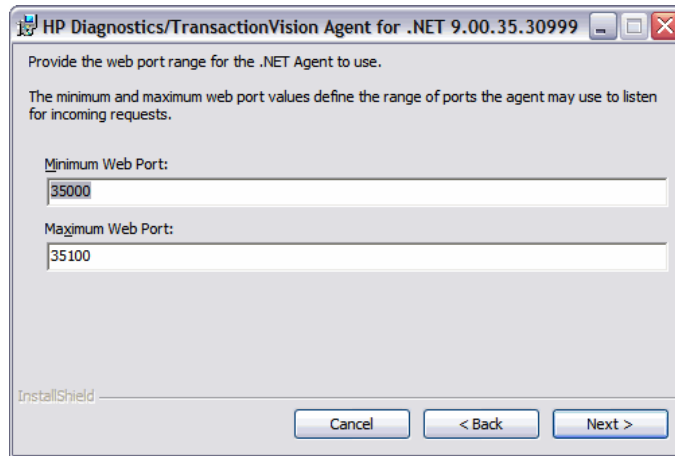
- ▶ If you are installing the Agent as the Diagnostics Profiler only, continue with “Installing the Agent as a Profiler Only” on page 241.
- ▶ If you are installing the Agent to work with a Diagnostics Server, continue with “Installing the Agent to Work with a Diagnostics Server” on page 243.
- ▶ If you are installing the Agent to work in a TransactionVision environment, continue with “Installing the Agent to Work in a TransactionVision Environment” on page 249.
- ▶ 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 243 and then “Installing the Agent to Work in a TransactionVision Environment” on page 249.

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, you want to assign to the Agent.
- **Maximum Web Port.** Type the highest port number, in a range of ports on the Agent host, 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 could already be in use if another Agent or application previously claimed them.

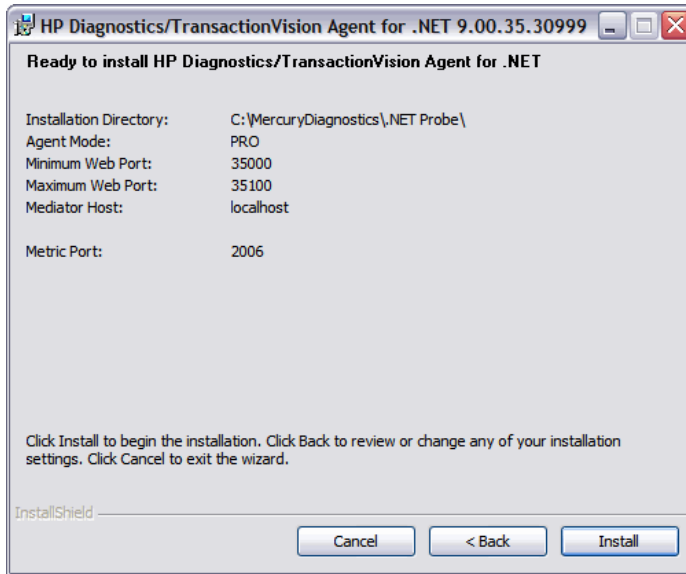
The minimum size for the port range is equal to the maximum number of Agents that will be concurrently running on the Agent's host.

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 might 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 251 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 253.

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 domain name of the monitored application. The agent name is assigned as the probe entity name.

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 is being monitored, and the type of instrumentation.

For example, the agent name for the .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.

The default agent name auto-generated by the agent when the agent name field is left blank is equivalent to specifying \$(APPDOMAIN).NET.

To override the default name, 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

\$(WEBSITENAME): The IIS Web site under which the application is hosted.

Note: For applications which are not hosted in IIS the agent name will be reverted to the default, that is, \$(APPDOMAIN).NET. An example of this would be console applications.

Note: For newly installed IIS applications you may need to run **Rescan ASP.NET Applications** from the HP Diagnostics .NET Agent program group in the Windows Start menu.

- **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 could assign all of the probes for a particular enterprise application to a single probe group so that you can monitor both the performance at the group level and the performance based on individual probe entities.

Click **Next** to continue.

- 2 Provide the information needed to enable the .NET Agent to communicate with the Diagnostics mediator server.

If you selected to install the Probe Aggregator Service, you will see the Probe Aggregator Data Port instead of the Diagnostics Server Data Port and Probe Aggregator Metric Port instead of Diagnostics Server Metric Port.

HP Diagnostics/TransactionVision Agent for .NET 9.00.35.30999

Provide the location of the Diagnostics Server in Mediator mode.

Diagnostics Server (Name or IP address): localhost Port (Default is 2006): 2006

Diagnostics Server Data Port (Default is 2612): 2612

Diagnostics Server Metric Port (Default is 2006): 2006

Test

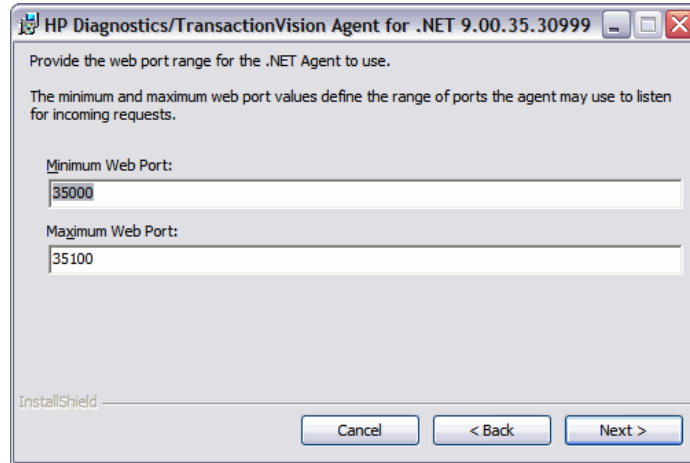
InstallShield

Cancel < Back Next >

- In the **Diagnostics Server (Name or IP address)** box, type the host name or IP address of the host for the Diagnostics mediator server.
- 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.

- ▶ 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 changed the port since the Diagnostics Server was installed, specify that port number here instead of using the default.
- ▶ If you selected to install the Probe Aggregator Service, you will see the **Probe Aggregator Data Port** box instead of for the Diagnostics Server data port. Type in the port number where the Diagnostics mediating server is listening for the Agent communication when probe aggregation is installed. The default port number is 2626. If you changed the port since the Diagnostics Server was installed, specify that port number instead of using the default.
- ▶ 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 changed the port since the Diagnostics Server was installed, specify that port number here instead of the default.
- ▶ If you selected to install the Probe Aggregator Service, you will see the **Probe Aggregator Metric Port** box instead of for the Diagnostics Server metric port. Type in the port number where the Diagnostics mediating server is listening for the Agent communication when probe aggregation is installed. The default port number is 45000. If you changed the port since the Diagnostics Server was installed, specify that port number instead of using the default.
- ▶ 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 made an error in the information you provided about the Diagnostics mediator server, or if there is a connection problem between the Diagnostics Server's host and the Agent's host. If the connection to the Diagnostics mediator server host cannot be resolved, an error message is displayed.
- ▶ 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, you want to assign to the Agent.
- **Maximum Web Port.** Type the highest port number, in a range of ports on the Agent host, 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 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 could already be in use if another Agent or application previously claimed them.

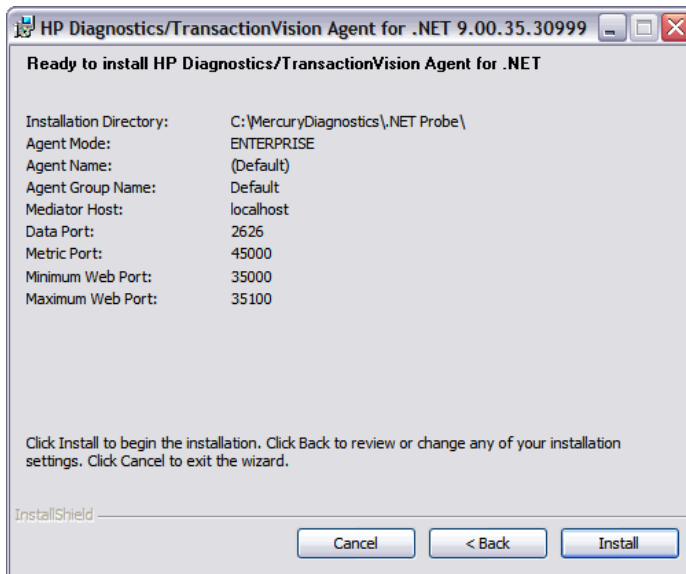
The minimum size for the port range is equal to the maximum number of Agents that will be concurrently running on the Agent's host.

Considerations when setting the Web Port Range:

- ▶ If the Agents are working with ASP.NET applications, 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, open the firewall for the ports within the range. 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 information about additional configuration for the .NET Agent:

- ▶ “About Custom Configuration and Instrumentation for the .NET Agent” on page 250.

- “Enabling and Disabling Standard Instrumentation for Applications” on page 251 for more information.
 - “Restarting IIS” on page 253.
 - “Verifying the .NET Agent Installation” on page 254.
- 6** After you exit the installer and finish any modification/customization to the .NET Agent configuration and .NET application instrumentation, restart the IIS service before using the .NET Agent with ASP.NET applications. See “Restarting IIS” on page 253.

Installing the Agent to Work in a TransactionVision Environment

If you selected to install the agent to work with both Diagnostics Server and TransactionVision Server you will see additional screens in the installation. See the *TransactionVision Deployment Guide* for details.

Verifying the Connection with the Diagnostics Server

If you are running the .NET Agent in a production environment (with Business Service Management or Diagnostics Standalone), or in a test environment (with LoadRunner or Performance Center), verify that the probe is connected to the Diagnostics Server using the following instructions.

To verify that the probe is connected:

- 1** Open the web browser and browse to http://<diagnostics_server_host>:<diagnostics_server_port>/registrar for the Diagnostics command server. You might be prompted to log on to the server.
- 2** Under **View** click on **Registered Components**.
- 3** Look for your application 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 probes only.

- 4 To find the Probe Aggregator Service, look for the **Probe** component type and the name **ProbeAggregator<hostname>** where <hostname> is the name of the host where the Probe Aggregator Service is running.
- 5 Look for the probe's status in the **Active** column. An Active status of True indicates that the probe is connected.

About Custom Configuration and Instrumentation for the .NET Agent

You can customize the .NET Agent configuration and add custom instrumentation to suit your environment and the performance issues you would like to diagnose.

The installer configures your ASP.NET applications and the .NET Agent to work together to capture the basic workload of the applications. It is possible that one or more of your ASP.NET applications was deployed in a manner that prevents the installer from detecting it. Or, you might want to enhance the standard instrumentation to capture the performance metrics for the custom classes in the application.

In Diagnostics, you can do additional configuration using the **probe_config.xml** file. For details on this file see Chapter 13, "Understanding the .NET Agent Configuration File." For instructions on advanced .NET Agent configuration, see Chapter 14, "Advanced .NET Agent Configuration."

Also in 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 of the `enablealldomains` attribute in the `<process>` element and attributes in the `<appdomain>` element 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.

These are the rules for the `enablealldomains` attribute of the `<process>` element:

- ▶ `enablealldomains = false` : If there are no domains in the list of `<appdomain>` No domains should be enabled.
- ▶ `enablealldomains = false` : If there are domains in the list of `<appdomain>` Domains should be enabled if the "enable" attribute is set to true or not defined in the enable attribute of the `<appdomain>`.
- ▶ `enablealldomains = true` : If there are domains in the list of `<appdomain>` Only Domains in the list should be enabled disregarding their "enable" attribute.
- ▶ `enablealldomains = true` : If there are no domains in the list of `<appdomain>` All domains should be enabled.
- ▶ `enablealldomains` attribute is not defined: same as if `enablealldomains = true`.

To enable or disable the instrumentation for an application:

- 1** Set the **enablealldomains** attribute in the **<process>** element 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, no applications are enabled.
- 2** Set the **enabled** attribute in the **<appdomain>** element to **true** for each application where you want to enable the instrumentation.
- 3** Set the **enabled** attribute in the **<appdomain>** element to **false** for each application that is to have its instrumentation disabled.

The following example shows instrumentation 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>
```

To enable the instrumentation for ALL applications:

- Set the **enablealldomains** attribute in the **<process>** element to **true**. This overrides the settings of the attributes in each **<appdomain>** element so that the instrumentation can be enabled without having to set numerous attributes.

The following example shows instrumentation 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 install the .NET Agent and modify the instrumentation or configuration for the Agent, restart IIS to allow 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 command server.

Note: ASP.NET automatically restarts applications under various circumstances, including when it detects that applications are redeployed, or when applications exceed 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 can be a period of overlap where there are multiple agents simultaneously registered with the Diagnostics command server and connected to the Diagnostics mediator server. This condition could cause LoadRunner / Performance Center and Business Service Management to report errors during the application restart sequence.

Verifying the .NET Agent Installation

This section applies only if you installed the .NET 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 followed the recommended installation sequence, after you install the .NET Agent and restart the instrumented application, you can verify the following:

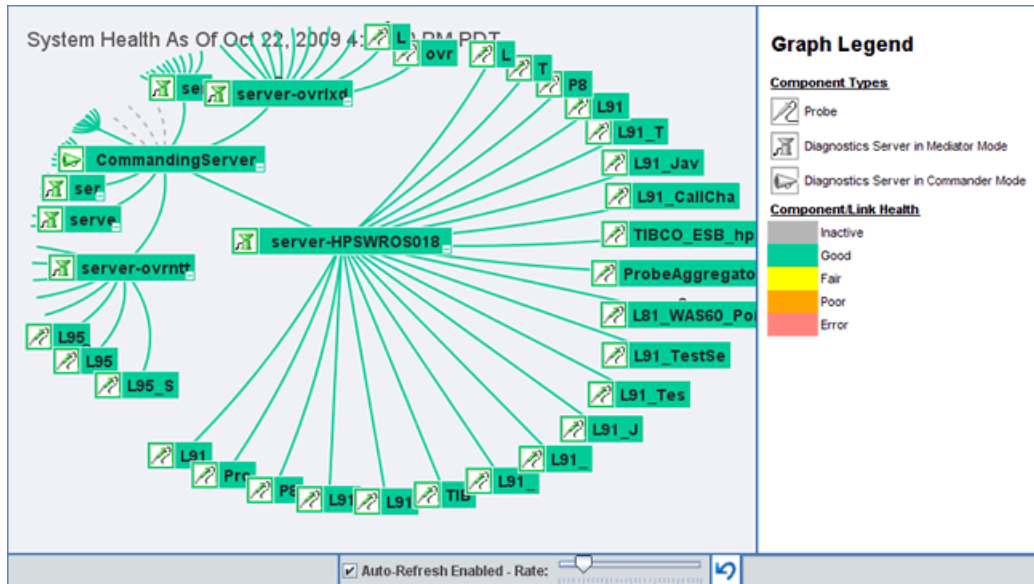
- ▶ The Diagnostics command server was successfully installed.
- ▶ Where relevant, additional Diagnostics Servers in Mediator mode are successfully installed and communicating with the Diagnostics command server.
- ▶ The .NET Agent is successfully installed and probe instances are communicating with the Diagnostics Server. And, optionally, the Probe Aggregator Service is installed and active.

Note:

The .NET agent does not register with the Diagnostics Server until a probe is started. The probe is started and registered with the Diagnostics Server when the instrumented application is run. For ASP.NET applications, this happens the first time a page is requested for the instrumented application.

The default logging level for the probe is set to **info** so that, if the probe is not displayed in System Health when you believe it should be, you can check the log file to troubleshoot the problem. If you do not see a log file, check the Windows Event Viewer.

Depending on your deployment, the new .NET Agent (and optional Probe Aggregator Service) is shown on the System Health Monitor to be reporting to either the Diagnostics command server or the Diagnostics mediator server.



If the Probe Aggregator fails to start (or starts but does not continue running) consult the log file
 <probe_install_dir>\ProbeAggregator\log\ProbeAggregatorJSW.log.

Determining the Version of the .NET Agent

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

Or you can use the System Health Monitor in Diagnostics.

Uninstalling the .NET Agent

To uninstall the .NET Agent:

- 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.

To remove the Probe Aggregator Service you can uninstall the .NET Agent which will also remove the Probe Aggregator Service. Or you can run the .NET Agent installation again, selecting the **Change** button and then de-select the check box for installing the **Probe Aggregator Service**.

Enabling and Disabling the Diagnostics Agent for .NET

The .NET Agent is enabled when it is installed. After you restart your Web server and a URL in the application is accessed, the .NET Agent begins to gather performance information.

You can 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 collection is controlled through the standard Windows services manager. The effect of enabling or disabling probes only happens the next time the probed application restarts. It has no affect on currently running applications.

Once the Probe Aggregator Service is installed and running, you can disable and enable it from the Start Menu. Select **Start > Programs > HP Diagnostics .NET Probe > Disable HP .NET Probe** or **Enable HP .NET Probe**. Selecting **Disable HP .NET Probe**, in addition to disabling the .NET probes will mark the Probe Aggregator Service as disabled, but not stop the service (in case there are running probes remaining). Selecting **Enable HP .NET Probe**, in addition to enabling the .NET probes will change the Probe Aggregator Service back to type **automatic** and start it if needed.

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

This section includes:

- Custom Instrumentation for Java Applications
- Custom Instrumentation for .NET Applications

9

Custom Instrumentation for Java Applications

This section explains how to control the instrumentation that Diagnostics applies to the classes and methods of the applications to enable the Java Agent to gather the performance metrics.

This chapter includes:

- ▶ About Instrumentation and Capture Points Files on page 262
- ▶ Coding Points in the Capture Points File on page 264
- ▶ Defining Points With Code Snippets on page 273
- ▶ Controlling Class Map Capture on page 288
- ▶ Instrumentation Examples on page 289
- ▶ Understanding the Overhead of Custom Instrumentation on page 305
- ▶ Instrumentation Control on a Per Layer Basis on page 306
- ▶ Advanced Instrumentation on page 307
- ▶ Maintaining Instrumentation from the Java Profiler UI on page 322
- ▶ Default Layers Defined for Typical Java Classes and Methods on page 334

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 a capture points file.

When you install the Java Agent, a predefined default capture points file is installed with a set of points for the platform you are using. A default capture points file containing pre-defined Java points is located at `<probe_install_dir>\etc\auto_detect.points`.

The capture points file enables you to control the scope of the instrumentation so that Diagnostics can give you all the information you need to understand the performance of the applications without overwhelming you with costly or confusing extraneous information. The instrumentation definitions in the capture points files are called *points*. The points define 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 "Using Regular Expressions" on page 782.

To add custom instrumentation, make a copy of the default **auto_detect.points** file, give it a different name, and use it to make all your instrumentation customizations. This precaution prevents you from losing your custom instrumentation when you upgrade to a new version of the Java Agent and the installer overlays the `auto_detect.points` file.

You can customize the points in the capture points files to include methods, classes, packages, and namespaces for areas of the application that do not fall within the default points. A common situation that might require custom points is when a J2EE 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 organize the performance metrics into meaningful tiers of information that can be compared as part of a triage process. They control the collection behavior of the instrumentation.

The points in the capture points file installed with the Java Agent are grouped into default layers. You can customize the default layers and create new layers. For description of the default layers see “Default Layers Defined for Typical Java Classes and Methods” on page 334.

Notes:

- The default capture points file name is specified in `<probe_install_dir>\etc\probe.properties`.
 - To override the default file name so that the copy with your custom points is used instead, use the `-Dprobe.points.file.name=<newPointsFileName_NoExtension>` JVM property.
-

Coding Points in the Capture Points File

The following arguments 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 following sections describe the arguments.

- “Mandatory Point Arguments” on page 265
- “Optional Point Entries” on page 267

Mandatory Point Arguments

Every point, except for the points for CLP, 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	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 can be used.
method	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 can be used.
signature	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	<p>Specifies a layer, sublayer, or tier under which the data from this point is grouped. Layers are a part of the instrumentation collection control.</p> <p>Layers in a point can be specified with nested layers or sublayers by separating the layer names with a / (slash). The layer specified following the slash is a sublayer of the layer specified before the slash. A sublayer can have its own sublayers by coding another slash and layer name following a sublayer name.</p> <p>In the UI, the sublayers for a layer are displayed under their parent layer. For example, the sublayers JSP and Struts would be displayed under the web layer and a drilldown would exist from Web to JSP and Struts.</p>

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 can 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 “Using Regular Expressions” on page 782.

Optional Point Entries

Point definitions can contain one or more of the following arguments:

Argument	Description
keyword	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 can use implementation-specific arguments not documented here.
ignore_cl	Specifies 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	Specifies a comma-separated list of methods to ignore. Any method matching one of the methods specified with ignore_method is not instrumented.
ignore_tree	A list of classes or regular expressions. Any subclass of a class matching one of the list items is excluded from the instrumentation.
method_access_filter	A list of method specifiers, separated by commas. The available specifiers are static , private , protected , package , and public . Any method matching this point is not instrumented if its access specifier matches any of the listed values.

Argument	Description
deep_mode	<p>Specifies how subclasses are handled. This argument accepts three values:</p> <ul style="list-style-type: none"> ▶ 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 subinterfaces 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 subinterfaces 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	<p>Constrains the context in which instrumentation is performed. If specified, the inserted bytecode will be caller side. Any valid regular expression can 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.</p>
ignoreScope	<p>Lists method names or regular expressions and excludes certain packages, classes, and methods from those included in the scope specified in the scope argument.</p>

Argument	Description
detail	<p>Specifies more specific capture instructions. It is a comma-separated list of the following:</p> <ul style="list-style-type: none"> ▶ 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 runs is displayed in the method's argument field in the Diagnostics console and can also be used as the aggregation parameter in the layer argument. The code-key argument specifies the secure code key you generated for the code snippet you created for the point. See “Defining Points With Code Snippets” on page 273 for information about code snippets and “Securing Code Snippets” on page 286 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 run.

Argument	Description
detail (continued)	<ul style="list-style-type: none"> ▶ disabled – prevents the instrumentation inserted into the bytecode from reporting data. A disabled point can be dynamically enabled using the Instrumentation control web page so that it will begin reporting data. This web page can be accessed using the Profiler URL <a href="http://<probe_install_dir>:<probe_port>/inst/layer">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 that 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”, that is, 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 <code>java.sql.Statement</code> class. ▶ is-prepare-statement – marks calls returning <code>java.sql.Statement</code> 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 (<code>cpu.timestamp.collection.method = 0</code>).

Argument	Description
<p>detail (continued)</p>	<ul style="list-style-type: none"> ➤ condition – prohibits instrumentation by this point unless the specified condition is met. The conditions are static and are defined by the <code>details.conditional.properties</code> 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 are 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 of the thread stack trace onto stdout whenever this point is used for instrumentation. ➤ gen-runtime-trace – causes printing of the thread stack trace onto stdout whenever the methods instrumented by this point are executed. ➤ trace – causes printing of verbose instrumentation information into <code>probe.log</code> 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)	<ul style="list-style-type: none"> ▶ 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	Identifies server requests whenever the when-root-rename detail is in effect.
layerType	<p>Specifies special handling for some instrumented methods and accepts the following values:</p> <ul style="list-style-type: none"> ▶ method – no special handling (default). ▶ trended_method – identifies methods to be displayed in the Trended Methods view. ▶ Portlet – identifies portlet lifecycle methods that are used for the Portal Components views. These are set by HP Diagnostics and should not be modified. ▶ sql – identifies methods that are used to capture SQL for the SQL views. These are set by HP Diagnostics and should not be modified.
priority	Whenever there is more than one instrumentation point that can be applied to a given method, and the Diagnostics Agent cannot resolve the conflict on its own, the point's priority determines which point to use. Higher priority wins. The default is zero.
active	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 to execute the code snippet before or after the instrumented method. The code snippet is typically secured using a code-key argument to prevent unauthorized modifications of the code snippet. The values for the code-key arguments can be generated using any running probe's code-key generator page and are valid on any Java Agent installation. For more information on the code-key see “Securing Code Snippets” on page 286.

The actual code snippets for a point are entered into the `<probe_install_dir>/etc/code/custom_code.properties` file. These snippets are then 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 bytecode 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 displayed in the Diagnostics console.

Code snippets can also be used to store values for a particular fragment directly or 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 274
- ▶ “Code Snippet Grammar” on page 275
- ▶ “Securing Code Snippets” on page 286

Using Code Snippets

To use code snippets when specifying a point in `<probe_install_dir>/etc/auto_detect.points`, the following detail:

```
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 was entered for the point. The code-key value secures the code in the code snippet and ties 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 must 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
null constant	null
a no-type, no-value constant	void

► **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 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. Must 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 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 (<i>required</i>)	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 must not be used in combination with the store-fragment special fields for web service inbound data (special fields starting with ##WS_inbound_*). Use ONLY one 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. The following variables must be used with the **store-fragment** detail setting.

Variable	Use
##WS_consumer_id	Stores the consumer Id for a particular fragment.
##WS_SOAP_fault_code	Stores the SOAP fault code.
##WS_SOAP_fault_reason	Stores the SOAP fault reason.
##WS_SOAP_fault_detail	Stores the SOAP fault detail.
##WS_inbound_service_name	Stores the inbound web service name.
##WS_inbound_operation_name	Stores the inbound web service operation name.
##WS_inbound_target_namespace	Stores the inbound web service target namespace.
##WS_inbound_port_name	Stores 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. Use these thread local storage variables with caution because they have overhead associated with them. Use them only 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 without the leading `##` 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
<code>##SOAPHandler_wsname</code>	Stores the web service name for later use by the SOAP Handler.
<code>##<any_string></code>	Stores any value for later retrieval in a following code snippet.

► Class references and static members

Static members/methods can be accessed by pre-pending the class with an `@` symbol to identify it as a Static, and 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 `ProbeCodeSnippetHelperV5`. The `CodeSnippetHelperV5` uses some APIs available only with Java 5 or later.

The following shows `ProbeCodeSnippetHelper` functionality.

```

/*
 * (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 following shows ProbeCodeSnippetHelperV5 functionality.

```

/*
 * (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 that builds the bytecode requires a “\” (backslash) before each carriage return when it must continue parsing the stack of statements. The final line of the Code Snippet stack of statements should not contain a backslash 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 are 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 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. It can be handy in other situations as well.

```
@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. 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 a pre-defined special field to store the object references returned by called methods. 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 a special field to store the object references returned by called methods. 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@.getThreadContextValue("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 is 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 snippet syntax allows for limited conditional logic that is equivalent to the Java if-else statement. This syntax enables you to compare object references of the same type or integer or boolean primitives using both the `==` and `!=` operators. Literal value and other 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();
```

► Exception Handling

A limited form of exception handling is provided by the following syntax:

```
!{<code-snippet-code>}!
```

The specified code is executed and the value of the above expression is the thrown exception, or null if no exception was thrown during the execution of the code.

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 accidentally 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 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 it matches the code-key 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 Agent is installed with a tool that generates the code-key from the code snippet 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:

HP Diagnostics

This page provides you with the ability to validate a snippet of code for use in the probe's points file, as well as generate the required secure code-key.

If a point's code does not match its key, the probe will refuse to use that code during instrumentation.

Input your code snippet:

Submit

Resulting point section:

HP Diagnostics J2EE Probe "WAS6_Plants"

- 2 Enter the code snippet 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 is 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 is 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 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: Be very careful when using this property. If you disable this check, you could experience unexpected processing overhead and unpredictable performance monitoring results.

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 and help you instrument the application code properly. The cost for using class map comes from the additional overhead that creating the map places upon the agent's host system.

By default the property **use.class.map=false** is set in the `probe.properties` file. Changing this to **true** provides a class map.

Instrumentation Examples

The examples in this section 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 sublayer
- 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 lightweight memory diagnostics (LWMD)
- Configuring collection leak pinpointing
- 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)

Custom layer and sublayer

- The following point creates a custom sublayer 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 following point captures all methods in the MyCompany.MyFoo class:

```
[myCompany.myFoo_AllMethods]
class = myCompany.myFoo
method = !.*
signature = !.*
layer = FOO/BAR
```

Ignore Specified Methods

- The following point 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 following point 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 following point 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 following point captures all methods in the constructor for the MyCompany.MyFoo class:

```
[myCompany.myFoo_Constructor]
class = myCompany.myFoo
method = <init>
signature = !.*
layer = FOO/BAR
```

- ▶ The following point 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 following point captures the setFoo method in the MyCompany.MyFoo class:

```
[myCompany.myFoo_setFoo]
class = myCompany.myFoo
method = setFoo
signature = !.*
layer = FOO/BAR
```

- ▶ The following point captures all "set" methods in the MyCompany.MyFoo class:

```
[myCompany.myFoo_AllSets]
class = myCompany.myFoo
method = !set.*
signature = !.*
layer = FOO/BAR
```

- ▶ The following point 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 following point captures the getFoo method with no arguments 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 following point 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 307.

```
[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 following point 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 307.

```
[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 is used for both soft and hard deep_mode examples:

```
public interface Interface1 {
    public void callerMethod();
}
```

The following class is 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 following point captures the "callerMethod" in the Class1 class:

```
[Training-1]
class    = Interface1
method  = !.*
signature = !.*
deep_mode = soft
layer   = Training
```

- The following point captures all methods in Class 1 (for example, "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.

Important: Extreme caution has to be exercised when using argument capture. Unless the set of all possible values of the captured argument is finite, the agent will run out of Java heap space.

- Suppose that you instrument a method called `myCompany.myFoo.myMethod()`, and `myFoo` has another method called `getComponentName()` that returns a `String`. The following example shows the result of `getComponentName()` as the argument in Diagnostics (`#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 following point captures the first argument to myMethod and shows it as the captured argument in Diagnostics. It also uses it as the sublayer 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 the value myArg, the layer is 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. If you use `#arg2`, you would capture 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 the following:

```
"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 the following:

```
"DIAG_ARG:type=ws&ws_name="+<WebServiceName>+"&ws_op="+
<OperationName>+"&target="+<TargetName>
```


Here is an 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().getWsdIService();\
#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

- 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, the name of such a server request is Background-mySuffix, and the type of the server request is RootRename.

- Consider the following point instead:

```
class    = Statement
method  = execute
layer   = Database/JDBC/Execute
detail  = when-root-rename
```

Notice that the rootRenameTo property is skipped. The name of such a server request is Background-Database (because Database is the first sublayer) and the server request type is RootRename again.

Adding a field to a class

- Consider the following point:

```
class    = com.corp.Foo
method  = bar
detail  = add-field:protected:Object:serviceName
```

The detail causes 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 attaches `my_attribute` name to every captured instance of `com.corp.Foo.bar()`.

The name prefixed with `display_` and its corresponding value is 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 instruments 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 is not recorded. This is caused by the **detail = no-layer-recurse**.

This, however, is only for direct recursion. If `com.corp.Foo.bar` calls an instrumented method from another layer that calls this method again, all methods are recorded.

```
class    = com.corp.Foo
method  = bar
layer   = Example/MyBar
detail  = no-layer-recurse
```

Performing caller side instrumentation

- The following point causes caller side instrumentation to be performed (as opposed to the default callee instrumentation). This is caused by the **detail = caller**.

Another way to do caller side instrumentation is to use the “scope” property as described in “Using Caller Side Instrumentation” on page 307.

```
class    = com.corp.Foo
method  = bar
detail  = caller
```

Configuring allocation analysis

Both of the following examples track allocations of `java.lang.Integer` in the package `com.mycompany.mycomponent`. There are, however, two differences:

- In the first example (**detail = leak**), tracking is managed. It starts when the user clicks **start** in the profiler and stops when the user clicks **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 you will not see “new Integer” show up on an instance tree. In the second example, you will.

Example 1 – Managed. Tracking starts when the user clicks **start** and stops when the user clicks **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
```

Neither of these points captures reflected allocation. To enable reflected allocation capture, simply append the detail “reflection” to the point (**detail = leak,reflection**).

Configuring lightweight 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.

```
[Light-Weight Memory Diagnostics]
scope = !com\.mycompany\.mycomponent\.*
class = java.lang.Integer
keyword = lwmd
active = true
```

You also need to set the property `lwm.diagnostics.capture=true` in the `dynamic.properties` file. For more information, see the *HP Diagnostics User's Guide* chapter on the "Collections and Resources View."

Configuring collection leak pinpointing

Regardless of JRE version, you must run the JRE Instrumenter if you want to use the collection leak pinpointing (CLP) feature in the Java Agent.

By default collection leak pinpointing is enabled in the **auto_detect.points** file.

```
[Collection Leak Pinpointing]
keyword = clp
```

In the **dynamic.properties** file you can set the following properties to configure collection leak reporting. These same values can also be set in the Java Profiler Configuration tab UI (see “Enabling and Configuring Collection Leak Reporting” on page 453).

clp.diagnostics.reporting=true

Enable reporting in the Diagnostics UI. You can disable reporting in the UI for this feature by unchecking the checkbox.

clp.diagnostics.growth.time.threshold.flag = 60m

The threshold of time duration in which the collection has size growth. If a collection's size growth period exceeds this threshold, it will be flagged as a memory leak by the probe. To avoid false positives, this value should be larger than the time required by your application to fully initialize all its caches.

clp.diagnostics.nongrowth.time.threshold.unflag = 60m

For an already flagged leaking collection, if its size stops growing continually for this threshold time period, the probe will unflag it as a leak.

Enabling object lifecycle monitoring for JDBC result set

A few preconfigured instrumentation points allow object lifecycle monitoring but are disabled by default. Two of them are shown in the following example.

The example shows how to enable object lifecycle monitoring for JDBC Result Sets. For a more detailed discussion on object lifecycle monitoring, see 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, two actions are 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 following, the probe performs 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\.jdbc\.WSJdbcResultSet)
method = !(close)|(closeWrapper)
signature = !.*
deep_mode = soft
detail =
condition:mercury.enable.resourcemonitor.jdbcResultSet,before:code:513a2b36,method-trim
```

Adding a disabled point and enabling it at runtime

- In the following example, the point is disabled. This does not mean that instrumentation does not happen. Instrumentation happened but did not collect any data. This significantly lowers the runtime overhead of such a point.

To enable 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 do not want instrumentation to happen at all, 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 does 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()` is not reported. However, any code snippets associated with the point will always be executed.

```
[My Example]
class    = Example
method  = myMethod
detail  = method-trim, before:code:...
```

Enabling collection of CPU time for a method

- In the following example, the detail “method-cpu-time” causes 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 prints several pieces of debug information in standard out and `probe.log`.

- The **gen-instrument-trace** detail causes printing to stdout the thread stack trace whenever this point is used to instrument a method.
- The **gen-runtime-trace** causes printing to stdout the thread stack trace whenever `Example.myMethod()` is run.
- The **trace** detail causes printing in the `probe.log` verbose instrumentation information whenever `Example.myMethod()` is run.

```
[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, beware of over-instrumenting the application because it can introduce excessive latency into the probed application. 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 run.

For example, if a method takes 100ms, and instrumentation makes it run in 101ms, overhead is 1%. If a method takes 10ms and instrumentation changes its response to 11ms, 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 can affect the latency of the application's response even though its overhead percentage is much smaller.

Unlike a traditional profiler, HP Diagnostics uses bytecode instrumentation. This allows the default instrumentation to be selective 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 (for example, JNDI lookups).

You should also consider Diagnostics data overhead when you are customizing the instrumentation for an application. The more methods you instrument, the more data the probe must serialize and pass over the network to the Diagnostics Server. You can tune the Java 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 tuning of a probe can cause CPU, Memory and Network overhead on the physical machine where the Java Agent is installed. For more information about managing Latency, CPU, Memory and Network overhead, see Chapter 12, "Advanced Java Agent and Application Server Configuration."

Instrumentation Control on a Per Layer Basis

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 to apply changes 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 (see the Instrumented Layers page below).

You can access the Instrumented Layers page from the URL: <http://<probe-host>:<probe-port>/inst/layer>.

Layer	Hits	Active Points	Actions
(Other)	0	2 / 2	[Disable] [Clear # Hits]
(keyword) http	129707	6 / 6	[Disable] [Clear # Hits]
(keyword) lwmd	31377106	4159 / 4217	[Enable] [Disable] [Clear # Hits]
(keyword) remote-http	0	15 / 15	[Disable] [Clear # Hits]
(keyword) rmi	0	206 / 206	[Disable] [Clear # Hits]
(keyword) soap fault	0	2 / 2	[Disable] [Clear # Hits]
Business Tier/EJB/Entity Bean	989945	63 / 63	[Disable] [Clear # Hits]
Business Tier/EJB/Session Bean	110926	35 / 35	[Disable] [Clear # Hits]
Database/JDBC/Connection	107755	49 / 49	[Disable] [Clear # Hits]
Database/JDBC/Execute	105821	79 / 79	[Disable] [Clear # Hits]
Directory Service/JNDI	175	4 / 4	[Disable] [Clear # Hits]
Legacy/JCA/Connection	52877	1 / 1	[Disable] [Clear # Hits]
Legacy/JCA/ECIConnectionFactory	51936	2 / 2	[Disable] [Clear # Hits]
Legacy/JCA/ManagedConnectionFactory	2	2 / 2	[Disable] [Clear # Hits]
Web Tier/Servlet	55198	11 / 11	[Disable] [Clear # Hits]

HP Diagnostics J2EE Probe "WAS6_Plants_T155_W2k3", version 7.0.9.214

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 307
- “URI Aggregation Instrumentation” on page 309
- “CORBA Cross VM Instrumentation” on page 310
- “Using RMI Instrumentation” on page 311
- “Using thread local storage to store the SOAP payload” on page 311
- “Performing Correlation Across Multiple Threads” on page 312
- “Using fragment local storage to store Web service field” on page 315
- “Using Annotations for Custom Instrumentation” on page 319

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 can increase application classloader time because each class specified in the scope must be checked for references to the class/method 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 following example, the parse methods for the `javax.xml.parsers.SAXParser` and `javax.xml.parsers.DocumentBuilder` are 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
```

- 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;
scope = !com.myCompany.myFoo.*
deep_mode = soft
layer  = FOO/JNDI
```

Notes:

- 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 (that is, `[JNDI-lookup-FOO]`).
-

- During final triage steps for a performance issue, it can be impractical to use the classmap and individual 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 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 example also captures 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 (that is, <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 following example allows the ServerRequests to appear uniquely in the Diagnostics console:

```
http://server/myApplication?page=home  
http://server/myApplication?page=openReport
```

```
[HttpCorrelation]  
args_by_class=!*&page
```

The following example shows that more than one URI parameter can be used for URI parsing:

```
args_by_class=!*&page&role
```

Note: Avoid using a session parameter or highly unique URI value because of the impact to overhead and data storage.

In a WebLogic environment, 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.

CORBA Cross VM Instrumentation

The Common Object Requesting Broker Architecture (CORBA) standard enables components written in different computer languages and running on different systems to work together.

Instrumentation for correlating CORBA cross VM instance trees is provided in the `<probe_install_dir>\etc\auto_detect.points` file.

Follow these steps in to enable cross-VM instance trees for CORBA:

- 1 Uncomment the Corba cross-VM points in the `auto_detect.points` file.
- 2 Specify the following JVM argument at Application Server startup:
-Dorg.omg.PortableInterceptor.ORBInitializerClass.com.mercury.opal.javaprobe.handler.corba.CorbaORBInitializer

- 3 Put the following jar file in the classpath:

```
<java-agent-install-dir>/lib/probeCorbalInterceptors.jar
```

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 Java 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 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 Java 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

The following example demonstrates usage of thread local storage. In particular, it shows how to store (and clean) the SOAP payload from thread local storage. SOAP payload is captured by default only for certain application servers. For more information on the support matrix, see “Configuring SOAP Fault Payload Data” on page 444.

The following example is applicable only for application servers where Diagnostics does not capture payload out of the box.

First, it is necessary to identify where to access the payload from. Assume that the payload is the second argument of a method called `DispatchController.dispatch()`.

The keyword `store-thread` causes the Java 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 provided 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;"";
```

Performing Correlation Across Multiple Threads

Asynchronous Server Requests are server requests that switch threads between server request start and end events. In the most simple case, one thread receives the request, partially processes it, and then hands it off to another thread to complete processing and to send the response back to the requesting party.

Diagnostics offers two operations, available through code snippets, to allow the Java agent to perform correlation across multiple threads:

- `parkFragment(Object anchor)`

This operation is executed to indicate that the current thread will no longer run the current server request. All method invocations, as recorded by the Java Agent, are artificially terminated at this point. This is to indicate that even though some of these methods will continue execution, their activity will have nothing to do with the current server request. Furthermore, even if the current thread will invoke some instrumented methods after calling `parkFragment`, these calls will not be reported. The server request is no longer considered running, and the specified object (anchor) is used by the application as a unique identification of the server request to be resumed later (presumably, by another thread).

► `resumeFragment(Object anchor)`

This operation is executed to indicate that the current thread resumes execution of previously parked server request. The argument (anchor) is used to identify the server request. All active method invocations will have their start time artificially reset to the current time. This is to indicate that even though some time may have elapsed while these method were executing, their execution had nothing to do with the server request being resumed. If the current thread was already running a server request, it will be ignored (dropped).

When using these operations, it is essential that the correct anchor object, as well as the correct thread switching points are identified by the application specialist.

Beware of race conditions: if the fragment is reported "parked" too late, after the corresponding resume operation is performed, the fragment will get lost (and a warning will appear in `probe.log`). Two useful techniques to avoid the race condition are: first, calling `parkFragment` slightly before the current thread really abandons the server request, and second, try to piggyback the application built-in synchronization which is often used to hand off an object from one thread to another.

A "parked" fragment can be seen using the `pending-fragment` servlet, as "PARKED SERVER REQUEST" displayed in place of the currently running method.

The feature usually requires you to identify the thread switching points in the monitored application, and to provide the corresponding instrumentation points with code snippets. Out of the box support is provided for BEA AquaLogic.

Examples of two instrumentation points with the corresponding code snippets are presented below. They are a part of the AquaLogic support.

The first point presented below is executed whenever AquaLogic sends a sub-request to another server. The instrumented method, `PipelineContextImpl.dispatch(...)` returns true if the sub-request was successfully sent. The thread sending the sub-request does not wait for a response, but proceeds to pick up the next server request from a pipeline.

Therefore, the code snippet examines the return value, and if it is true, signals to the probe that the current server request will be suspended. The server request is identified by a `MessageContext` object, which AquaLogic creates for every incoming server request.

```
[BEA_ALSB_AsyncDispatch]
; instrumentation point for AquaLogic Service Bus asynchronous dispatch
class    = com.bea.wli.sb.pipeline.PipelineContextImpl
method   = dispatch
signature = !\Lcom/bean/wli/sb/context/MessageContext;.*
detail   = after:code:549b1b59
layer    = Service Bus/AquaLogic

# Used by [BEA_ALSB_AsyncDispatch]
# Asynchronously dispatches a subrequest for a service, the response will be
# processed on another thread
549b1b59 = (#return == true ?
@ThreadContextProxy@.parkFragment(#location,#arg1) : void);
```

Upon receiving a response from the sub-request, AquaLogic executes `RouterCallback.onReceiveResponse(...)`, possibly on another thread. The processing of the original server request resumes, and this is signaled to the probe by the code snippet.

In this case, the `MessageContext` object representing the server request is not available as an argument of the instrumented method and needs to be extracted from the `RouterCallback` object.

```
[BEA_ALSB_ProxyService_Callback_Response]
; instrumentation point for AquaLogic Service Bus callback function
class    = com.bea.wli.sb.pipeline.RouterCallback
method   = !(onError)|(onReceiveResponse)
signature = !.*
layer    = Service Bus/AquaLogic
detail   = before:code:dba72078

# Used by [BEA_ALSB_ProxyService_Callback_Response]
# Resume processing of a server request when the reply for a subservice comes back
# (or when the server request was moved to the response pipeline internally)
dba72078 =
@ThreadContextProxy@.resumeFragment(#location,#callee._context.getMessageContext());"";
```

Using fragment local storage to store Web service field

The following 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`, and so on). This is an alternative to specifying the “DIAG_ARG” string.
- How to retrieve (and remove) the stored payload from thread local storage (which was 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.

Because web services are treated in a special way, several fields must be captured. These fields are described in “Code Snippet Grammar” on page 275.

The first step is to find the instrumentation points that will give access to the required fields (Web Service name, operation, namespace, port name). Suppose that there is a single class in the instrumented application that has access to all these fields. 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()`. This is the method we want to instrument.

Here is our instrumentation point (each line is explained below):

```
class    = com.myCompany.MyWSFactory
method  = create
signature = !(Lcom/myCompany/MyWSContext; *
layer   = Web Services
detail  = ws-operation, before:code: f334f0b4,store-fragment
```

The first three lines of the point shown above cause the probe to instrument anything that matches `com.myCompany.MyWSFactory.create(MyWSContext, *)`.

The fourth line specifies the layer for this point.

The fifth line provides the probe with additional information about this point (details):

- ▶ The first detail (`ws-operation`) is important because it causes the probe to treat this as an inbound Web Service.
- ▶ The second detail (`before:code: f334f0b4`) causes 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`) causes 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 below).

```
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@.getThreadContextValue("My_Inbound_Payload");\
#consumer_id = (#soap_payload == null ? null :
@com.mercury.opal.capture.proxy.ProbeCodeSnippetHelper@.getConsumerIdFromDocument(##WS_inbound_service_name<java.lang.String>,#soap_payload<org.w3c.dom.Document>));\
##WS_consumer_id = (#consumer_id == null ?
@ProbeCodeSnippetHelper@DO_NOT_STORE : #consumer_id);"";
```

First line: `f334f0b4 = #wsContext=#arg1;\`

As mentioned previously, 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`. 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 275.

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 275.

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 275.

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 275.

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 supported for capturing SOAP payload out of the box. See the previous example for details. In the followings 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 retrieves and removes the stored payload from thread local storage (see the previous example on how this was stored) and stores 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_payload
d<org.w3c.dom.Document>));\

```

This expression sets a `consumer_id` local variable. If the payload is null, the `consumer_id` is set to null. Otherwise, we use the service name to perform consumer ID matching based on the `consumer.properties` entries. For more information on consumer ID matching, see “Configuring Consumer IDs” on page 434.

```

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 275.

Using Annotations for Custom Instrumentation

Applications that use version 1.5 or greater of the JVM can “force” the instrumentation of methods by simply using a custom annotation (`InstrumentationPoint`) that is contained in the `annotation.jar` file 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()
```

In this instance, ARGS includes the following (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

The following example shows code that uses the InstrumentationPoint annotation.

```
/*
 * (c) Copyright 2008 Hewlett-Packard Development Company, L.P.
 *-----
 */

import com.mercury.diagnostics.common.api.InstrumentationPoint;

...

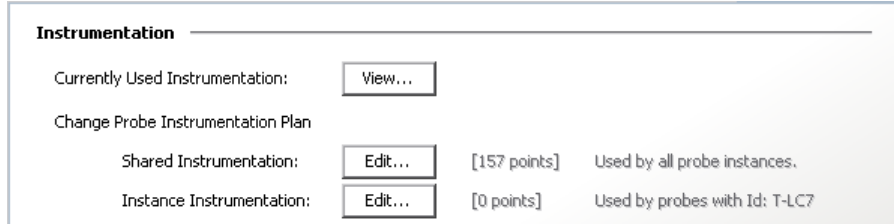
@InstrumentationPoint(layer="my_app",detail="diag,method-no-trim,method-cpu-time")
public void myMethod1(Object x, String y) {
    ...
}
```

In the example, myMethod1 will get instrumented and 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). The inclusive (including children) CPU consumption by this method will be measured and reported.

Maintaining Instrumentation from the Java Profiler UI

You can use the Configuration tab in the Java Diagnostics Profiler to maintain the instrumentation points and edit the probe configuration without having to manually edit the Java Agent capture points file or property files. You can access the Configuration tab from the Java Diagnostics Profiler whether profiling has been started or not.

The Instrumentation section of the Java Diagnostics Profiler gives you access to view and update the instrumentation for the application the probe is monitoring. The edit dialogs enable you to view and edit the instrumentation points as defined in the capture points file that Diagnostics uses to instrument your applications.



Reviewing the Current Instrumentation

To review the layers, classes, and methods that were 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:

Diagnostics			
Instrumented layers (no particular sorting)			
Layer	Hits	Active Points	Actions
(Other)	97539	1 / 1	[Disable] [Clear # Hits]
(keyword) http	78356	13 / 13	[Disable] [Clear # Hits]
(keyword) lwmd	1004363	2861 / 2861	[Disable] [Clear # Hits]
(keyword) remote-http	0	12 / 12	[Disable] [Clear # Hits]
(keyword) soap fault	0	1 / 1	[Disable] [Clear # Hits]
Business Tier/EJB/Entity Bean	436449	596 / 596	[Disable] [Clear # Hits]
Business Tier/EJB/Session Bean	48922	110 / 110	[Disable] [Clear # Hits]
Database/JDBC/Connection	93203	57 / 57	[Disable] [Clear # Hits]
Database/JDBC/Execute	45968	64 / 64	[Disable] [Clear # Hits]
Directory Service/JNDI	479	5 / 5	[Disable] [Clear # Hits]
HttpStatus	0	20 / 20	[Disable] [Clear # Hits]
Legacy/JCA/Connection	23075	1 / 1	[Disable] [Clear # Hits]
Legacy/JCA/ECIConnectionFactory	22918	2 / 2	[Disable] [Clear # Hits]
Legacy/JCA/ManagedConnectionFactory	20	2 / 2	[Disable] [Clear # Hits]
Messaging/JMS/Listener	0	1 / 1	[Disable] [Clear # Hits]
SOAPHandler	0	1 / 1	[Disable] [Clear # Hits]
Web Services	0	1 / 1	[Disable] [Clear # Hits]
Web Tier/Servlet	24073	23 / 23	[Disable] [Clear # Hits]
Web Tier/Struts	0	2 / 2	[Disable] [Clear # Hits]

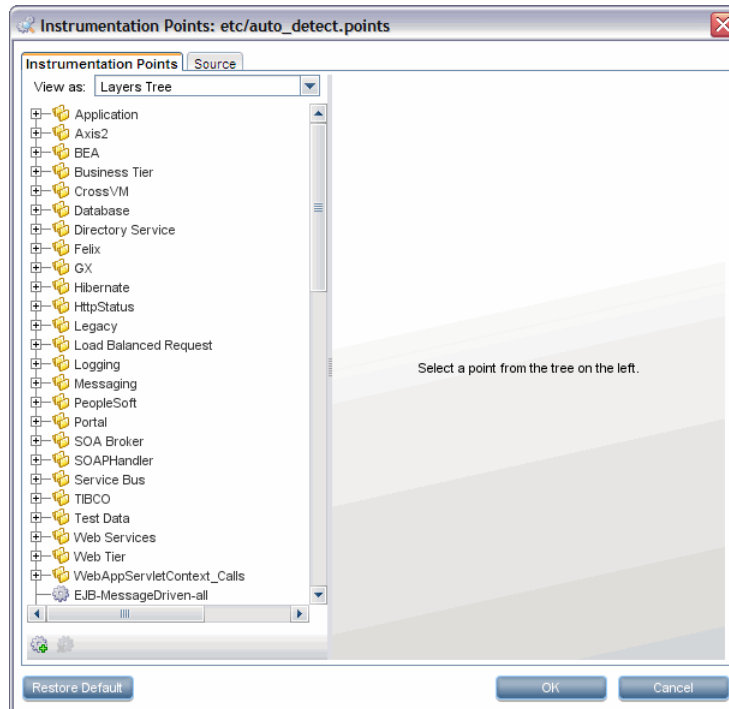
HP Diagnostics J2EE Probe "P81_WA55_Plants_ovrmtt152_W2k", version 9.00.70.1002

The Instrumented Layers page lists the layers that were instrumented, the number of times the instrumentation points in the layer were triggered, and the number of points currently active in the layer. The following columns are provided:

Column	Description
Layer	Lists the layers that were instrumented. The layer names in this column are links to a page that provides 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 were 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 were defined for the particular layer.
Actions	<p>Contains links that enable you to manipulate the information for the listed layers. The available action are:</p> <ul style="list-style-type: none"> ▶ 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 264. ▶ 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 that 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 opens:

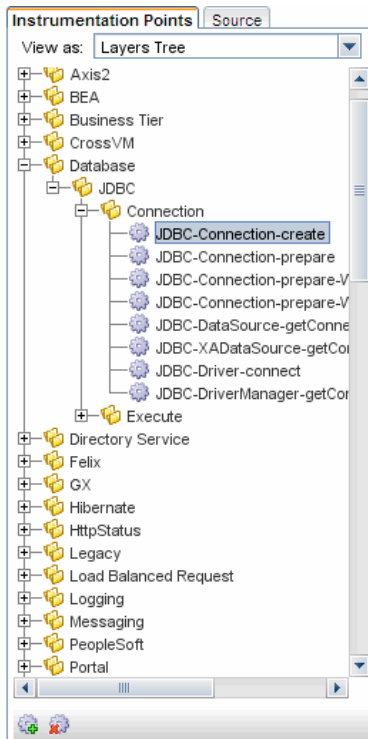


You can edit the instrumentation in two ways: visually, using a list or tree of points on the **Instrumentation Points** tab; or via the source of the capture points file on the **Source** tab.

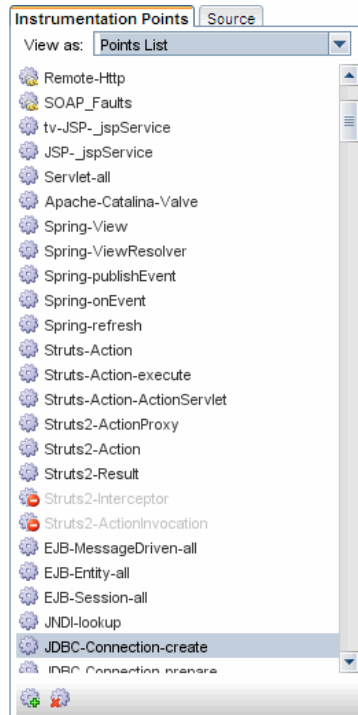
Selecting and Viewing an Existing Point

The navigation bar in the Instrumentation Points dialog helps 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 **Layers Tree** from the dropdown, you see a list of the points in the capture points file in a tree structure according to the layers and sublayers you assigned to the point:



When you select **Points List** from the dropdown, you see a list of the points in the capture points file in ascending alphabetical order:

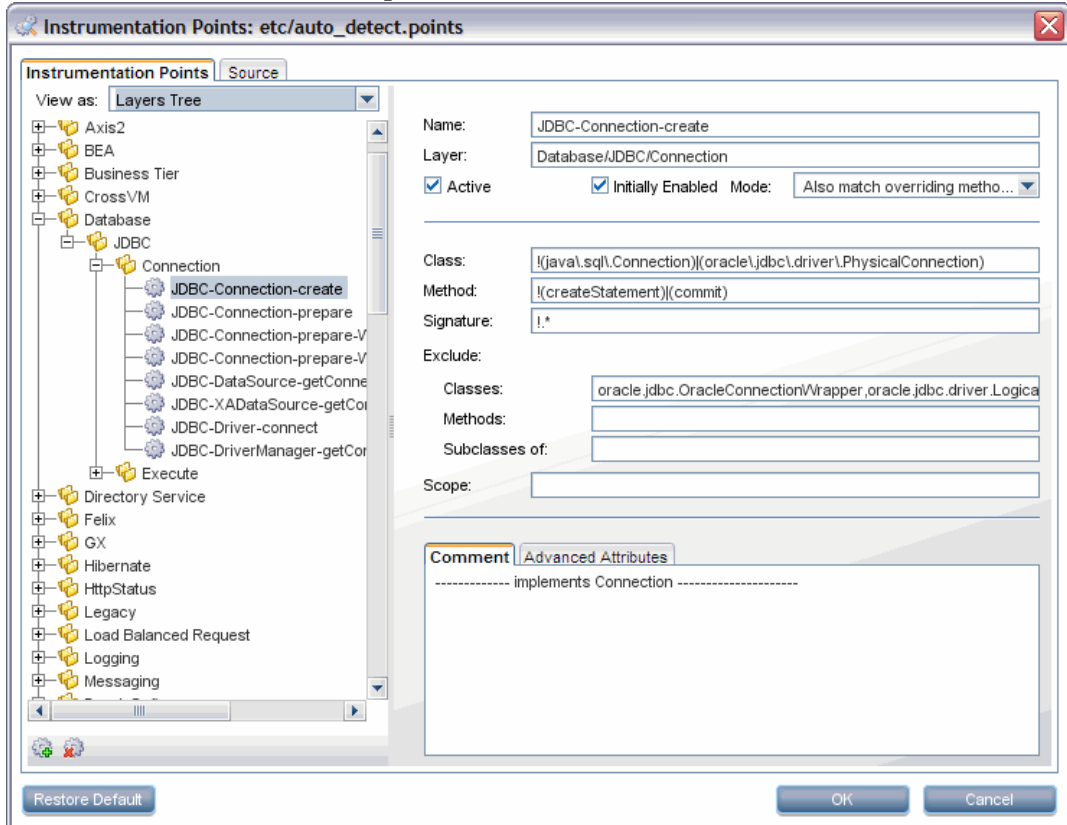


When you locate the point you want to view or maintain, select the point in the navigation bar. Then you see the details of the selected point in the view/edit panel where you can maintain the point.

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:



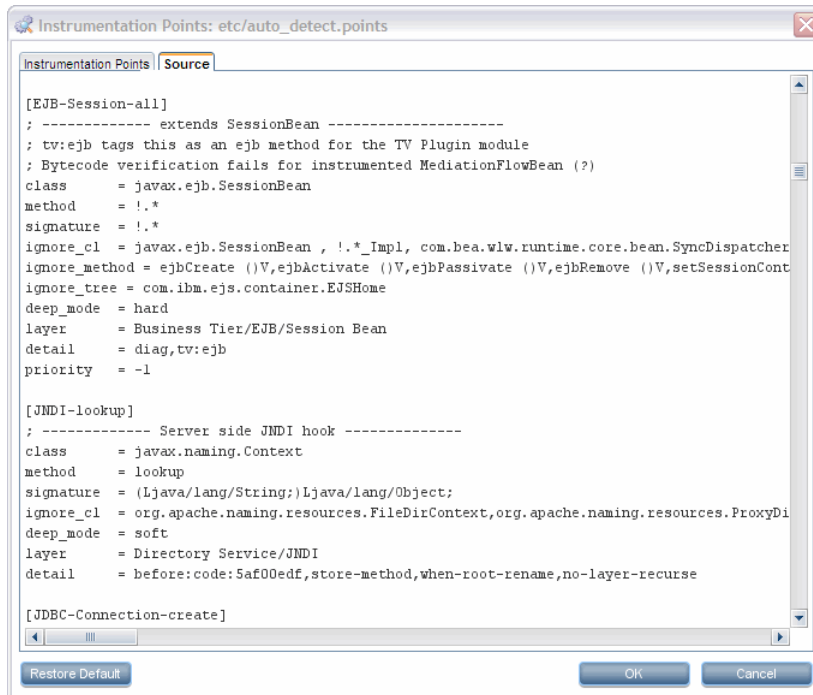
The arguments that are commonly used when defining a point in the capture points file are displayed as separate data fields to make it easier for you to make any necessary updates. More advanced arguments are displayed in the **Advanced Attributes** tab at the bottom of the display. Comments for the point are displayed in the **Comment** tab. After making changes click **OK**. And remember to apply all of the changes made using the Configuration tab by clicking Apply Changes.

The arguments that can be used to define a point in the capture points file are documented in “Coding Points in the Capture Points File” on page 264.

The following table cross-references the arguments displayed in the Instrumentation Points tab with the capture points file arguments documented in “Coding Points in the Capture Points File” on page 264. You might 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

The following is an example of the Source tab:



Deleting an Existing Point or Layer

You could 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**.

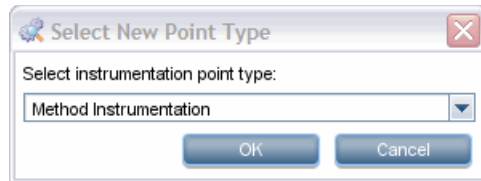
Adding a New Point

You could add a point to the instrumentation.

To add a point:



- 1 Click New Point. The Profiler displays the Select New Point Type dialog box:



- 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**.

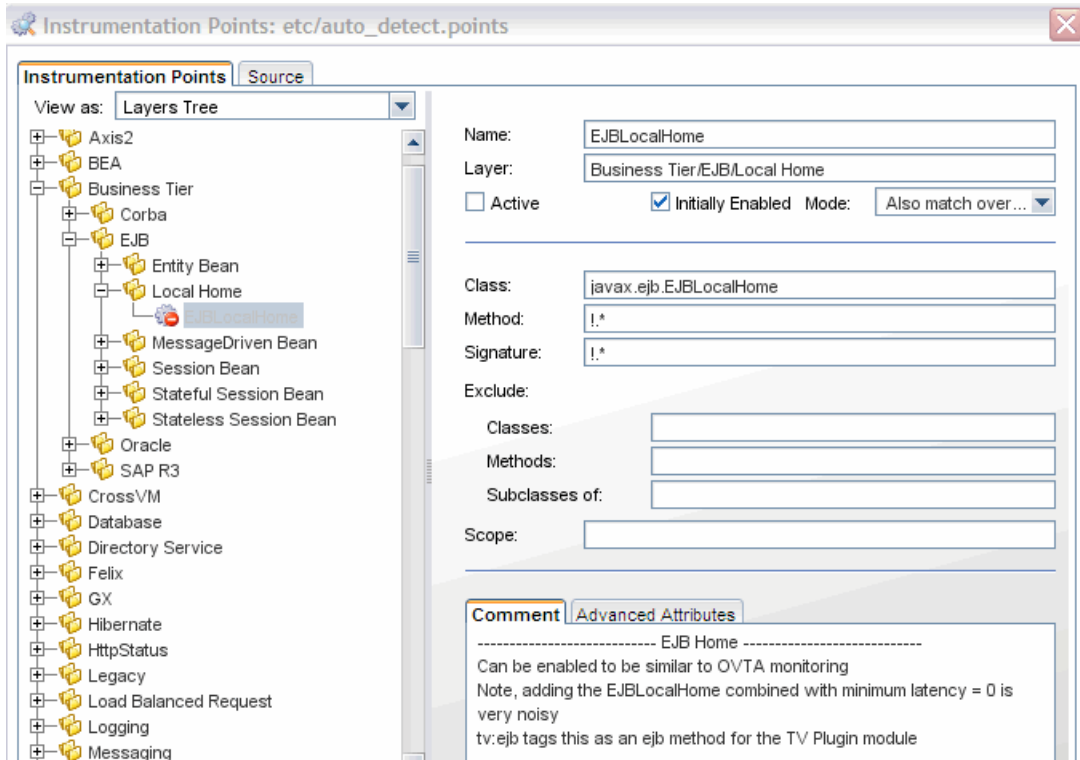
Activating OVTA-like Points

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 below. To use these points, set `active=true` in the **auto_detect.points** file 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 326 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:

- 1 Select the point from the Instrumentation Points navigation bar so that the Profiler displays the details for the point. Check the **active** check box.
- 2 Close the Instrumentation Points dialog by clicking **OK**.
- 3 Apply all the changes made using the Configuration tab by clicking **Apply Changes**. Restart your application server (which restarts the probe) for the newly activated points to take effect.

Restoring Default Points

When you finish diagnosing a problem using the Profiler or HP Diagnostics, you can 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**.

Default Layers Defined for Typical Java Classes and Methods

HP Diagnostics groups the performance metrics for classes and methods into *layers* and *sublayers* according to 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 could be contributing to performance issues.

The following table lists the default layers and sublayers that are defined for typical Java classes and methods.

Platform-specific layers are also defined in the capture points file. These layers are, for the most part, sublayers of the top-level parent layers defined in the following tables. You can see performance data for layers in the Load View in the Diagnostics UI.

Java EE Layers

Layer	sublayers	Parent Layer
Web Tier	JSP Servlets Struts Session Spring Struts2	
Business Tier	EJB Corba	
Web Services		
EJB	Entity Bean Session Bean Local Home Stateless Session Bean Stateful Session Bean MessageDriven Bean	Business Tier
Directory Service	JNDI	
Database	JDBC	
JDBC	Execute Connection	Database
Messaging	JMS Spring	
JMS	Producer Listener Consumer	Messaging
Spring	Producer Consumer	Messaging
Hibernate		

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 into layers for the portal lifecycle methods. For more information about portal layers, see the *HP Diagnostics User's Guide*.

10

Custom Instrumentation for .NET Applications

This section explains how to control the instrumentation that HP Diagnostics applies to the classes and methods of applications to enable the .NET Agent to gather the performance metrics.

This chapter includes:

- ▶ About Instrumentation and Capture Points Files on page 338
- ▶ Locating the .NET Capture Points Files on page 339
- ▶ Coding Points in the Capture Points File on page 340
- ▶ Instrumentation Examples on page 344
- ▶ Understanding the Overhead of Custom Instrumentation on page 369
- ▶ Default Layers for Typical .NET Applications on page 370

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, and catch exceptions; and to correlate method calls and threads. The instrumentation points for each probe are specified in the capture points file.

The capture points file enables you to control the scope of the instrumentation so that Diagnostics can give you all the information 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 "Using Regular Expressions" on page 782.

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 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 organize the performance metrics into meaningful tiers of information that can be compared as part of a triage process and control the collection behavior of the instrumentation.

The points in the capture points files are grouped into default layers. You can customize the default layers and create new layers (see "Default Layers for Typical .NET Applications" on page 370).

Locating the .NET 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** as well as other points files shown in the table below.

In addition, the .NET Agent installer automatically creates a separate capture points file for each IIS deployed ASP.NET Application Domain 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 in the `<probe_install_dir>\etc\<ApplicationDomain>.points` file. These points files and the default points files are read 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 installed in the `<probe_install_dir>/etc` directory 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 (MSMQ instrumentation)
Remoting.points	.NET Remoting
WebServices.points	ASP.NET Web Services

You can enable the points files by adding a reference to them in the `<points>` element in the scope of the appdomain in the **probe_config.xml** file. See Chapter 13, "Understanding the .NET Agent 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*.

Coding Points in the Capture Points File

The following arguments can be used to define a point in the points files:

```
[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>
detail     = <list of specifiers>
keyword    = <keyword>
layer      = <layer name>
layerType  = <layer type>
```

Caution: Do not modify any of the default points files because, in an installation upgrade, modifications are lost. Store your application-specific instrumentation points in a custom capture points file.

All arguments that can be specified as a regular expression list have an effective maximum limit of 260 characters, which if exceeded results in a truncated value. 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	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 can be used.
method	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 can be used.
layer	Specifies a layer, sublayer, 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 sublayers by separating the layer names with a / (slash). The layer specified following the slash is a sublayer of the layer specified before the slash. A sublayer can have its own sublayers by coding another slash and layer name following a sublayer 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 can 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 "Using Regular Expressions" on page 782.

Optional Point Entries

Point definitions can contain one or more of the following arguments:

Argument	Description
keyword	<p>Indicates special instrumentation. The keyword argument can be used to enable specific features; for example, the WCF keyword turns on the WCF feature. The keyword argument can also relate point definitions to special functionality; an example of this with the RemotingServer keyword is described in "Remoting" on page 359.</p> <ul style="list-style-type: none"> ▶ HttpCorrelation. Turns on correlation of client/server method calls via HTTP. ▶ HttpClientCorrelation. Turns on correlation of raw HTTP client request calls across both the .NET and Java technologies. ▶ 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. Associates points in a .NET Remoting server to special .NET Remoting logic for these points.
ignoreClass	<p>Specifies a comma-separated list of classes to ignore. Any class matching one of the classes specified with ignoreClass is not instrumented.</p>

Argument	Description
ignoreMethod	Specifies a comma-separated list of methods to ignore. Any method matching one of the methods specified with ignoreMethod is not instrumented.
ignoreTree	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	<p>Specifies how subclasses are handled. This argument accepts three values:</p> <ul style="list-style-type: none"> ▶ 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 subinterfaces 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 subinterfaces 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	Constrains the context in which instrumentation is performed. If specified, the inserted bytecode is caller side. Any valid regular expression can be used for the value of this argument. Scope values are expressed as a comma-separated list of method names.
ignoreScope	Excludes certain methods from those included in the scope specified by the scope argument. Any valid regular expression may be used for the value of this argument. ignoreScope values are expressed as a comma-separated list of method names.

Argument	Description
detail	<p>Provides more specific capture instructions. The string that is returned is displayed in the method's Argument field in the details pane of the Call Profile view. It is a comma-separated list of the following:</p> <p>args:n – Captures all supported types of arguments for the method(s) that match. A value of 'n' captures all arguments. Or you can enter a value for n from 1 through 256.</p> <p>args:0 – Calls the ToString() on the current class instance or callee object. This is invalid for static methods.</p> <p>*args:1 – Marks (*) the argument as a key argument for the server requests if the method is a top-level request.</p>
layerType	<p>Specifies special handling for some instrumented methods and accepts three values:</p> <ul style="list-style-type: none"> ▶ trended_method – Identifies methods to be displayed in the Trended Methods view. ▶ sql – Identifies methods used to capture SQL for the SQL views. These are set by HP Diagnostics and should not be modified.
signature	<p>Specifies the signature (return and parameter types); for example, System.String(System.int32, System.String). Any valid regular expression can be used.</p>

Instrumentation Examples

The following examples illustrate how you can customize the instrumentation of an application by creating and modifying the points in the capture points file.

This section includes:

- ▶ "Custom layer and sublayer" on page 345
- ▶ "Wildcard method" on page 345
- ▶ "Ignore Specified Methods" on page 346

- "Capture Methods for the Trended Methods View" on page 346
- "Capture Only a Specific Method In a Class" on page 346
- "Capture a Specific Method That Returns a String" on page 347
- "Caller Side Instrumentation" on page 348
- "HTTP Client" on page 349
- "Argument Capture" on page 349
- "Deep_mode Examples" on page 354
- "How to Configure and Set Up Points for Non-ASP.NET or Windows Applications" on page 355
- "Remoting" on page 359

Custom layer and sublayer

- The following point creates a custom sublayer 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 following point captures all methods in the MyCompany.MyFoo class:

```
[myCompany.myFoo_AllMethods]
class = myCompany.myFoo
method = !.*
layer = FOO/BAR
```

Ignore Specified Methods

- ▶ The following point 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 following point 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 following point 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 following point captures all non-static constructor methods for the MyCompany.MyFoo class:

```
[myCompany.myFoo_Constructor]
class = myCompany.myFoo
method = .ctor
layer = FOO/BAR
```

- The following point captures all static constructor methods for the `MyCompany.MyFoo` class:

```
[myCompany.myFoo_Singleton]
class = myCompany.myFoo
method = .cctor
layer = FOO/BAR
```

- The following point captures the `setFoo` method in the `MyCompany.MyFoo` class:

```
[myCompany.myFoo_setFoo]
class = myCompany.myFoo
method = setFoo
layer = FOO/BAR
```

- The following point captures all methods in the `MyCompany.MyFoo` class whose name includes “set”:

```
[myCompany.myFoo_AllSets]
class = myCompany.myFoo
method = !.*set.*
layer = FOO/BAR
```

- The following point 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 following point 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 the method.

Caller side instrumentation allows for finer control of instrumentation placement, but can increase the application initialization time because each class specified in the scope must be checked for references to the class/method 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 side instrumentation.

- The following point 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 following point 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
```

HTTP Client

The HTTP client instrumentation enables you to monitor and correlate raw (non-proxied) HTTP client request calls across both the .NET and Java technologies. By default this feature is disabled.

To enable this feature, configure the points file associated with the application you want to monitor to set **keyword = HttpClientCorrelation**.

For example, let's assume the name (either the appdomain or process name) of the .NET application you want to monitor is MyHttpClient. Also assume that the application implements a method named SendHttpRequest() which makes an HTTP client request call that is implemented in the My.Namespace.MyHttpClient class. In this example the MyHttpClient.points file would contain:

```
[HTTP Client Correlation]
keyword = HttpClientCorrelation

[MyHttpClient]
class = My.Namespace.MyHttpClient
method = SendHttpRequest
layer = MyHttpClient
```

If you are having problems with capturing data for any HTTP Client calls, there are logging settings that will log both Instrumentation and Capture side logs that are specific to HTTP Client:

```
<instrumentation><logging level="HTTP" /></instrumentation>
```

and

```
<logging level="HTTP" />
```

Argument Capture

The arguments to be captured are specified in the detail key of a points file section. For example:

```
detail=args:1,...args:4, *args:3
```

Calls the ToString() method of the n-th argument. The string that is returned is displayed in the method's Argument field in the Call Profile view.

There are several special values to note:

- ▶ args:n – Captures all supported types of arguments for the method(s) that match. A value of 'n' captures all arguments. Or you can enter a value for n from 1 through 256.
- ▶ args:0 – Calls the ToString() method on the current class instance or callee object.
- ▶ Adding a * to the args element (*args:1) marks a key argument.

To see the arguments for each method call, do not specify a key argument. This is a way to get more detailed information on the captured instance tree and could help answer questions about why this instance is a MAX tree or what values were passed in when there was an exception.

To group server requests for a method by arguments, specify a key argument. The key arguments, aggregate server requests with distinct values. Arguments that have a large number of distinct values are not good candidates for key arguments because this will lead to unique server requests for every distinct value.

Note: Even if you haven't specified argument capture, arguments are captured when a method in the call tree throws an exception. These arguments are displayed in the Call Profile view, in the Stack Trace section of the Exceptions detail pages. See the Call Profile View online help for more details.

The following argument capture example relates to the code shown below:

```
[ILTest]
class = !ILTest_NameSpace.ILTest_Class
method = methodWithParams
detail = args:0, *args:3, args:5, args:7
layer = myFunctionLayer
```

Here is the relevant code example:

```
class ILTest_Class
{
    public bool methodWithParams
    (string param1, int param2, string QnameParam3, long param4, object param5, int
    param6, double param7)
    {
        ... some implementation
    }
}

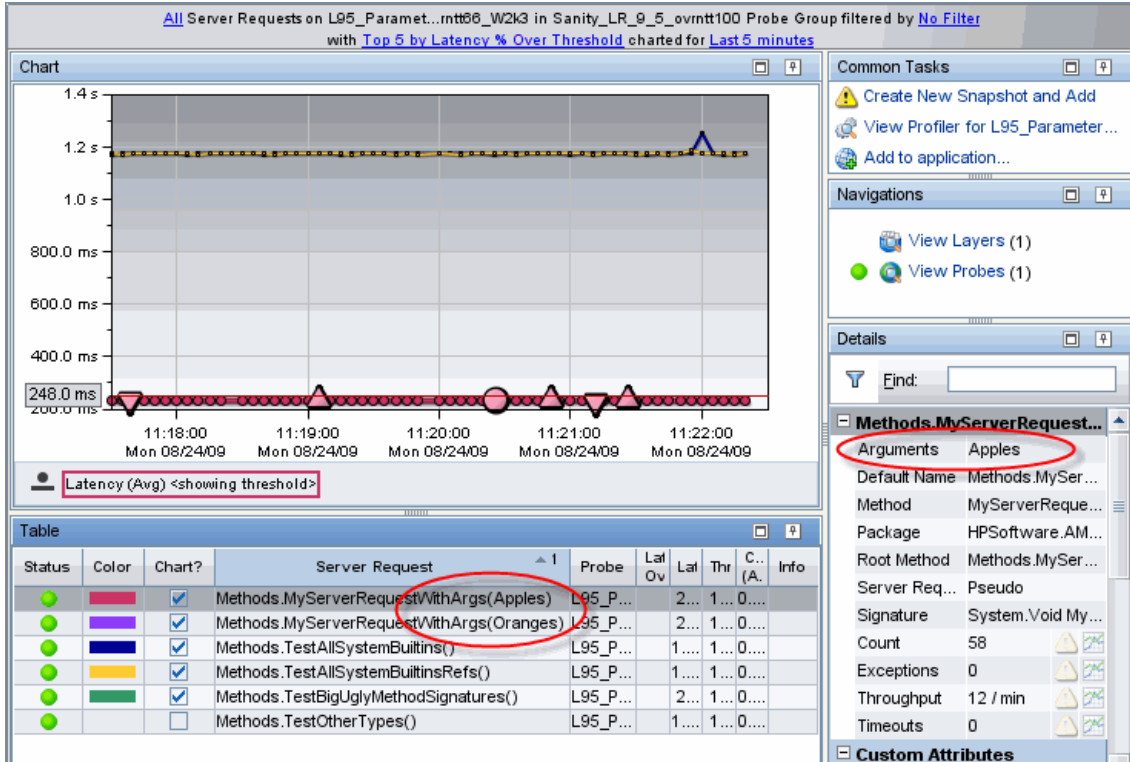
```

In this example the defined detail will capture ILTest_Class.ToString(args:0) param1, QnameParam3, param5 and param7.

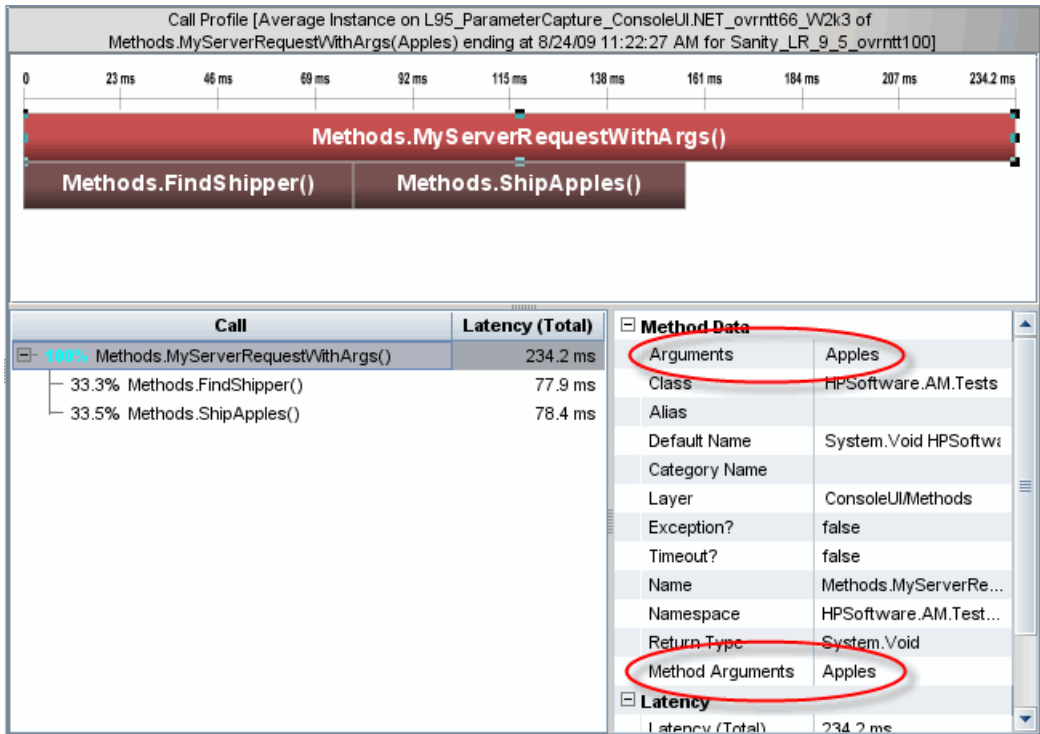
The value of QnameParam3 will be part of the identity of the sever request if the top level method is methodWithParams.

When an argument to be captured is marked as a key argument (with an asterisk *) and the method is a top-level method, the argument value becomes part of the Server Request identity.

For example, if Shipping Type is a parameter of a method processing different shipments and you specify the Shipping Type argument as a key argument, you will be able to see aggregated views for each different shipment (apples and oranges) being processed by the method.



When you specify a key argument, the Call Profile view shows key arguments in the Arguments field in the Details pane. You will also see the arguments displayed under Method Arguments in the Details pane.



When arguments to be captured are NOT marked as key arguments (with asterisk *), they are displayed in the Call Profile view under Method Arguments only.

Deep_mode Examples

The following interface definition is used for both soft and hard deep_mode examples:

```
public interface Interface1 {  
    public void callerMethod();  
}
```

The following class is 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 following point captures the callerMethod in the Class1 class:

```
[Training-1]  
class    = Interface1  
method  = !.*  
deep_mode = soft  
layer   = Training
```

- The following point captures all methods in Class 1; that is, 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 explains 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 and are referred to as such.

Windows Application Design

The critical point to consider when contemplating how to configure a Windows application 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. 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 simple Windows application illustrates 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 cread;

        public String readCl()
        {
            System.Console.WriteLine("Continue?");
            cread = Console.ReadLine();
            return cread;
        }
    }
}
```

The Hello_dotNet.exe Windows application has Main() that calls a method, waits for the user to enter something on the command line, and then exits. Until the application exits, the probe is active.

Creating the Hello_dotNet.points File

In the <probe_install_dir>\bin folder there is a **Reflector.exe** command line utility 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 522 for more information on the reflector utility.

When both the Reflector.exe and the Hello_dotNet.exe application are in the same folder, you would use the following command:

```
Reflector.exe Hello_dotNet.exe
```

The output is sent to stdout. Among other information you will see the following suggested Hello_dotNet.points:

```
-----  
Sample .points by Namespace  
-----
```

```
[Hello_dotNet_nameSpace]  
class = !Hello_dotNet_nameSpace.*  
layer = Hello_dotNet_nameSpace
```

The suggested points can be used as is, except when the Windows application has a method like Main(); that is, a method that, if instrumented, does not return an exit until the application exits. In this case, the method spans 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, 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 instruments other methods in Hello_dotNet_nameSpace.someclass if there are 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, 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.

```
<process name="Hello_dotNet">
  <points file="Hello_dotNet.points" />
  <instrumentation>
    <logging level="" />
  </instrumentation>
  <logging level="" />
</process>
```

Note: You must 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:

```
<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

Configuration

By default, the .NET probe is not enabled to instrument Remoting applications. You must add custom instrumentation points for both the Client and Server applications.

Two instrumentation keywords are related to Remoting:

Remoting. The Remoting keyword enables instrumentation for various points in the Remoting Framework.

RemotingServer. The RemotingServer keyword identifies 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 very simple Windows application example illustrates a number of crucial concepts that 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 "How to Configure and Set Up Points for Non-ASP.NET or Windows Applications" on page 355, you can use the Reflector utility to help determine how to configure the Remoting Client points file.

To configure the probe to instrument the SimpleConsoleClient Remoting Windows application, add the following XML to the **probe_config.xml** file:

```
<process name="SimpleConsoleClient">
  <points file="Remoting.points" />
  <points file="SimpleConsoleClient.points" />
  <instrumentation><logging level="" /></instrumentation>
  <logging level="" />
</process>
```

You must 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
```

These settings, however, would not create instrumentation that would produce any data. The reason, as discussed in "How to Configure and Set Up Points for Non-ASP.NET or Windows Applications" on page 355, is that you must ignore methods like Main(). If you factor in the need to ignore Main(), you would be left 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
```

Although these settings might be useful and would produce data, you should make them more precise. This is primarily due to probe performance. The more methods that are instrumented, the greater will be the probe's performance hit on the instrumented application. For example, if you can remove the wildcards "!.*" from the settings, 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

You 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 because it wraps the Remoting client call. The following settings are recommended 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 illustrates 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.

Here is the 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;
        }
    }
}
```

Here is the 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, add the following XML to the **probe_config.xml** file:

```
<process name="SimpleConsoleServer">
  <points file="SimpleConsoleServer.points" />
  <instrumentation><logging level="" /></instrumentation>
  <logging level="" />
</process>
```

It is not required to add the `<points file="Remoting.points" />` entry.

Point files for the Remoting Server can have one or more sections. The first section relates to the Remotable Object and is a required section. A second section that relates to the Remoting Server instrumentation can be added. Other optional sections can also be added to instrument other methods that can 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.

When you run the Reflector against the RemotableObjects.dll, you 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)
 __RaceSetServerIdentitySystem.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. You must be certain that you identified the class that implements the Remotable Object. You do this by observing that the Remotable Object is required to inherit from System.MarshalByRefObject and therefore 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. 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 you 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 you can construct the optional Remoting Server section. You only need to create this section if you want to monitor the Server logic that is invoked independent of the Remoting methods.

When you run the Reflector against the SimpleConsoleServer.exe, 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()
```

As explained in "How to Configure and Set Up Points for Non-ASP.NET or Windows Applications" on page 355, you cannot just use the suggested points settings. You must 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, you can add other optional sections to instrument other methods that can be called by either the Remoting methods or the Remoting Server.

Understanding the Overhead of Custom Instrumentation

When creating custom instrumentation, beware of over-instrumenting the application because that 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. 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 takes 100ms and instrumentation makes it execute in 101ms, overhead is 1%. If a method takes 10ms and instrumentation changes its response to 11ms, 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 that can 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 you instrument, the more data 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 14, "Advanced .NET Agent Configuration."

Default Layers for Typical .NET Applications

HP Diagnostics groups the performance metrics for classes and methods into *layers* and *sublayers* according to the instructions provided in the 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 could be contributing to performance issues.

The following table lists the default layers and sublayers that are defined for typical .NET applications.

.NET Layers

Layer	sublayers	Parent Layer
Web Tier	IIS	
IIS	ExecutionSteps	
Database	ADO	
ADO	Execute Connection Fill Update Cache	Database
Messaging	Sender Receiver	
Web Services	Soap Http WCF	
LWMD		
HTTP Client		
Outbound Calls		

Part V

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

This section includes:

- Advanced Diagnostics Server Configuration
- Advanced Java Agent and Application Server Configuration
- Advanced .NET Agent Configuration

11

Advanced Diagnostics Server Configuration

This section describes 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 component properties.

This chapter includes:

- ▶ Synchronizing Time Between Diagnostics Components on page 374
- ▶ Configuring the Diagnostics Server for a Large Installation on page 378
- ▶ Overriding the Default Diagnostics Server Host Name on page 383
- ▶ Changing the Default Diagnostics Server Port on page 383
- ▶ Configuring the Diagnostics Server for Multi-Homed Environments on page 384
- ▶ Reducing Diagnostics Server Memory Usage on page 388
- ▶ Configuring Fragment Name Based Trimming on page 388
- ▶ Preparing a High Availability Diagnostics Server on page 390
- ▶ Configuring Diagnostics for HP ServiceGuard (HA solution) on page 391
- ▶ Diagnostics Server Assignments (LoadRunner/Performance Center Runs) on page 393
- ▶ Configuring the Diagnostics Server for LoadRunner Offline Analysis File Size on page 394
- ▶ Configuring Business Service Management Samples Queue Size and Web Services CI Frequency on page 397

- ▶ Configuring Diagnostics Using the Diagnostics Server Configuration Pages on page 398
- ▶ Optimizing the Diagnostics Server in Production to Handle More Probes on page 398

Synchronizing Time Between Diagnostics Components

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 can be located.

The following sections describe how time synchronization works, and how to configure the components properly so that the time will be synchronized.

Probe collections running in VMware hosts have additional time synchronization requirements. See “Time Synchronization for Probes Running on VMware” on page 421.

Understanding Time Synchronization

Time synchronization in Diagnostics begins with the Diagnostics command server 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: Make sure 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 Service Management core server is to be used as the source of GMT time.

Note: If Business Service Management is configured to use Database time, you should also configure the Diagnostics command server to use this setting as the time source.

- **SERVER.** Indicates that the Diagnostics command server 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 did not yet synchronize with the **Time Source**, the Diagnostics Servers in Mediator mode are considered to be “unsynched.” The Diagnostics Servers in Mediator mode that are unsynched 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,” the probe connection is not allowed and is dropped.

Because the data is stored based on 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 is adjusted and saved to the synchronized GMT time of the Diagnostics Server in Commander mode. If the UI is running on a machine whose time was not synchronized properly with the **Time Source**, the data displayed in the UI appears 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 can synchronize the Diagnostics Server in Commanding mode by performing the following procedure.

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, the default configuration will work and no changes are necessary.

Because Business Service Management 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, you *must* do one of the following:

- ▶ Set up 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: Have backup NTP servers in case the primary NTP server is not available.

- ▶ If you are using Diagnostics in a Business Service Management or HP Software as a Service (SaaS) environment, you can set the **timemanager.time_source** property in `<diagnostics_server_install_dir>/etc/server.properties` to **BAC** to indicate Business Service Management. This causes the Diagnostics Server to connect to the registered Business Service Management core server to establish the time.

Note: To set up Business Service Management to use Diagnostics, see Chapter 21, “Setting Up the Integration Between Business Service Management and Diagnostics.”

- ▶ If the Diagnostics Server in Commander mode is to be used in Standalone mode, with no intention of using it with Business Service Management, and there is no internet connection allowing time synchronization with an NTP server, 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 is captured and persisted using the designated **Time Source**. Changing the **Time Source** after data is captured can result in a significant corruption to the data that was captured and persisted. This is because the data that was persisted might 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, the 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, the Diagnostics Server in Mediator mode could “hangs” for periods of time. If the heap is too large, the Diagnostics Server in Mediator mode could 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 Use the following table to determine the amount of heap the Diagnostics Server in Mediator mode will need:

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 GB, the heap size must 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 require significant network bandwidth between the probe and server, and significant CPU resources on the server.

If the network becomes a bottleneck—for example, network utilization above 25% on a mediator as observed in Windows task manager, or probes report less than 100% availability although they were up—you should reduce the data that is generated via trimming, to enable compression, if the probe system's CPU is not fully used. You can also 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` (for example, 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` (for example, can be increased to 101ms)

For more information on trimming, see “Configuring Fragment Name Based Trimming” on page 388 for the server, “Configuring Latency Trimming and Throttling” on page 527 and “Configuring Depth Trimming” on page 531 for .NET Agent, “Controlling Automatic Method Trimming on the Agent” on page 417 for Java Agent.

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) require additional CPU for compression.

Another way of decreasing network traffic is to change the frequency that data is pulled from the probe. By default, trends are pulled every 5 seconds and trees (Call Profiles) are pulled every 45 seconds. To lower the frequency for call trees, change **probe.trees.pull.interval** on the mediator in the **server.properties** file—for example, 90 seconds or 240 seconds depending on how many methods a Call Profile contains. First, lower the pull frequency of call trees. If this is not enough, lower the trend pull frequency by changing **probe.trends.pull.interval**—for example, 10 seconds.

Changing any of these parameters requires restarting the probe or server.

Additional Adjustments

If more than 50 probes are connected, 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.

You can also increase the number of threads available for jetty by setting **webserver.properties, jetty.threads.max=500**.

If call tree and trend files (see also Appendix E, “Diagnostics Data Management”) become too large (greater than 4 GB) in their uncompressed state, offload some of the probes to a new mediator. Otherwise, the aggregation and compression of the files could start to lag due to the large amount of data.

When many probes are connected to a server, the default purging setting of 5 GB might not be enough. For more information, see “Data Retention” on page 739.

Overriding the Default Diagnostics Server Host Name

When a firewall or NAT is in place, or the host for the Diagnostics Server in Mediator mode was configured as a multi-homed device, the Diagnostics Server in Commander mode might not be able to communicate with the Diagnostics Server in Mediator mode using the host name assigned when the Diagnostics Server in Mediator mode was installed. The **registered_hostname** property enables you to override the default host name 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 does not allow the default Diagnostics port to be used, 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 deploy Diagnostics, you must update the properties in the following table for each of the indicated components in your deployment with the new port number to ensure that the proper communications can take place.

Component Type	Properties
Diagnostics command server	<code><diagnostics_server_install_dir>/etc</code> <ul style="list-style-type: none"> ▶ <code>webserver.properties – jetty.port</code> ▶ <code>mediator.properties – commander.url</code> ▶ <code>probe/etc/dispatcher.properties – registrar.url</code>
Diagnostics mediator server	<code><diagnostics_server_install_dir>/etc</code> <ul style="list-style-type: none"> ▶ <code>mediator.properties – commander.url</code> <code><diagnostics_server_install_dir>/probe/etc</code> <ul style="list-style-type: none"> ▶ <code>dispatcher.properties – registrar.url</code>
Probes	<code><probe_install_dir>/etc</code> <ul style="list-style-type: none"> ▶ <code>dispatcher.properties – registrar.url</code>

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). 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 must 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 that 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. 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. Some systems could 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 383.

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.

```

<!-- 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 prevents 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 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 on which the Diagnostics Server in Commander mode is running 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 Service Management and the Diagnostics command server, or between Business Process Monitor and Business Service Management, the Diagnostics Server might not receive the ELT that lets it know when a transaction 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 the memory the transaction data is using.

The **correlation.txn.timeout** property sets the duration of the transaction timeout period. If you experience out-of-memory conditions in the Diagnostics Server, you could 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 could be sending data to the Diagnostics Server, and an active transaction could be idle in one Diagnostics Server. Setting the value of this property too low can cause transactions to be reported incorrectly. 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 used by the probes.

Note: Fragment name-based trimming is not intended to be used instead of the latency and depth trimming you configure for 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.properties**. 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 more than two fragments need to be trimmed, create additional **trim.fragment** properties. Make 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 high availability, you can create a standby Diagnostics Server for each Diagnostics Server. The standby is then ready to be used during a hardware failure or 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 downloads 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, 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 use the **bin/server.sh** or **bin\server.cmd** scripts. 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.
- 3** 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 390.

Note: When the failed Diagnostics Server host is recovered, do not make it the primary Diagnostics Server because it loses any data gathered from the probes while the new primary Diagnostics Server is being used.

Configuring Diagnostics for HP ServiceGuard (HA solution)

You can configure Diagnostics for HP ServiceGuard as a HA (High Availability) solution. This section outlines the necessary steps for configuring HP Diagnostics (Version 7.50 and higher) to run under HP Serviceguard (Linux).

Note: It is assumed that you are familiar with both, Diagnostics and HP Serviceguard.

The configuration steps described in this section can be used for other HA solutions as well (for example Microsoft Cluster Service).

The Diagnostics server should be installed on the shared disk with enough room for the Diagnostics time series database (TSDB) and other configuration items (for example user rights, custom dashboard screens, etc).

Both Diagnostics servers (active and standby) need to be time synchronized via NTP or Business Service Management. It is not recommended to use SYSTEM as the time synchronization mechanism since the "clock" used by the Diagnostics server needs to be the same on both servers.

The Diagnostics server uses the hostname as a prefix for sub-directories in the archive and storage directory. This needs to be overwritten on the Java command line that starts the server by specifying **-Dmediator.id=cluster -Dserver.id=<cluster>** (<cluster> can be replaced by any other unique name)

Example: <installdir>/bin/server.sh

```
$JAVA1_5_HOME/bin/java -Dserver.id=cluster -Dmediator.id=cluster -server  
-Xmx512m  
$SERVER_BCP $JAVA_OPTS -Dsun.net.client.defaultReadTimeout=70000  
-Dsun.net.client.defaultConnectTimeout=30000  
-classpath $SERVER_HOME/lib/mediator.jar$PATHSEP$SERVER_HOME/lib/  
loading.jar$PATHSEP$SERVER_HOME/lib/common.jar$PATHSEP$SERVER_HOME/  
lib/mercury_picocontainer-1.1.jar com.mercury.opal.mediator.util.DiagnosticsServer
```

Note: All command line components need to be on the same line.

The Serviceguard package requires start and stop commands for the application. The start command for Diagnostics is the <server_install_dir>/bin/server.sh script.

The stop command requires a new script that should reside in <server_install_dir>/bin as well, with the following content:


```
<installdir>/bin/stop.sh
```

```
#!/bin/sh
PID=`ps -ef | grep -v grep | grep DiagnosticsServer | awk '{ print $2 }'`
kill $PID
sleep 10
```

Note, make sure that the script has execute permissions (chmod u+x stop.sh).

In the Serviceguard package script, add the following lines:

```
stop_command:
  <installdir>/bin/stop.sh

start_command:
  <installdir>/bin/server.sh &
```

Note, replace <installdir> with the Diagnostics' server install directory and make sure that there is an ampersand (&) at the end of server.sh.

Diagnostics Server Assignments (LoadRunner/Performance Center Runs)

By default, a probe that is selected for a LoadRunner or Performance Center run uses the Diagnostics Server specified in its `<probe_install_dir>/etc/dynamic.properties`.

It is possible to override the configuration when the probe is started for a run. To do so, modify a mapping file on the Diagnostics Commanding Server. This enables 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 Service Management environment. You could have the probes use different Diagnostic Servers when they are in a LoadRunner / Performance Center run than when they are monitoring Business Service Management. Following the override instructions in this section.

It might be more convenient to use this mechanism than to edit the probe configuration file.

Note: When the probe is not in a run, it uses 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. Changes made to this file become effective without restarting 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 containing the Java data captured during the scenario. The size of this file can grow quite large. Make sure 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 ends.

Estimating the Size of the LoadRunner Offline File

Estimating the size of the offline file is highly dependent upon the data and 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 the 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 1 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 the load test by multiplying the 1 hour file size calculated in the previous step by the number of hours 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 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 makes the offline file smaller because fewer data points need to be stored when the aggregation period is longer. For example, increasing the offline aggregation period properties to **45s** reduces the file size by 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 to 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** Make sure 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 can take effect.
- 2** Access the Mediator Configuration Page 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, 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, 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 appears 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. The **Restart Mediator** button is also displayed.

For more information on updating property values from the Configuration Page and a screen image showing the command buttons, see “Making Server Configuration Changes” on page 654.

To cause the configuration changes to take effect, restart the Diagnostics Server in Mediator mode by clicking **Restart Mediator**.

Configuring Business Service Management Samples Queue Size and Web Services CI Frequency

The following configurations are applicable to Business Service Management integrations.

Configuring Business Service Management Samples Queue Size

Business Service Management Samples queue size, by default, is set to 100. When more than 100 samples are created at once, some of the samples are dropped, resulting in missing data in BAC for SOA. You can 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.

You could change the timing of the process that adds the Web Services CIs to the uCMDB. The Web Service CI population process has the following 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 enable you to set the property values that control how the Diagnostics Server communicates with the other Diagnostics components, and how it processes the data it receives from the probes.

Note: To ensure that you are entering valid property values, 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 Administration UI.”

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-minute interval, and the number of methods and layers in each server request. The following optimizations 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 45 seconds. If a single diagnostic server process is handling more than 25 probes, this could be optimized such that the trends and trees are pulled less often. A suggested optimal setting in production is a 30-second 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 512 MB for maximum heap size. Increase the maximum heap size according to 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 1 Gbps 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 of 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
```


12

Advanced Java Agent and Application Server Configuration

This section discusses advanced configuration of the Diagnostics Java Agent and the application server environment. Advanced configuration is for experienced users with in-depth knowledge of this product. Use caution when modifying any of the component properties.

This chapter includes:

- ▶ Advanced Configuration on page 402
- ▶ Controlling Which HP Software Products the Agent can Work With on page 403
- ▶ Configuring the Agent for Multiple Application Server JVM Instances on page 406
- ▶ Disabling the Java Diagnostics Profiler on page 412
- ▶ Specifying Probe Properties as Java System Properties on page 413
- ▶ Controlling Probe Logging on page 414
- ▶ Setting the Probe's Host Machine Name on page 415
- ▶ Controlling Automatic Method Trimming on the Agent on page 417
- ▶ Configuring an Agent for a Proxy Server on page 418
- ▶ Configuring Reverse HTTP for an Agent in SaaS on page 419
- ▶ Time Synchronization for Probes Running on VMware on page 421
- ▶ Limiting Exception Tree Data on page 422
- ▶ Diagnostics Probe Administration Page on page 425

- ▶ Authentication and Authorization for Java Diagnostics Profilers in Standalone Mode on page 428
- ▶ Configuring Collection of CPU Time Metrics on page 431
- ▶ Configuring Consumer IDs on page 434
- ▶ Configuring SOAP Fault Payload Data on page 444
- ▶ Configuring REST Services on page 445
- ▶ Customizing Grouping JMS Temporary Queue/Topics on page 445
- ▶ Configuring Display of Application Name for Server Requests on page 445
- ▶ Maintaining Probe Settings from the Java Profiler UI on page 446

Advanced Configuration

The following bullet points list the probe configuration sources of information to consult to configure your environment.

- ▶ If you have an agent that was installed to work with one HP Software product and you now would like to work with other products, see “Controlling Which HP Software Products the Agent can Work With” on page 403.
- ▶ If your application server is using multiple JVMs or you want to capture data for multiple JVMs, follow the steps described in “Configuring the Agent for Multiple Application Server JVM Instances” on page 406.
- ▶ If you have a probe that you want to prevent others from using in Profiler mode, see “Disabling the Java Diagnostics Profiler” on page 412.
- ▶ If you have more than one JVM using a single probe installation, you might need to set some of the probe properties in the Java system properties in the application start up commands, see “Specifying Probe Properties as Java System Properties” on page 413.
- ▶ To have log messages posted to the probe logs for lower level messages, adjust the log level as described in “Controlling Probe Logging” on page 414.

- ▶ If you have more than one agent installed on the same host, make sure the log messages for each agent are stored in a different file, as explained in “Changing the Log Directory for a Probe” on page 415.
- ▶ 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 Agent” on page 417.
- ▶ If there is a proxy between the agent and the Diagnostics command server, you must set the correct property to tell the agent the URL of the Diagnostics command server, see “Configuring an Agent for a Proxy Server” on page 418.
- ▶ If you installed a Java Agent in an HP Software as a Service (SaaS) environment, disable the reverse http (rhttp) communication between the agent and the Diagnostics mediator server, see “Configuring Reverse HTTP for an Agent in SaaS” on page 419.
- ▶ If you are running in a virtual environment, see “Time Synchronization for Probes Running on VMware” on page 421.
- ▶ If you need to limit the amount of exception data, see “Limiting Exception Tree Data” on page 422.
- ▶ If you want to use some of the collection options that require property file changes, see the other topics in this section such as “Configuring Consumer IDs” on page 434.

Controlling Which HP Software Products the Agent can Work With

The Java Agent can be configured to monitor applications from development through implementation and production. The Java Agent can be configured to work with several HP Software products or as a Diagnostics Profiler without other Diagnostics components or HP Software products.

The products with which the Java Agent can work are 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 you install the Java Agent. See Chapter 5, “Installing Java Agents.” To enable the agent to capture data for different products, set the value of the **active.products** property by editing the property file and restarting the application server.

Note: To use the Java Agent that was downloaded with the Diagnostics Profiler for Java with other HP Software products, contact HP Software Customer Support. The instructions that follow do not work without intervention from HP Software Customer Support.

To see Diagnostics data in the user interface of the interfacing HP Software products, you must perform additional configuration steps. See the chapters in this guide that correspond to the HP Software product you are using to complete the configuration.

The sections that follow provide instructions for configuring each product mode of the **active.products** property.

PRO Product Mode – Diagnostics Profiler for Java

When PRO mode is set, the agent gathers additional metrics and presents them on a user interface it makes available through a URL on the agent host.

If you are running the Java Agent as part of the Java Diagnostics Profiler in Development Mode, restrictions are placed on the agent to limit the load it can handle.

If you are running the Java Agent as part of Diagnostics enterprise, or along with another HP Software product, the Profiler is enabled without the load restrictions.

Enterprise Product Mode

When configured in Enterprise mode, the agent works with HP Software products such as Business Service Management, LoadRunner, Performance Center, AND as Diagnostics Standalone.

To configure the agent for Enterprise mode:

- 1 Set the **active.products** property to **Enterprise** and restart the Application Server.
- 2 Configure the agent to register with the Diagnostics command server.

One of the functions of the Diagnostics command server 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 agent 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 agent to connect to the Diagnostics mediator server.

The agent must be able to transmit the processing metrics it gathers to the Diagnostics mediator server in order for Business Service Management, 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 agent. In Business Service Management, you must tell the agent which Diagnostics Server to work with.

To configure the agent to communicate with the Diagnostics mediator server, set the host name and the port using the **mediator.host.name** and **mediator.port.number** properties. These are found in `<probe_install_dir>\etc\dynamic.properties`.

The default port for the Diagnostics mediator server is 2006. If, when installing the Diagnostics mediator server, you set the port for the Diagnostics mediator server to a value other than the default, use that same port number when you set the **mediator.port.number** property for the agent.

The following excerpt from the **dynamic.properties** file shows 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

You can set **active.products** to AM mode to protect an agent in a production Business Service Management deployment from accidentally being included in a LoadRunner or Performance Center run. In AM mode, the agent is not listed as an available agent in LoadRunner or Performance Center.

AD Product Mode

In AD mode the Java agent will only capture data during a LoadRunner or Performance Center run. When the agent is in this mode it will not use resources or send any data to the server unless the probe is part of a LoadRunner/Performance Center run. Use this mode to prevent an agent in a QA environment from using additional resources and continually report data to the Diagnostics console dataset when a load test is not running.

Configuring the Agent 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 agent configuration steps. You have two options. You can configure a probe entity for each JVM on a host, or you can configure one probe entity to be shared by all of the JVMs.

This section includes:

- “Configuring a Single Probe Entity for Multiple JVMs” on page 407
- “Configuring Separate Probe Entities For Each JVM” on page 411

Configuring a Single Probe Entity for Multiple JVMs

To allow multiple JVMs to share a single probe entity, 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 single probe entity to monitor multiple JVMs:

- 1 When a single probe entity is 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 running on the JVMs.

If you did not run the JRE Instrumenter for each of the JVM versions the probe will be working with, do so now. See “Running the JRE Instrumenter” on page 165.

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 ran the JRE Instrumenter multiple times, 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 is 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 Agent communicates using the mini web server. A separate port is assigned for 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, configure the firewall to allow communications using the ports in the range you specify. For more information, see Appendix , “Configuring Diagnostics to Work in a Firewall Environment.”

If you configure the firewall to allow probe communications on a range of ports that is different than the default, update the port range values discussed in the following bullets.

a Locate the **webservice.properties** file in the folder **<probe_install_dir>/javaprobe/etc**.

b Set the following properties to adjust the range of ports available for probe communications.

➤ The minimum port in the port number range uses the following property:

jetty.port=35000

➤ The maximum port in the port number range uses the following property:

jetty.max.port=35100

3 Assign a unique probe name using one of the following methods.

The command-line properties must be entered on one line, without any breaks. The probe ids defined on the Java command line override the probe names defined in the **probe.properties** file using the probe’s **id** property.

a Assign a custom probe Identifier to the probe for each JVM, using the Java command line or startup script.

-Dprobe.id=<Unique_Probe_Name>

The following example shows 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 example shows 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
```

- b** When a single Java parameter is specified but multiple probes are started, use the %0 string to generate a custom probe identifier for each probe—for example, in a clustered environment where a single startup script is used to start multiple probed application server instances.

-Dprobe.id=<probeName>%0

On Windows, use %%0. Use the first % to escape the second %.

The %0 is replaced dynamically with a number to create a unique probe name for each probe; for example, <probeName>0, <probeName>1, and so on.

- 4 Specify the points file the probe will use. By default, LoadRunner/Performance Center sets the `points.file.name` to `auto_detect.points`. You can specify that a custom points file be used when you must use more than one custom instrumentation plan, or where you have several JVMs on the same machine using a single probe, 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 Entities For Each JVM

When there are multiple JVMs on a single host, you can configure a separate probe for each JVM. To use a separate probe for each JVM, install the agent multiple times and define an instance of each probe by setting the probe's `id` property in the `probe.properties` file in each agent's installation directory.

To define an instance of each installed agent:

- 1 If you did not manually run the JRE Instrumenter for the JVMs the probe will be working with, do so now. To run the JRE Instrumenter, see “Running the JRE Instrumenter” on page 165.
- 2 Locate the `probe.properties` file in the `<probe_install_dir>/javaprobe/etc` directory.

Here is an example:

```
C:\\MercuryDiagnostics\\JAVAProbe\\etc\\probe.properties
```

- 3 Assign a name to the `id` property that is unique on the server and on the Diagnostics Server, as follows:

```
id=<uniqueProbeName>
```

When the probe 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 must spend time configuring each agent installation. A single probe for multiple JVMs can provide the same diagnostics detail.

Disabling the Java Diagnostics Profiler

You can disable the Diagnostics Profiler for Java on a Java Agent so that it cannot be accessed accidentally. 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 is disabled, change the value of the **disable.profiler** property from **true** to **false**.

Specifying Probe Properties as Java System Properties

All of the 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 property as a Java System property, pre-pend the letter **D** and the first part of the properties file name to the property name. The following examples explain this.

- ▶ To set the **id** property in **probe.properties** from the startup command, concatenate the **D** and **probe** from the property file name, and then tack on the name of the property you are specifying; that is, **id**, as follows:

```
-Dprobe.id=SomeId
```

- ▶ To set the **active.products** property in **probe.properties** from the startup command, concatenate the **D** and **probe** from the property file name, and then tack on the name of the property 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, concatenate the **D** and **dispatcher** from the property file name, and then tack on the name of the property 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 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**. This prevents almost all agent 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 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 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 an Agent for a Proxy Server” on page 418.

Setting the Probe’s Host Machine Name

The probe’s host name registers 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 agent host machine was configured as a multi-homed device, it might not be possible for the agent 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 without 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 enables you to manually set a host machine name for the probe and stop the probe from doing the automatic lookup.

To set a default host machine name for a probe, set the **registered_hostname** property (located in the property file `<probe_install_dir>/etc/dispatcher.properties`) to a machine name or IP address.

When you set the **registered_hostname** property, 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.

When you set the **registered_hostname** in a production environment where you are using Business Service Management or Diagnostics Standalone, the name you specify is shown as the host name in System Health.

Controlling Automatic Method Trimming on the Agent

Default configuration for the agent includes settings that control the trimming of methods. Trimming can be controlled according to 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 depth.

You could reduce the level of trimming, or turn off trimming completely. You can control 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 less interesting methods.

Note: In the following situations, latency is not trimmed when its latency is less than the trimming property:

- Methods that are the root for a call tree.
 - Methods that threw an exception.
-

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 might not be useful to specify this value in increments of less than 10 ms.

Controlling Depth Trimming

Methods that are called at a stack depth less than or equal to the value of the **maximum.stack.depth** property are captured. Those called at a stack depth greater than this limit are trimmed to avoid incurring overhead for less interesting methods.

Here is an example:

- If **maximum.stack.depth** is 3
- /login.do calls a() calls b() calls c()
- Only /login.do, a, and b are 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.

Configuring an Agent for a Proxy Server

Important! This section only applies if you are using the Java Agent with a Diagnostics Server.

Two properties are used to specify for the Java Agent, the URL of the Diagnostics command server. The property you set depends on whether or not there is a proxy.

- **registrar.url** in **dispatcher.properties**

The **registrar.url** property in `<probe_install_dir>\etc\dispatcher.properties` is set when you install the agent. When there is a direct connection between the agent and the URL of the Diagnostics command server, the agent uses the value of this property.

► **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 command server.

Configuring Reverse HTTP for an Agent in SaaS

When you install a Java Agent in a HP Software as a Service (SaaS) environment, you must disable the reverse http (rhttp) communication between the agent and the Diagnostics mediator server. This configuration prevents other customers in the SaaS environment from inadvertently getting the metrics from your applications.

The following instructions explain how to disable or enable rhttp after the agent is installed.

Disabling Reverse HTTP

If you did not disable rhttp when the agent was installed, 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 follows:

```
#####
## 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 disable rhttp for an agent, enable rhttp manually using the following instructions.

To enable rhttp, uncomment the **dispatcher.objects** property value in `<probe_install_dir>/etc/modules.properties` as follows:

```
#####
## 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 could 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 "Synchronizing Hosts and Virtual Machines with Real Time." VMware Tools must be installed in each VMware guest operating system that hosts a Diagnostics probe. The time synchronization option in VMWare Tools must be turned on.

This option in VMware Tools works only if the guest operating system time is initially set earlier than that of the VMware host. For instructions on how to install VMware Tools, see the "Basic System Administration" document for VMware ESX Server. 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.

If you encounter negative latency issues when running the probe on VMware guest with the probe property `attempt.vmware.timestamp.adjustments` set to true, you should set the following property in the probe **etc/capture.properties** file:

```
use.vmware.timestamp.workaround=true
```

When `use.vmware.timestamp.workaround` is set to true, the probe will use the alternative call to get the VMware host timestamps, so as to workaround the negative latency issue.

Limiting Exception Tree Data

The agent 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. For example, filtering application server-based errors such as **javax.naming.AuthenticationException** allow the exception trees to contain more application-specific errors.

The exception tree data collected is controlled by 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

You can limit the exception tree data collected by specifying the number of different types of exceptions by setting the **exception.instance.tree.count** property in **server.properties**. By default, this property is set to 4, which indicates that only the first four exceptions types encountered during the probe's data collection cycle are used in building the exception trees. You can raise or lower this setting.

Examples

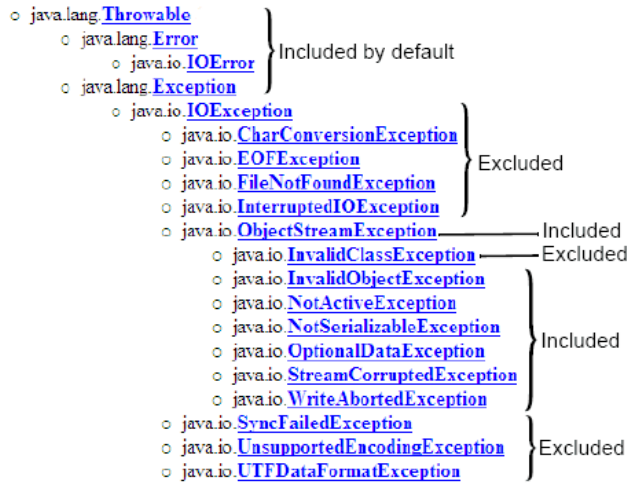
The following example causes exceptions of type `ClassNotFoundException` and all its subtypes to be ignored.

```
...  
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 Agent and Profiler settings. Access the Diagnostics Probe Administration page directly from your browser.

Accessing the Diagnostics Probe Administration Page

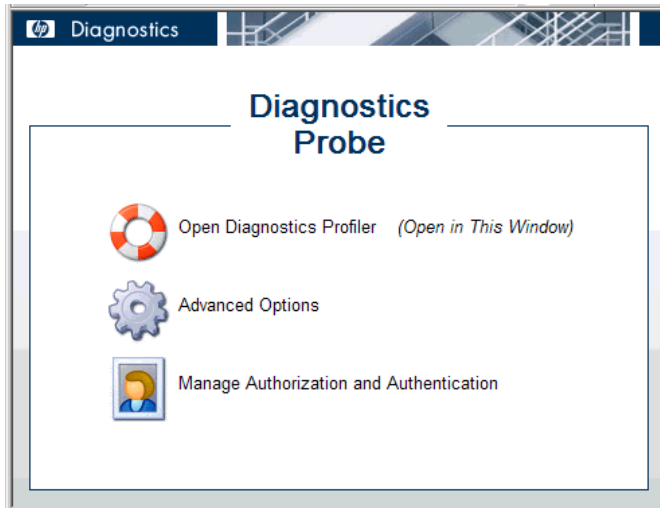
Open the Diagnostics Probe Administration page inside your browser.

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 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 427.
 - ▶ **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 command server to which this probe is connected. When you click this option, you are redirected to that Diagnostics command server. 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 428.

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 425.
- 2 Click **Advanced Options**.
- 3 If prompted, enter your user name and password.

The Components page opens.

Component Name	Component Description
query	Query API - allows you to download diagnostics data in HTML, XML or as Java objects
inst	Instrumentation Control
security	Built-In User Management
scheduler	See and control regularly scheduled background tasks
infrequentLogger	See the current status of entries in the infrequent logging table
files	Installation directory browser - upload and download property files, log files, etc

HP Diagnostics J2EE Probe "WLS91_MedRec_T155_W2k3", version 7.1.100.10

4 Click one of the following options:

- **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 261.
- **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 command server to which this probe is connected. When you click this option, you are redirected to that Diagnostics command server. For more information, see “User Authentication and Authorization” on page 657.

- ▶ 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 428.
- ▶ **scheduler.** Enables 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 Agent as a Profiler only (not connected to any Diagnostics Server), you can manage the authentication and authorization of users of the Profiler from the Diagnostics Probe User Administration page.

Note: If the Java Agent is configured to work with a Diagnostics Server, the probe (Profiler) authorization and authentication settings are managed from the Diagnostics command server to which this probe is connected. For more information, see “User Authentication and Authorization” on page 657.

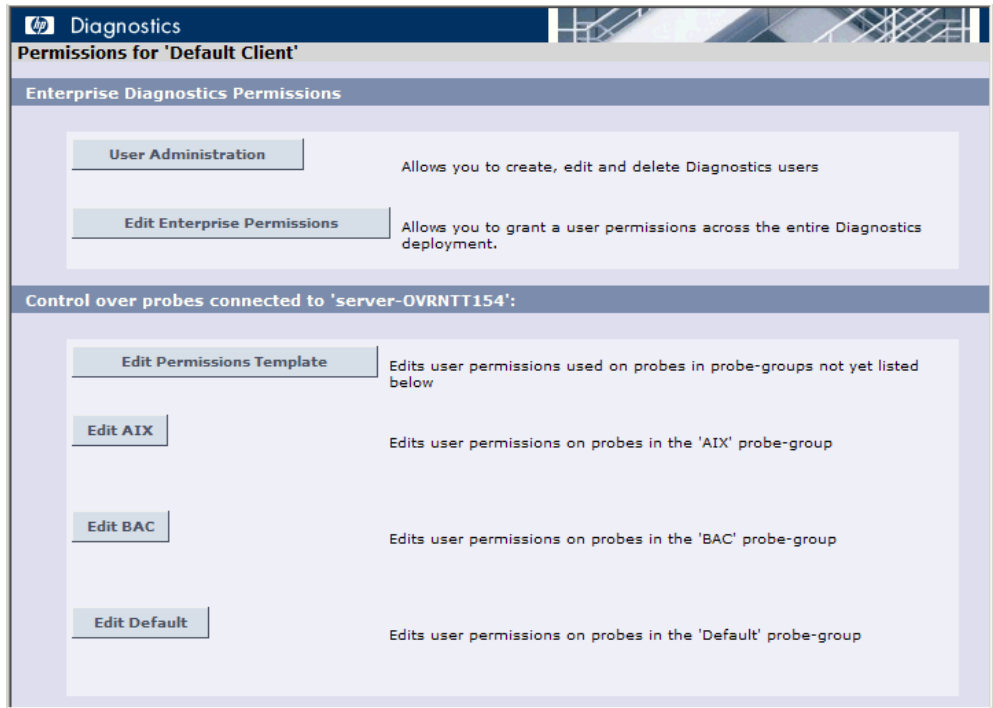
To manage authentication and authorization for users of the standalone Java Diagnostics Profiler:

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 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 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, AIX, HP-UX and Linux kernels 2.6.10 or later (for example RedHat 5.x, SUSE 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, see 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 “Time Synchronization for Probes Running on VMware” on page 421.

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. You can configure collection of the following CPU Time metrics:

- Server Requests only
- Server Requests and Portlet Methods
- Server Requests and All Methods

For a Java Agent, 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. Collection of any CPU timestamps is controlled by a second property listed below. If you set the `use.cpu.timestamps` property to `false`, the CPU time metrics are not 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 collected only for server requests.

The default value is **1**, which means CPU times can be reported at the server request level but not the transaction level. However, if the setting is removed or commented out of the properties file, the default is 0.
- **2** – CPU timestamps collected for All server requests and ALL methods.
- **3** – CPU timestamps 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 dropdown list.

CPU Timestamp Collection Method	Description
None	No CPU Timestamps.
For Server Requests Only	CPU timestamps are only collected for server requests.
For Server Requests and Portlet Methods	CPU timestamps are collected for ALL server requests and the lifecycle methods instrumented for portal components.
For Server Requests and All Methods	CPU timestamps are collected for ALL server requests and ALL methods.

- 3** When you complete your changes, click **Apply Changes**.

Note: Your changes take effect immediately. You do not need to restart the application (or 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 “Loading the Diagnostics SOAP Message Handler” on page 221 for details.

The Diagnostics SOAP message handler is not available for all application servers. Custom instrumentation is not available to capture SOAP faults or consumer IDs from SOAP payloads. Therefore, this feature is not available on all versions of all application servers. For the most recent information on Diagnostics SOAP message handler support, see 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 Service Management. Business Service Management users can look at the performance of the same application based on consumers to compare their performance characteristics.

Note: When 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, IP address is used as consumer ID for SOAP over HTTP/S web services and inbound queue name (or topic name) is used by default as consumer ID for SOAP over JMS web services.

This section includes:

- ▶ “Basic Procedure for Consumer ID Configuration” on page 435
- ▶ “About Consumer ID Rules” on page 437
- ▶ “Consumer ID Rules Syntax and Examples for Java Agents” on page 438

Basic Procedure for Consumer ID Configuration

The basic procedure to configure consumer IDs is as follows:

- 1** (Optional). Specify ***dump-payload** in the **consumer.properties** file to print the entire SOAP payload out to the **consumer.log** file. Use this output to plan how to create the specific rules to configure consumer IDs.

Before you configure consumer IDs, familiarize yourself with the SOAP payload data to determine how best to create the specific rules Diagnostics will use to find the value for consumer IDs.

The dump-payload option should only be used when help is required to locate the element that contains the Consumer Id.

This option should be the only value on the right side of the equal(=)sign when used: `DumpTest;HTTP_WS;TraderService = *dump-payload`

Important: Do not try to use the same service name to extract a value AND dump the payload at the same time.

For example, to use this feature, enter:

```
SoapTest1;HTTP_WS;TraderService = *dump-payload
```

This results in printing the SOAP Payload for a rule that matches TraderService. The content of the consumer.log file is:

```
2009-01-15 14:42:13,653 INFO consumer [[ACTIVE] ExecuteThread: '0' for queue:
'weblogic.kernel.Default (self-tuning)'] [PAYLOAD:] <?xml version="1.0" encoding="UTF-8"
standalone="yes"?><soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:trad="http://
www.bea.com/examples/Trader" xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/">
  <soapenv:Header>
    <CallerA>customerA</CallerA>
  </soapenv:Header>
  <soapenv:Body>
    <trad:buy soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <string xsi:type="xsd:string">hpq</string>
      <intVal xsi:type="xsd:int">11</intVal>
    </trad:buy>
  </soapenv:Body>
</soapenv:Envelope>
```

- 2 For each Java Agent you want metrics grouped by consumer, update the **consumer.properties** file as described in “Consumer ID Rules Syntax and Examples for Java Agents” on page 438.
- 3 To track more than five consumer types, update the **max.tracked.ids.per.probe** setting in the **dispatcher.properties** file.
- 4 Review the **<probe_name>_id.properties** file located in the **probe/files/log** directory. The **<probe_name>_id.properties** file might need to be completely deleted or modified to match the **consumer.properties** changes made in the previous steps. The file goes together with the **max.tracked.ids.per.probe** (dispatcher.properties) setting, once the limit is reached, per probe, all other consumers are 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 not applied according to how 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. 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.

All rule types do not need to be used. There might be SOAP rules, no HTTP rules, and IP rules. If there is no match on any of these rules, 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. See “Using Regular Expressions” on page 782 for help using regular expressions.

If there is a match, the probe attempts to find the text element also specified in the rule. If the text element is not found in the SOAP header, this rule is skipped and the probe goes 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.

Consumer ID Rules Syntax and Examples for Java Agents

The assignment of consumer IDs is controlled by specifying rules in the `consumer.properties` file.

Important: ALL configuration items are case sensitive. 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
```

When the callers of the TraderService Web service have a value defined for CallerA, the metrics are grouped by the different values for CallerA. The following excerpt from the soap header maps 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>
```

By default, Diagnostics looks for CallerA in the first-level element (the element directly under the SOAP env:Header). You can configure Diagnostics to look into a deeper-level xml element for consumer ID. The dynamic property **max.search.level.depth** in the **consumer.properties** file controls the depth at which to search for consumer ID (default value is 1 level deep). For example, max.search.level.depth = 2 would find consumer ID:

```
<env:Header>
  <test:id>
    <test:CallerA>consumerA</test:CallerA>
  </test:id>
</env:Header>
```

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
```

When the callers of the Web service have a value defined for User-Agent, the metrics are grouped by the different values for User-Agent. The following excerpt from the HTTP header maps 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; and 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 the following format:

```
<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 range> = <consumerID-string>
```

where <IP address range> can be defined with integers, wildcards specified with *, 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.

Manual configuration is required for WebSphere 5.1 JAX-RPC and Oracle 10g JAX-RPC. See “Loading the Diagnostics SOAP Message Handler” on page 221 for details.

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. Therefore, this feature is not available on all versions of all application servers. For the most recent information on Diagnostics SOAP message handler support, see the Diagnostics Product Availability Matrix at http://support.openview.hp.com/sc/support_matrices.jsp.

For a Java Agent, 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).

Customizing 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` according to 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`.

Configuring Display of Application Name for Server Requests

With 9.0 the Deployed Into value displayed in the Diagnostics UI in the Server Requests details pane can show the Application Name of the server request for most application servers. Prior to 9.0 this information was only available for WebLogic application servers so only a WebLogic probe could fill in the Application Name identifier on a server request.

To ensure backward compatibility with the server request trend lines, the Application Name is not filled in for the server request, except in WebLogic server requests.

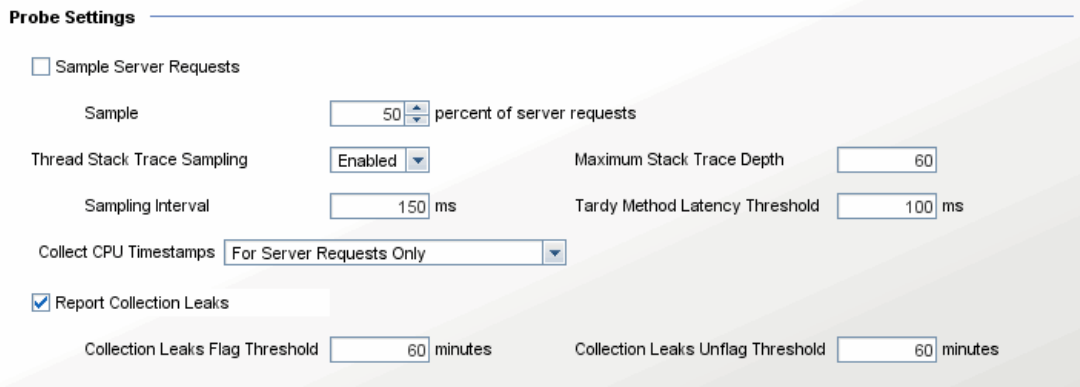
This is configurable using the **keep.fragment.data.compatible** property in the **capture.properties** file. By default `keep.fragment.data.compatible=true` which means the Application Name is not filled in for the server requests, except in WebLogic server requests.

You can set this property to **false** if you want the Diagnostics UI to show the J2EE application server name of each server request (shown as Deployed Into in the details pane of the Server Requests view).

Maintaining Probe Settings from the Java Profiler UI

You can use the Configuration tab in the Java Diagnostics Profiler to maintain the instrumentation points and edit the probe configuration without having to manually edit the Java Agent capture points file or property files. You can access the Configuration tab from the Java Diagnostics Profiler whether profiling has been started or not.

The Probe Settings section of the Java Diagnostics Profiler Configuration tab enables you to configure probe settings for server request sampling, thread stack trace sampling, collection of CPU time metrics (using timestamping) and reporting collection leaks.



Probe Settings

Sample Server Requests

Sample percent of server requests

Thread Stack Trace Sampling Maximum Stack Trace Depth

Sampling Interval ms Tardy Method Latency Threshold ms

Collect CPU Timestamps

Report Collection Leaks

Collection Leaks Flag Threshold minutes Collection Leaks Unflag Threshold minutes

Controlling Server Request Sampling

Server request sampling allows Diagnostics to gather detailed information for a subset of all 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. You can control server request sampling from the Java Profiler Configuration tab as described below or you can use the following properties 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.
- 3** When you finish making changes to the Configuration tab, click **Apply Changes**.

Your changes take effect immediately. There is no need to restart the application (or probe).

To disable server request sampling from the Profiler:

- 1** Uncheck the **Sample Server Requests** check box.
- 2** When you finish making changes to the Configuration tab, click **Apply Changes**.

Your changes take effect immediately. There is no need to restart the application (or 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.

Several properties enable and configure thread stack trace sampling.

The following properties are in **dynamic.properties**:

- ▶ **enable.stack.trace.sampling** – enables asynchronous thread stack trace sampling; possible values are false, auto (the default), and true.

When the dynamic property `enable.stack.trace.sampling` is set to auto, stack trace sampling is enabled IF the probe is running on selected (certified) platforms and JVMs. For other JVMs, the setting must be set to true explicitly. Use caution because the JVM could generate errors or abort. See the Diagnostics Readme.

- ▶ **tardy.method.latency.threshold** – the minimum time that an instrumented method must run without hitting any instrumentation points before stack trace sampling is 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** – the time that must elapse before the next consecutive sampling attempt is made.

Small values for **stack.trace.sampling.rate** cause frequent sampling and provide 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 can go up as well, but this effect is not as profound as the latency increase. For systems with many CPUs, the process CPU consumption can actually go down (not a good thing).

- ▶ **stack.trace.depth.max** – the limit for the depth of stack traces obtained from the JVM. You will most likely not need to adjust this value.

The following properties are in **dispatcher.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 is proof that the nodes must not represent a single method invocation. When set to true, it could decrease the number of additional call tree nodes created, but could create a false impression that the number of calls to the additional nodes is known and is small. When set to false, it creates a node for each method and each stack trace it was visible on, 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 are reported. This setting controls noise elimination and memory footprint, especially on the server side.

All of the properties can be dynamically changed so no restart of the application is required.

You can change the first four properties (from **dynamic.properties**) remotely, using the Configuration tab in the Diagnostics Java Profiler. After making changes remember to apply all of the changes made using the Configuration tab by clicking **Apply Changes**.

Thread Stack Trace Sampling	<input type="button" value="Enabled"/> ▾	Maximum Stack Trace Depth	<input type="text" value="60"/>
Sampling Interval	<input type="text" value="150"/> ms	Tardy Method Latency Threshold	<input type="text" value="100"/> ms

Example Thread Sampling Configurations

Use Case 1: A particular method has average latency of about 170 milliseconds, but from time to time it takes 1.4 seconds for this method to complete. Most of the methods visible in Call Profiles for any fragment execute in 550 milliseconds or less. Because the method in question makes multiple calls to its callees, you do not want to instrument them.

Instead you enable stack trace sampling to find out what the cause for long execution times. To minimize overhead, 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, this should theoretically give up to eight 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.

If you examine the Call Profile for long running instances of the server request, you 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 agent 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 200 milliseconds. 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 three or more consecutive stack traces, or methods with estimated latency of 600 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: Why do I not see any new nodes in my Call Profile after I enabled stack trace sampling.

Answer: Go through this checklist and see if any of the following applies to your case:

- Are you using Java 1.5 or newer? Stack trace sampling does not work for earlier versions of Java.
- Was the last method visible in the Call Profile 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).
- Was the last method visible in the Call Profile 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.)
- Did you try reducing `tardy.method.latency.threshold` 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 is never an inactive period of `tardy.method.latency.threshold` for the caller.
- Did you try reducing `aggregated.stack.trace.validity.threshold` or check if there are warnings in the `probe.log` file about the stack depth being too shallow? Possibly, the observed stack traces changed too quickly to get reported.
- Did you try reducing the `stack.trace.sampling.rate`? Perhaps your methods simply miss the opportunities to get sampled.
- Did you verify that the latency of the last visible method in Call Profile is not caused by having run garbage collector? Java code, including the stack trace sampling code, does not run during garbage collection.

Question 2: What is the minimum value of `stack.trace.sampling.rate` that can be used?

Answer: You can use any positive value, but remember that each platform will refuse to sample more frequently than it possibly can. The three determining factors 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` that can be used?

Answer: There is no limit. The usefulness of a high setting depends entirely on the latency of the server requests for the application. To get any results, plan for at least a few samples for each server request you are concerned with. Even that could require tuning other sampling parameters as well.

Controlling CPU Timestamp Collection

The CPU timestamps calculate the amount of exclusive CPU time that a method uses. You can view this information 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 “Time Synchronization for Probes Running on VMware” on page 421.

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 431) 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 dropdown list.

CPU Timestamp Collection Method	Description
None	No CPU Timestamps.
For Server Requests Only	CPU timestamps are only collected for server requests.
For Server Requests and Portlet Methods	CPU timestamps are collected for ALL server requests and the lifecycle methods instrumented for portal components (layertype=portlet).
For Server Requests and All Methods	CPU timestamps are collected for ALL server requests and ALL methods.

- 3** When you finish making changes to the Configuration tab, click **Apply Changes**.

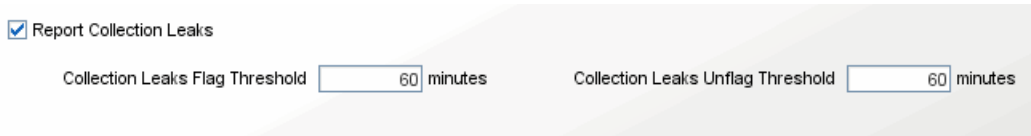
Note: Your changes take effect immediately. There is no need to restart the application (or probe).

Enabling and Configuring Collection Leak Reporting

Data collection and reporting for collection leak pinpointing is enabled by default for the probe. The [**Collection Leak Pinpointing**] keyword = **clp** point in the probe's **etc/auto_detect.points** file is set to true by default.

Note: You must initially run the 9.0 JRE Instrumenter if you want to use the collection leaks pinpointing (CLP) feature in the Java Agent.

You can set the following configuration items for collection leak reporting using the Java Profiler Configuration tab:



Report Collection Leaks

Collection Leaks Flag Threshold minutes

Collection Leaks Unflag Threshold minutes

- ▶ **Report Collection Leaks.** You can disable reporting in the UI for this feature by unchecking the checkbox.
- ▶ **Collection Leaks Flag Threshold.** The threshold of time duration in which the collection has size growth. If a collection's size growth period exceeds this threshold, it will be flagged as a memory leak by the probe.
- ▶ **Collection Leaks Unflag Threshold.** For an already flagged leaking collection, if its size stops growing continually for this threshold time period, the probe will unflag it as a leak.

These same values can also be set in the **dynamic.properties** file for the probe: **clp.diagnostics.reporting**, **clp.diagnostics.growth.time** and **clp.diagnostics.nongrowth.time**.

Applying Changes

When you click **Apply Changes** on the Java Diagnostics Profiler Configuration tab, all the updates 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.

13

Understanding the .NET Agent Configuration File

You control the configuration of the .NET Agent by modifying the elements and attributes in the .NET Agent configuration file: `<probe_install_dir>/etc/probe_config.xml`.

This chapter includes:

- Understanding the .NET Agent Configuration File on page 455

Understanding the .NET Agent Configuration File

The topics in this section describe the elements and attributes that make up the .NET Agent configuration file `<probe_install_dir>/etc/probe_config.xml`.

Each element is defined by describing its purpose, attributes, and parent and children elements. For information on additional .NET Agent 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. If no AppDomain elements are defined for a process then all application domains for that process will be included.

Attributes

Attributes	Valid Values	Default	Description
enabled	true false	true	Determines if the AppDomain should be instrumented. Is overridden by enableallappdomains attribute of a process element.
name	string	none	Name of the .NET AppDomain.
website	string	none	The name of the Website for those appdomains that are Websites.

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"/>

<appdomain enabled="false" name="1" website="Default Web Site">
  <points file="1-Default Web Site.points"/>
  <id probeid="Default Web Site" />
</appdomain>
```


<authentication> element**Purpose**

List of authenticated user names and passwords.

Attributes

Attributes	Valid Values	Default	Description
username	string	admin	User name account.
password	string	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 authenticate="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	string	none	User name to validate with the Diagnostics Server.
password	string	none	Password to validate with the Diagnostics Server.
authenticate	true, false	true	Enables and disables authentication.

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	none

Example

```
<credentials username="test" password="diag" authenticate="true"/>
```

<demomode> element**Purpose**

This configures demo mode. Demo mode makes it easier to show capability and value of the .NET agent because it requires less custom points to be defined. With demo mode turned on, all outbound calls will be shown irrespective of any other instrumentation. It is recommended under production environments that this be turned off and calls leading to the outbound calls of interest be "custom" instrumented. This will ensure that call stacks leading to the outbound calls are apparent.

Attributes

Attributes	Valid Values	Default	Description
enabled	true, false	false	Enables or disables demo mode.

Elements

Number of Occurrences	Zero or one.
Parent Elements	probeconfig
Child Elements	none

Example

```
<demomode enabled="false"/>
```

<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	Proxy user account.
proxypassword	password for proxy	none	Proxy user account's password.
registered_host 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.
timeskewcheck interval	number	60	Number of seconds to wait for getting the time skew from the Diagnostics server.

Elements

Number of Occurrences	1 per parent
Parent Elements	probeconfig
Child Elements	none

Example

This is a general example showing the setting for the <diagnosticsserver> element. The question marks (?) indicate that appropriate values need to be substituted.

```
<diagnosticsserver url="http://localhost:2006/commander" delay="2"
keepalive="15" proxy="?" proxyuser="?" proxypassword="?"
registerhostname="?" register_byip="false"/>
```

For the steps involved in using the registered_hostname attribute to override the default probe host machine name see "Overriding the Default Probe Host Machine Name" on page 539.

<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.

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 from the Collections by Growth and Collections by Size tables in the .NET Profiler's Collections tab and the Diagnostics UI's Collections view.

The specified collection classes may include classes that implement **ICollection**. Note that this setting does not affect the instrumentation of LWMD points; it only affects the presentation of the LWMD data and the amount of LWMD data that is sent to the Diagnostics Server.

Attributes

None

Elements

Number of Occurrences	Zero to many
Parent Elements	lwmd
Child Elements	None

Example

```
<lwmd enabled="true" sample="15s" autobaseline="1h" growth="10" size="10">
  <exclude>System.Collections.ArrayList</exclude>
  <exclude>System.Data.DataView</exclude>
</lwmd>
```

Note that System.Data.DataView implements System.Collections.ICollection.

<excludeassembly> element**Purpose**

Excludes the instrumentation of an assembly. An assembly is an .exe or .dll file. Provides the ability to exclude sensitive assemblies from instrumentation (for example, when a product was used to obfuscate and encrypt code in sensitive assemblies and exceptions would be thrown if instrumented).

Add <excludeassembly name=<AssemblyNameToExclude> as a child to a process element.

Attributes

Attributes	Valid Values	Default	Description
name	string	none	Name of assembly to exclude (without the file extension).

Elements

Number of Occurrences	zero to many
Parent Elements	process
Child Elements	none

Example

```
<process enablealldomains="true" name="ASP.NET">
  <logging level="" />
  <points file="ASP.NET.points" />
  <points file="ADO.points" />
  <points file="WCF.points" />

  <excludeassembly name="Acme.Encryption" />

  <appdomain enabled="false" name="TestWebService">
    <points file=" TestWebService .points" />
  </appdomain>
</process>
```

<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"/>
```

<httpclient> element**Purpose**

This configures whether the URL will be included as part of an HTTP outbound call's identity. The default is true and should be kept so unless there are many distinct URLs for the outbound HTTP calls. This could potentially overwhelm the performance of the Diagnostics Server because of the number outbound calls created (one for each distinct URL). You may also want to turn it off if you do not care about the URL of the HTTP outbound call. The identity of the HTTP outbound call will then be the Server and port number to which the request is being made to.

Attributes

Attributes	Valid Values	Default	Description
showurl	true, false	true	Enables/disables the inclusion of the URL as part of the identity of an outbound call made by a client using HTTP. Setting to false can be used to protect against symbol table explosion on the server/agent side if there are too many distinct http client calls.

Elements

Number of Occurrences	Zero to one.
Parent Elements	probeconfig, process, appdomain
Child Elements	none

Example

```
<httpclient showurl="true"/>
```


<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	String containing: Letters, digits, underscore, dash, period and internally defined \$() variable values: \$(APPDOMAIN), \$(MACHINENAME), \$(WEBSITENAME), \$(PID)	\$(APPDOMAIN).NET	The name of the probe as recognized by LoadRunner / Performance Center and System Health.
probegroup	string	Default	Defines the grouping recognized by the Diagnostics Server for reporting of system metrics and probe metrics.

Elements

Number of Occurrences	1 per parent
Parent Elements	probeconfig, process, appdomain
Child Elements	none

Example

Default setting example.

```
<id probeid="$(APPDOMAIN).NET" probegroup="Default"/>
```

Example

Example for a probe running in a LoadRunner 8.1 environment reporting to "myDiagServer" with the probe's name comprised of valid characters, the name of the Web site the application is deployed under, plus the name of the machine the application is deployed on.

```
<id probeid="LR_81_$(WEBSITENAME)_$(MACHINENAME).NET"  
probegroup="LR_81_myDiagServer"/>
```

<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="property lwmd" />  
</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.

Elements

Number of Occurrences	1
Parent Elements	trim
Child Elements	none

Example

```
<trim>
  <latency enabled="true" throttle="true" min="2" max="100" increment="2"
  incrementthreshold="75" decrementthreshold="50"/>
</trim>
```

<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 .NET Agent instrumentation processing.

Attributes

Attributes	Valid Values	Default	Description
level	off assert break severe warning info debug points eh sig chi cil classmap ilasm symbols deepmode load all checksum property remoting lwmd http	"" which is equivalent to "info"	Level of logging.
threadids	true false	true	Should thread IDs be included in the log.

Note: Valid Values below "info" should typically not be used. These are diagnostic settings that can produce extremely large log files.

Elements

Number of Occurrences	zero to many
Parent Elements	instrumentation
Child Elements	none

Example

```
<instrumentation>  
  <logging level="property lwmd" />  
</instrumentation>
```

<logging> element (when parent is appdomain, probeconfig, or process)

Purpose

Sets the logging level for the .NET Agent 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 tvdebug	"" which is equivalent to "info"	

Note: Valid Values below "info" should typically not be used. These are Diagnostic settings that can 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 (LWMD) feature.

Attributes

Attributes	Valid Values	Default	Description
enabled	true false	false	Enables sampling for lwmd capturing.
sample	string	1m	Sample interval (h-hour/m-minute/s-second).
autobaseline	string	1h	Auto baseline interval.
manualbaseline	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.
metrichost	string		The host to which metric data is sent.
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 mediator server. Used only when ipaddress is specified.
localportend	number	5000	End of port range to use for tcp event channel connection to the Diagnostics mediator server. 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"/>
```

<metrics> element**Purpose**

This element contains all of the <metric> elements.

Attributes

None

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, process
Child Elements	metric

Example

```
<metrics>
  <metric name="% Time in GC" group="Memory" units="percent"
category=".NET CLR Memory" counter="% Time in GC"/>
</metrics>
```

<metric> element

Purpose

Specifies additional probe metrics that you want the Diagnostics .NET to collect from perfmon. See "Collecting Additional Probe Metrics or Modifying Probe Metrics" on page 549 for additional information.

Attributes

Attributes	Valid Values	Default	Description
name	string		Name of the metric as you would like to see it in the Diagnostics UI.
group	string		Group (Category) of the metric as you would like to see it in the Diagnostics UI.
units	microseconds, milliseconds, seconds, minutes, hours, days, bytes, kilobytes, megabytes, gigabytes, count, percent, fraction_percent, load, status		Units of measure for the perfmon metric.
category	string		The performance counter category as specified in perfmon.
counter	string		The performance counter as specified in perfmon

Note: The instance of the counter is automatically assigned as the process instance for the counter or application domain instance for ASP.NET application counters. Counters that do not have process or application domain instances are not collected; you should define system metrics instead.

Elements

Number of Occurrences	1 or more per parent
Parent Elements	metrics
Child Elements	none

Example

```
<metrics>
  <metric name="% Time in GC" group="Memory" units="percent"
category=".NET CLR Memory" counter="% Time in GC"/>
</metrics>
```

<modes> element

Purpose

Specifies which mode(s) the .NET Agent should run in.

See "Controlling Which HP Software Products the Agent can Work With" on page 524 for more information about using the different modes.

Attributes

Attributes	Valid Values	Default	Description
enterprise	true false	Depends on mode chosen in installation. <ul style="list-style-type: none"> ▶ true if pro is false ▶ false if pro is true 	Sets agent to run in enterprise mode (probe is working with Diagnostics Server). Enterprise mode is like a combination of <i>ad</i> , <i>am</i> and <i>pro</i> mode. It will capture data for LoadRunner runs as well as data outside of LoadRunner runs.
ent	true false	Depends on mode chosen in installation. true if pro is false false if pro is true	This is a short form of the enterprise attribute.
ad	true false	false	<i>ad</i> mode supersedes all other modes. If <i>ad</i> mode and any other modes are set, then mode will be set to <i>ad</i> . In <i>ad</i> mode the .NET Agent will only capture runs from LoadRunner and put the results in a specific database for that run (for example, Default21).

Attributes	Valid Values	Default	Description
am	true false	► false	<i>am</i> mode supersedes all other modes except for <i>ad</i> . In <i>am</i> mode the .NET agent will ignore runs. If LoadRunner is executing an application then you will see the data in the normal Diagnostics database.
pro	true false	Depends on mode chosen in installation. ► true if enterprise is false ► false if enterprise is true	Sets the agent to run in Profiler mode. This mode sends data to the profiler. This mode can be combined with other modes.
tv	true false	false	Enables the capture of TransactionVision events. See the <i>HP TransactionVision Deployment Guide</i> for how to configure the required transport settings. This mode will send events to TransactionVision. This mode can be combined with other modes.

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	none

Example

```
<modes enterprise="false" ad="false" am="false" pro="true"/>
```

<points> element

Purpose

Specifies the capture points file to use for 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 .NET Agent configuration.

Attributes

None.

Elements

Number of Occurrences	1
Parent Elements	None
Child Elements	appdomain, bufferpool, captureexceptions, consumeridrules, credentials, diagnosticserver, eventserverhost, id, instrumentation, ipaddress, logging, lwmd, mediator, modes, points, process, profiler, rum, sample, soappayload, trim, webserver, topology, vmware, xvm

Example

```
<probeconfig>
</probeconfig>
```

<process> element

Purpose

Provides an inclusion filter list of which processes will be monitored.

If no process elements are defined then no processes will be monitored.

Attributes

Attributes	Valid Values	Default	Description
enablealldomains	true false	true	When set to true the enable attribute on all appdomains that are part of the process is overridden so that all will be enabled.
name	string	none	Name of the .NET process that these setting apply to.

These are the rules for the enablealldomains attribute of the <process> element:

- enablealldomains = false : If there are no domains in the list of <appdomain> No domains should be enabled.
- enablealldomains = false : If there are domains in the list of <appdomain> Domains should be enabled if the "enable" attribute is set to true or not defined in the enable attribute of the <appdomain>.
- enablealldomains = true : If there are domains in the list of <appdomain> Only Domains in the list should be enabled disregarding their "enable" attribute.
- enablealldomains = true : If there are no domains in the list of <appdomain> All domains should be enabled.
- enablealldomains attribute is not defined: same as if enablealldomains = true.

Elements

Number of Occurrences	zero or more
Parent Elements	probeconfig
Child Elements	appdomain, bufferpool, credentials, diagnosticserver, mediator, id, instrumentation, ipaddress, logging, lwmd, modes, points, profiler, sample, trim, webserver, filter, symbols, topology

Example

```
<process enablealldomains="true" name="ASP.NET">
```

<profiler> element

Purpose

Contains settings for the Profiler feature.

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.
inactivitytimeout	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/ RemoteDeskTop into the machine and run the Profiler locally.

Elements

Number of Occurrences	1 per parent
Parent Elements	appdomain, probeconfig, process
Child Elements	authentication

Example

```
<profiler authenticate="true" register="false" samples="60" best="1" worst="3"
inactivitytimeout="10m">
  <authentication username="admin" password="admin"/>
</profiler>
```

<rum> element**Purpose**

Controls the settings 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

Example

```
<rum enabled="true" responseheader="X-HP-CAM-COLOR"
encryptedkey="OBF:3pe941vx43903wre40303xxz3q6r42ob43n93wre3io03xjs4
0h940pc3wir3q233jur3zir3yi03zir3vc03wre3xpi3r8o3olr44na3zor3v6m3vc03zir4
4u03ohb3rdi3xjs3wx03v6m3zor3yc63zor3jqz3q6r3wd740vi40b53xpi3ike3wx04
3gp42ur3q233y3r3zwy3wx0432i42293p9p"/>
```

To create 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: <ul style="list-style-type: none"> ▶ 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"/>
```


<soapcapture> element**Purpose**

Configures whether SOAP requests and responses are captured.

Attributes

Attributes	Valid Values	Default	Description
enabled	true false	true	Enables or disables the capture of SOAP requests and responses. If this is disabled it will affect the following: <ul style="list-style-type: none"> ➤ SOAP request capture for SOAP faults ➤ SOAP requests and responses capture for TV mode ➤ ConsumerID assigned via the SOAP rules.
size	number	0	This is an optional attribute that specifies the maximum size in characters of the SOAP request or response captured. 0 indicates unlimited.

Elements

Number of Occurrences	one per parent
Parent Elements	probeconfig
Child Elements	none

Example

```
<soapcapture enabled="true" maxsize="0" />
```

<soappayload> element

Purpose

This element is deprecated and replaced by <soaprequestforsoapfault>.

Configures the SOAP payload capture on SOAP faults feature which 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 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" />
```

<soaprequestforsoapfault> element**Purpose**

Configures the SOAP request capture on SOAP Faults feature. The SOAP request capture on SOAP Faults feature provides the SOAP request associated with a SOAP fault.

NOTE: If the soapcapture setting is disabled it will override this setting. Please refer to the documentation for the <soapcapture> element.

Attributes

Attributes	Valid Values	Default	Description
enabled	true false	true	Enables or disables the SOAP request capture for SOAP fault feature.
size	number	5000	This is an optional attribute that specifies the maximum size in characters of SOAP request 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

```
<soaprequestforsoapfault 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 be 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 whether topology information will be collected and sent to the Diagnostics server.

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

```
<trim>  
</trim>
```


<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
attempttimestampadjustments	true false	false	Enables time stamp adjustments in VMware environments.
useworkaround	true false	false	If you encounter negative latency issues when running the .NET Agent on a VMware guest with the attempttimestampadjustments attribute set to true you should set this attribute to true. When this attribute is set to true the .NET Agent will use an alternative call to get the VMware host timestamps to workaround the negative latency issue.

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	IP address		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

```
<xvm><ws>ws</xvm>
```

<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

```
<xvm></xvm>
```

14

Advanced .NET Agent Configuration

Instructions are provided for advanced configuration of the .NET Agent. 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 .NET Agents Running on VMware on page 518
- ▶ Customizing the Instrumentation for ASP.NET Applications on page 518
- ▶ Discovering the Classes and Methods in an Application on page 522
- ▶ Controlling Which HP Software Products the Agent can Work With on page 524
- ▶ Configuring Support for MSMQ Based Communication on page 526
- ▶ Configuring Latency Trimming and Throttling on page 527
- ▶ Configuring Depth Trimming on page 531
- ▶ Configuring the .NET Agent for Light-Weight Memory Diagnostics on page 533
- ▶ Limiting Exception Stack Trace Data on page 536
- ▶ Disabling Logging on page 538
- ▶ Overriding the Default Probe Host Machine Name on page 539
- ▶ Listing the Probes Running on a Host on page 540
- ▶ Authentication and Authorization for .NET Profilers in Standalone Mode on page 541
- ▶ Configuring Consumer IDs on page 543

- ▶ Configuring SOAP Fault Data on page 548
- ▶ Collecting Additional Probe Metrics or Modifying Probe Metrics on page 549

Time Synchronization for .NET Agents Running on VMware

.NET Agents running in VMware hosts have additional time synchronization requirements. For agents 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 could 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, see 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 Agent is installed, the **ASP.NET.points** file is created with the standard instrumentation that the agent 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 want to monitor, you can use the Reflector tool that was installed with the .NET Agent 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 522 for instructions on using Reflector.

To let the .NET Agent 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 agent captures custom business logic for the applications.

The following example illustrates how to modify the capture points file so that the agent 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 .NET Agent 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 were 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 253.

Discovering the Classes and Methods in an Application

To monitor the performance of an application that you are not familiar with, use the Reflector automatic discovery tool that is installed with the .NET Agent to find the classes and methods in the application that you want to add to the instrumentation used by a probe. 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 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 518.

Controlling Which HP Software Products the Agent can Work With

The .NET Agent can be configured to monitor applications from development through implementation and production. The .NET Agent can be configured to work with several HP Software products or as a Diagnostics Profiler without other Diagnostics components or HP Software products.

To see Diagnostics data in the user interface of the interfacing HP Software products, you must perform additional configuration steps. See the chapters in this guide that correspond to the HP Software product you are using to complete the configuration.

The sections that follow provide instructions for configuring each product mode. The **<modes>** element is configured in the **probe_config.xml** file on the .NET Agent system (see "**<modes>** element" on page 494).

PRO Mode - Diagnostics Profiler for .NET

When PRO mode is set the agent gathers additional metrics and presents them in the Diagnostics Profiler for .NET user interface which is made available through a URL on the agent host.

In this mode the profiler is always collecting data even when the profiler UI is not in use. This mode can be combined with other modes. If this mode is not set when in AD or AM mode then profiler data will not be available.

AD Mode

In AD mode the .NET agent will only capture data during runs from LoadRunner/Performance Center and the results will be stored in a specific Diagnostics database for that run, for example, Default Client:21.

When the agent is in this mode it will not use resources or send any data to the server unless the probe is part of a LoadRunner/Performance Center run.

AD mode supersedes all other modes. So for example, if AD mode and any other modes are set then the mode will be set to AD.

See Chapter 23, "Setting Up LoadRunner and Diagnostics Integration" for how to set up LoadRunner integration or see Chapter 24, "Setting Up Performance Center to Use Diagnostics" for how to setup Performance Center integration.

AM Mode

In AM mode the .NET agent will capture all instrumentation data. If LoadRunner/Performance Center is executing an application, then you will see the data in the normal Diagnostics database (for example, Default Client). AM mode supersedes all other modes except for AD.

Enterprise Mode

Enterprise mode is a combination of AD, AM and PRO modes. It will capture data for LoadRunner/Performance Center runs in a separate database as well as capture data outside of LoadRunner/Performance Center runs.

In this mode data will also be sent to the profiler.

Both AD and AM modes will override this mode.

If the PRO mode is set along with Enterprise mode then the .NET agent will collect data continuously for the profiler even if the profiler UI is not in use.

If PRO mode is not set then the agent will not start collecting until the profiler UI is started.

TV mode

This mode will send events to Transaction Vision. This mode can be combined with other modes.

Note about AD Mode and Enterprise Mode

The .NET agent gets notified of LoadRunner/Performance Center runs by the Diagnostic Mediator.

If LoadRunner/Performance Center starts testing an instrumented application that is not running, for example, a web application getting hit the first time, then when the application starts executing the Diagnostics agent will not be notified of the run. This is because the agent will not have had enough time to get initialized and start listening to the mediator for this notification.

To work around this problem, the .NET agent needs to be "primed"(initialized) by a call to the web application before a LoadRunner/Performance Center run is started. This initializes the web application's process (worker process) and the probe so that it is ready to accept run information from the mediator.

Configuring Support for MSMQ Based Communication

To configure the .NET Agent to support MSMQ based communication, include the **msmq.points** file in the scope of the appdomain as shown in the example excerpt from a <probe_install_dir>/etc/probe_config file:

```
<process name="SimplestQueuingSender">  
  <points file="msmq.points"/>  
  <modes enterprise="true"/>  
</process>
```

Configuring Latency Trimming and Throttling

When the .NET Agent determines that it is running out of resources because the Diagnostics Server is not keeping up with the amount of data that the probes are capturing, the agent can automatically reduce the number of methods the probe captures using a process called *latency trimming*. By default, latency trimming is enabled so that the probe's work load can be adjusted as necessary.

When latency trimming is enabled, the .NET Agent trims the number of methods captured by a probe 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 can 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 can be configured to apply additional trimming of the probe's data which will affect the granularity of the data shown by the Diagnostics user interface.
-

Disabling Latency Trimming

By default, trimming is enabled for the .NET Agent. To disable trimming you must change the 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 13, "Understanding the .NET Agent Configuration File."

Enabling Latency Trimming

By default, trimming is enabled for the .NET Agent. If you subsequently disabled trimming, you must change the 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 13, "Understanding the .NET Agent 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 13, "Understanding the .NET Agent Configuration File."

Configuring Latency Trimming Throttling

Latency trimming is throttled by default. When throttling is enabled, the amount of trimming that is done 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 13, "Understanding the .NET Agent Configuration File".

Configuring Depth Trimming

The .NET Agent 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 Agent 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 13, "Understanding the .NET Agent Configuration File".

Configuring the .NET Agent for Light-Weight Memory Diagnostics

The Light-Weight Memory Diagnostics (LWMD) feature refers to the ability to capture and analyze usage data that relates to Collections. Specifically Collections refer to any class that implements either the **System.Collections.ICollection** or **System.Collections.Generic.ICollection** interfaces. Examples of such Collections are ArrayList, HashTable, DataView etc. The most common form of .NET memory leaks occur in Collections that are not properly maintained.

When the .NET Agent is installed, the default configuration for the .NET Agent probe is to have LWMD turned off. To enable the LWMD feature you must perform two modifications to the **probe_config.xml** file:

- ▶ You must enable the <lwmd> element (see "<lwmd> element" on page 488).
- ▶ You must add one or more references to the **Lwmd.points** file as described in the instructions below.

Note: Enabling the probe to capture collections metrics could incur additional overhead on the host for an application.

To enable the capture of collection metrics for a process or for an appdomain:

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 Agent, 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.

To enable LWMD instrumentation for all enabled appdomains that run under a process, you add the `points` tag to the `process` tag in the `probe_config.xml` configuration file. For example, to enable LWMD instrumentation for all enabled ASP.NET appdomains:

```
<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>
```

To enable LWMD instrumentation for a specific enabled appdomain that runs under a process, you add the `points` tag to an `appdomain` tag in the `probe_config.xml` configuration file. You can add the `points` tag to one or more of the `appdomain` tags. For example, to enable LWMD instrumentation for the "your_app_name" appdomain running in the ASP.NET process:

```
<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:

To disable the LWMD feature you must perform two modifications to the `probe_config.xml` file:

- ▶ Disable the `<lwmd>` element (see "`<lwmd>` element" on page 488).
- ▶ Delete the `points` tags for the `Lwmd.points` file from all `process` tags and from the appropriate `appdomain` tags.

Without the LWMD `points` tags in the configuration file, the probe cannot locate the LWMD instrumentation instructions contained in the `Lwmd.points` file and so the probe will not instrument for Collection usage.

To control LWMD Instrumentation:

When the .NET Agent is installed, the default configuration for the Lwmd.points file contain instructions to instrument Collection usage in a wide range of assemblies, appdomains, namespaces and classes. You can modify the your application's points file to narrow the scope of the Collections that you want to inspect. LWMD Instrumentation is implemented as Caller side Instrumentation, refer to "Caller Side Instrumentation" on page 348 for a description of how this instrumentation works.

Note: Narrowing the scope of LWMD instrumentation is a recommended best practice.

To narrow the scope of the Collections that you want to inspect perform the following steps:

- 1** Delete the points tags for the Lwmd.points file from the process tags and from the appropriate appdomain tags. This will remove the LWMD settings that specify a wide instrumentation scope.
- 2** Add an LWMD section to the points file for your process or appDomain. As an example, to do this copy and paste the following into your_app.points file:

```
[LWMD]
keyWord = lwmd
scope =
ignoreScope =
```

- 3** Set the scope and ignoreScope Arguments in the LWMD section to narrow the scope of the Collections that you want to inspect. Example:

```
[LWMD]
keyWord = lwmd
scope = !my_namespace\.*
ignoreScope = !my_namespace.my_class1\.*
```

The example above instruments all the Collections that are constructed from the my_namespace namespace except for any Collections that are constructed from any method in the my_namespace.my_class1 class.

For LWMD Instrumentation there is an internal default value for `ignoreScope` that is unpublished and is always included with any value you enter. The default value includes namespaces and classes relating to the .NET Infrastructure that if instrumented would adversely affect the application, for example, `!System.*`, `!Microsoft.*`, and so on.

Limiting Exception Stack Trace Data

The agent 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. 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.

This section includes:

- Limit Specific Exception Types
- Limit the Number of Exceptions per Server Request
- Limit the Size of the Stack Trace
- Example

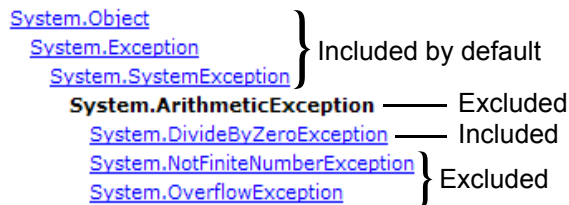
Limit Specific Exception Types

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 Agent probes collect stack trace data on only the first 4 exceptions encountered 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 can 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.

```
<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 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 instrumentation logging by changing the **logging level** tag of the instrumentation section, as shown in the following example:

```
<instrumentation>
  <logging level="off" />
</instrumentation>
```

Overriding the Default Probe Host Machine Name

The **registered_hostname** property enables you to override the default host machine name that a probe uses to register itself with the Diagnostics command server. 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 might not be possible for the Diagnostics command server to communicate with the probe unless you override the default host machine name.

To override the default host machine name for a probe there is a three step process.

- 1 First, set the **registered_hostname** attribute, located in the .NET Agent **<diagnosticsserver>** element of the **probe_config.xml** file, to an alternate machine name or IP address that allows the Diagnostics command server to communicate with the Probe.

For example:

```
<diagnosticsserver url="http://localhost:2006/commander"
  registered_hostname=" my_host_name "/>
```

- 2 Second, register the alternate machine name or IP address of the host with the .NET Metrics Agent. To do this, make a **metrics.agent.registered_hostname** entry in the **metrics.config** file. You can add the entry just under the **metrics.systemgroup** entry.

For example:

```
metrics.systemgroup = Default
metrics.agent.registered_hostname = my_host_name
```

- 3 Finally, you must restart both the .NET Agent and the .NET Metrics Agent for this change to take affect.

Notes:

- ▶ You need to set the **registered_hostname** attribute to deal properly with the use of the IIS Host Header technology.
 - ▶ 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 Service Management or Diagnostics Standalone, the name that you specify is shown as the host name in System Health.
-

Listing the Probes Running on a Host

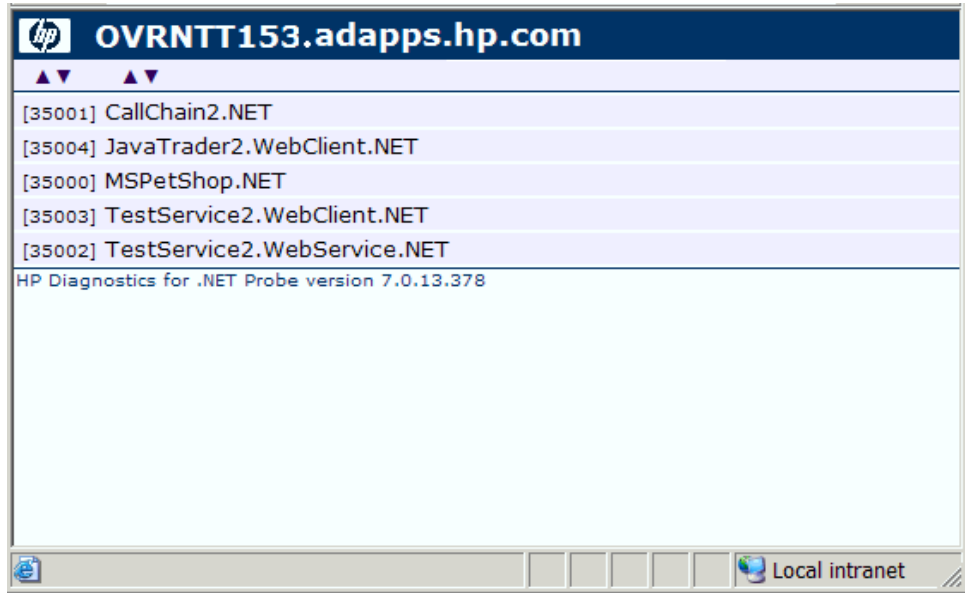
When more than one probe is running on a single host, you cannot know which port each probe is using since the port that is assigned is based on the one that is available at the time the application (and probe) is started. As the applications are started and stopped, the port that is assigned to the probe for a given application is likely to change.

You can determine which probes are running 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 Agent 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 Agent is configured to work with a Diagnostics Server, the probe (Profiler) authorization and authentication settings are managed from the Diagnostics command server to which this probe is connected. For more information, see "User Authentication and Authorization" on page 657.

When you access the probe from the Diagnostics Server, the default username is `admin` and the default password is `admin`.

If the .NET Agent is installed as a profiler only, 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.

Caution: 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 must 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 request
- ▶ 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 "Consumer ID Rules Syntax and Examples for .NET Agent" on page 545.
- 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 could be SOAP rules, no HTTP rules, and IP rules. If there is no match on any of these rules, 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, envelope or body. The rule specifies a regular expression that is used to match against the web service name being called by the consumer. See "Using Regular Expressions" on page 782 for help using regular expressions.

If there is a match with the web service name, the agent/probe attempts to find the element defined in `consumeridfield` in the appropriate SOAP location defined by the SOAP rule. If the element is not found, this rule is skipped and the agent/probe goes 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 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.

Consumer ID Rules Syntax and Examples for .NET Agent

The rules syntax and examples are specific to how the consumer ID is being defined.

SOAP Rules

The SOAP rules allow for the consumer ID to be obtained from an XML element in the SOAP header, envelope or body.

An example of configuring consumer ID based on a value in the SOAP header is shown below:

```
<consumeridrules enabled="true">
  <soaprules>
    <soaprule id="SOAP1" rule="TestService" location="soap-header"
consumeridfield="Caller"/>
  </soaprules>
</consumeridrules>
```

`id=` attribute can be any name you would like to use to identify the rule; this attribute is not used by the .NET probe.

`rule=` attribute must be defined for a `soaprule`. The rule is a regular expression that is used to match against the web service name being called by the consumer or you can use the exact Web service name.

`location=` can be set to "soap-header", "soap-envelope", "soap-body". If you do not specify a location, it defaults to use "soap-header." If you configure a location for any soap rule, you must configure a location for all soap rules, or a severe error will occur and the consumer ID based on SOAP logic will be disabled.

`consumeridfield=` attribute must be defined for a `soaprule`. The element in the SOAP header, envelope or body whose value you want to use as the consumer ID.

If there is a match with the pattern specified in the `rule=` attribute, the .NET agent attempts to find a text element for the element defined in the `consumeridfield`. The element in the `consumeridfield` can be a qualified name—that is, 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 specified location, this rule is skipped and the probe goes on to the next rule that is defined.

For example, the following rule matches on a Web service named `TestService` and uses the `Caller` element's value as the consumer ID:

```
<soaprule id="SOAP1" rule="TestService" location="soap-header"
consumeridfield="Caller"/>
```

As long as the callers of the `TestService` Web service have a value defined for `Caller`, the metrics will be grouped by the different values for `Caller`. Here is an excerpt from the soap header that would map to a consumer ID of `"Customer2"` for this caller of the `TestService`:

```
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>
    <Caller>Customer2</Caller> <!-- The consumer id returned is
                               "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>
```

Enable SOAP Capture

SOAP envelopes can be very large so the `<soapcapture>` element is provided to enable you to control the overhead, mainly memory overhead, of capturing SOAP requests and responses.

```
<soapcapture enabled="true">
```

The <soapcapture> element controls whether SOAP requests and responses are captured. If it is disabled, SOAP requests and responses will not be captured. This means there will not be SOAP requests or responses included in TransactionVision events, nor will there be any SOAP requests available with SOAP faults, and you cannot configure consumer ID based on SOAP header, envelope, or body.

The <soapcapture> setting overrides the settings in <soaprequestforsoapfault> which controls SOAP payload capture on SOAP faults. See "Configuring SOAP Fault Data" on page 548.

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, 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, 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 can be defined as a single IP address; for example, 19.225.17.125. The rule can also define a range; for example, 19.255.17.125,19.255.17.255.

An asterisk can also be used in an octet of an IP address to match against anything in that octet; for example, 19.255.17.*. A range can be defined in an octet to match a range of values in that octet; for example, 19.255.17.20-255. Combinations of these can also be used; for example, 19.*.17.20-255, 20*.10-55.*. If there is a match, 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"
  </iprules>
  consumerid="Client3"/>
</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 SOAP fault 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:

```
<soaprequestforsoapfault enabled="true" maxsize="10000"/>
```

Set `enabled` to "false" to disable this feature.

The `<soapcapture>` element overrides the `<soaprequestforsoapfault>` element. So that if `<soapcapture>` is disabled, `<soaprequestforsoapfault>` is disabled even if `<soaprequestforsoapfault>` is set to true. Also whatever `<soapcapture>` `maxsize` value is set, overrides the `<soaprequestforsoapfault>` `maxsize`. So that is `<soapcapture>` `maxsize` is set to 5000 and `<soaprequestforsoapfault>` `maxsize` is set to 10000, the payload size will be maximum of 5000.

Collecting Additional Probe Metrics or Modifying Probe Metrics

You can configure the .NET agent to collect additional probe metrics based on perfmon counters using the `<metrics>` and `<metric>` elements in the `<probe_install_dir>\etc\probe_config.xml` file. See "`<metric>` element" on page 492 and "`<metric>` element" on page 492 for details.

You can also modify probe metrics using the `<metric>` element. But note the following special cases:

- ▶ If you want to move a metric from one metric category to another, you must change the metric's `group` attribute as well as the `metric` name attribute. This is because the existing metric name is already registered to its old group on the Diagnostics mediator and this association cannot be changed.
- ▶ If you want to redefine an existing probe metric it is better to create a completely new metric entry rather than assigning a different perfmon counter to the metric. This ensures that you avoid aggregating disparate data.

Performance Counter Security

The .NET Agent uses Performance Counters to collect probe metrics. This requires the application process that is being monitored by the .NET Agent to have access rights to performance counters. Each process runs as a user account therefore this user account must have access rights to performance counters. The simplest way to do this is to add the user account that the process runs as to the **Performance Monitor Users** group.

However Microsoft has introduced the concept of a **virtual accounts** in Windows Vista SP2, Windows Server 2008 SP2, Windows 7 and Windows Server 2008 R2 (see [http://technet.microsoft.com/en-us/library/dd548356\(WS.10\).aspx](http://technet.microsoft.com/en-us/library/dd548356(WS.10).aspx) for details). These operating systems have used the virtual accounts concept in IIS and by default, application pools in IIS run as **ApplicationPoolIdentity**. Because this user account is virtual, it requires special steps to add the user account to the Performance Monitors Users group.

In Windows 2008 R2 and Windows 7 do the following:

- 1** Open the **Server Manager** tool, there are many ways to do this but one is through Administrative Tools.
- 2** In the left hand pane find **Local Users and Groups** under Configuration.
- 3** Click the + to expand it.
- 4** Double-click Groups.
- 5** Double-click the **Performance Monitor Users** group.
- 6** Click the **Add...** button.
- 7** Click the **Locations...** button.
- 8** Select the local computer.
- 9** Click the **OK** button.
- 10** Make sure that object types includes **Built-in security principals**.
- 11** Enter **IIS APPPOOL\<name of the application pool>**, (example IIS APPPOOL\My Webservice App Pool, where My Webservice App Pool is the name of the application pool), in the text box.
- 12** Click the **OK** button.

In Windows 2008 SP2 and Windows Vista SP2 do the following:

- 1** Open a Command Prompt window.
- 2** Type `net localgroup "Performance Monitor Users" "IIS APPPOOL\<name of application pool> /ADD` (where **<name of the application pool>** is the application pool name).
- 3** **The command completed successfully** will be displayed if this is successful.

Part VI

Configuring Communications through Proxies and Firewalls

This section includes:

- Configuring Diagnostics Servers and Agents for HTTP Proxy
- Configuring Diagnostics to Work in a Firewall Environment

15

Configuring Diagnostics Servers and Agents for HTTP Proxy

Configuration steps are provided for you to enable HTTP proxy communications between the Diagnostics components. These configuration instructions are intended for experienced administrators with in-depth knowledge of 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 556
- ▶ Enabling HTTP Proxy Communications for the Java Agent on page 557
- ▶ Enabling HTTP Proxy Communications for a .NET Agent on page 557

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 command server 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 63.

Enabling HTTP Proxy Communications for the Java Agent

The following section describes how to configure the Java Agent to communicate with Diagnostics command server through an HTTP proxy:

To configure the Java Agent 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 Agent

The following section describes how to configure the .NET Agent to communicate with the Diagnostics command server through an HTTP proxy:

To configure a .NET Agent 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.

16

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 Service Management. See the LoadRunner, Performance Center and Business Service Management documentation for more details.

This chapter includes:

- ▶ Overview of Configuring Diagnostics for a Firewall on page 560
- ▶ Collating Offline Analysis Files over a Firewall on page 563
- ▶ Installing and Configuring the MI Listener on page 564
- ▶ Configuring the Diagnostics mediator server to Work with a Firewall on page 565
- ▶ Configuring LoadRunner and Performance Center to Work with Diagnostics Firewalls on page 571

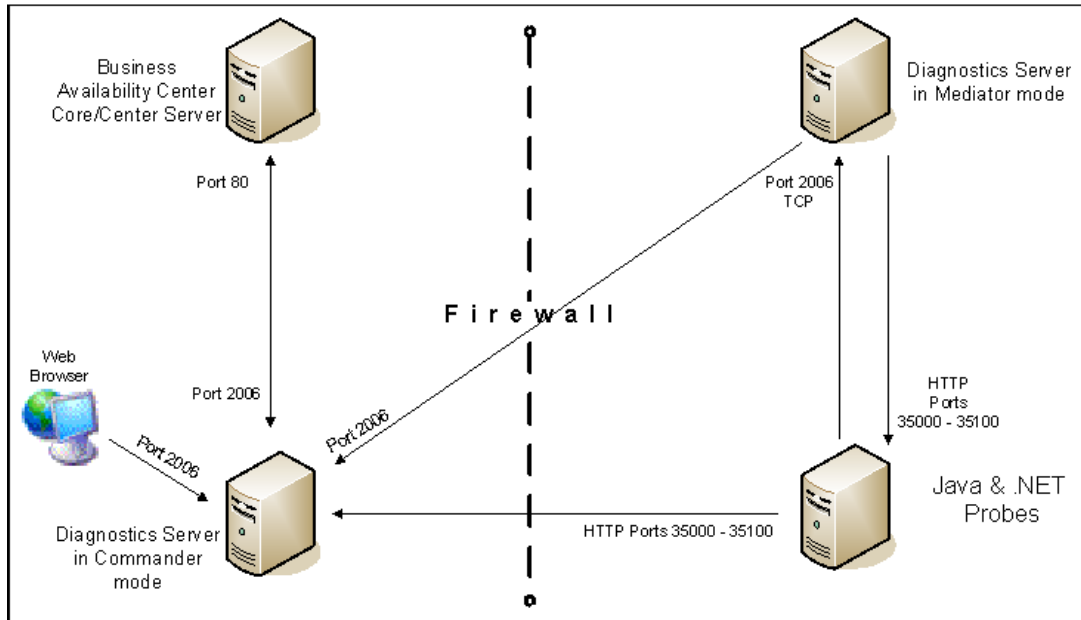
Note: The configuration instructions should be used only by experienced users with in-depth knowledge of HP Diagnostics. Use caution when modifying any configuration settings for the Diagnostics components.

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 Service Management

The diagram below shows a typical Diagnostics deployment where a firewall separates the probe from the other Diagnostics and Business Service Management components.



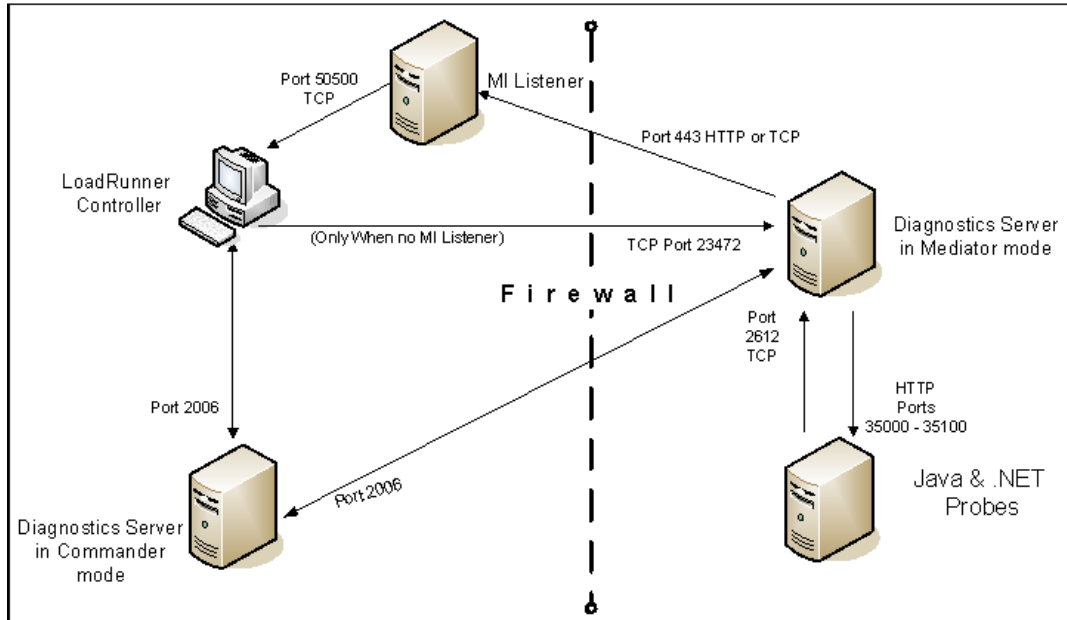
To configure the firewall to enable the communications between the Diagnostics components, open the ports that will allow:

- ▶ HTTP requests from the Business Service Management server(s) to the Diagnostics Server in Commander mode, on port 2006.
- ▶ HTTP requests from the Diagnostics Server in Commander mode to Business Service Management 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 mediator server to the Diagnostics Server in Commander mode on port 2006.
- TCP requests from the probe to the Diagnostics mediator server on port 2612.
- HTTP requests from the probe to the Diagnostics mediator server 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 on 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 Agent for Multiple Application Server JVM Instances” on page 406.

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:

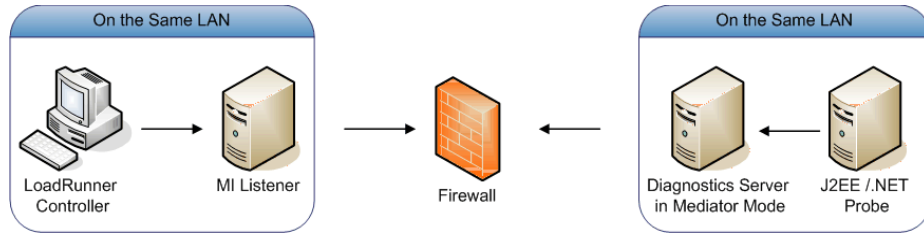
- ▶ HTTP requests from the Diagnostics Server in Mediator mode to the Diagnostics Server in Commander mode on ports 2612 and 2006.
- ▶ TCP requests from the probe to the Diagnostics Server in Mediator mode on port 2612.
- ▶ HTTP requests from the probe to the Diagnostics mediator server on port 2006.
- ▶ 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 on 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 Agent for Multiple Application Server JVM Instances” on page 406.

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 mediator server 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, see the *HP LoadRunner Controller User Guide*. For more information about how to configure the MI Listener for Performance Center, see the *HP Performance Center Administrator's Guide*.

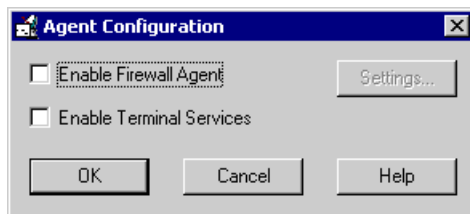
Configuring the Diagnostics mediator server to Work with a Firewall

To configure the Diagnostics mediator server so that it can work across a firewall, you must complete the following additional configuration steps. If you did not install and configure the Diagnostics mediator server you must do so before attempting these steps. For instructions on installing the Diagnostics mediator server, see Chapter 2, “Installing the Diagnostics Server.”

To configure the Diagnostics mediator server for a firewall on a Windows machine:

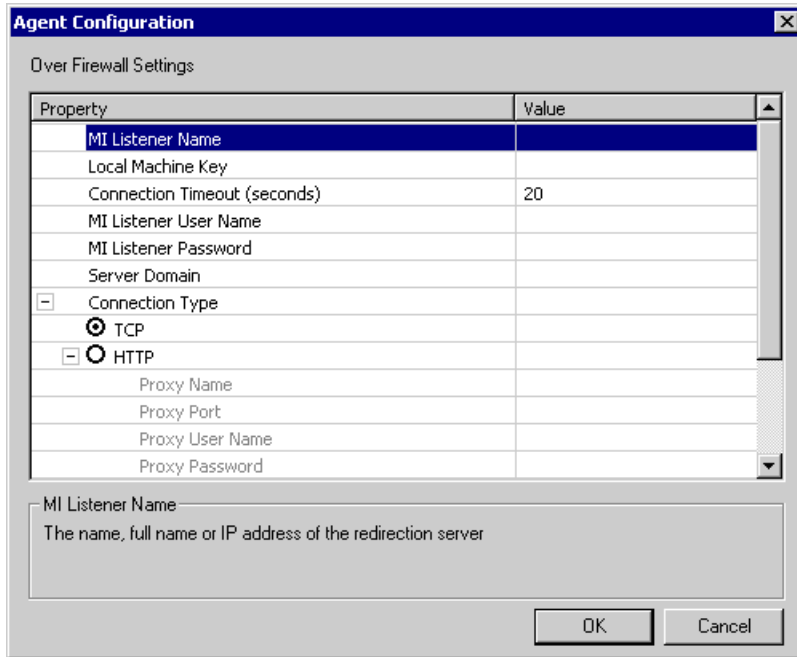
- 1 Launch the Agent Configuration by running `<diagnostics_server_install_dir>/nanny/windows/bin/AgentsConfig.exe`.

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.

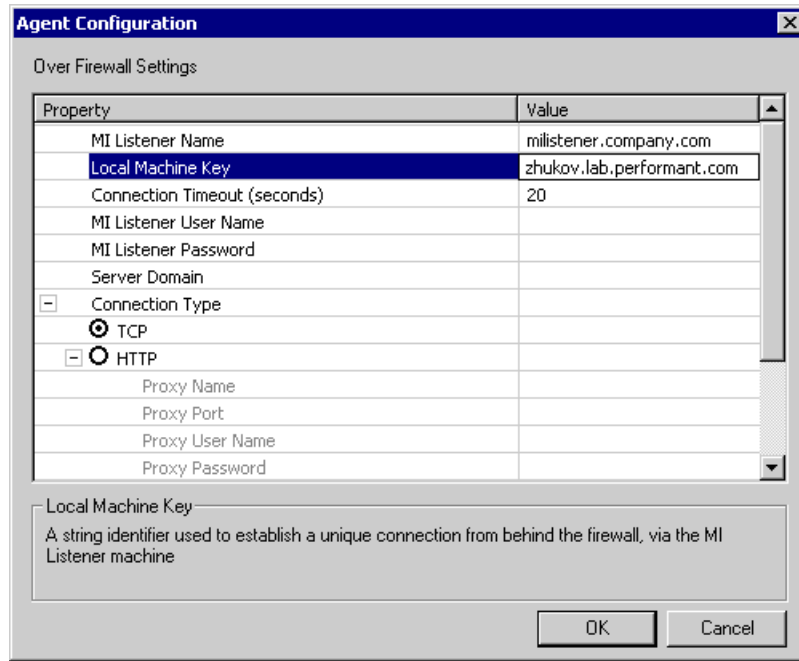


- 5 For the **Local Machine Key** property, enter the machine name of the host of the Diagnostics mediator server.

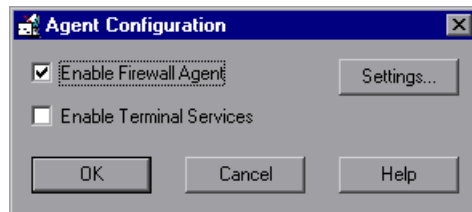
Important:

- ▶ 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 mediator server. For more information on the System Health Monitor, see Appendix D, “Using the System Health Monitor.”
-

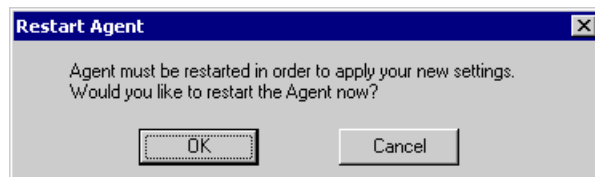
6 Click OK to close the dialog box.



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 mediator server for a firewall on a UNIX/Linux machine:

- 1 Modify the `<diagnostics_server_install_dir>/nanny/<platform>/dat/br_lrch_server.cfg` file.

Change the value of the `FireWallServiceActive` property to 1.

- 2 Run the following commands to launch the Agent Configuration utility.

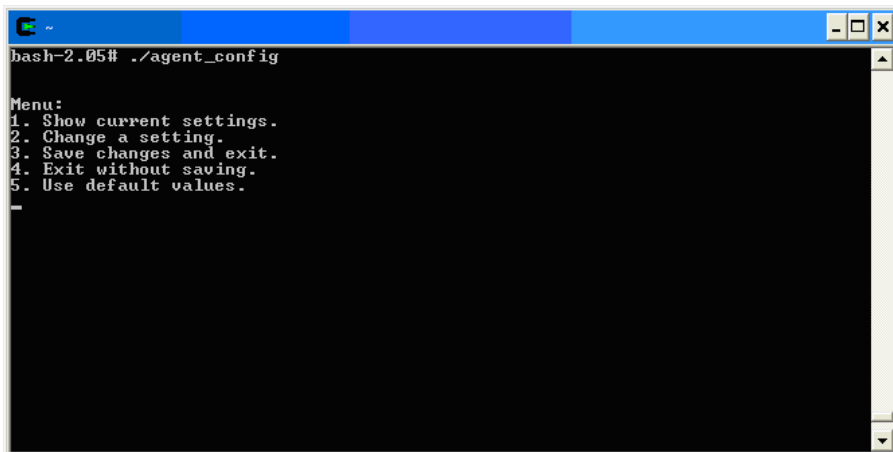
For Solaris and Linux, where `<platform>` is either `solaris` or `linux`:

```
export LD_LIBRARY_PATH=.
export M_LROOT=<diagnostics_server_install_dir>/nanny/<platform>
cd $M_LROOT/bin
./agent_config
```

For HP-UX:

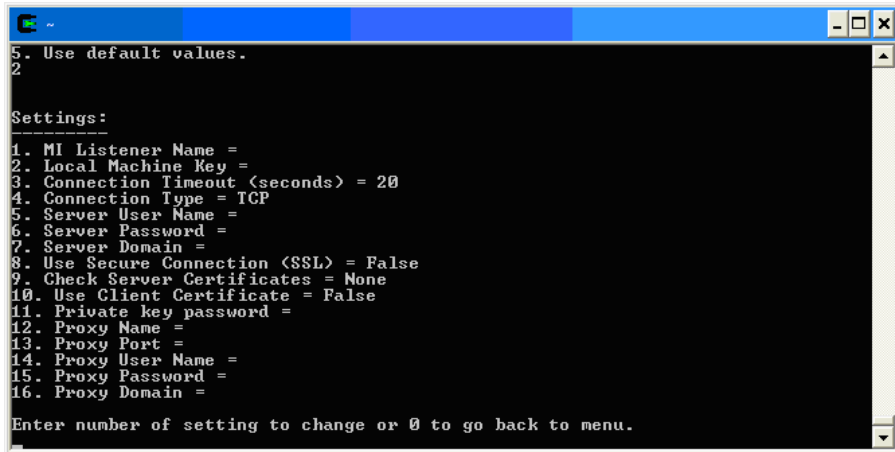
```
export SHLIB_PATH=.
export M_LROOT=<diagnostics_server_install_dir>/nanny/hpux
cd $M_LROOT/bin
./agent_config
```

- 3 In the Agent Configuration Utility window, press **2**, **Change a Setting**.



4 A list of settings appears.

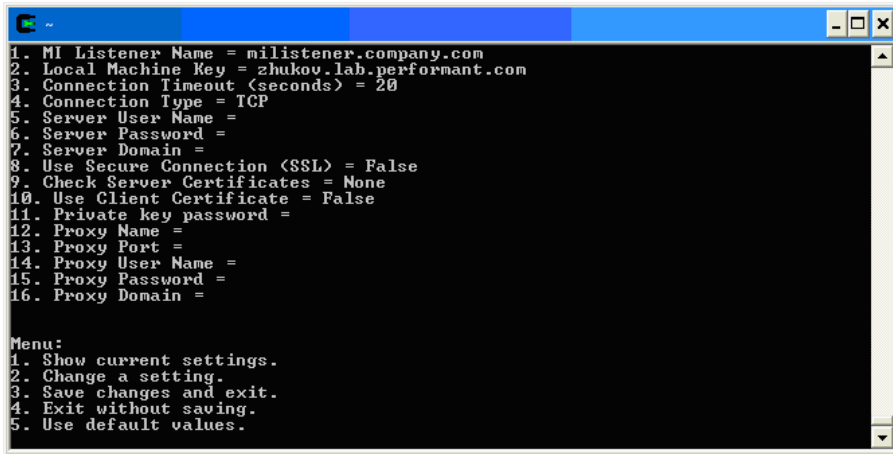
Press **1** to select **MI Listener Name**, and enter the machine name or IP address of the MI Listener host.

A screenshot of a Windows command prompt window. The window title bar is blue with standard minimize, maximize, and close buttons. The text inside the window is as follows:

```
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 mediator server 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 mediator server. For more information on the System Health Monitor, see Appendix D, “Using the System Health Monitor.”



```
1. MI Listener Name = milistener.company.com
2. Local Machine Key = zhukov.lab.performant.com
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 =

Menu:
1. Show current settings.
2. Change a setting.
3. Save changes and exit.
4. Exit without saving.
5. Use default values.
```

- 6 Press **3**, **Save changes and exit**, to complete the updates.
- 7 Restart the Diagnostics mediator server.

`./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 might 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 are 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:

Make sure that you 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 Listeners in the *HP Performance Center Administrator's Guide*. Also make sure that Diagnostics is enabled in Performance Center. For details, see the section about HP Diagnostics in the *Performance Center User's Guide*.

For LoadRunner:

Make sure that Diagnostics is enabled in LoadRunner. When you configure your load test scenario to work with Diagnostics, make sure 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

This section includes:

- ▶ .NET System Metrics Agent - Systems Metrics Capture
- ▶ Java Agent Metrics Collectors
- ▶ Java Agent - System Metrics Capture
- ▶ Java Agent - JMX Metrics Capture

17

.NET System Metrics Agent - Systems Metrics Capture

Information is provided about system metrics capture and how to configure the system metrics collector installed with the .NET Agent.

This chapter includes:

- About the .NET System Metrics Agent on page 575
- System Metrics Captured by Default on page 576
- Configuring .NET System Metrics Capture on page 577
- Adding System Metrics Using the Windows Performance Monitor on page 580
- Default Entries in the .NET Agent metrics.config File on page 582
- Keywords in the metrics.config File on page 583

About the .NET System Metrics Agent

A system metrics collector is installed with the .NET Agent and run as a Windows Service (HP Diagnostics Metrics Agent). The .NET system metrics agent gathers system level metrics, such as CPU usage and memory usage, from the agent's host. It is configurable so you can control which metrics are collected as well as aspects of how the metrics are collected and published.

Only one instance of the .NET system metrics agent is run on a given host, no matter how many instances of the probe were started on the host.

Note: To configure additional probe metric capture with the .NET Agent (other than system metrics capture described here) see “Collecting Additional Probe Metrics or Modifying Probe Metrics” on page 549.

System Metrics Captured by Default

The following are the system metrics that the .NET system metrics agent collects by default for all supported platforms (excluding z/OS):

- CPU
- MemoryUsage
- VirtualMemoryUsage
- ContextSwitchesPerSec
- DiskBytesPerSec
- DiskIOPerSec
- NetworkBytesPerSec
- NetworkIOPerSec
- PageInsPerSec
- PageOutsPerSec

In addition to the default system metrics listed above, the following system metrics are also captured by default on .NET Agent systems. (The layout of these entries is described in “Understanding the system/ Metrics Collector Entries” on page 577).

.NET CLR Memory\# Total committed Bytes_Global_
ASP.NET\Application Restarts
ASP.NET\Requests Queued
ASP.NET Apps v2.0.50727\Requests/sec__Total__

You can control which of the default system metrics the .NET system metrics agent gathers and you can capture custom system metrics with the .NET system metrics agent.

Configuring .NET System Metrics Capture

The configuration file for the .NET system metrics agent is the `<probe_install_dir>/etc/metrics.config` file. Changes to the `metrics.config` file are processed dynamically by the .NET Agent.

```

metrics.config - Notepad
File Edit Format View Help
system/.NET CLR Memory\# Total committed Bytes\_global_ = Bytes Used|bytes|.NET
system/ASP.NET\Application Restarts = Application Restarts|
system/ASP.NET\Requests Queued = Requests queued|count
system/ASP.NET Apps v2.0.50727\Requests/sec\_Total_ = ASP.NET v2.0 Requests
system/CPU = CPU|percent|System
system/MemoryUsage = MemoryUsage|percent|S
system/VirtualMemoryUsage = VirtualMemoryUsage|pe
system/ContextSwitchesPerSec = ContextSwitchesPerSec
system/DiskBytesPerSec = DiskBytesPerSec|bytes
system/DiskIOPerSec = DiskIOPerSec |count|D
system/PageInsPerSec = PageInsPerSec |count|S
system/PageOutsPerSec = PageOutsPerSec |count|
system/NetworkBytesPerSec = NetworkBytesPerSec |by
system/NetworkIOPerSec = NetworkIOPerSec |count

metrics.server.uri = http://127.0.0.1:45000/metricdata/
metrics.systemgroup = Default
credentials.username =
credentials.password =

```

Note: There is a different `metrics.config` file included with the Java Agent (see Chapter 18, “Java Agent Metrics Collectors”).

Understanding the system/ Metrics Collector Entries

Metrics collector entries in the `metrics.config` file instruct the .NET system metrics agent to gather specific metrics. Entries that begin with `system/` are processed as Windows Performance Monitor Counters.

These system metrics collector entries use the following layout:

```
system/<Counter_name>\<Performance_object>\<Instance>\<Remote_machine> =  
<metric_id>|<metric_units>|<category_id>
```

All fields are required except for the optional <Instance> and <Remote_machine> fields.

Where:

- ▶ **Counter_name** indicates the Windows Performance Monitor counter. See “Adding System Metrics Using the Windows Performance Monitor” on page 580 for details on how to identify the counter, performance object and instance in the Windows Performance Monitor UI.
- ▶ **Performance_object** indicates the Windows Performance Monitor performance object associated with the Counter_name.
- ▶ **Instance** indicates the Windows Performance Monitor instance of a counter. You may use a wildcard (*) to indicate that all instances are desired. If you wish to specify a specific enumeration of all instances, you precede the enumeration index number with the hash sign (#1). The enumeration index number must be a positive number.
- ▶ **Remote_machine** is only required if the Windows Performance Monitor Counter is running on a machine that is different (remote) from the machine that the .NET system metrics agent is running on. The minimum requirement for this configuration to work is that the Network Service User on the machine that the .NET system metrics agent is running on must have permissions to read the Windows Performance Monitor Counters from the remote machine.
- ▶ **<metric_id>** indicates the name that represents the metric in the Diagnostics 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, 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 Details pane of the Diagnostics UI. This parameter has no impact on the data displayed in the Diagnostics views.

Example without an <Instance>:

```
system/ASP.NET\Requests Queued = Requests Queued|count|ASP
```

Example with an <Instance>:

```
system/Processor\% Processor Time\_Total = CPU|percent|System
```

Example with an integer <Instance>:

```
system/Processor\% Processor Time\#1 = CPU 1|percent|System
```

Example without an <Instance> and running on a <Remote_Machine>:

```
system/ASP.NET\Requests Queued\IISAQUAH = Requests  
Queued(IISAQUAH)|count|ASP
```

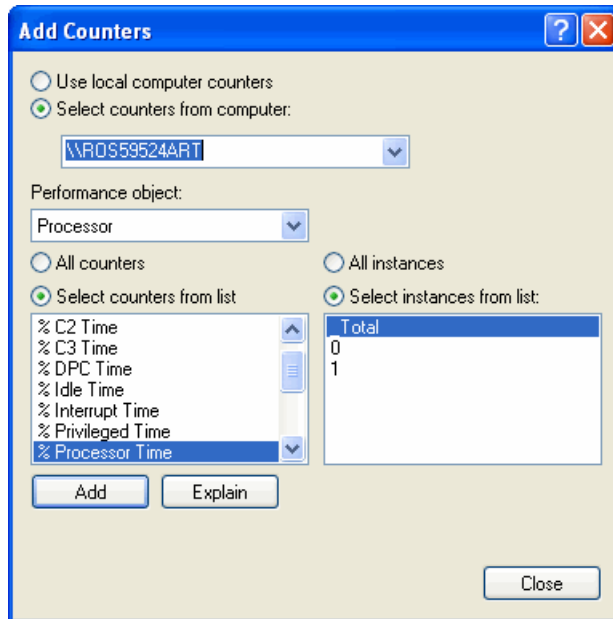
Adding System Metrics Using the Windows Performance Monitor

To add a system metric counter to the metrics.config file you must first find its definition using the Windows Performance Monitor (Perfmon). The following example uses version 5.x of Perfmon. Version 6.x is similar but the UI is a little different.

To add counters in Perfmon:

- 1 Start the Windows Performance Monitor. For example select **Start > Control Panel > Administrative Tools > Performance**.
- 2 The Perfmon Performance dialog box is displayed showing the System Monitor graph with a table of the current counters beneath the graph. Right-click the System Monitor graph and select **Add Counters...** from the pop-up menu.

The Add Counters dialog box is displayed:



- 3 Select the **Select counters from computer** entry and make sure the host computer is select in from the drop down list.

- 4 In the Performance object list, select the object that the counter belongs to.
- 5 Choose **Select counters from list** and select an instance from the list of instances.
- 6 Click the **Add** button to add the counter. The following instructions tell you how to create an entry for a counter using the system/ metrics entry described in “Understanding the system/ Metrics Collector Entries” on page 577.

To collect metrics for a Perfmon counter:

- 1 Open the `<probe_install_dir>/etc/metrics.config` file on the .NET agent system.
- 2 Create the **system/** metrics entry for the counter using the layout described in “Understanding the system/ Metrics Collector Entries” on page 577.

You can add this entry anywhere in the file, however best practice is to add it to the bottom of existing collection of these type of entries. In the example shown in the screen shot above:

- The selected host computer is ROS59524ART
- The selected Performance object is Processor
- The selected Counter is % Processor Time
- The selected Instance is _Total

So if the host computer is local, the entry in the metrics.config file for the Performance Monitor counter would be:

```
system/Processor\% Processor Time\_Total = CPU|percent|System
```

And if the host computer is remote, the entry in the metrics.config file for the Performance Monitor counter would be:

```
system/Processor\% Processor Time\_Total\ROS59524ART = CPU(ROS59524ART)|percent|System
```

Performance Counter Security

The .NET metrics agent uses Performance Counters to collect system metrics. The metrics agent runs as a Network Service and this account needs to be added to the **Performance Monitor Users** group.

Troubleshooting Added System Metrics Counters

If you specify a new counter that appears to not be functioning, you can use the Windows Event Viewer to look at the Diagnostics logs for the .NET system metrics agent source for errors and warnings.

For example:

A **Could not locate Performance Counter with specified category name** warning entry typically indicates that you may have mis-typed the name of the counter. This can happen, for example, if you read a counter name from the PerfMon Performance pane that has embedded blanks. The default font used by PerfMon is not a monospaced font and as such makes it difficult to see embedded blanks in the name of the counters, categories and instances. You can change the font to a monospaced font type and then more clearly see the exact format of counter names.

For example:

An **Instance does not exist in the specified Category** warning entry typically indicates that the instance you have chosen is not active at this time. We do not recommend that you use transient instances. Permanent instances like `__Total__` are appropriate.

Default Entries in the .NET Agent metrics.config File

Upon installation, the `<probe_install_dir>/etc/metrics.config` file has three entries:

- ▶ A grouping of default **system/** entries for PerfMon counters
- ▶ A **metrics.server.uri** entry that specifies how the .NET system metrics agent publishes its data
- ▶ A default **metrics.systemgroup** entry

Other additional entries can be added after these default entries.

Keywords in the metrics.config File

The keywords that can be used in entries in the <probe_install_dir>/etc/metrics.config file are as follows:

- credentials.password
- credentials.username
- default.sampling.rate
- metrics.server.uri
- metrics.systemgroup
- metrics.agent.publish.interval
- metrics.agent.registered_hostname
- proxy.password
- proxy.user
- proxy.uri
- system/

The use of the **system/** keyword is described in “Configuring .NET System Metrics Capture” on page 577.

The use of each of the other keywords is described in the following section.

credentials.password	This setting must match the setting for the password attribute of the <credentials> element in the probe_config.xml file. See “<credentials> element” on page 462 for more details.
credentials.username	This setting must match the setting for the username attribute of the <credentials> element in the probe_config.xml file. See “<credentials> element” on page 462 for more details.
default.sampling.rate	This setting defines the rate at which the .NET system metrics agent samples the configured system metric counters. The default rate is every 5 seconds. Values are expressed as a number of Seconds, Minutes, Hours or Days, for example, nS, nM, nH or nD. The following example sets the rate to every 10 seconds: default.sampling.rate = 10s

<p>metrics.server.uri</p>	<p>This setting is automatically generated at install time. It defines the URI that the .NET system metrics agent uses to publish the system metric counters to the Diagnostic Mediator Server. The following example is for a Diagnostic Mediator Server running on the my_diag_server machine, and using a metricport of 2006 to publish the metrics:</p> <pre>metrics.server.uri = http://my_diag_server:2006/metricdata/?sleep=false</pre> <p>Any changes to the probe_config.xml settings for either the metrichost attribute or the metricport attribute of the <mediator> element must also be reflected at the same time in the metrics.server.uri setting.</p> <p>The ?sleep setting controls whether the Diagnostic Mediator Server that receives the published metrics will respond immediately or delay its response to the .NET system metrics agent. A setting of ?sleep=false responds immediately, a setting of ?sleep=true delays its responds by a default of 5 seconds.</p>
<p>metrics.systemgroup</p>	<p>This setting is automatically generated at install time. Do not change this setting.</p>
<p>metrics.agent.publish.interval</p>	<p>This setting defines the interval between publishes of the current values of the System Metric Counters by the .NET system metrics agent to the Diagnostic Mediator Server. The default interval is 5 seconds. Set values can be expressed as a number of Seconds or Minutes, for example, nS or nM. The following example sets the publish interval to 10 seconds:</p> <pre>metrics.agent.publish.interval = 10S</pre>
<p>metrics.agent.registered_host name</p>	<p>Refer to the “Overriding the Default Probe Host Machine Name” on page 539 for a description of when and how to use this setting.</p>
<p>proxy.password</p>	<p>This setting must match the setting for the proxypassword attribute of the <diagnosticsserver> element in the probe_config.xml file. See “<diagnosticsserver> element” on page 465 for more details. Also refer to Chapter 15, “Configuring Diagnostics Servers and Agents for HTTP Proxy.”</p>

proxy.user	This setting must match the setting for the proxyuser attribute of the < diagnosticserver> element in the probe_config.xml file. See “<diagnosticserver> element” on page 465 for more details. Also refer to Chapter 15, “Configuring Diagnostics Servers and Agents for HTTP Proxy.”
proxy.uri	This setting must match the setting for the proxy attribute of the < diagnosticserver> element in the probe_config.xml file. See “<diagnosticserver> element” on page 465 for more details. Also refer to Chapter 15, “Configuring Diagnostics Servers and Agents for HTTP Proxy.”

18

Java Agent Metrics Collectors

This section describes Java Agent metrics capture and how to configure the metric collectors.

This chapter includes:

- ▶ About Metrics Capture on page 587
- ▶ What Metrics are Being Collected by the Java Agent on page 588
- ▶ Understanding Metric Collector Entries on page 590
- ▶ About Collecting Additional Probe Metrics on page 592
- ▶ Modifying Probe Metrics Already Being Captured on page 592
- ▶ Stopping Capture of a Metric on page 592

About Metrics Capture

With the Java Agent you can configure metrics collectors by modifying the entries in the metrics configuration file, `<probe_install_dir>/etc/metrics.config`.

Note: There is a different `metrics.config` file included with the .NET Agent (see “.NET System Metrics Agent - Systems Metrics Capture” on page 575).

The system and JMX metric collectors for your agent installation are defined in the metrics configuration file. The properties and entries in the metrics configuration file, `<probe_install_dir>/etc/metrics.config`, enable you to control the metric collectors.

Note: If you update the metrics configuration file, the metric collectors automatically restarts so that your changes can take effect.

What Metrics are Being Collected by the Java Agent

In the `metrics.config` file you can see what metrics are being collected by the Java Agent.

```

metrics.config - Notepad
File Edit Format View Help

#####
### Collector configuration ###
#####

#####
#### Basic JVM collector
JVM/jvmHeapUsed = HeapUsed|bytes|Probe
JVM/jvmHeapFree = HeapFree|bytes|Probe
JVM/jvmHeapTotal = HeapTotal|bytes|Probe
JVM/jvmheapusedpct = HeapUsedPct|percent|Probe
#####

#####
#### Process Metrics Collector
ProcessMetrics/processCpuUtil=ProcessCpuUtil|percent|Probe
ProcessMetrics/processCpuUtilAbs = ProcessCpuUtilAbs|percent|Probe

#####
#### system metric collector

system/CPU = CPU|percent|System
system/MemoryUsage = MemoryUsage|percent|system
system/VirtualMemoryUsage = VirtualMemoryUsage|percent|System
system/ContextSwitchesPerSec = ContextSwitchesPerSec|count|System
system/DiskBytesPerSec = DiskBytesPerSec|bytes|Disk
system/DiskIOPerSec = DiskIOPerSec|count|Disk
system/NetworkBytesPerSec = NetworkBytesPerSec|bytes|Network
system/NetworkIOPerSec = NetworkIOPerSec|count|Network
system/PageInsPerSec = PageInsPerSec|count|System
system/PageOutsPerSec = PageOutsPerSec|count|System

```

Listing Available Metrics

The Java Agent `metrics.config` file has a feature to write a list of all the available metrics for each JMX collector into a file. When the `default.dump.available.metrics` property in the `metrics.config` file is set to `true`, the probe will write this list of available metrics to text files in the probe log directory. The files are named as follows: `<probe_install_dir>/log/<probe-id>/jmx_metrics_<collector-name>.txt`. See “List of Available JMX/PMI Metrics” on page 607 for details and examples of how to use this information as a template for configuring additional metrics capture.

Understanding Metric Collector Entries

Metric Collector entries instruct the Java Agent 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 Additional Custom System Metrics” on page 597. For information on formatting the **metric_config** property for JMX metrics, see “Creating New JMX Metrics Entries” on page 610.
- **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, 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 pane in the Diagnostics UI views.

Note: After you create the metric collector entry, 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.

About Collecting Additional Probe Metrics

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 syntax described in “Understanding Metric Collector Entries” on page 590.

See “Capturing Additional Custom System Metrics” on page 597 for details on capturing additional system metrics.

See “Capturing Additional Custom JMX Metrics” on page 606 for details on capture addition JMX metrics.

Modifying Probe Metrics Already Being Captured

You can update both the default and the custom metric 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.

19

Java Agent - System Metrics Capture

Information is provided on the process for capturing system metrics and how to configure the Java Agent system metric collector to capture them.

This chapter includes:

- ▶ About System Metrics on page 593
- ▶ System Metrics Captured by Default on page 594
- ▶ Configuring the System Metrics Collector on page 595
- ▶ Capturing Additional Custom System Metrics on page 597
- ▶ Enabling z/OS System Metrics Capture on page 603

About System Metrics

The system metric collector is installed with the Java Agent. The system metric collector gathers system level metrics, such as CPU usage and memory usage, from the agent's host. The system metric collector is configurable so you can control which 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 were 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, 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 the System Metrics Collector” on page 595 for more information. For certain platforms, such as Windows, Solaris, and Linux, you can create custom system metrics that can be gathered by the system metric collector. For details, see “Capturing Additional Custom System Metrics” on page 597.

For information on z/OS system metrics see “Enabling z/OS System Metrics Capture” on page 603.

Configuring the System Metrics 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 entries in the metrics configuration file, `<probe_install_dir>/etc/metrics.config`. See Chapter 18, “Java Agent Metrics Collectors” for general information on the metrics collector and see “Understanding Metric Collector Entries” on page 590 for an explanation of the metrics collector entries and syntax.

Note: There is a different `metrics.config` file included with the .NET Agent (see “.NET System Metrics Agent - Systems Metrics Capture” on page 575).

Note: If you update the metrics configuration file, the systems metric collector automatically restarts so that your changes can take effect.

Example System Metrics Collector Entry

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

Modifying the Default Port

The default port for the metric collector is **35000**. This value can be modified using the **system.udp.port** property if the configuration for your agent 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 agent's Web server.

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 Additional Custom System Metrics

You can capture custom system metrics on Windows, Solaris, and Linux platforms using the Java Agent system metric collector.

The following sections provide instructions for capturing the metrics and updating the entries in the system metric collector so that the custom metrics can be monitored.

This section includes:

- ▶ Capturing Custom System Metrics on Windows Hosts
- ▶ Capturing Custom System Metrics on Solaris Hosts
- ▶ Capturing Custom System Metrics on Linux Hosts

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 Java Agent system metric collector.

To add counters using the Windows System Monitor:

1 Start the Windows Performance Monitor:

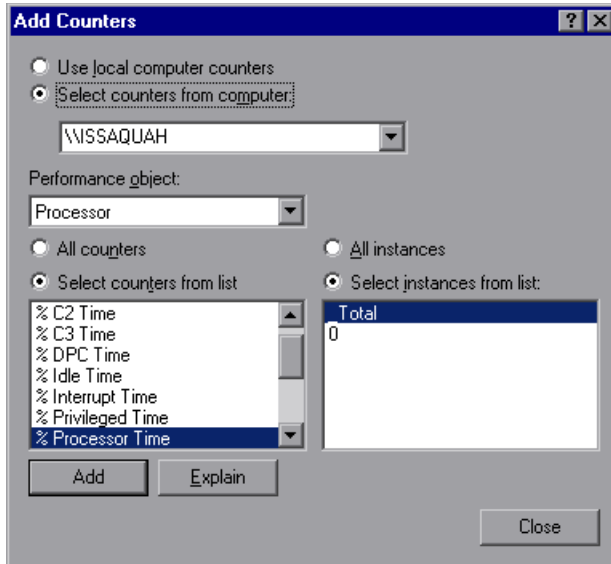
- a** Select **Start > Run** from the Start menu.
- b** In the **Open** box on the **Run** dialog box type `perfmon`.

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** Make sure 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 on the following template:

```
<collector_name>/<metric_config>= <metric_id>|<metric_units>
```

This template is described in “Understanding Metric Collector Entries” on page 590.

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 **kstat** command and identify the metric that you want to monitor.

A Solaris system metric has the following format:

```
module:instance:name:statistic
```

Here is an 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 590.

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, scan the **/proc** file system to locate the desired metric, and then create the system metric collector entry for the metric in **metrics.config** according to the location of the metric information.

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 590. 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, because this requires some system configuration changes. You must perform the following configuration steps 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 the following command:

```
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 the following command:

```
chmod -R 0+x ...
```

- 3** Start the RMF Monitor III and make sure 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** Make sure that the system is collecting SMF record 92 with subtype 5.

20

Java Agent - JMX Metrics Capture

Information is provided on the process for capturing JMX metrics and how to configure Java Agent metric collectors to capture them.

This chapter includes:

- ▶ About JMX Metrics on page 605
- ▶ Configuring WebSphere for JMX Metric Collection on page 606
- ▶ Configuring JMX Metric Collectors on page 606
- ▶ Capturing Additional Custom JMX Metrics on page 606

About JMX Metrics

The Java Agent 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 Agent can also collect JMX data from any J2EE server that supports the JMX standard.

The Java Agent 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 the Diagnostics Java Profiler.

Configuring WebSphere for JMX Metric Collection

For WebSphere JMX metric collection, you might need to configure the Performance Monitoring Service on the WebSphere server to start receiving JMX metrics.

See “Configuring WebSphere for JMX Metric Collection” on page 192 for information on how to configure WebSphere 5.x, 6.x and 7.0 servers for JMX metrics collection.

Configuring JMX Metric Collectors

The JMX metric collectors are configurable so that you can control which 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.

See Chapter 18, “Java Agent Metrics Collectors” for general information on the metrics collector and see “Understanding Metric Collector Entries” on page 590 for an explanation of the metrics collector entries and syntax.

Capturing Additional Custom JMX Metrics

The Java Agent is installed with a number of predefined JMX metric collectors for the application servers listed in “About JMX Metrics” on page 605. You configure these collectors by defining entries in the `metrics.config` file, see “Understanding Metric Collector Entries” on page 590 . You could 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 for creating new entries in the JMX metric collectors so that additional JMX metrics can be monitored.

List of Available JMX/PMI Metrics

The metric collectors installed with the Java Agent include entries for many of the JMX metrics that are available for each application server. However, there could be other JMX metrics that you could monitor, or new metrics could be exposed by the application server vendor.

In order to make it easier to configure new/additional JMX/PMI metrics for collection the `metrics.config` file has a feature to write a list of all the available metrics for each JMX collector into a file. When the **default.dump.available.metrics** property in the **metrics.config** file is set to true, the probe will write this list of available metrics to text files in the probe log directory. The files are named as follows: `<probe_install_dir>/log/<probe-id>/jmx_metrics_<collector-name>.txt`.

The **default.dump.available.metrics** property in the probe **metrics.config** file can be changed at runtime. It is recommended that the property is only set to true temporarily to write the list of available JMX/PMI metrics. After the metrics list is written to the file, the property should be set back to false (or commented out) to avoid the overhead of the probe periodically writing the metrics list to file.

In each JMX metrics list file the available MBean ObjectNames and their collectable attributes are listed as shown in the example below:

```

===== MBean ObjectNames and Available Attributes =====
MBean ObjectName:
WebSphere:J2EEServer=server1,JDBCProvider=Derby JDBC
Provider,JDBCResource=Derby JDBC
Provider,Server=server1,cell=yli87Node01Cell,diagnosticProvider=true,j2eeType=JDB
CDataSource,mbe
anIdentifier=cells/yli87Node01Cell/nodes/yli87Node01/servers/server1/
resources.xml#DataSource_12442
31364323,name=WST_PriceGen,node=yli87Node01,platform=dynamicproxy,process=
server1,spec=1.0,
type=DataSource,version=6.1.0.0
Available Attributes:
name: loginTimeout, type: int
name: statementCacheSize, type: int
name: testConnectionInterval, type: java.lang.Integer
.....

```

Use this information to add a JMX metric config in the probe etc/metrics.config file.

For example:

```
WebSphere6/WebSphere\type\=DataSource,*statementCacheSize = JDBC  
Statement Cache Size|bytes|JDBC DataSource
```

JMX GROUPBY and EXPAND_PMI Modifiers

Or, you may use the optional GROUPBY modifier to create a separate metric for each matched group of MBean ObjectNames with the same value of the key specified by GROUPBY. In the probe's etc/metrics.config file, for JMX metrics that describe an MBean object name pattern there is an optional modifier GROUPBY that can be added, which tells a JMX-based collector to treat the metric_config as multi-instance expression:

```
collector_name/GROUPBY[oname_key]/metric_config = ...
```

The collector will find all MBeans matching the metric_config and create a corresponding metric for each of them using the object name key oname_key to provide unique naming by appending it to category_id.

```
WebSphere6/GROUPBY[name]/  
WebSphere\type\=DataSource,*statementCacheSize = JDBC Statement  
Cache Size|bytes|JDBC DataSource
```

For WebSphere JMX collectors, besides the above generic MBean JMX metrics, the available WebSphere specific PMI metrics are also dumped to the WebSphere collector's dump file. This includes the PMI tree instance paths and their available statistics, and the PMI module config info.

For example:

```

===== PMI Tree and Available PMI Statistics =====
connectionPoolModule
Available Statistics:
CreateCount, CloseCount, AllocateCount, ReturnCount, PoolSize, FreePoolSize,
WaitingThreadCount, FaultCount, PercentUsed, PercentMaxed, UseTime, WaitTime,
ManagedConnectionCount, ConnectionHandleCount, PrepStmtCacheDiscardCount,
JDBCTime
connectionPoolModule->Derby JDBC Provider
Available Statistics:
CreateCount, CloseCount, AllocateCount, ReturnCount, PoolSize, FreePoolSize,
WaitingThreadCount, FaultCount, PercentUsed, PercentMaxed, UseTime, WaitTime,
ManagedConnectionCount, ConnectionHandleCount, PrepStmtCacheDiscardCount,
JDBCTime
connectionPoolModule->Derby JDBC Provider->jdbc/ALBUM
Available Statistics:
CreateCount, CloseCount, AllocateCount, ReturnCount, PoolSize, FreePoolSize,
WaitingThreadCount, FaultCount, PercentUsed, PercentMaxed, UseTime, WaitTime,
ManagedConnectionCount, ConnectionHandleCount, PrepStmtCacheDiscardCount,
JDBCTime

```

Use the above PMI information to add new PMI metric/statistic configs to the probe etc/metrics.config file.

For example:

```

WebSphere6/connectionPoolModule.CreateCount = JDBC Connection
Creates|count|JDBC ConnectionPools

```

```

WebSphere6/[connectionPoolModule][Derby\ JDBC\ Provider][jdbc/
ALBUM].AllocateCount = JDBCConnection Allocates|count|JDBC
ConnectionPools

```

Or, you may use the optional EXPAND_PMI modifier to group PMI metrics similar to how you group JMX metrics.

For PMI, the EXPAND_PMI modifier is specified to expand the PMI tree from the given module or StatDescriptor branch by the specified level. The expansion level "n" can be 1, 2, ..., or *, with the default level of 1 and * means expand all:

```

collector_name/EXPAND_PMI[n]/metric_config = ...

```

For example:

```
WebSphere6/EXPAND_PMI[*]/connectionPoolModule.AllocateCount = JDBC  
Connection Allocates|count|JDBC ConnectionPools
```

creates "JDBC Connection Allocates" metric for each JDBC connection pool provider and for each DataSource of the provider.

Creating New JMX Metrics Entries

The following instructions guide you through the process of creating the JMX metric entries according to the following template:

```
<collector_name>/<metric_config>= <metric_id>|<metric_units>
```

This template is described in “Understanding Metric Collector Entries” on page 590.

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 Agent.
- 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.
 - a 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>.<attribute name>
```

- `<MBean object name pattern>` is the string representation of the object name of an MBean.
- `<attribute name>` is the name of the MBean attribute that represents the metric. If `<attribute name>` has any '.' in it, it should be surrounded by parenthesis: `<MBean object name pattern>.(<attribute name>)`

As an 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 612.

- b** For WebSphere PMI metrics, the `<metric_config>` parameter is a pattern that the collector uses to find the matching PMI statistics. The pattern consists of two components separated by the `'.'` character.

```
<PMI StatDescriptor>.<statistics name>
```

- `<PMI StatDescriptor>` is used to locate and access particular Stats in the WebSphere PMI tree. It can be either a PMI module name (for example, `webAppModule`), or a PMI module branch (for example, `[webAppModule][AccountManagement#AccountManagementWar.war]`)
- `<statistics name>` is the name of the PMI statistics that represent the metric. If statistics name has any `'.'` in it, it should be surrounded by parenthesis:
`[webAppModule][AccountManagement#AccountManagementWar.war].(webAppModule.numLoadedServlets)`

See “JMX GROUPBY and EXPAND_PMI Modifiers” on page 608 for an example of the PMI module and PMI module branches and their available statistics names.

- 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 resolve 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 (for example, 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

This section includes:

- ▶ Setting Up the Integration Between Business Service Management and Diagnostics
- ▶ Installing the LoadRunner Diagnostics Add-in
- ▶ Setting Up LoadRunner and Diagnostics Integration
- ▶ Setting Up Performance Center to Use Diagnostics

21

Setting Up the Integration Between Business Service Management and Diagnostics

Information is provided on setting up the integration between HP Business Service Management and Diagnostics.

Note: This information is for Business Service Management 9.0 unless otherwise stated.

This chapter includes:

- ▶ About Setting Up the Integration Between Business Service Management and Diagnostics on page 617
- ▶ Registering the Diagnostics Server in Business Service Management 9.0 on page 618
- ▶ Changing the Diagnostics Server Details on page 624
- ▶ Understanding the Diagnostics Admin Page on page 624
- ▶ Assigning Permissions for Diagnostics Users in Business Service Management on page 626
- ▶ Accessing the Diagnostics Pages in Windows 2003 on page 627
- ▶ Accessing the Diagnostics Application from Business Service Management on page 628
- ▶ Data Samples Sent to Business Service Management on page 628

Chapter 21 • Setting Up the Integration Between Business Service Management and Diagnostics

- ▶ Diagnostics Populates CIs and Models in Business Service Management on page 629
- ▶ Synchronize CIs Between Diagnostics and Business Service Management on page 629
- ▶ Diagnostics Provides KPI/HI Coloring to Business Service Management on page 630

About Setting Up the Integration Between Business Service Management and Diagnostics

Before using HP Diagnostics with Business Service Management, you provide Business Service Management with the information that it needs to communicate with the Diagnostics components.

To set up Diagnostics in Business Service Management:

1 Specify the Diagnostics Server details.

Enter the Diagnostics Server details in Business Service Management. For more information, see “Registering the Diagnostics Server in Business Service Management 9.0” on page 618.

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 in Business Service Management” on page 626.

3 For Windows 2003 only: Change your Internet browser settings.

When your Internet browser is running in a Windows 2003 environment, you must change your Internet browser settings to access the Diagnostics configuration and application pages in Business Service Management. See “Accessing the Diagnostics Pages in Windows 2003” on page 627.

4 Enable cookies on the Diagnostics Commander host.

Cookies must be enabled to view Diagnostics data in Business Service Management. This can usually be accomplished by adding the registered Diagnostics Commander as a trusted site in the browser configuration.

Registering the Diagnostics Server in Business Service Management 9.0

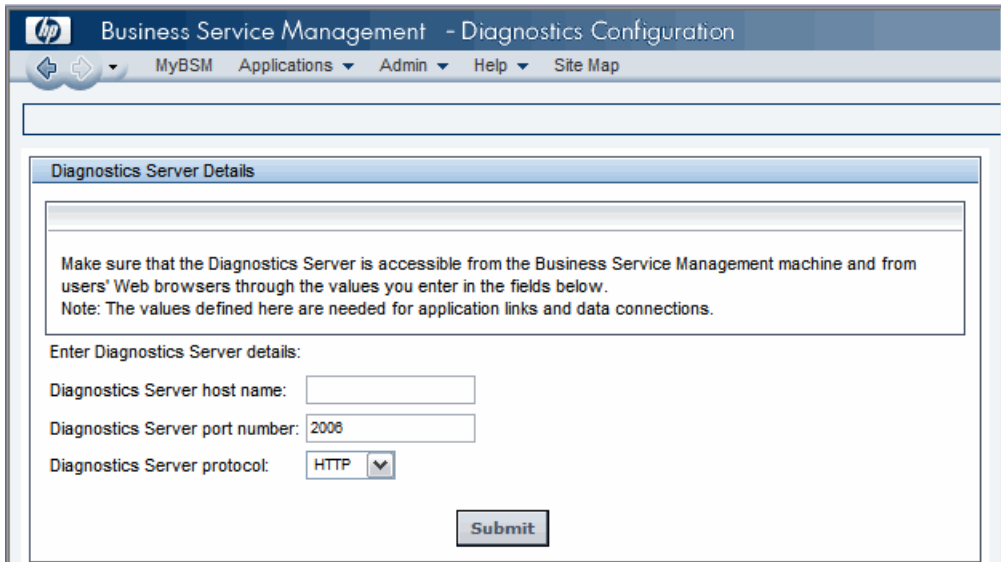
To make HP Diagnostics accessible from Business Service Management, you register the Diagnostics Server. The following section describes the steps for registering in Business Service Management 9.0. Differences for BSM 8.x are noted.

Note: After a Business Service Management upgrade, you must re-register the Business Service Management-Diagnostics integration in the **Admin > Diagnostics** page.

Note: If you are using Windows 2003, you must configure your Internet browser settings to access the Diagnostics Admin page. For more details, see “Accessing the Diagnostics Pages in Windows 2003” on page 627.

To initially register the Diagnostics Server in Business Service Management:

- 1 Log on to Business Service Management.
- 2 Select **Admin > Diagnostics**. When you are first setting up the integration, the Diagnostics Server Details page is displayed.



The screenshot shows a web browser window with the title "Business Service Management - Diagnostics Configuration". The browser's address bar shows "MyBSM Applications Admin Help Site Map". The main content area is titled "Diagnostics Server Details" and contains the following text:

Make sure that the Diagnostics Server is accessible from the Business Service Management machine and from users' Web browsers through the values you enter in the fields below.
Note: The values defined here are needed for application links and data connections.

Enter Diagnostics Server details:

Diagnostics Server host name:

Diagnostics Server port number:

Diagnostics Server protocol:

Note: If you try to access HP Diagnostics (by clicking **Diagnostics** on the Site Map page or by selecting **Applications > Diagnostics**) before you configure the Diagnostics Server, you will 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 command server.
 - **Diagnostics server host name.** Enter the name of the machine that is host to the Diagnostics command server.

Even if the Diagnostics Server is installed on the same system as Business Service Management, 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 Service Management will be accessed through a fully-qualified domain name, register the Diagnostics Server host with a fully-qualified domain name.

- ▶ **Diagnostics server port number.** Enter the port number used by the Diagnostics command server. The default port number is **2006**.
- ▶ **Diagnostics server protocol.** Select the communication protocol through which Business Service Management connects to Diagnostics, either **HTTP** or **HTTPS**.

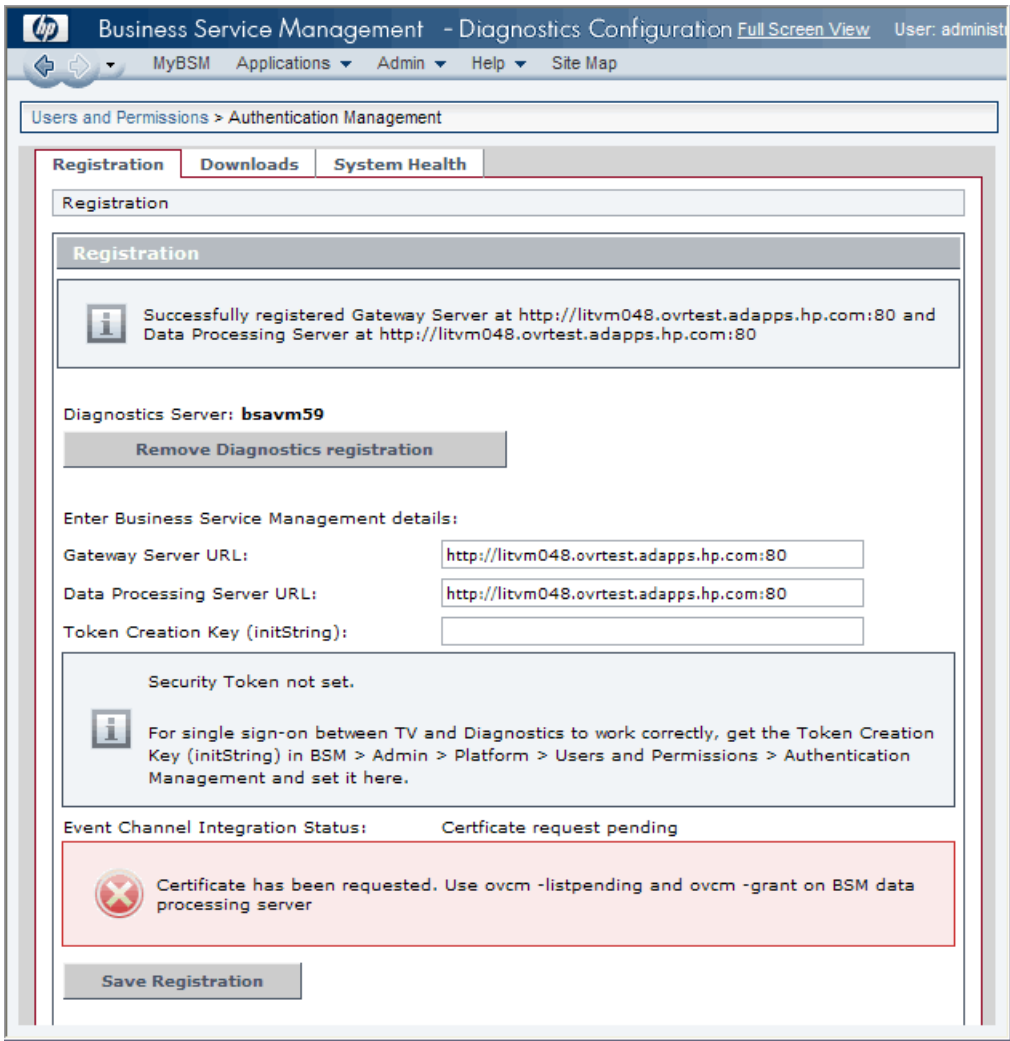
Note: If you select **HTTPS** as your communication protocol, additional configuration steps are required. For more information about the steps required, see Chapter , “Enabling HTTPS Between Components.”

- 4** After you enter the Diagnostics Server details and verify that these details are accurate, click **Submit** to complete the Diagnostics Server configuration process.

If the server name you 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 Service Management and the Business Service Management server details are automatically registered on the Diagnostics Server machine.

- 5 The **Registration** tab in the Diagnostics Configuration page opens, displaying the Business Service Management server details that were available.



Where necessary, you can manually change the Business Service Management server details in the **Enter Business Service Management details** section of the Registration tab.

Gateway Server URL. Verify that the root URL, matches the root URL that you use to access Business Service Management.

Data Processing Server URL. Verify that the root URL, matches the root URL that you use to access Business Service Management. If the Data Processing Server URL is different from the Gateway Server URL, typically you would use port 8080 in this URL.

Token Creation key (initString). If you are using TransactionVision, then to avoid a user having to login again when drilling from TransactionVision to Diagnostics you enter the Business Service Management token creation key in the field. Entering the token key provides the key to Diagnostics. If you are not using TransactionVision then this is not required.

You can find the Token Creation Key (initString) in Business Service Management in **Admin > Platform > Users and Permissions > Authentication Management**. Once you complete the registration, this token creation key is written to the Diagnostics **lwso.properties** file.

Single Sign-On Configuration	
Name	Value
Single Sign-On Mode	Lightweight
Token Creation Key (initString)	4G590seBB1rb

Event Channel Integration Status. When integrating Diagnostics with Business Service Management 9.0 this channel is used by Diagnostics (actually the OM agent and IAPA components are used) to send Health Indicator status events to the Business Service Management gateway server.

When the certificate request is pending you need to do the following to complete the event channel integration. The registration executes a script to generate the certificate (`<Diagnostics_install_dir>/server/bin/switch_ovo_agent.vbs` or `.sh`). But you must do the following additional steps to manually grant the certificate.

To manually grant the certificate:

Go to the Business Service Management Data Processing server and grant the OM agent’s certificates. If there is more than one certificate listed select the appropriate one.

ovcm -listpending -l

ovcm -grant <core id from above output>

See “OM Agent Troubleshooting” on page 774 if you have any problems with the OM Agent and IAPA installation or the certificates.

Note: When integrating Diagnostics with Business Service Management 8.x you will see Event Channel Integration Status: N/A since the OM agent and IAPA components are not used by Diagnostics when integrated with Business Service Management 8.x.

- 1 Click **Save Registration** and verify the Event Channel Integration Status is OK and the other values are correct.

hp Business Service Management - Diagnostics Configuration

MyBSM Applications Admin Help Site Map

Registration Downloads System Health

Registration

Registration

Diagnostics Server: **OVNTT150**

Remove Diagnostics registration

Enter Business Service Management details:

Gateway Server URL:

Data Processing Server URL:

Token Creation Key (initString):

Event Channel Integration Status: OK

Save Registration

Changing the Diagnostics Server Details

You can 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 the **Remove Diagnostics registration** button.
- 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 “Registering the Diagnostics Server in Business Service Management 9.0” on page 618.

Understanding the Diagnostics Admin Page

You access the Diagnostics Configuration page by selecting **Admin > Diagnostics**. The Diagnostics Admin page in Business Service Management consists of the following three tabs, which are described in this section:

Registration Tab

The Registration tab displays the following details:

- ▶ The Diagnostics Server details that you registered in Business Service Management. To change these details, see “Changing the Diagnostics Server Details” on page 624.
- ▶ The Business Service Management server details that were automatically registered on the Diagnostics Server machine. You can manually change the Business Service Management server details in the **Enter Business Service Management details** section.

For information about registering Diagnostics in Business Service Management for the first time, see “Registering the Diagnostics Server in Business Service Management 9.0” on page 618.

Downloads Tab

The Downloads tab provides links to the Diagnostics Agent and Collector installers, enabling you to download the agent or collector installation file for your relevant platform.

If you did not specify the path to the Agent and 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.”

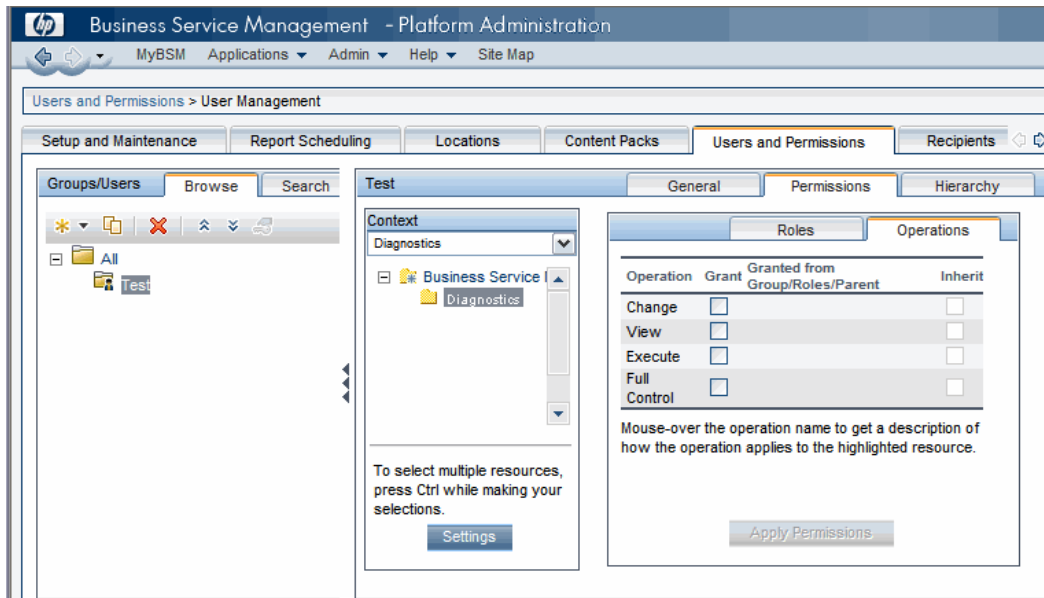
System Health Tab

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 in Business Service Management

Business Service Management 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 to Diagnostics users.

The following is an example of the **Admin > Platform > Users and Permissions > User Management** page in Business Service Management:



Permissions for Diagnostics are under a context called "Diagnostics". The tree shows Business Service Management and under it Diagnostics. In Business Service Management 8.x permissions for Diagnostics are under the Monitors context.

When applying permissions in Business Service Management, 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 Business Service Management.
- ▶ **Execute:** Enables setting thresholds in Diagnostics.
- ▶ **Full Control:** Enables performing all operations on Diagnostics, and granting and removing permissions for those operations.

Diagnostics permissions can be inherited from BSM roles.

For detailed information about how to assign user permissions in Business Service Management, see *Platform Administration* in the *HP Business Service Management Documentation Library*.

Accessing the Diagnostics Pages in Windows 2003

When your Internet browser is running in a Windows 2003 environment, you must change your Internet browser settings to access the Diagnostics configuration and application pages in Business Service Management.

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 Service Management, enter the IP address. If you entered a host name in Business Service Management, enter the host name.
 - ▶ Include the `http://` or `https://` prefix, and the port number, as illustrated in the following example:
`http://<Diagnostics_Commander>: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.

Accessing the Diagnostics Application from Business Service Management

Once you've set up Business Service Management to use Diagnostics you will be able to access the Diagnostics UI from within Business Service Management.

In Business Service Management, select **Applications > Diagnostics** to open the Diagnostics UI.

With the integration between Business Service Management and Diagnostics, data from Diagnostics is sent to Business Service Management.

You can see this Diagnostics data for application infrastructure elements and business transactions in various views within Business Service Management. And because Diagnostics and Business Service Management share a common data model you can select in context drill downs to Diagnostics directly from within Business Service Management.

For selected CIs in Business Service Management populated by Diagnostics you can right-click to select Drill to Diagnostics. In various reports in Business Service Management you can select an icon for an in context drill down to Diagnostics.

Data Samples Sent to Business Service Management

When you integrate Diagnostics with Business Service Management, Diagnostics monitors your enterprise applications and sends application performance and availability data to Business Service Management as **Data Samples**. See *HP Diagnostics User's Guide* section on Integrations for more information.

Diagnostics provides the following data samples to Business Service Management:

- ws_perf_aggr_t (SOA Sample)
- ws_event_aggr_t (SOA Sample)
- appmon_vu_t (Transaction (BPM) Sample)

- ▶ dg_trans_t (Business Transaction (Diagnostics) sample)

See the *HP Business Service Management Documentation Library* for more information on Data Samples.

Diagnostics Populates CIs and Models in Business Service Management

With Diagnostics 9.0 or later, Diagnostics populates CIs and model relationships in the Business Service Management Operational Database (ODB) for application infrastructure elements and business transactions. See *HP Diagnostics User's Guide* section on Integrations for more information.

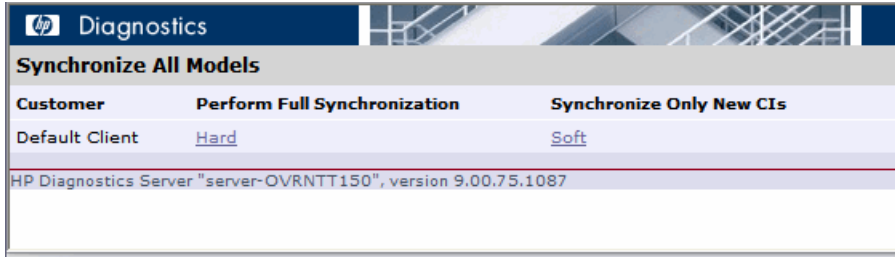
For ASP.NET applications see “Discovery of IIS Metadata for CI Population of IIS Deployed ASP.NET Applications” on page 229 for information on discovery of IIS metadata required for ODB population and running a Rescan if you modify an existing ASP.NET application deployment.

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.

Synchronize CIs Between Diagnostics and Business Service Management

If you need to force a synchronization between Diagnostics and Business Service Management for Diagnostics populated models (CIs), a `synchronize` function is available on the Diagnostics Server.

From the main Diagnostics UI select **Configure Diagnostics** (or from any Diagnostics view select the **Maintenance** link in the top right corner) and the Components page is displayed. Select the **synchronize** link to display the page for synchronizing models.



Anytime a Business Service Management system is upgraded or re-installed, a manual hard sync is needed (or a wait period of 24 hours) before CIs from Diagnostics are forwarded to Business Service Management. To do a hard sync, select **Hard**.

Diagnostics Provides KPI/HI Coloring to Business Service Management

Health Indicator status (coloring) for business transaction and web service CIs populated by Diagnostics is **metric-based**. Status for metric based KPIs and Health Indicators is sent to Business Service Management from Diagnostics in data samples. Diagnostics sends data samples to Business Service Management and rules in Business Service Management are used to evaluate the data and set the indicator's status.

You can change default objectives for business transaction and Web service Health Indicators in Business Service Management **Admin > Service Health**. See the *Business Service Management Documentation Library* for information on using Service Health Admin.

Health Indicator status (coloring) for application infrastructure CIs populated by Diagnostics is **event-based**. Status for event-based Health Indicators is sent to Business Service Management from Diagnostics when there is a threshold violation on relevant metrics.

The threshold violation event data is sent to Business Service Management using the OM agent and IAPA components installed with the Diagnostics Commander Server.

See Chapter 2, “Installing the Diagnostics Server” for information on installing the OM Agent/IAPA components. See “Registering the Diagnostics Server in Business Service Management 9.0” on page 618 for details on checking the Event Channel status. See “Event Based Health Indicator Status Troubleshooting Flow” on page 776 if there are issues.

See *HP Diagnostics User’s Guide* section on Integrations for overview information about KPI and HI status coloring and Business Service Management views populated with Diagnostics application and infrastructure performance data.

22

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 634
- ▶ Installing the LoadRunner Diagnostics Add-in on page 634

Before Installing the LoadRunner Diagnostics Add-in

Before installing the LoadRunner Diagnostics Add-in, Diagnostics Server in Commander mode and LoadRunner must be installed. To install LoadRunner, see the *HP LoadRunner Installation Guide*.

You need to use the Diagnostics 9.0x LoadRunner add-in with a Diagnostics 9.0x Server.

Note: Earlier versions of the LoadRunner add-in will NOT work against a Diagnostics 9.0x server.

Installing the LoadRunner Diagnostics Add-in

The LoadRunner Diagnostics Add-in is installed on the LoadRunner Controller host machine.

Note: The LoadRunner Diagnostics add-in uses 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.

LoadRunner Controller 9.10 J2EE\,.NET Diagnostics AddIn Setup

Registration Information

Please type your name, the name of your company and your maintenance number. The maintenance number was provided with your LoadRunner or Add-In package.

Name:

Company:

Maintenance number:

InstallShield

< Back Next > Cancel

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 **iKernel.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 must configure LoadRunner and provide the necessary information to enable communication with the Diagnostics components. See Chapter 23, “Setting Up LoadRunner and Diagnostics Integration” for information on configuring LoadRunner for integration with Diagnostics.

23

Setting Up LoadRunner and Diagnostics Integration

General information is provided on configuring LoadRunner and Diagnostics integration in load testing and offline analysis.

This chapter includes:

- ▶ About Setting Up LoadRunner to Use Diagnostics on page 638
- ▶ Configuring LoadRunner Scenarios to use HP Diagnostics on page 638
- ▶ Selecting Probe Metrics to Include in the Offline Analysis File on page 639
- ▶ Improving Transfer of Large Offline Analysis Files on page 642

About Setting Up LoadRunner to Use Diagnostics

Before you can access the Diagnostics UI from within LoadRunner, you must provide LoadRunner with the information it needs to communicate with the Diagnostics components.

To make the Diagnostics UI accessible from LoadRunner, you must configure the integration with the Diagnostics Server. You only need to specify the Diagnostics Server details the first time you use LoadRunner with Diagnostics. See the *HP LoadRunner Controller User Guide* for details about configuring LoadRunner to access the Diagnostics Server.

Note: Before specifying the Diagnostics Server details, make sure that the LoadRunner Controller is closed. When the Controller is open, you can view the Diagnostics configuration settings, but you cannot change them.

Configuring LoadRunner Scenarios to use HP Diagnostics

Each time you want to capture Diagnostics metrics in a load test scenario, you must 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 saved a scenario with the Diagnostics settings already configured, you do not need to reconfigure the Diagnostics parameters each time you run that scenario.

Selecting Probe Metrics to Include in the Offline Analysis File

Diagnostics probe metric data can be included for use in LoadRunner Offline Analysis.

By default, only *HeapUsed*, *GC Collections/sec* and *GC time Spent in Collections* metric data is included in offline Analysis. To select additional probe metrics to be included in offline Analysis (.eve file), you use the Diagnostics configuration file, **etc/offline.xml**.

Configure the offline.xml files on the Diagnostics mediators to specify the Diagnostics probe metrics that you want to be included in the offline analysis. You would configure this file for all mediators that have probes participating in runs.

Probe metrics are only available from the Java probe, not the .NET probe.

Note: Make sure that the clocks are synchronized between the Diagnostic servers and LoadRunner.

A list of all the possible Diagnostics metrics can be found in **metrics.config** in the Diagnostics probe install directory. This file contains all metrics, although some of these metrics might not be available for offline analysis, depending on the platform (Java probe only), application server type, and version being monitored.

In general, all the metrics that you can see under a Java probe in the Diagnostics UI Details pane can be used in the offline.xml file for the relevant mediator. The following collectors that are included in the metrics.config file cannot be used in the offline.xml file: "system" and "Mercury System".

An example of the **offline.xml** file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<probeMetrics xmlns="http://hp.com/diagnostics/offline/1.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="offline.xsd">
<metric>
  <name>HeapUsed</name>
</metric>
<metric>
  <name>GC Collections/sec</name>
</metric>
<metric>
  <name>GC Time Spent in Collections</name>
</metric>
</probeMetrics>
```

The `<metric>` element specifies a match condition. Matching is possible on metric name (`<name>`), category (`<category>`) and collector (`<collector>`). The `<name>`, `<category>` and `<collector>` elements can be combined, in which case all elements must match.

Matching examples:

Match and include a metric named "HeapUsed"

```
<metric>
  <name>HeapUsed</name>
</metric>
```

Match and include any metric that the "JVM" collector exposes.

```
<metric>
  <collector>JVM</collector>
</metric>
```

Match and include a metric named "HeapFree" that the "JVM" collector exposes.

```
<metric>
  <name>HeapFree</name>
  <collector>JVM</collector>
</metric>
```


By default, the matching is performed on a substring, meaning the specified text between <name>, <category> and <collector> needs to be within (or contain) the actual metric name, category or collector. For example, specifying "HeapFree" text for <name> would match any metric that has "HeapFree" in its name (for example "MyHeapFreeMetric", "YourHeapFreeMetric").

It is further possible to specify a regular expression for <name>, <category> and <collector> via the match attribute on <metric> (for example <metric match="regex">). Note, regular expressions are expensive and will impact the time it takes to write out .eve files.

Specify sending all probe metrics to LoadRunner as follows:

```
<metric match="regex">  
  <collector>.*</collector>  
</metric>
```

Note: Adding lots of metrics in offline.xml will increase the size of the offline (.eve) file, which in turn impacts the time it takes to analyze the offline file in the LoadRunner Analysis application.

Changes to the offline.xml file are automatically detected and applied every 15 seconds. Configuration errors (validated against offline.xsd) are logged to the **server.log** file.

Improving 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 enable you to define how many five second trend points are to be aggregated together to produce a single sample for offline analysis.

24

Setting Up Performance Center to Use 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 Diagnostics on page 644
- ▶ Configuring Performance Center Load Tests to Use Diagnostics on page 645
- ▶ Managing Performance Center Offline Files on page 646

About Setting Up Performance Center to Use Diagnostics

Performance Center and Diagnostics are integrated products that are 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 command server.
- ▶ **Port Number.** The port number used by the Diagnostics command server. 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 659.

- ▶ **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 Appendix C, “Enabling HTTPS Between 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.

Configuring Performance Center Load Tests to Use Diagnostics

Each time you want to capture Diagnostics metrics in a load test, you must 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 must 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 559.

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 must 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

This section includes:

- Diagnostics Administration UI
- User Authentication and Authorization
- Enabling HTTPS Between Components
- Using the System Health Monitor
- Diagnostics Data Management
- Diagnostics Technical Diagrams
- Upgrade and Patch Install Instructions
- Troubleshooting HP Diagnostics
- General Reference Information
- Data Exporting

A

Diagnostics Administration UI

Information is provided on how to access and use the Diagnostics Server Administration UI, where you can configure Diagnostics properties and manage your Diagnostics software.

This chapter includes:

- Accessing the Diagnostics Administration UI on page 649
- Using the Diagnostics Administration UI on page 652

Accessing the Diagnostics Administration UI

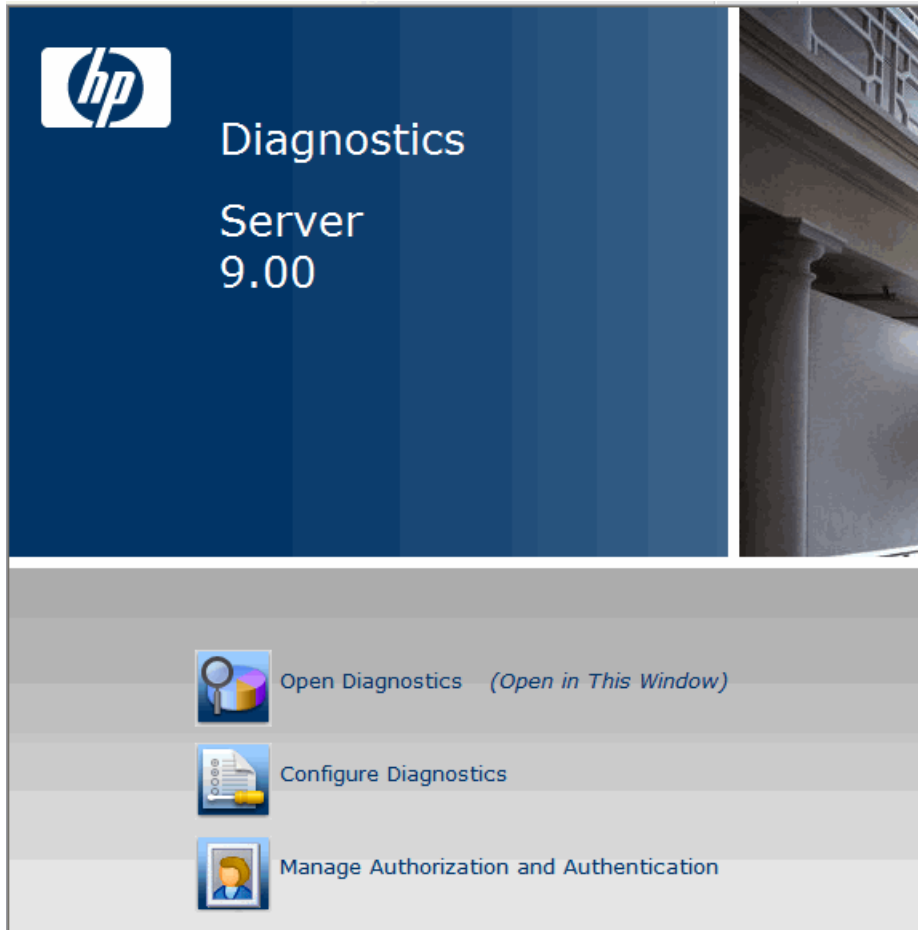
You can view information about the Diagnostics configuration, set the user privileges, configure Diagnostics settings and manage your Diagnostics software directly from the main UI of Diagnostics.

To access the Diagnostics Administration UI:

- 1 Open the main Diagnostics UI 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.

If you are 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.”

The main Diagnostics UI opens in your browser.



There are three options as described below.

- **Open Diagnostics.** Opens the Diagnostics UI where you can view the performance metrics collected by the agents that are reporting to the Diagnostics Server. The performance metrics are displayed in Diagnostics views.

For more information about the Diagnostics views, see the online help or the *HP Diagnostics User's Guide*.

- **Configure Diagnostics.** Opens the Components administration page, which has a link to the Diagnostics Server Configuration page.

For more information about configuring the Diagnostics properties, see “Making Server Configuration Changes” on page 654.

- **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 657.

- 2 From the main Diagnostics UI select **Configure Diagnostics**. See “Using the Diagnostics Administration UI” on page 652 for details.

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 must close your browser and reopen it.

Using the Diagnostics Administration UI

In the Diagnostics Administration UI, you view information about your Diagnostics configuration, 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.

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.

From the main Diagnostics UI select **Configure Diagnostics** and the **Components** page is displayed.

Component Name	Component Description
registrar	Central list of all Diagnostics component deployments
query	Query API - allows you to download diagnostics data in HTML, XML or as Java objects
security	Built-In User Management
logging	Configure log files and logging details.
configuration	Configuration
files	Installation directory browser - upload and download property files, log files, etc
license	License Management
synchronize	Synchronize uCMDB Models in BSM
BizTxnCreate	Business Transaction Creation

[\(Show Advanced Options\)](#)

HP Diagnostics Server "server-OVRNTT150", version 9.00.70.1005

You can also access this Components page by selecting the Maintenance link in any Diagnostics view.

You can select from the following links to go to different administration pages for Diagnostics. Some of the links take you to information pages and other links allow you to make configuration changes.

- ▶ **Registrar.** Central list of all Diagnostics component deployments.
- ▶ **Query.** Query API which enables 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.

If you select the **Active Users** link at the bottom of the initial query page you can get a list of active users seen by the Diagnostics server in the last 60 seconds. And you can see the Queries/sec indicating how much load the user generates with summary or trend queries.

- **Security.** Built-In User Management. See “Understanding the Diagnostics Server Permissions Page” on page 662.
- **Logging.** Configure log files and logging details.
- **Configuration.** Configure the Diagnostics Server. See “Making Server Configuration Changes” on page 654 for more information on additional configuration pages.
- **Files.** Installation directory browser for use in uploading and downloading property files, log files and other files.
- **License.** License management. See “Licensing HP Diagnostics” on page 73 for details.
- **Synchronize.** Synchronize CIs with Business Service Management. You can force a hard sync (perform full synchronization with Business Service Management) or soft sync (synchronize only new CIs with Business Service Management).
- **BizTxnCreate.** Use to create business transactions from server requests. See the *HP Diagnostics User’s Guide* section on Business Transactions for details.

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.

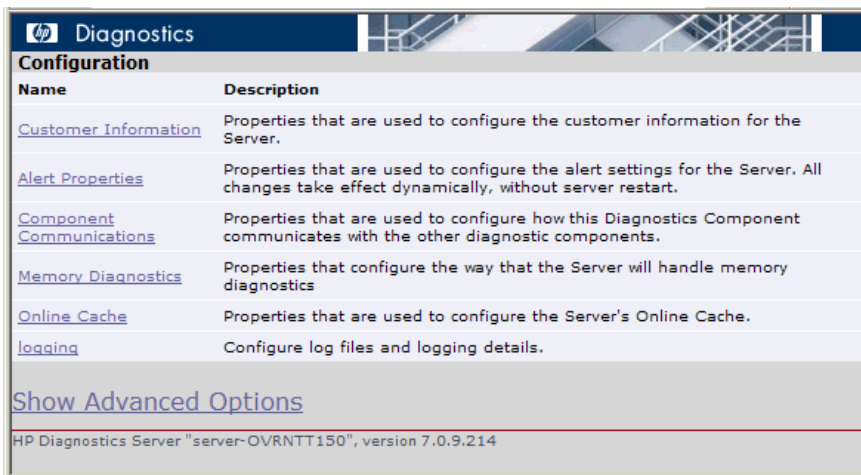
To hide the advanced options:

- 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**.

Making Server Configuration Changes

From the main Diagnostics UI, select **Configure Diagnostics** and then select the **configuration** link to access the Configuration page shown below.



- 1 Click the link to the page whose properties you want to update. For the Diagnostics Server you can configure:
 - Customer information
 - Alert properties
 - Component Communications
 - Memory Diagnostics
 - Online cache
 - Logging

The configuration options displayed on this page are the commonly configured options. The more advanced configuration options are hidden by default. Select **Show Advanced Options** to see more configuration options.

Important: Do not manipulate the advanced options without the guidance of your HP Software Customer Support representative.

- 2** For example if you select Customer Information the page below is displayed. Review the properties that are displayed and make updates.

Name	Value	Description	Default Value
Customer Name	<input type="text" value="Default Client"/>	The name of the customer that is associated with the Server. This information is only used for HP Managed Services.	Default Client

[Show Advanced Options](#)

- 3** 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 could 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 on page 658
- ▶ Understanding User Privileges on page 659
- ▶ Understanding Roles on page 660
- ▶ Accessing Diagnostics Using Default User Names on page 661
- ▶ Understanding the Diagnostics Server Permissions Page on page 662
- ▶ Creating, Editing and Deleting Users on page 669
- ▶ Assigning Privileges Across the Diagnostics Deployment on page 672
- ▶ Assigning Privileges for Probe Groups on page 673
- ▶ Authentication and Authorization for Users of Integrated HP Software Products on page 676
- ▶ Tracking User Administration Activity on page 677
- ▶ List of Active Users on page 679
- ▶ Configuring Diagnostics to use JAAS on page 679

About User Authentication and Authorization

User authentication and authorization settings for all the Diagnostics components are configured in the Diagnostics Commander.

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 Commander. 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.

Important:

- ▶ When you install the agent as a profiler only (not connected to any Diagnostics Server), you manage the authentication and authorization of users of the Profiler in the agent itself.
- ▶ For information about managing authentication and authorization for the Java Agent installed as a profiler only, see “Authentication and Authorization for Java Diagnostics Profilers in Standalone Mode” on page 428.
- ▶ For information about managing authentication and authorization for the .NET Agent installed as a profiler only, see “Authentication and Authorization for .NET Profilers in Standalone Mode” on page 541.

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 Commander 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, 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). **System** permission is required 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 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 672.

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 set up 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 agent system that they own and view permission on the agent system that they don't own. Both teams should have view and execute permissions for the UI.

The following roles must 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 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].

Accessing Diagnostics Using Default User Names

The following default user names are 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 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

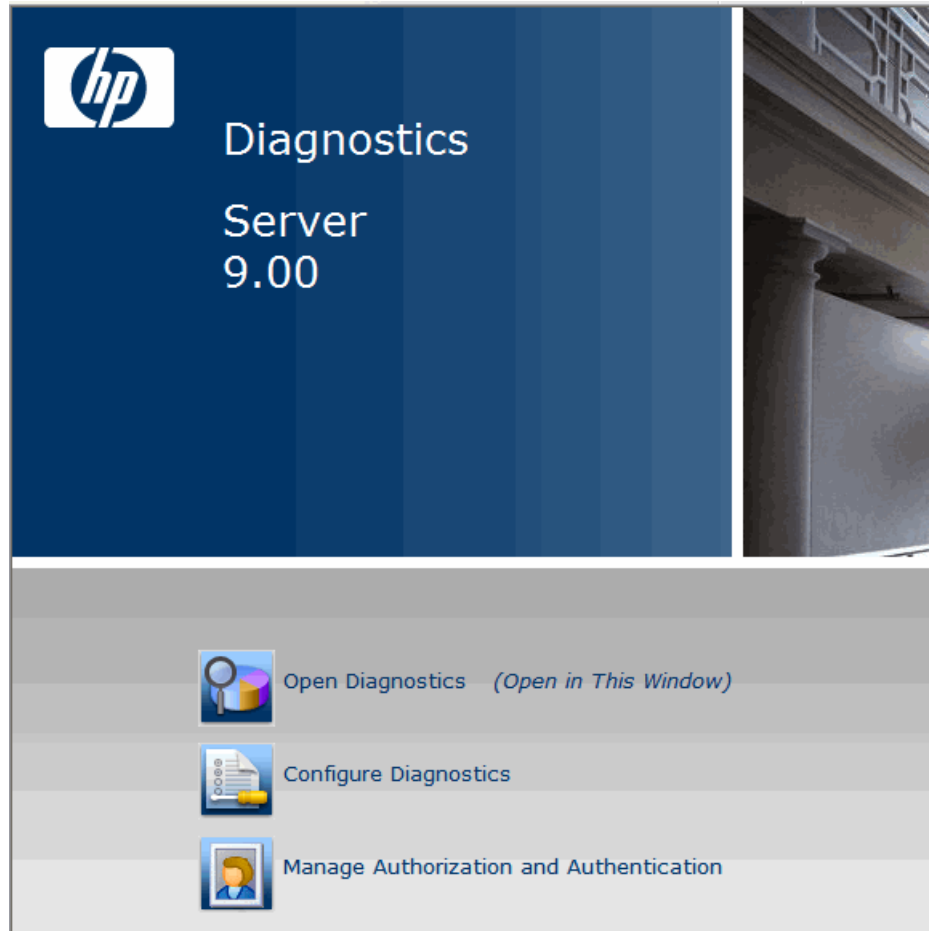
You manage users and assign user privileges in the Permissions page.

This section includes:

- ▶ “Accessing the Permissions Page” that follows.
- ▶ “The Permissions Page at a Glance” on page 665
- ▶ “Enterprise and Application Permissions” on page 666

Accessing the Permissions Page

You can access the Permissions page from the main Diagnostics UI by selecting **Manage Authorization and Authentication**.



You can also access this Permissions page by selecting the Maintenance link in any Diagnostics view and then selecting the security link.

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 Commander using a user name that has valid security access with the appropriate privileges.

When the Permissions page opens, if you are not already signed into the Diagnostics Server, you might be prompted for a user name and password. You must have at least **View** privileges 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 agents.

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.

Note: Diagnostics has a centralized permissions system whereby permissions can be set for a user and they will apply to all distributed servers and probes connected to the Diagnostics system. However, permissions are only pushed out to the distributed components once every 5 minutes, so permission changes do not take effect immediately.

- ▶ **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	modify	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			

Appendix B • User Authentication and Authorization

Area and Action	Enterprise Permissions			Application Permissions		
	view	execute	change	view	modify	edit screens
Access maintenance page			X			
Working with incidents	X					
Working with customer views	X					
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	

Area and Action	Enterprise Permissions			Application Permissions		
	view	execute	change	view	modify	edit screens
Remove entity from application	X				X	
Edit entity properties (thresholds, comments, etc)		X		X		
View application	X			X		
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 663.

- 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 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 672.

To assign roles:

- 1 Access the Diagnostics Server Permissions page as described in “Accessing the Permissions Page” on page 663.
- 2 On the User Administration page, assign the roles for a user. Make sure that the roles are enclosed in brackets ([aRole]). Roles can be separated by comma ([Role1],[Role2]).

Note: Permissions must be set up for roles under the Enterprise and/or Per Probe Group dialogs (see “Assigning Privileges Across the Diagnostics Deployment” on page 672 and “Assigning Privileges for Probe Groups” on page 673).

To delete a user:

- 1** Access the Diagnostics Server Permissions page as described in “Accessing the Permissions Page” on page 663.
- 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 663.
- 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 663.
- 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 659.

To assign user privileges across the entire Enterprise:

- 1** Access the Diagnostics Server Permissions page as described in “Accessing the Permissions Page” on page 663.
- 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 enables 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 669.

- 4** Add the privileges to the username as comma separated values.

For example, if you defined a user by the name of **newuser** and you want to assign this user with view and execute privileges you must 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 661. 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 659.

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 could take up to 1 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 663.
- 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 enables 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 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, 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 663.
- 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 enables 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 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, 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 Service Management, 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 Service Management Users” on page 676
- ▶ “Authentication and Authorization for Performance Center and LoadRunner Users” on page 677

Authentication and Authorization for Business Service Management Users

In Business Service Management, you can define user permissions for Diagnostics. For more information, see “Assigning Permissions for Diagnostics Users in Business Service Management” on page 626.

When an existing or new Business Service Management user opens Diagnostics from Business Service Management, their permissions are picked up from the Business Service Management session and copied into the Diagnostics permissions system (under the SaaS customer, if relevant).

Updates to Business Service Management 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 reopened).

Business Service Management passwords are never sent to Diagnostics—Diagnostics trusts a successful Business Service Management login.

If privileges of Business Service Management users change, the changes are not picked up until that user reopens Diagnostics.

If a Business Service Management 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 669.

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 UI main page opens.

- 2** Click **Configure Diagnostics**.
- 3** If you are 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 659.

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.

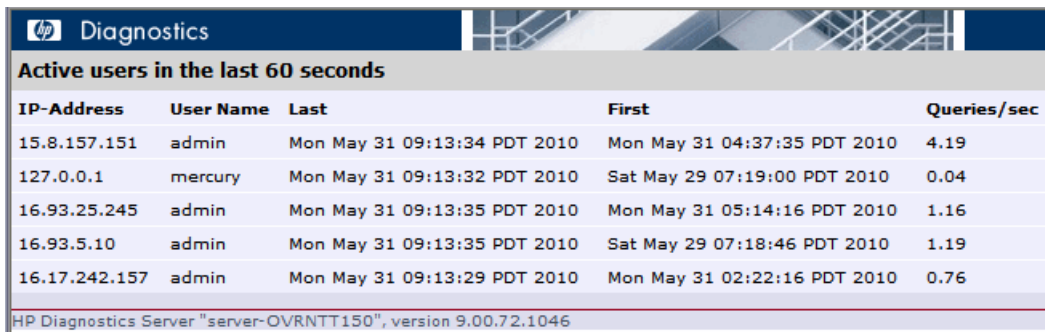
The log file is displayed at the bottom the page.

List of Active Users

You can get a list of active users seen by the Diagnostics server in the last 60 seconds. And you can see the Queries/sec indicating how much load the user generates with summary or trend queries.

From the main Diagnostics UI select **Configure Diagnostics** and the Components page is displayed. (You can also access this Components page by selecting the Maintenance link in any Diagnostics view).

Select the **query** link and then select the **Active Users** link at the bottom of that page to display a list of active users.



IP-Address	User Name	Last	First	Queries/sec
15.8.157.151	admin	Mon May 31 09:13:34 PDT 2010	Mon May 31 04:37:35 PDT 2010	4.19
127.0.0.1	mercury	Mon May 31 09:13:32 PDT 2010	Sat May 29 07:19:00 PDT 2010	0.04
16.93.25.245	admin	Mon May 31 09:13:35 PDT 2010	Mon May 31 05:14:16 PDT 2010	1.16
16.93.5.10	admin	Mon May 31 09:13:35 PDT 2010	Sat May 29 07:18:46 PDT 2010	1.19
16.17.242.157	admin	Mon May 31 09:13:29 PDT 2010	Mon May 31 02:22:16 PDT 2010	0.76

HP Diagnostics Server "server-OVRNTT150", version 9.00.72.1046

You can configure limits on the queries the UI is executing if you find there are very high query loads. Use the **ui.properties** file on the server to set properties to throttle update frequency of ui queries to the server.

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 commanding server.

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 must restart the commanding server.
- ▶ When using a JAAS authentication provider that is also used in other applications, such as 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 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 679) 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=People,o=yourdomain.com"

Option Name	Description	Required /Optional	Default Value	Example
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="employeeNumber"
useSSL	Whether or not connecting to the LDAP server requires SSL.	optional	false	useSSL="true"
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"

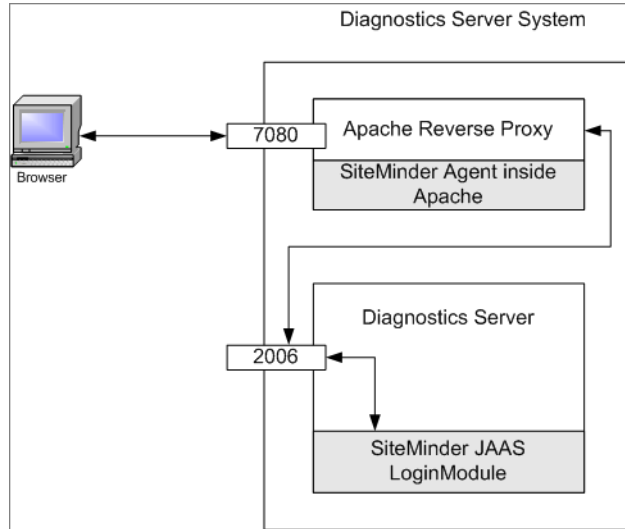
Option Name	Description	Required /Optional	Default Value	Example
roleAttributes	Comma-delimited list of the user's DN attributes whose values will be used as the user's roles. If defaultRoles is also set, the resulting roles will be the union of defaultRoles and roleAttributes.	optional	roles	roleAttributes="employeeType, 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 must restart 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) 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, 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:

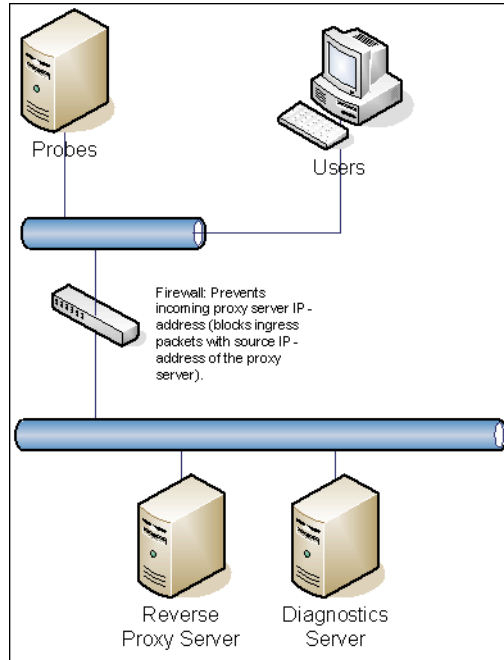
- ▶ ProxyPass / http://<IP-address of Diagnostics Server>:2006/
(2006 is the default Diagnostics Server port, use the port configured for your Diagnostics Server)
- ▶ ProxyPassReverse / http://<IP-address of Diagnostics Server>:2006/
(2006 is the default Diagnostics Server port, use the port configured for your Diagnostics Server)

Note: After making any changes to the httpd.conf file, you must restart 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 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 679).
- 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; for example:

```

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 must restart the Diagnostics commanding server.

C

Enabling HTTPS Between Components

Information is provided on the configuration steps to enable HTTPS communications between the HP Diagnostics components and with Business Service Management.

This chapter includes:

- ▶ About Configuring HTTPS Communications on page 689
- ▶ HTTPS Checklist per Diagnostics Component on page 690
- ▶ Enabling Incoming HTTPS Communication for Diagnostics Components on page 692
- ▶ Enabling Outgoing HTTPS Communication from Diagnostics Components on page 702
- ▶ Enabling HTTPS Communications for the Business Service Management Server on page 708

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 Technical Diagrams.”

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: enablesl=true, keystorepassword, keypassword	Yes	Yes	Yes	Yes	No

Configuration step	Commanding Server	Mediator Server	Java Probe	Collector	.NET Probe
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
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

Enabling Incoming HTTPS Communication for Diagnostics Components

This section includes instructions for configuring the Diagnostics Server, the Java Agent 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 692
- ▶ “Configuring the Java Agent for Incoming HTTPS Connections” on page 696
- ▶ “Configuring the Collector for Incoming HTTPS Connections” on page 699

Configuring the Diagnostics Server for Incoming HTTPS Connections

Notes:

- ▶ To avoid issues with DNS and host name resolution, the Commander URL for the Diagnostics command server 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 must 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. You can 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:1ytc1vu91v2p1y831y7v1v1p1vw11yta
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 command server.
 - 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 Agent for Incoming HTTPS Connections

Note:

- ▶ Enabling SSL and HTTPS Communications for the Java Agent 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 agent system, you must use port 45000 to run the Profiler UI: **https://probe_system:45000**.
-

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.

To configure the Java Agent for incoming HTTPS connections:

- 1 Generate a keystore in the <probe_install_dir>/etc directory using the following command:

```
/opt/MercuryDiagnostics/JavaAgent/_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 Agent.

- Replace **<probe_hostname>** with the machine name for the host of the Java Agent. 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. You can assign different passwords to the **storepass** and the **keypass**.

After you run this command, a keystore is created in **<probe_install_dir>/etc/keystore** with an entry called **PROBE** for the host of the Java Agent.

- 2 Export the certificate for the PROBE entry in the keystore using the following command.

```
/opt/MercuryDiagnostics/JavaAgent/_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 Agent.
- 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 Agent.

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 Agent. 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 Agent.
 - b** Replace **<password>** with the string that you assigned as the password when you created the keystore.

```
/opt/MercuryDiagnostics/JavaAgent/_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. You can assign different passwords to the **storepass** and the **keypass**.

After you run 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>`.

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
```

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.

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.

To enable the Diagnostics command server for outgoing communication to the Diagnostics mediator server 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 command server.
- 2** Change the value of the `trusted.certificate` property in the file `<diagnostics_server_install_dir>/etc/security.properties` for the Diagnostics command server.
- 3** 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.

- 4 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 command server via HTTPS:

- 1 Copy the certificate file from `<diagnostics_server_install_dir>/etc/diag_server_commander.cer` on the Diagnostics command server to `<diagnostics_server_install_dir>/etc/diag_server_commander.cer` on the Diagnostics mediator server.
- 2 Change the value of the `trusted.certificate` property in the file `<diagnostics_server_install_dir>/etc/security.properties` for the Diagnostics mediator server.
- 3 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.
- 4 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.

Set `commander.url` to `https://<diagserver_commander_hostname>:8443`.

Note: When you enable HTTPS on the server, you must 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 Agent for outgoing communications to the Diagnostics mediator server via HTTPS:

- 1** Copy the certificate file from `<diagnostics_server_install_dir>/etc/diag_server_mediator.cer` on the Diagnostics mediator server to `<probe_install_dir>/etc/diag_server_mediator.cer` on the Java Agent.
- 2** Change the value of the `trusted.certificate` property in the file `<probe_install_dir>/etc/security.properties` for the Java Agent.

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 Agent communication, indicate the URL for the Diagnostics mediator server 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 must 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 mediator server via HTTPS:

- 1** Copy the certificate file from `<diagnostics_server_install_dir>/etc/diag_server_mediator.cer` on the Diagnostics mediator server 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 mediator server 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 must use port 8443 in the URL to run the Diagnostics UI.

To enable the .NET Agent for outgoing communications to the Diagnostics command server via HTTPS:

- 1** Copy the certificate for the Diagnostics mediator server to the host for the .NET Agent. The certificate was generated when the Diagnostics mediator server was configured to receive HTTPS. See “Enabling Incoming HTTPS Communication for Diagnostics Components” on page 692 for instructions to configure the Diagnostics mediator server 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 mediator server.
 - a** Select All Files (*.*) in Files of type.
 - b** Navigate to the directory where the keystore for the Diagnostics command server 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" />`
- 22** Change the mediator host and port and add `ssl="true"`: `<mediator host="<diagnostics_mediator_server_host>" port="2612" metricport="8443" ssl="true"/>`
- 23** 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.

- 24** Restart IIS. For instructions on restarting IIS see “Restarting IIS” on page 253.
To verify that you successfully configured the .NET probe for HTTPS communication with the Diagnostics command server:
 - 1** Browse to your .NET application to activate the .NET Agent.
 - 2** Verify that the .NET Agent node is shown in System Health.

Enabling HTTPS Communications for the Business Service Management Server

The following instructions will guide you through the process of configuring Business Service Management for HTTPS communication with Diagnostics.

To enable HTTPS communications between the Diagnostics command server and Business Service Management:

- 1** Copy the Diagnostics certificate file, **diag_server_commander.cer**, from the Diagnostics command server installation directory, **<diagnostics_server_install_dir>/etc/**, to the Business Service Management host.
- 2** Import the copied certificate, **diag_server_commander.cer**, into the Business Service Management server cacert keystore by running the following command on the Business Service Management 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 Service Management.
- 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 Service Management 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 command server.
- Replace **<copied_BAC_certificate_directory>** with the path to the copied Business Service Management 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 Service Management server to the HTTPS port on the Diagnostics command server.
 - a** Open the Diagnostics Administration in Business Service Management 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 command server exactly as it was specified in the **CN** parameter when you created the keystore for the Diagnostics command server. 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 692.
 - Enter **HTTPS** for the protocol.
 - Enter **8443** for the Web port of the Diagnostics Server.
 - e** Click **Submit Configuration**.

D

Using the System Health Monitor

You can use the System Health Monitor to review the configuration of the Diagnostics components and verify that they are working properly.

This chapter includes:

- ▶ Introducing the System Health Monitor on page 712
- ▶ Accessing the System Health Monitor on page 712
- ▶ System Health Monitor Views on page 713
- ▶ Understanding the System Health Graph on page 714
- ▶ Viewing Component Status and Host Configuration Tooltips on page 717
- ▶ Viewing Detailed Component Information on page 719
- ▶ Viewing Troubleshooting Tips on page 723
- ▶ Viewing Log Information for the Whole System on page 723
- ▶ Customizing the System Health Monitor Display on page 724
- ▶ Filtering the System Health Monitor Component Map by Customer on page 726
- ▶ Controlling the Refresh Rate of the System Health Monitor on page 727
- ▶ Creating and Using System Health Monitor Snapshots on page 728

Introducing the System Health Monitor

The System Health Monitor provides you with a map of all of the components of your 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 could help you diagnose the issues you see on the map.

For very large deployments there are additional system health views available in the main Diagnostics UI. See *HP Diagnostics User's Guide* section on System Views. To access the System Views you must access the Diagnostics UI as the Mercury System customer from `http://<Diagnostics_Commanding_Server_Name>:2006/query/`.

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 Service Management.

Note: To access the System Health Monitor, you require **system** permissions. For more information about permissions, see “Understanding User Privileges” on page 659.

To access the System Health Monitor from your Web browser:

- 1 Enter the following URL into your browser:

http://<Diagnostics command server host>:<Diagnostics command server port>/registrar/health

The default port for the Diagnostics command server is 2006. If you 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 the main Diagnostics UI by selecting **Configure Diagnostics > Registrar > System health graph** or by selecting the **Maintenance** link at the top right corner of any Diagnostics view and then selecting **Registrar > System health graph**.

To troubleshoot 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.

System Health Monitor Views

There are several System Health views available on the Registrar page:

- System health graph (Requires JRE 1.5 or later)
- System health xml with table-view stylesheet
- System health xml with node-view stylesheet

► Raw system health xml

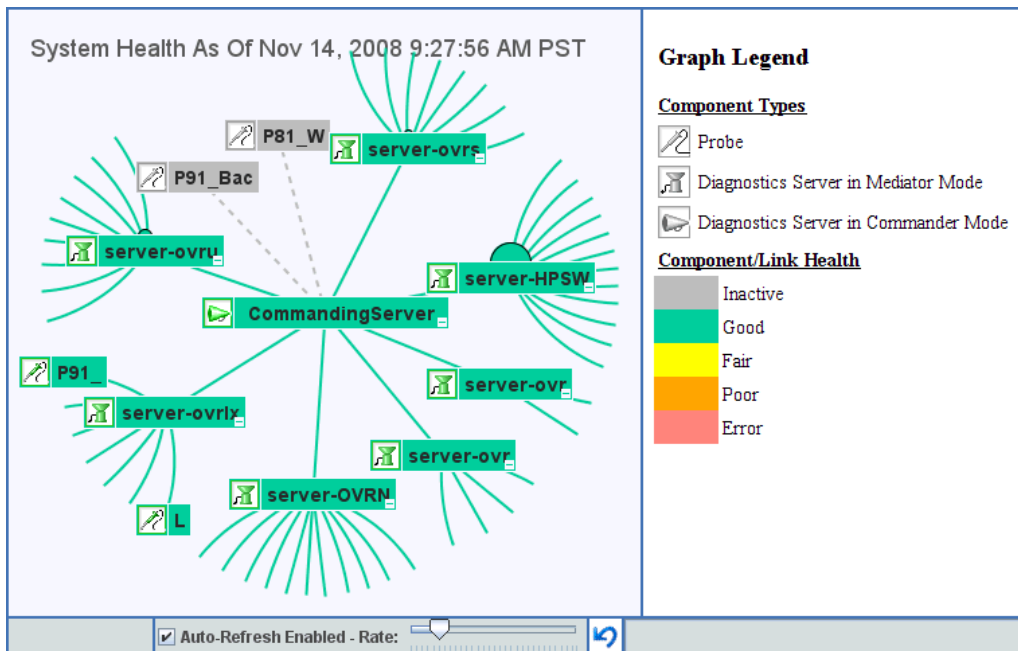
For very large deployments there are additional system health views available in the main Diagnostics UI. See *HP Diagnostics User's Guide* section on System Views. To access the System Views you must access the Diagnostics UI as the Mercury System customer from http://<Diagnostics_Commanding_Server_Name>:2006/query/.

Understanding the System Health Graph

The System Health Monitor Graph is comprised of two panes:

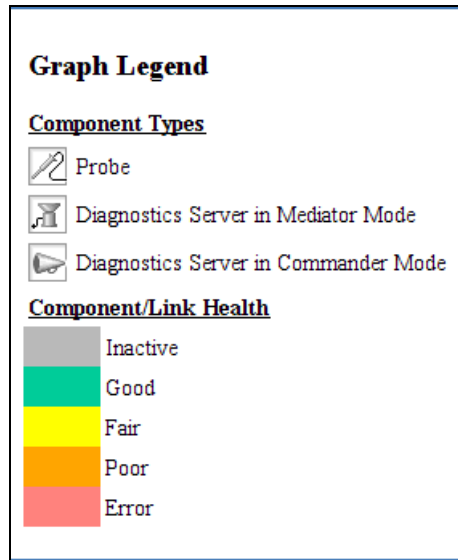
- **The graph legend** (in the right pane)
- **The System Health Monitor component map** (in the left pane)

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 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.

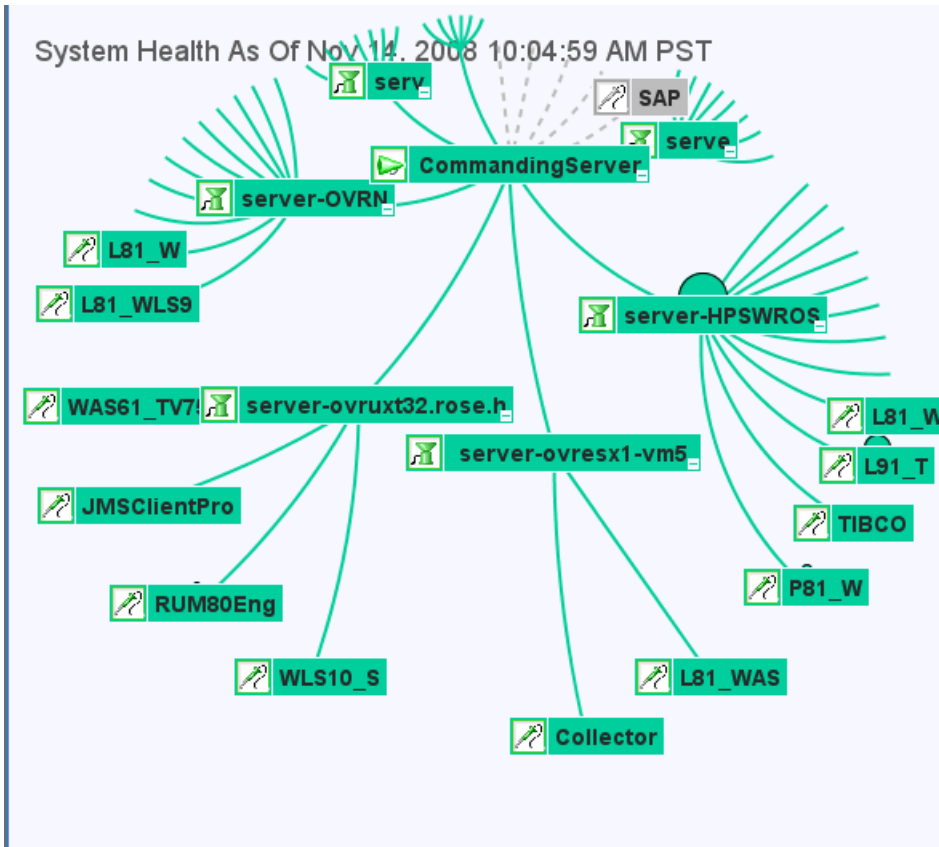
In the following image of a Diagnostics deployment you can see the following:

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.

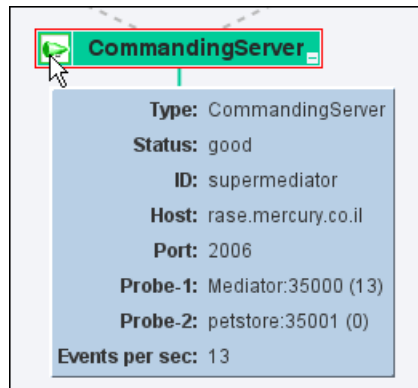
To view component status and host configuration tooltips:

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 were configured to work with the Diagnostics Server.

► Diagnostics Commanding Server



► **Diagnostics Mediator Server**

The screenshot shows a tooltip for a component named 'server-ovruxt32.rose.hp.com'. The tooltip contains the following information:

- Type:** mediator
- Status:** good
- ID:** server-ovruxt32.rose.hp.com
- Host:** ovruxt32.rose.hp.com
- Port:** 2612
- Probe events per sec:** 0
- Probe-1:** WLS10_SOAP_over_JMS_ovrntt206_W2k3:35001 (0)
- Probe-2:** RUM80Engine_ovrntt206_W2k3:35000 (0)
- Probe-3:** JMSClientProbe:35002 (0)
- Probe-4:** ServerProbe-Server:35000 (0)
- Probe-5:** WAS61_TV75_AMKILAB02_W2k3:35000 (0)

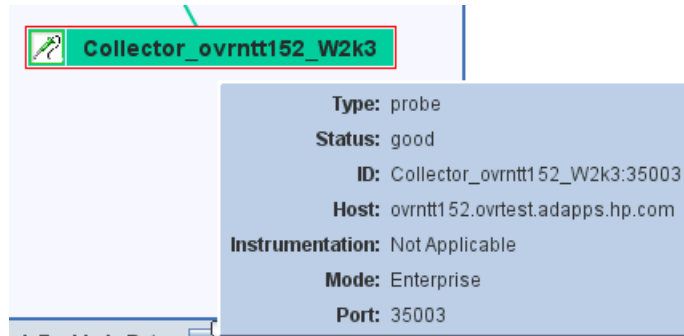
► **Diagnostics Probe**

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.

The screenshot shows a tooltip for a component named 'WAS61_TV75_AMKILAB02_W2k3'. The tooltip contains the following information:

- Type:** probe
- Status:** good
- ID:** WAS61_TV75_AMKILAB02_W2k3:35000
- Events per sec:** 13
- Host:** amkilab02.ovrtest.adapps.hp.com
- Instrumentation:** auto_detect.points
- Mode:** Enterprise, PRO, TV
- Port:** 35000

► Diagnostics Collector

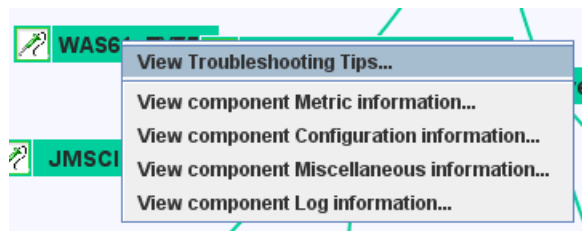


Viewing Detailed Component Information

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.

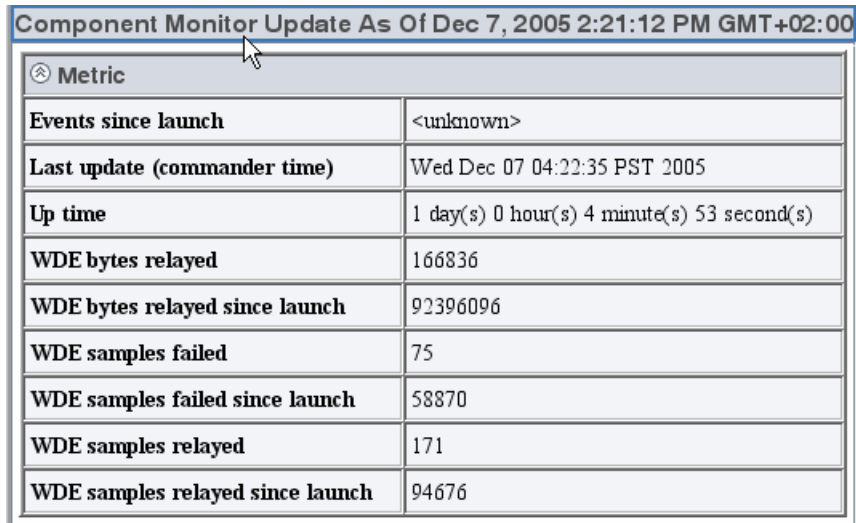
You can view the following information:

- **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 command server:



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 Agent:

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 mediator server:

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.I
Tue Dec 06 05:51:44 PST 2005	SEVERE: out of order com.mercury.diagnostics.server.persistence.impl.I
Tue Dec 06 05:46:01 PST 2005	SEVERE: out of order com.mercury.diagnostics.server.persistence.impl.I

Component Status Information

Below is an example of Component Status information for the Java Agent. 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 Agent:

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.

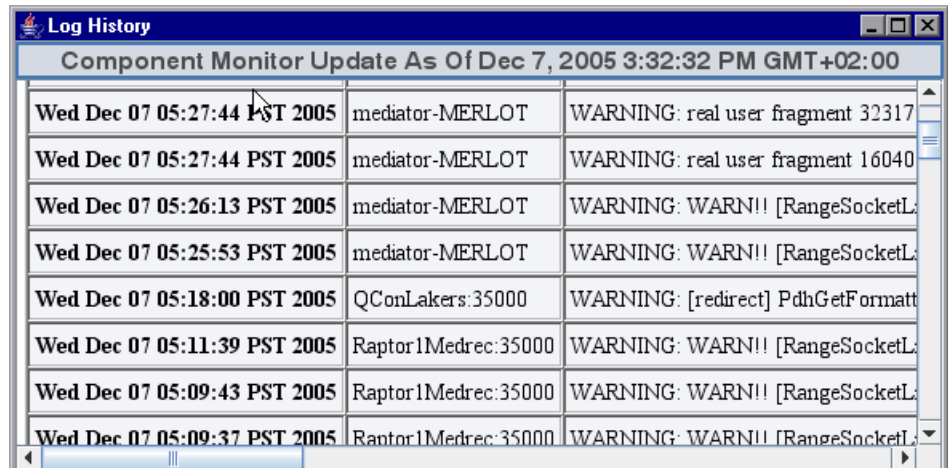
Scan the information displayed for the symptoms that you are investigating, to see if there is a recommended solution documented in the troubleshooting tips.

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.



The screenshot shows a window titled "Log History" with a subtitle "Component Monitor Update As Of Dec 7, 2005 3:32:32 PM GMT+02:00". The window contains a table with three columns: Date and Time, Component Name, and Message. The table lists several warning messages from various components.

Date and Time	Component Name	Message
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	Rantor1Medrec: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.

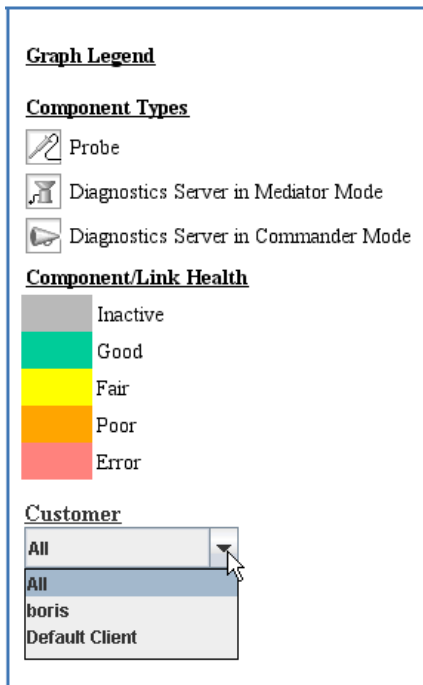


Filtering the System Health Monitor Component Map by Customer

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 command server 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 command server host>:<Diagnostics command server port>/registrar/xml
```

The default port is **2006**. If you 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. Give the file a name that helps 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 could 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 command server host machine.
- 2 Enter the following URL in your Web browser:

```
http://<Diagnostics command server host>:<Diagnostics command server port>/  
registrar/health?xml=<XML_file>
```

where <XML_file> is the path to the .xml file on the Diagnostics command server host machine.

For example, if you copied **sys_health_mediator_yyyymmdd.xml** to the hard drive (C:) on the Diagnostics command server 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 command server port>, by default, should be 2006. If you 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 on page 732
- ▶ Custom View Data on page 732
- ▶ Performance History Data on page 733
- ▶ Data Retention on page 739
- ▶ Pre-Installation Data Management Considerations on page 744
- ▶ Backing Up Diagnostics Data on page 745
- ▶ Handling Diagnostics Data when Upgrading Diagnostics on page 749

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 command server 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.

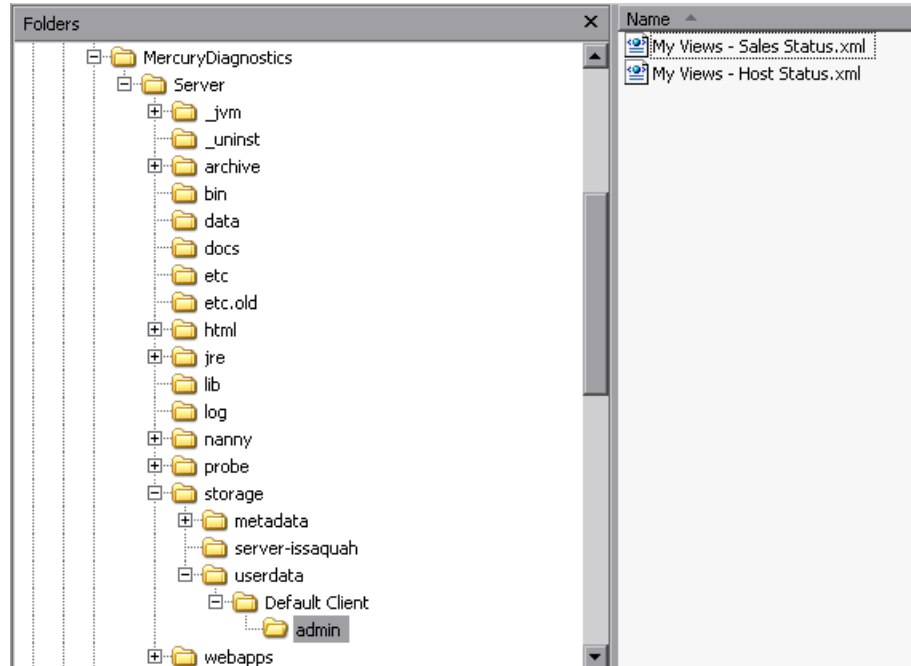
Custom View Data

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 command server.

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 command server. 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 command server as shown in the following example.



Performance History Data

Diagnostics stores the historical performance data in a time series database (TSDB) on the Diagnostics mediator server. 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 Retention” on page 739.

This section includes:

- “Performance History Data Organization” on page 734
- “Performance History Data File Types” on page 737

Performance History Data Organization

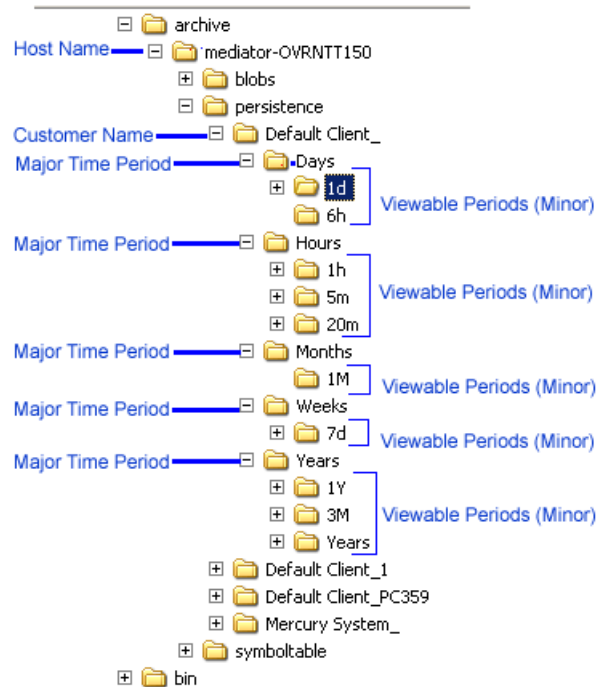
The Diagnostics performance history data is in the `<diagnostics_server_install_dir>/archive/mediator-<host_name>/persistence/<customer_name>_` directory of the Diagnostics mediator server where:

- `<host_name>` is the name of the host for the Diagnostics mediator server.
- `<customer_name>` is the customer name that you entered when you installed the Diagnostics mediator server. The name of this directory is the customer name with an appended underscore.

Notes:

- Unless you are an HP Software-as-a-Service (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.
-

In the **/persistence** directory, the performance data is organized into directories as shown below:



The directory levels are referred to as **Major** time periods (Days, Hours, Months, Weeks, and Years) and the subdirectories are referred to as **Minor** time periods (1d, 6h, 1h, 20m, 5m, 1M, 7d, 1Y, 3M, Years). These time periods are also referred to as data granularity.

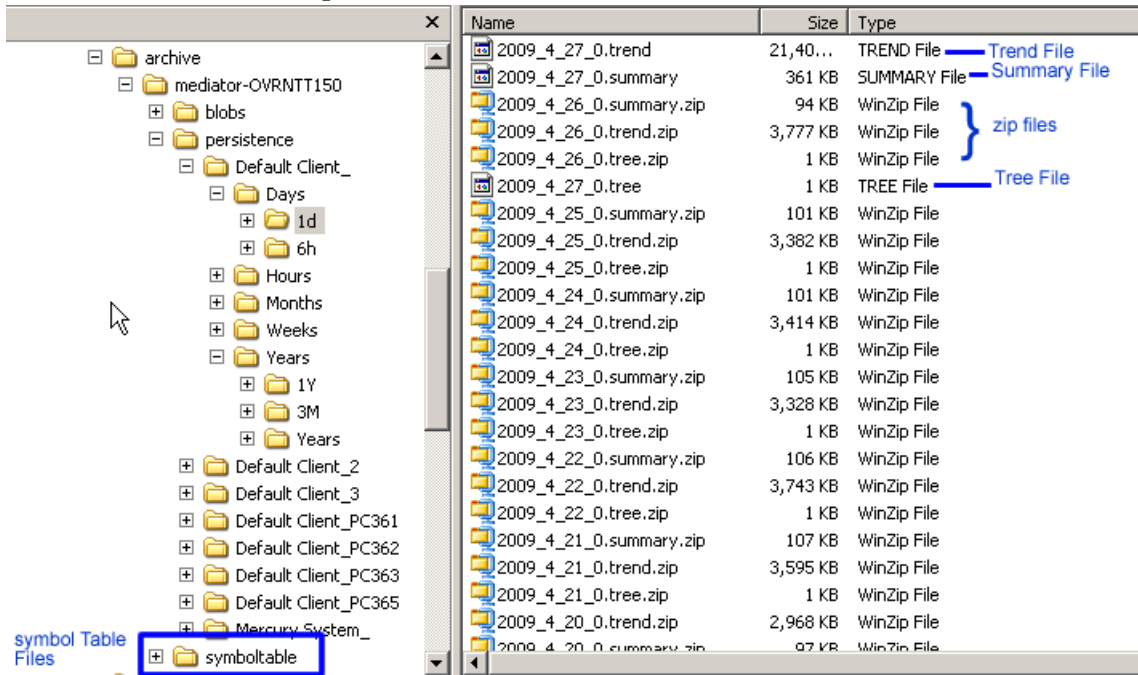
As seen in the directory example above, the Hours directory represents a major time period and has three subdirectories for the minor periods 5m, 20m, 1h.

These same minor time periods serve as the viewing periods in the UI. An example of the Viewing filter in Diagnostics is shown below:



Performance History Data File Types

The Diagnostics performance history data is stored in several types of files. And example of the directories with these files is shown below:



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.

Summary Files

The summary files are accessed to display data in the Diagnostics View's entity tables and details pane. The Status shown in Diagnostics Views is based on summary files. Each viewable time period is stored in separate *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 April 24, 2009 would be named **2009_4_27_0.summary**.

Trend Files

The trend files are accessed to display graph (charted) data in Diagnostics Views. To retrieve a trend for a minor time period, such as 5 minutes, only a small portion of the data in its major time period (1 hour in this case) trend file is read. Diagnostics stores the charted trend data for each major time period in *trend* files.

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 April 24, 2009 would be named as follows: **2009_4_27_0.trend**.

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 were created by consolidating raw data points for twelve 5-second time periods.

Instance Tree Files

Instance tree files are accessed when you drill down to an instance call tree on the Diagnostics Server Requests view.

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, **2009_4_27_0.tree**.

Compressed Zip Files

Data in the instance tree, trend and summary files is for current time periods. The data in these files is uncompressed. The data files are quite large so they are automatically compressed after each time period is complete to save disk space. Compressed files have the same file names as the uncompressed file, but with a .zip extension; for example, **2009_4_26_0.summary.zip**.

Data Retention

Diagnostics uses data retention strategies that allow it to optimize its use of disk storage.

This section includes:

- “Data Retention on the Mediators” on page 739
- “Data Retention Configuration” on page 740
- “Symbol Table Purging Process” on page 742

Data Retention on the Mediators

To make optimum use of disk space, historical performance data is stored in major and minor time periods with the data in each period retained based on the diagnostics data retention policy.

The data in the time periods with lower resolution data points is kept for longer periods of time to assist with such activities as capacity planning. The data in the time periods 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.

In the table below you can see the Major time periods (directories as described in “Performance History Data Organization” on page 734) and the Minor time periods (viewable periods or subdirectories). The viewable periods under each directory are grouped together because they have the same resolution. So for example in the Days directory, 1 day and 6 hour data both have 5-minute resolution.

The following table illustrates the general Diagnostics data retention policy.

Major Time Period (Directory)	Minor Time Period (Viewable Period)	Trend Resolution	Data is kept for ... (Retention)
Hours	Hour, 20 minutes & 5 minutes	5 seconds	72 Hours
Days	Day & 6 Hours	5 minutes	93 Days
Weeks	Week	1 Hour	52 Weeks
Months	Month	6 Hours	24 Months
Years	Year, Quarter	1 Day	5 Years

The above table only applies when the entities don't change and are constantly available for the specified periods of time. See “Symbol Table Purging Process” that follows.

As shown in the table above, data for the last 1 hour, 20 minutes and 5 minutes viewing periods is kept for 72 hours while data for the last quarter is kept for 5 years.

Data Retention Configuration

You can configure data retention using the **server.properties** file on each mediator server.

An example showing the persistence section of the server.properties file is shown below:

```

persistence.major.durations.num=5
persistence.major.0.length=1
persistence.major.0.unit=h
persistence.major.0.retention=72
persistence.major.0.name=Hours
persistence.major.0.datapoints=720
persistence.major.0.minors=3
persistence.major.0.minor.0.name=5m
persistence.major.0.minor.0.length=5
persistence.major.0.minor.0.unit=m
persistence.major.0.minor.1.name=20m
persistence.major.0.minor.1.length=20
persistence.major.0.minor.1.unit=m
persistence.major.0.minor.2.name=1h
persistence.major.0.minor.2.length=1
persistence.major.0.minor.2.unit=h

```

Default retention is 72 hours

Major time period is Hours

Contains 3 minor time periods

5 minute minor viewable period

20 minute minor viewable period

1 hour minor viewable period

The retention level from the major time period is inherited by the minor time period.

To set the retention for a minor time period to a different value, add a line as shown below for the 1h minor time period. This sets retention of hourly data to 90 hours:

```

persistence.major.0.minor.2.name=1h
persistence.major.0.minor.2.length=1
persistence.major.0.minor.2.unit=h
persistence.major.0.minor.2.retention=90

```

To retain the hourly data for one week, set retention to 168 hours. Units are in hours (unit=h) so we compute one week in hours as [7 (days) x 24 (hours) = 168].

```

persistence.major.0.minor.2.name=1h
persistence.major.0.minor.2.length=1
persistence.major.0.minor.2.unit=h
persistence.major.0.minor.2.retention=168

```

To retain daily data for six months you would set retention to 186 days. Units are in days (`unit=d`) so we compute six months in days as $[6 \text{ (months)} \times 31 \text{ (days/month)} = 186]$.

```
persistency.major.1.minor.1.name=1d
persistency.major.1.minor.1.length=24
persistency.major.1.minor.1.unit=d
persistency.major.1.minor.1.retention=186
```

Symbol Table Purging Process

The purging mechanism provides for the following:

- ▶ Data that is part of a snapshot is not purged since doing so renders the snapshot 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 **`persistency.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 modified in **`server.properties`** by changing 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 is removed. To allow multiple snapshots at the same time, each snapshot creates a new preservation entry. Identical entries are not merged because 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 rerun 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 does not 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.

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 (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.

Note: If your Diagnostics deployment requires that the Diagnostics Server have high availability, you can create a standby Diagnostics Server for each Diagnostics Server. The standby is then ready to be used during a hardware failure or other problem with the host of the Diagnostics Server. See “Preparing a High Availability Diagnostics Server” on page 390

This section includes:

- “Backing up Data Remotely” on page 745
- “Configuring Symbol Table Backup” on page 747
- “Restoring Data After a Failure” on page 748

Backing up Data Remotely

Remote backup is possible by downloading the Diagnostics data files over HTTP to a local directory, and 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 make sure the files are backed up in the proper order to ensure data integrity.

Diagnostics is installed with a remote backup script stored at `<diagnostics_install_dir>/server/bin/remote-backup.sh`. The UNIX script uses the `wget` utility (<http://www.gnu.org/software/wget/wget.html>) to download incrementally over HTTP. On Windows, Cygwin (<http://www.cygwin.com/>) can be used to run this script. There is also a `remote-backup.cmd` for Windows. The `.cmd` script requires `wget.exe` in `<diagnostics_install_dir>/server/bin/wget/wget.exe` (the `.sh` script just requires `wget` in the path).

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
<code>-h</code>	The host (or IP address) to download from
<code>-o</code>	The directory to store the backup in
<code>-u</code>	The HTTP username to use Default: <code>admin</code>
<code>-p</code>	The HTTP password to use Default: <code>admin</code>
<code>-P</code>	The HTTP port number to use (optional) Default: <code>2006</code>
<code>-r</code>	The ID of the Diagnostics Server in Mediator mode being read from (for <code>rhttp</code> backups) (optional). For example, if you have 2 servers "commander" and "mediator", you could backup mediator over <code>rhttp</code> with: <code>-h commander -r mediatorId</code> .

Parameter	Description
-c	The clean option. When specified, files that exist in the output directory and do not exist on the server will be removed (the others need to be kept for better performance with the timestamping feature on download).
-v	Specified for more verbose output.

For example, to back up 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
Raw performance history data	archive/.../persistence/
Symbol table	archive/.../symboltable/

Configuring Symbol Table Backup

Sometimes the backup folder for the jdb files under symboltable use up a lot of disk space when there are a large number of symbol files. So you can configure backing up the symbol table as follows:

- ▶ You can enable or disable backing up the symbol table by setting the **symboltable.backup** property to true or false in the **server.properties** file.
- ▶ You can configure symbol table backup frequency by setting the **symboltable.backup.majors** property in the **server.properties** file.

Set the `symboltable.backup.majors` property using a comma separated list, to the desired backup frequency (Days,Weeks, Months). The frequency values are the same as defined by the `persistency.major.<n>.name` property (see “Data Retention Configuration” on page 740). For example, to backup the symbol table weekly, use the `persistency.major.2.name` which is Weeks.

The default configuration for symbol table backup as defined in `server.properties` is:

```
# Should the server backup the symboltable?  
symboltable.backup = true  
# Which majors should be backed up?  
symboltable.backup.majors = Days,Weeks,Months
```

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** Make sure 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 on 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 Diagnostics, 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, “Upgrade and Patch Install Instructions.”

F

Diagnostics Technical Diagrams

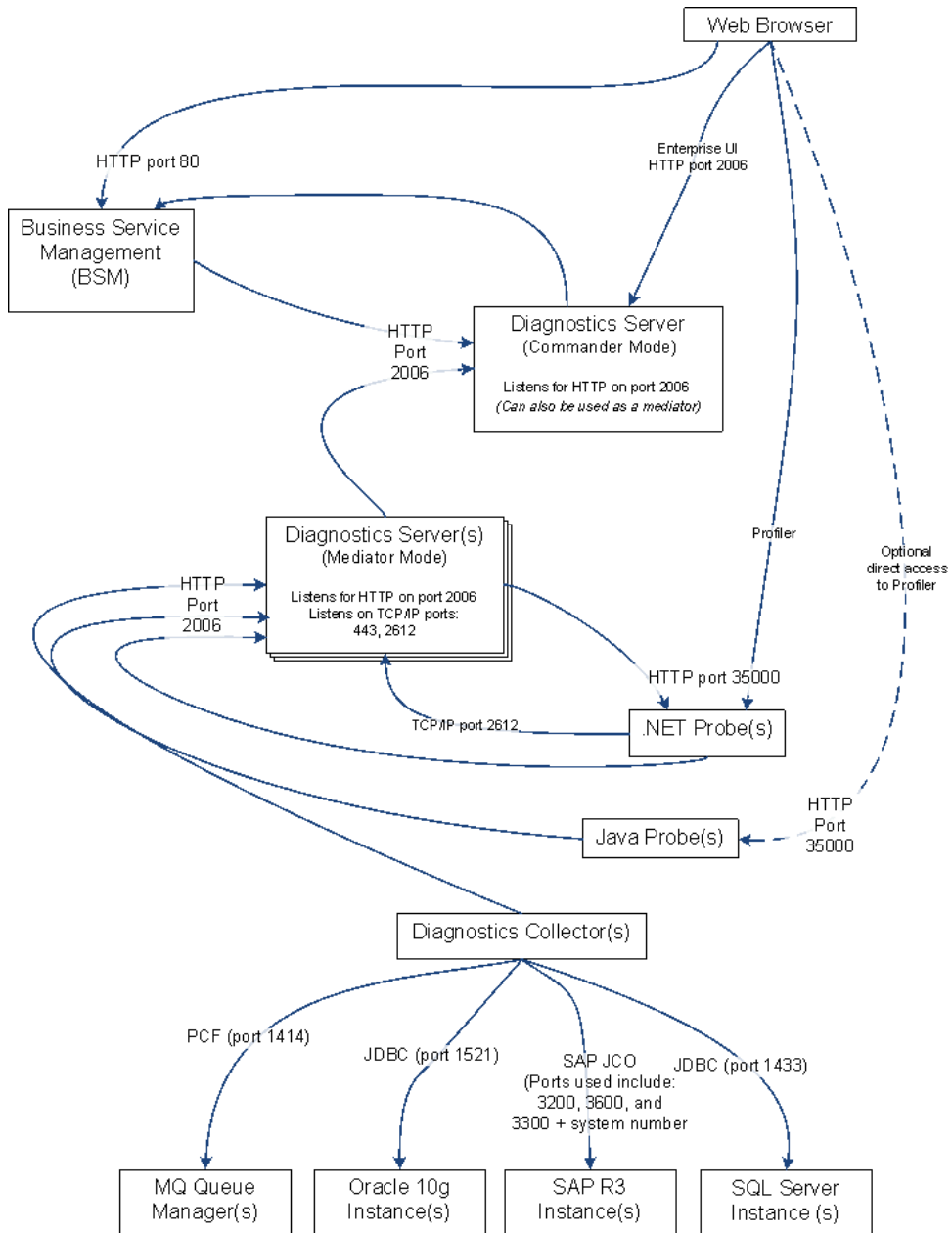
Data flow and communication diagrams are provided to assist you as you deploy Diagnostics components and integrate Diagnostics with other HP Software Products.

This chapter includes:

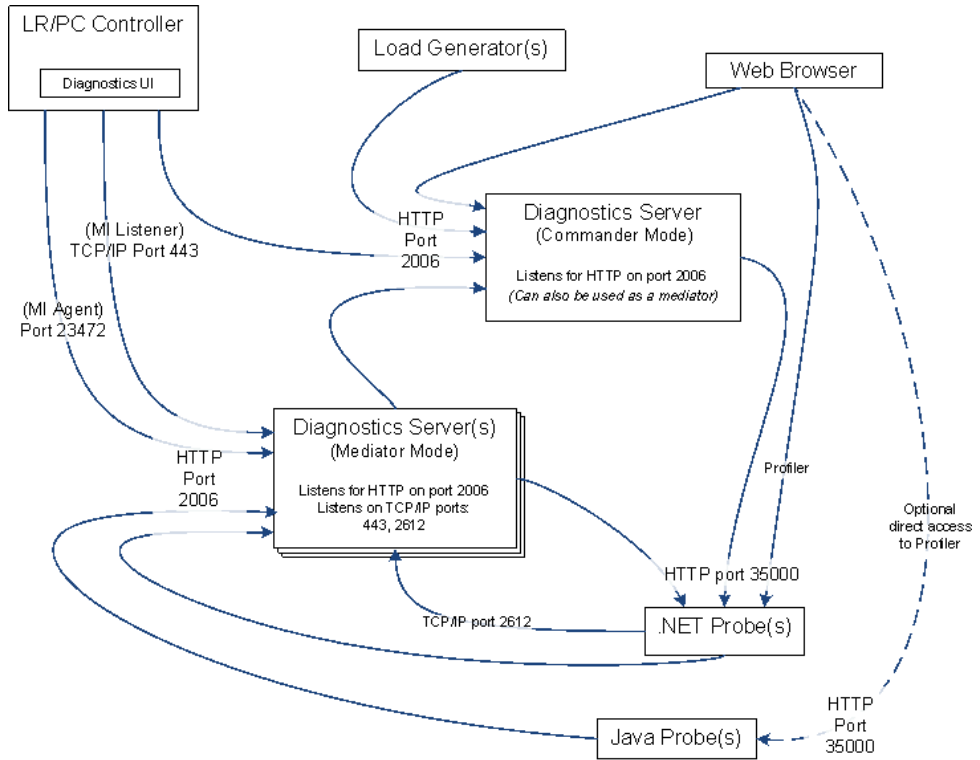
- Communications with Business Service Management on page 752
- Communications with LoadRunner and Performance Center on page 753
- .NET Probe Aggregator Data Flow on page 754

Note: The diagrams are intended to provide a high-level view, not provide an in-depth knowledge of the working of the components.

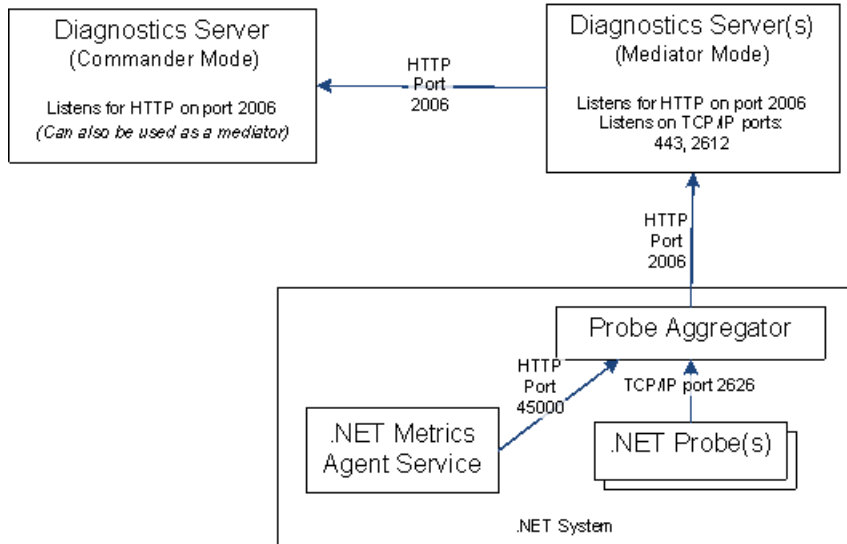
Communications with Business Service Management



Communications with LoadRunner and Performance Center



.NET Probe Aggregator Data Flow



G

Upgrade and Patch Install Instructions

Instructions are provided for upgrading between major releases of Diagnostics (for example 8.0 to 9.0). You follow these same instructions when installing a patch release. Patch releases contain a full replacement of the Diagnostics product components so you need to follow the same instructions as for an upgrade.

Note: The Java Agent instructions apply when upgrading or installing a patch release of a Diagnostics or TransactionVision Java Agent.

This chapter includes:

- ▶ General Recommendations on page 756
- ▶ Diagnostics Compatibility with Earlier Diagnostics Versions on page 756
- ▶ Upgrade or Patch Install Instructions for Diagnostics Components on page 756
- ▶ Diagnostics Compatibility with Other HP Software Products on page 767

General Recommendations

The following recommendations are generally applicable when upgrading from earlier versions of Diagnostics or installing patch releases.

- ▶ Before you upgrade to a newer 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 version of the component. See the readme or the *HP Diagnostics Installation and Configuration Guide* for system requirements.
- ▶ You have to upgrade the Diagnostics Server before upgrading an agent.
- ▶ You should contact HP Software Customer Support when you need to upgrade from an earlier version of Business Service Management, or Performance Center.

Diagnostics Compatibility with Earlier Diagnostics Versions

The 9.0 version of the Diagnostics Server is capable of working with the following earlier agent and collector versions:

- ▶ Java Agent 7.x, 8.x
- ▶ .NET Agent 7.x, 8.x
- ▶ Collectors 7.x, 8.x

Upgrade or Patch Install Instructions for Diagnostics Components

The following instructions guide you in the process of upgrading an existing Diagnostics component or installing a component from a patch release.

This section includes instructions for the following:

- ▶ “Diagnostics Server” on page 757
- ▶ “Java Agent” on page 760
- ▶ “.NET Agent” on page 763

- ▶ “Diagnostics Collector” on page 764

Diagnostics Server

This section contains instructions for upgrading your Diagnostics Server from an earlier version to 9.0. The same instructions would apply for installing patch releases.

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 HP Software-as-a-Service (SaaS) customer, contact SaaS Support for upgrade instructions.
-

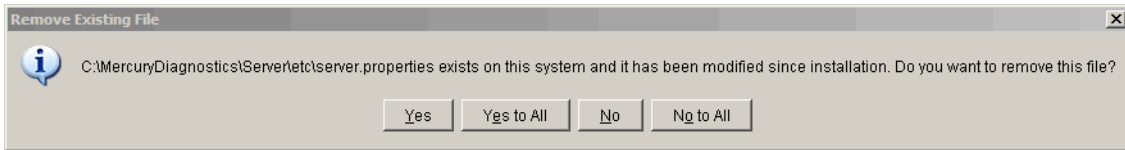
Important: During the installation the keystore is overwritten along with the JRE. As a result your trusted certificates will be unavailable after the upgrade.

To upgrade a 7.x or 8.x Diagnostics Server to 9.0:

- 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 could 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.

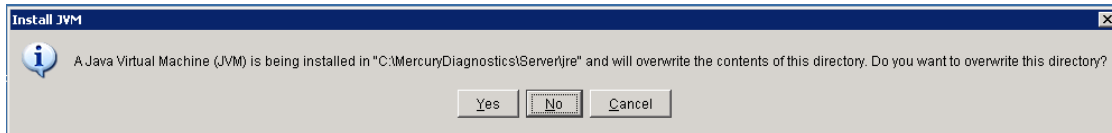
- 3** Uninstall the current Diagnostics Server, **but retain the modified files when prompted**. For example, on Windows click **No to All** at the following prompt:



For information about uninstalling and removing the Diagnostics Server, completely see Chapter 2, “Installing the Diagnostics Server”.

- 4 On Unix, rename **etc** directory to **etc.old** (replacing etc.old if it already exists).
- 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 Compare the **etc** directory and the **etc.old/etc.old<timestamp>** directory so that you can determine the differences between the two. It might 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, Commander's URL.
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 command server, 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 the latest version 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 9.0, 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.

Java Agent

This section contains instructions for upgrading your Diagnostics Java Agent from an earlier version to 9.0. The same instructions would apply for installing patch releases.

Note: You must upgrade the Diagnostics Server before upgrading the agents that are connected to it because Diagnostics Servers are not forward compatible.

To upgrade a 7.x or 8.x Java Agent to 9.0:

Note: The new agent installation will not begin monitoring your applications until you have updated the startup scripts to start the new agent and restarted the applications as described in these instructions.

- 1 Install the Diagnostics Agent for Java **into a different directory** than the current agent's installation directory.

During the installation, for Diagnostics be sure to:

- configure the Java Agent to work 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 agent
- ▶ for the agent group name, use the same group name as used by the previous agent
- ▶ for the mediator server name and port, use the same information as used by the previous agent
- ▶ run the **JRE instrumenter** regardless of the JRE version you use and even if you did not run the JRE instrumenter before.

This ensures that the persisted data for your application will match up with the metrics captured by the new agent.

- 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 agent's `\etc` directory and the previous agent's `\etc` directory so that you can determine the differences between the two.

HP recommends that you execute the **Property Scanner** utility provided with the Java Agent 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\Snapins>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 agent's **\etc** directory to the new agent'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 Agent
auto_detect.points	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.
capture.properties	Depth and latency trimming.
inst.properties	define.pre.process
dispatcher.properties	minimum.sql.latency 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** Update the applications startup script to point to the upgraded agent installation. This includes the -javaagent and/or -Xbootclasspath.
- 5** At an approved time, shut down the applications that were being monitored by the old agent.
- 6** Restart the applications to allow the new version of the agent 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 Agent is running and properly registered, by checking the version in System Health. Browse to the System Health http://<commanding_server>:<port>/registrar/health, and double-click the agent icon. The version listed under Configuration should be the latest version, if the upgrade was successful and the new version of the Java Agent was started.
- 9 When all your applications have been migrated over to be version 9.0 and everything is working properly, you can delete the old directory. Don't try to uninstall the old version because this will actually uninstall the new version.

.NET Agent

This section contains instructions for upgrading your Diagnostics .NET Agent from an earlier version to 9.0. The same instructions apply for installing patch releases.

Note: You must upgrade the Diagnostics Server before upgrading the .NET Agents 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.

To upgrade a 7.x or 8.x .NET Probe to 9.0:

- 1 Install the new Diagnostics Agent 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 Agent 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 the latest version, if the upgrade was successful and the .NET Agent was started.

Diagnostics Collector

This section contains instructions for upgrading your Diagnostics collector from an earlier version to 9.0. The same instructions apply for installing a patch release.

Note: You must upgrade the Diagnostics Server before upgrading the Collectors that are connected to it because Diagnostics Servers are not forward-compatible.

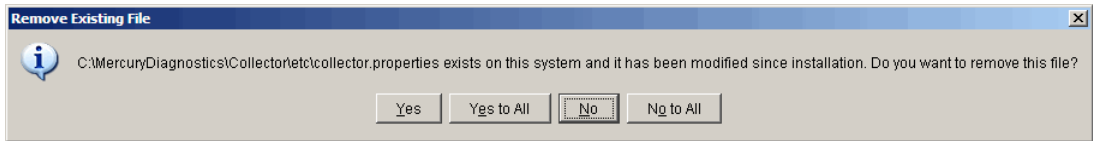
To upgrade a 7.x or 8.x Diagnostics Collector to 9.0:

- 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 retain the modified files** when prompted. For example, on Windows click **No to All** at the following prompt:



For information about uninstalling and removing the Collector completely, see “Uninstalling the Diagnostics Collector” on page 117.

- 4** On Unix, rename **etc** directory to **etc.old** (replacing etc.old if it already exists).
- 5** Install the new Collector into the same installation directory that was used for the old version of the Collector.

Make sure to **use the same Collector name and Mediator host** 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.

- 6** 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.
- 7** Compare the new **etc** directory and the **etc.old/etc.old<timestamp>** directory to determine the differences between the two. (It might be helpful to use a diff/merge tool for this purpose.)

Apply any differences that were caused by the customizations that you made from the **etc.old** 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.
sqlserver-config.xml	If the collector is monitoring an SQL Server environment. If you are upgrading from 7.x you must remove the databaseName attribute from the sqlserver-config.xml file because in later Collector versions the 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.

- 8** Start the Diagnostics Collector.
- 9** 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 the latest version, if the upgrade was successful and the Collector was started.
- 10** 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

For the most recent information on version compatibility, see the Diagnostics Product Availability Matrix at http://support.openview.hp.com/sc/support_matrices.jsp.

H

Troubleshooting HP Diagnostics

Troubleshooting tips are provided for problems that could occur when using HP Diagnostics.

This chapter includes:

- ▶ Component Installation Interrupted on a Solaris Machine on page 770
- ▶ Java Agent Fails to Operate Properly on page 770
- ▶ Error During WAS Startup with Diagnostics Profiler for Java on page 771
- ▶ Missing Server-Side Transactions on page 772
- ▶ Event Capture Buffer Full Warning on page 772
- ▶ Java Agent Support Collector on page 773
- ▶ OM Agent Troubleshooting on page 774
- ▶ Event Based Health Indicator Status Troubleshooting Flow on page 776

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 Agent Fails to Operate Properly

If the Java Agent 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.”

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=!aik\.security\.*;!c8e\.*;!org\.jboss\.net\.protocol\.file\.Handler,!org\.jboss\.net\.protocol\.file\.URLConnection,!*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 388.

"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 Business Process Monitor is running in Business Service Management or HP Software-as-a-Service (SaaS) and that the monitor is running.
- 2 Verify the state of the profile in the Business Process Monitor Console.

Event Capture Buffer Full Warning

Symptoms:

Some Diagnostics data loss is occurring and the following error appears in the probe log file:

"The event capture buffer is full, at least one event dropped."

Reason:

The log entry indicates that the application load is too high, or that the application is excessively instrumented.

Solution:

In some cases, increasing the value of the **event_buffer.size** property in the `etc/capture.properties` file can help avoid dropping events, but often reducing the application instrumentation is necessary.

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 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.

OM Agent Troubleshooting

When integrating Diagnostics with Business Service Management 9.0, the OM Agent and IAPA components must be installed on the Diagnostics Commander Server. These components are used by Diagnostics to send Health Indicator status events to the Business Service Management Gateway Server.

If there are problems with the installation, errors are reported in the `<Diagnostics_install_dir>/server/log.txt` file.

After installing the OM Agent and IAPA components you must complete additional configuration which is performed when you register Diagnostics with Business Service Management 9.0. The required certificates are requested during the registration and then you perform an additional step to grant the certificate on the gateway server.

To make sure the Diagnostics Commander can ping the Business Service Management Gateway Server:

- 1 Execute the following on the Diagnostics Commander:

```
bbcutil -ping <gateway server>
```

This is the expected output:

```
bsavm12.rose.hp.com: status=eServiceOK  
coreID=6c852d02-9ae6-7543-1d6e-b6fab24428f0  
bbcV=06.20.101 appN=ovbbccb appV=06.20.101 conn=2  
time=218 ms
```

- 2 If you get an eSSLError, remove the certificates (unless the OM Agent on the Diagnostics Commander is used by another OM sever):

```
C:\ovcert -list
+-----+
| Keystore Content |
+-----+
| Certificates: |
| ad2c79b2-9af7-7543-002d-ceeb548960bc (*) |
+-----+
| Trusted Certificates: |
| CA_6c852d02-9ae6-7543-1d6e-b6fab24428f0 |
+-----+
C:\ovcert -remove ad2c79b2-9af7-7543-002d-ceeb548960bc
C:\ovcert -remove CA_6c852d02-9ae6-7543-1d6e-b6fab24428f0
```

- 3 And then run <server_install_dir>\server\bin\switch_ovo_agent.vbs again:

```
cscript switch_ovo_agent.vbs -server <GATEWAY SERVER> -cert_srv <DATA PROCESSING SERVER>
```

To check if the agent cannot communicate with the server:

- 1 If you get errors in the C:\Documents and Settings\All Users\Application Data\HP\HP BTO Software\log\System.txt file indicating that the agent cannot communicate with the server, verify that **com.hp.ov.opc.msgr** is running on the Business Service Management Gateway Server:

```
bbcutil -reg
BasePath=/com.hp.ov.ctrl.ovcd/
  Protocol=HTTPS
  BindAddress=localhost
  Port=1057
  Authentication=REMOTE
BasePath=/com.hp.ov.opc.msgr/
  Protocol=https
  BindAddress=ANY
  Port=2506
  Authentication=REMOTE
```

To fix the problem if the Registration times out because of the OM Agent:

- 1 If you see an error like the one below:

```
Timed Out: cscript //NoLogo C:\MercuryDiagnostics\Server\bin\switch_ovo_agent.vbs
-server hpswros055.ovrtest.adapps.hp.com -cert_srv
hpswros055.ovrtest.adapps.hp.com
64-bit OS
Server is currently set to " need to register 'hpswros055.ovrtest.adapps.hp.com'
Certificate server is currently set to " need to register
'hpswros055.ovrtest.adapps.hp.com'
```

- 2 Select Save Registration again in the Business Service Management Registration page for Diagnostics.

Event Based Health Indicator Status Troubleshooting Flow

When integrated with Business Service Management 9.0 Diagnostics sends Health Indicator status events to the Business Service Management gateway server.

The following gives the troubleshooting flow you can use if there are problems with Health Indicator status events sent to Business Service Management.

To troubleshoot problems with HI status events:

- 1 Verify whether Diagnostics writes to **bachi_data.log** on Health Indicator status change on a probe metric:

Entry:

```
C:\latency\jbossas\bsavm12.ovrtest.adapps.hp.com|17b87476ac6de3938ff9898cd19c8
bd8|mercury|Default Client|1|J2EE|PROBE|2010-05-17 13:40:51|latency
[BpmTxJpalImpl.getBamNodeStatusKey() (144.5µs > 122.6µs)]
```

- 2 Verify whether **opcle** is up via **ovc -status**:

opcle	OVO Logfile Encapsulator	AGENT,EA	(5528)	Running
opcmsga	OVO Message Agent	AGENT,EA	(5460)	Running

- 3 Check the agent log file for errors (for example, unable to communicate to BSM server or certificate errors).

C:\Documents and Settings\All Users\Application Data\HP\HP BTO Software\log\System.txt

- 4 Enable tracing if necessary:

```
ovconfchg -ns eaagt -set OPC_TRACE TRUE -set OPC_TRC_PROCS opcle -set  
OPC_TRACE_AREA ALL
```

Traces are written to **C:\Documents and Settings/All Users/Application Data/HP/HP BTO Software/tmp/OpC** and **/var/opt/OV/log/tmp**.

To test the event channel on the Business Service Management Gateway Server:

- 1 Check if OPR (hpbsm_opr-backend) is running by manually submitting an event.
 - a First get the CMDBID of a CI that Diagnostics populates (look in the BSM Diagnostics Probe and Infrastructure view).
 - b Then go to **opr\support** and run the following (replace the CMDBID in bold with the one from the previous step):

```
sendEvent.bat -s critical -t foo -eh CPU Critical -rch  
UCMDB:7b75a57ee89fe6c076ce8d258be4a971
```

2 Verify the flow in `log\opr-backend\opr-backend.log`:

```

2010-05-11 06:17:18,981 [Thread-37] INFO Step.process(45) - Pipeline Step
'PipelineEntry': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO Step.process(45) - Pipeline Step
'CIResolver': got 1 events
2010-05-11 06:17:18,981 [Thread-37] ERROR
EventChannelCiResolver.resolveHost(172) - No host CI found for event
com.hp.opr.common.model.Event@4ef05e84[865b7200-5cff-71df-00eb-0f2bf8e70000,
Back to normal: Threshold violation(s) for latency on
mercury:bsavm12.ovrtest.adapps.hp.com,<null>,OPEN,NORMAL,NONE,J2EE,<null>,
<null>,UCMDB:17b87476ac6de3938ff9898cd19c8bd8,<null>,<null>,<null>,com.hp.opr.
common.model.ResolutionHints@6cd5499[<null>,ROS84604HAE.ovrtest.adapps.hp.c
om,15.43.248.231,ad2c79b2-9af7-7543-002d-ceeb548960bc],com.hp.opr.common.mo
del.ResolutionHints@126d0c4c[Diagnosics:mercury,ROS84604HAE.ovrtest.adapps.h
p.com,15.43.248.231,ad2c79b2-9af7-7543-002d-ceeb548960bc],<null>,<null>,false,-1,
-1,[],{}},Tue May 11 06:17:18 PDT 2010,Tue May 11 06:17:18 PDT 2010,Tue May 11
06:17:18 PDT
2010,0,latency:Normal,<null>,<null>,Diagnostics,latency,N:17b87476ac6de3938ff9898
cd19c8bd8:latency,^<*>:17b87476ac6de3938ff9898cd19c8bd8:latency$,N|latency|jbos
sas|bsavm12.ovrtest.adapps.hp.com|17b87476ac6de3938ff9898cd19c8bd8|mercury|D
efault Client|1|J2EE|PROBE|2010-05-11
06:16:48|latency,false,com.hp.opr.common.model.MatchInfo@35425b07,<null>,<null>]
2010-05-11 06:17:18,981 [Thread-37] INFO Step.process(45) - Pipeline Step
'CiVariableReplacer': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO Step.process(45) - Pipeline Step
'DowntimeProvider': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO Step.process(45) - Pipeline Step
'EtiResolverByHint': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO Step.process(45) - Pipeline Step
'EtiResolverByRule': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO Step.process(45) - Pipeline Step
'HIUpdater': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO MarbleHealthSubmitter.submit(129) -
submitting 1 health updates for customer 1
2010-05-11 06:17:18,981 [Thread-37] INFO Step.process(45) - Pipeline Step
'ResolutionCompleted': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO Step.process(45) - Pipeline Step
'IMDBStore': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO Step.process(45) - Pipeline Step
'PairwiseCorrelation': got 1 events

```

3 Enable more tracing if needed:

```
HPBSM\conf\core\Tools\log4j\opr-backend\opr.backend.properties
```


General Reference Information

This section includes general reference topics.

This chapter includes:

- Using UNIX Commands on page 781
- Using Regular Expressions on page 782
- Multi-Lingual User Interface Support on page 790

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**. After you finish selecting all your options, type 0 and press **Enter** again to confirm your choices.
- If you 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**.

Using Regular Expressions

When you specify the instrumentation definitions for each probe 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.

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, see 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 785).

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.

Here is an 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 787. For more information on the | regular expression character, see "Matching One of Several Regular Expressions" on page 788.

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.

Here is an 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, `_`).

Here is an 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**.

Here is an 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.

Here is an 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 must 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)`

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 Service Management 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 are not yet installed. If you choose a language in your Web browser whose fonts are not 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.



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 731.

This chapter includes:

- ▶ Task 1: Prepare the target database on page 793
- ▶ Task 2: Determine which metrics you want to export on page 794
- ▶ Task 3: Determine the frequency and the recovery period on page 798
- ▶ Task 4: Modify the data export configuration file on page 799
- ▶ Task 5: Monitor the data export operation on page 802
- ▶ Task 6: Verify the results on page 804
- ▶ Task 7: Select the data from the target database on page 805
- ▶ Sample Queries on page 805

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 Diagnostics commanding 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 must 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

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:

$$(\text{current time}) - (\text{recovery-periods} * \text{frequency})$$

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 were 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-uri>jdbc:sqlserver://testapps.hp.com:1433;databaseName=DIAG</
connection-uri>
  <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 112.

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 in the <query>:

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>
```

The following entry creates a query that runs every hour and returns every hour's web services latency metrics for all probe groups starting from April 22 at 3pm:

```
<query id="Web-Services" frequency="1h" recovery-periods="2">
  <entity-path>/probegroup/probe/index[name='services']/service</entity-path>
  <init-query-time>2009-04-22T15:00:00</init-query-time>
</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 “Using Regular Expressions” on page 782.

- 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"/>
```

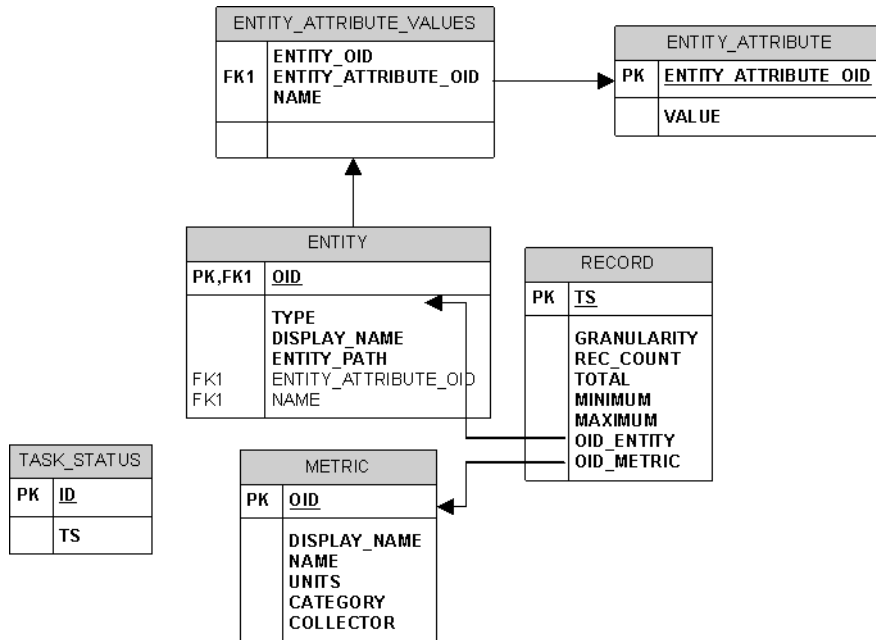
- 9 Save the changes to the `data-export-config.xml` file.

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.

- ▶ 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>
```

Table - dbo.METRIC	Table - dbo.TASK_STATUS	Table - dbo.RECORD	Table - dbo.ENTITY*	Summary
OID	DISPLAY_NAME	NAME	UNITS	CATEGORY
39fe732d4a8be...	JDBc [MedRec...	JDBc [MedRecEAR@MedRecAp...	COUNT	JDBc [MedRec
3aa2764fb92b5...	EJB Response Time	EJB Response Time	MILLISECOND5	EJB
3aa4caf58987c7...	IOBusyTime	IOBusyTime	MILLISECOND5	SqlServer
3ad15a083c484...	User Commits Pe...	User Commits Percentage	PERCENT	Oracle
3ae792e1509c6...	Servlets Respon...	Servlets Response Time	MILLISECOND5	Web Applicat
3b49b2d59252d...	JDBc [MedRecGl...	JDBc [MedRecGlobalDataSourc...	MILLISECOND5	JDBc [MedRec
3b769a0780452...	Physical Writes ...	Physical Writes Direct Per Txn	COUNT	Oracle
3be4dbe933822...	JDBc [MedRecP...	JDBc [MedRecPool-PointBase] ...	COUNT	JDBc [MedRec
3c0afe8166fbc...	JDBc [MedRecE...	JDBc [MedRecEAR@MedRecAp...	COUNT	JDBc [MedRec
3c34553aeb51a...	JDBc [wlsbjmsrp...	JDBc [wlsbjmsrpDataSource] St...	COUNT	JDBc [wlsbjms

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 the "Diagnostic Fragments.sln" file.

Sample Queries

This section includes some query examples for dealing with time, querying totals and querying averages.

Dealing with Time

To query for data from 8 am to 5 pm, your query should specify the start time as 8:00 and the end time as 16:59. If you specify 17:00 as your query's end time, you will include data from the next time bucket which could be 17:00 to 18:00. This is reflected in the examples that follow.

Querying for totals

Use the sum() function to calculate totals of metrics.

Example:

This query will calculate the total number of threshold violations between 6pm and 8pm for the Server Request with the entity path: Default Client / Default / ROS54770TST_Diag80_JDK_15.

```
select entity_display_name as Server_Request, sum(total) as
Avg_Latency,sum(total) as Tot_Latency, sum(rec_count) as count, metric_name,
units, name, entity_path
from DIAG.DBO.REG_FRAGMENT_TYPE_METRICS_VIEW
where metric_name = 'threshold_violations'
  and ts between '2009-06-11 18:00:00.000' and '2009-06-11 19:59:00.000'
  and entity_path = 'Default Client / Default / ROS54770TST_Diag80_JDK_15 / '
group by entity_path, entity_display_name , metric_name,units, name
order by entity_path, entity_display_name
```

Querying for averages

To calculate the average metric value, divide the total metric value by its count. The rec_count field will contain the count.

Note: The count for the Soap Fault metric is set to its total and hence it is not possible to calculate an average value for this metric. Only a total calculation is possible for this metric. In Diagnostics version 7.5, this was also true for the Threshold Violations metric. From Diagnostics 8.0 onwards, the count is available for Threshold Violation and hence it is possible to calculate an average for this metric.

Example:

This query will calculate the average latency between 6pm and 8pm for the Server Request with the entity path: Default Client / Default / ROS54770TST_Diag80_JDK_15.

```
select entity_display_name as Server_Request, sum(total)/sum(rec_count) as
Avg_Latency,sum(total) as Tot_Latency, sum(rec_count) as count, metric_name,
units, name, entity_path
from DIAG.DBO.REG_FRAGMENT_TYPE_METRICS_VIEW
where metric_name = 'latency'
  and ts between '2009-06-11 18:00:00.000' and '2009-06-11 19:59:00.000'
```

```
and entity_path = 'Default Client / Default / ROS54770TST_Diag80_JDK_15 / '
group by entity_path, entity_display_name , metric_name,units, name
order by entity_path, entity_display_name
```

Example:

This query will calculate the average number of threshold violations between 6pm and 8pm for the Server Request with the entity path: Default Client / Default / ROS54770TST_Diag80_JDK_15.

```
select entity_display_name as Server_Request, sum(total)/sum(rec_count) as
Avg_Latency,sum(total) as Tot_Latency, sum(rec_count) as count, metric_name,
units, name, entity_path
from DIAG.DBO.REG_FRAGMENT_TYPE_METRICS_VIEW
where metric_name = 'threshold_violations'
and ts between '2009-06-11 18:00:00.000' and '2009-06-11 19:59:00.000'
and entity_path = 'Default Client / Default / ROS54770TST_Diag80_JDK_15 / '
group by entity_path, entity_display_name , metric_name,units, name
order by entity_path, entity_display_name
```

Index

Symbols

- .NET advanced configuration
 - ASP.NET applications 228
- .NET agent
 - .NET framework requirements 233
 - add an HP software product to configuration 524
 - advanced configuration 517
 - collect additional metrics 549
 - configure for outgoing HTTPS 706
 - enabling and disabling 256
 - HTTP proxy 557
 - installing 225
 - installing as a Profiler 241
 - installing with a Diagnostics Server 243
 - launching the installers 236
 - performance counters 550
 - points files 226
 - probe_config file elements 455
 - running the installation 237
 - system metrics collector 575
 - system requirements 234
 - uninstall 256
 - uninstalling 256
 - upgrade 763
 - verifying connection 249
 - verifying installation 254
- .NET agent installer
 - how it works 226
- .NET agent version information 255
- .NET configuration file 455
- .NET instrumentation 337
- .NET layers 370
- .NET points files 339

A

- active 272
- active users
 - list of 653, 679
- active.products property 404
- add-in
 - LoadRunner Diagnostics 633
- adonet.points 339
- advanced options
 - show or hide 653
- after
 - code 269
- agent
 - .NET system requirements 36
 - Java 121
 - agent, .NET, *See* .NET agent
 - agent, Java, *See* Java agent
 - agent.java properties file 262
 - alert properties 654
 - AM product mode 405, 406
 - appdomain element 456
 - application level permissions 666
 - application name
 - configure show in server request 446
 - application server
 - multiple JVM instances 406
 - application server configuration
 - generic 216
 - JBoss 210
 - Oracle 201
 - SAP NetWeaver 214
 - WebLogic 8.1 195
 - WebLogic 9.x and 10 200
 - WebSphere 181
 - WebSphere 5.x, 6.0 182
 - WebSphere 6.1 190
 - application server startup script

Index

- generic 216
- application server startup scripts
 - modifying 180
- application servers
 - supported 28
- application servers, supported 28
- ApplicationPoolIdentity 550
- args 269
- argument capture 349
- arguments
 - mandatory for .NET 341
 - mandatory for Java 265
 - optional for .NET 342
 - optional for Java 267
- ASP.NET applications
 - automatic configuration 229
 - discovering 228
- aspnet.points 339
- asynchronous thread sampling 448
- authentication element 457
- authorization and authentication 658
- auto_detect.points file 262
- Automatic Method Trimming
 - controlling 417
- B**
- backslash (\) 784
- backup 745
- backup directories 747
- backup symbol table
 - configuring 747
- before
 - code 269
- bufferpool element 458
- Business Availability Center
 - assigning permissions for Diagnostics users 626
 - Diagnostics configuration 624
- Business Service Management
 - Admin>Diagnostics 624
 - changing the Diagnostics Server details 624
 - configure HTTPS communications 708
 - Diagnostics downloads 625
 - Diagnostics registration 624
 - samples queue size 397
 - setup to use Diagnostics 615
 - specifying Diagnostics Server details 618
- Business Service Management server HTTPS communication 708
- business transactions
 - create 653
- C**
- caller 269
- capture points
 - .NET 338
- capture points files
 - mandatory point entries 265, 341
 - optional point entries 267, 342
 - using for instrumentation 262, 338
- captureexceptions element 459
- change
 - privilege level 659
- CI population
 - discovery of IIS metadata 230
- CIs
 - Diagnostics populates in BSM 229, 629
 - synchronize with BSM 653
- class 265, 341
- class map capture 288
- ClassLoader class, recreating 770
- CLP 301
- code snippet helper 279
- code snippets 273
 - secure code-key 286
- code-key
 - generating 287
- collect CPU timestamps 452
- collector
 - configure for incoming HTTPS 699
 - configure for outgoing HTTPS 706
 - configuring active system property files 100
 - installing on UNIX 90
 - installing on Windows 82
 - starting and stopping 115

- supported platforms 82
 - uninstalling 117
 - upgrade 764
 - verifying installation 114
 - version information 117
 - collector in System Health Monitor 719
 - communication diagrams 751
 - compatibility with other HP Software 767
 - component and communication diagrams 751
 - component communications 654
 - Components page 652
 - compressed files 739
 - configure Diagnostics 649
 - configuring application servers, *See* application server configuration
 - consumer ID
 - look deeper in xml to find 439
 - consumer IDs
 - configuring 434
 - consumeridrules element 460
 - CORBA instrumentation 310
 - CPU 431
 - CPU time metrics 431
 - configuring 431
 - CPU timestamp 452
 - cpu.timestamp.collection.method 432
 - cptime element 461
 - credentials element 462
 - cross VM
 - RMI instrumentation 311
 - custom dashboard 652
 - custom data 732
 - custom sub-layer instrumentation 290
 - custom_code.properties 274
 - customer information 654
- D**
- data compression 739
 - Data exporting
 - about 793
 - configuration file 799
 - frequency 798
 - recovery period 798
 - sample scripts 805
 - supported metrics 794
 - target database 793, 799
 - data management 731
 - data management, *See* Diagnostics data management
 - data port 246
 - data retention 739
 - database name
 - automatically discovered 109
 - data-export-config.xml file 799
 - days data 736
 - deep_mode 268, 293, 343
 - deployed into
 - show for server request 446
 - depth element 464
 - depth trimming 418, 531
 - detail 269
 - Diagnostics
 - integration with LoadRunner 637
 - integration with Performance Center 643
 - integration with Business Service Management 615
 - Diagnostics components
 - description 26
 - host requirements 29
 - synchronizing time between 374
 - Diagnostics data management
 - backing up data 745, 749
 - custom screen data 732
 - data sizes & data retention 739
 - performance considerations 744
 - performance history data 734, 737
 - Diagnostics mediator
 - firewall configuration 565
 - Diagnostics Server
 - adjusting heap size 378
 - administration 649
 - advanced configuration 373
 - changing default port 383
 - configuration pages 394
 - configure for incoming HTTPS 692
 - configure for outgoing HTTPS 702
 - configuring 66
 - configuring for large installation 378
 - configuring for multi-homed

- environments 384
- configuring LoadRunner offline file 394
- configuring time synchronization 376
- configuring, advanced 373
- high availability 390
- HTTP proxy 556
- information required for installation 38
- installing 52
- jetty.xml file, modifying 385
- jetty.xml file, sample 387
- launching the installers 52
- optimized to handle more probes 398
- override assignment for a probe 393
- overriding default host name 383
- reducing memory usage 388
- reducing memory use 388
- running the installation 54
- setting event host name 384
- setup in BSM 618
- starting and stopping 63
- synchronizing time between
 - Diagnostics components 374
- system requirements 44
- UI 649
- uninstalling 67
- upgrade 757
- verifying installation 65
- Diagnostics Server host name
 - overriding 383
- Diagnostics Server installation 52
- Diagnostics Server port
 - changing default 383
- Diagnostics Server version information 66
- Diagnostics setup menu 53
- Diagnostics UI 650
- diagnosticsserver element 465
- documentation updates 21

E

- embedded java probe and HTTPS 705
- enable.stack.trace.sampling 448
- encrypted password 112
- EncryptPassword.jsp 112

- enterprise level permissions 666
- eve files 642
- event host, setting name for 384
- exception data
 - limiting 536
- exception tree data
 - limiting 422
- exceptiontype element 467
- exclude element
 - parent is captureexceptions 468
 - parent is lwmd 469
- excludeassembly element 470
- execute
 - privilege level 659
- exporting data 793

F

- files 653
- filter
 - system health monitor 726
- filter element 471
- firewall
 - enabling communications through 559

G

- granularity 735
- GROUPBY for JMX metrics 608

H

- heap size 378
 - adjusting for large installation 378
 - adjusting in startup script 218
- high availability 390
- host requirements, Diagnostics components 29
- hours data 736
- HP Software Support Web site 20
- HP Software Web site 20
- HTTP proxy communication
 - .NET Probe 557
 - Diagnostics Server in Mediator Mode 556
 - enabling 555

- Java Probe 557
- httpclient element 472
- httpheaderrule element 473
- httpheaderrules element 474
- HTTPS
 - configuration steps 690
 - configure for incoming communication 692
 - configure for outgoing communication 702
- HTTPS communication
 - enabling for Business Service Management server 708
- HTTPS communications
 - enabling between components 689
- I**
- IBM JVM
 - instrumentation notes 167
- id element 475
- ignore_cl 267, 342
- ignore_method 267, 343
- ignore_tree 267
- ignoreScope 268, 343
- IIS
 - restarting 253
- IIS Host Header 540
- iis_discovery_data.xml 230
- include element
 - parent is captureexceptions 477
 - parent is lwmd 478
- installation
 - .NET agent 225
 - collector 82
 - Diagnostics Server 51
 - Diagnostics Server (Windows) 52
 - gathering information for 37
 - interruption during 770
 - Java agent 121
 - order of 45
 - planning 37
 - recommended order 45
- installation requirements
 - .NET agent 36
 - .NET Diagnostics Profiler 36, 235
 - Diagnostics Server 30
 - Java agent 34
 - Java Diagnostics Profiler 35
 - instance tree files 738
 - instant on license 74
 - instrumentation
 - deep_mode examples 354
 - instrumentation
 - .NET applications 337
 - .NET remoting 359
 - access filter 298
 - advanced 307
 - allocation analysis 299
 - always trim 303
 - argument capture 295, 349
 - attributes in instance trees 298
 - caller side 299, 348
 - capture for trended methods view 291, 346
 - capture with controlled scope 293
 - collection leak pinpointing 301
 - correlation across multiple threads 312
 - CPU time collection 304
 - custom layers 290, 345
 - deallocation 299
 - deep_mode hard and soft 293
 - direct recursion 299
 - edit points 325
 - edit points from the profiler 322
 - enable at runtime 302
 - enabling for .NET 251
 - fragment local storage 315
 - HTTP client calls 349
 - ignore specific methods 290, 346
 - Java applications 261
 - LWMD 300
 - never trim 303
 - non-ASP.NET applications 355
 - object lifecycle 301
 - printing 304
 - RMI 311
 - RootRename 297
 - thread local storage 311
 - TransactionVision related 271
 - URI aggregation 309

Index

- using annotations 319
- using wildcards 290, 345
- view current 323
- web services 296
- instrumentation element 479
- instrumentation examples
 - .NET 344
 - Java 289
- instrumentation overhead 305, 369
- iprule element 480
- iprules element 481

J

- JAAS authentication 679
- Java agent
 - add an HP software product to configuration 403
 - advanced configuration 401
 - automatic method trimming 417
 - configure for incoming HTTPS 696
 - configure for outgoing HTTPS 704
 - configuring and installing, about 122
 - configuring for multiple JVMs 406
 - configuring for proxy server 419
 - controlling log messages 414
 - edit probe settings from the profiler 446
 - failed operation 770
 - HTTP proxy 557
 - installing 121
 - installing as a Profiler 126
 - installing using generic installer 156
- JMX metric collector 605
- JMX metrics collector 605
- launching unix installer 141
- launching windows installer 123
- log messages 414
- metrics collectors 587
- proxy server 418
- reverse HTTP for agent in SaaS 419
- running the installation 142
- running windows installation 125
- silent installation 158
- system metrics 593
- system requirements 34

- uninstalling 163
- Unix install 139
- upgrade 760
- upgrade steps 760
- Java agent installer
 - how it works 122
- Java agent windows install 123
- Java Diagnostics Profiler
 - disabling 412
 - WAS startup error 771, 772
- Java instrumentation 261
 - code snippets 273
- Java layers 335
- Java Profiler
 - PRO product mode 404, 406
- Java profiler
 - configuration tab 322, 446
- Java system properties 413
- JBoss application server startup scripts
 - modifying 210
- JBoss, application server configuration 210
- JDK/JRE executable 128, 136
- jetty.xml 385
- jetty.xml file
 - modifying 385
 - sample 387
- JMS temporary queues
 - grouping 445
- JMX metrics
 - accessing 605
 - add custom 606
 - collecting 607
 - custom 610
 - GROUPBY 608
 - understanding patterns 612
- JMX metrics collector 605
 - configuring 606
- JRE Instrumenter 166
 - about 166
 - running 169
- JRE instrumenter
 - how it works 167

K

- keyword 267, 342

Knowledge Base 20

L

large deployments
 data management 744
 latency element 482
 latency trimming 417, 527
 layer 266, 341
 layer_type 344
 layers
 .NET 370
 .NET layers 370
 about instrumentation 334, 370
 Java 334
 Portals 336
 view current 324
 layers page
 instrumentation control and edit 306
 layers tree 326
 layerType 272
 LDAP authentication 681
 license 653
 light-weight memory Diagnostics (LWMD)
 533
 Linux custom metrics 601
 llicensing 74
 LoadRunner
 Diagnostics probe metrics 639
 Diagnostics, configuring scenarios to
 use 638
 Diagnostics, setting up 638
 firewall configuration 571
 installing add-in for Diagnostics 633
 integration with Diagnostics 637
 LoadRunner offline analysis
 reducing file size 395
 LoadRunner Offline File
 advanced Diagnostics Server
 configuration 395
 estimating size of 395
 reducing size of 395
 LoadRunner scenarios
 configure Diagnostics parameters 638
 log messages 414
 logdirmgr element 483

logging 653, 654
 controlling 414
 disable .NET agent 257
 disabling 538
 logging element
 parent is appdomain probeconfig
 process 486
 parent is instrumentation 484
 logging HTTP 349
 LR/PC runs
 assign Diagnostics server for a probe
 393
 lwmd element 488
 LWMD *See* light-weight memory Diagnostics
 (LWMD)

M

MAC address 77
 major time periods 735
 max.search.level.depth 439
 mediator element 489
 memory diagnostics 654
 memory usage, reducing 388
 message handler 219
 method 265, 341
 method_access_filter 267
 metric collector default port 596
 metric element 492
 metric patterns 612
 metric port 246
 metrics
 add custom JMX 606
 add custom system 580, 597
 list available 589
 list available JMX 607
 metrics agent 575
 metrics collector
 modifying default port 596, 597
 system metrics 595
 metrics element 491
 metrics entries 577, 590, 610
 metrics, Diagnostics system health monitor
 720
 metrics.config
 .Net 577

Index

- Java 587, 595
 - keywords 583
- MI listener 564
- minor time periods 735
- modes
 - .NET agent 524
 - Java agent 403
- modes element 494
- months data 736
- MQ probe
 - configuring 105
 - permissions required 105
- multi-homed environments 384
- multiple JVM instances 406

N

- nanny
 - start and stop using 63
- NET Diagnostics Profiler
 - enabling 527
- non ASP.NET applications 232
- non-ASP.NET applications
 - instrumenting 355

O

- ODB
 - timing for adding web service CIs 629
- offline analysis files
 - improve transfer 642
- offline.xml 639
- online cache 654
- Oracle 10g JAX-RPC
 - SOAP message handler 222
- Oracle application server startup scripts
 - modifying 201
- Oracle probe, configuring 103
- Oracle, application server configuration 201
- order of installation, recommended 45
- OVTA like points 332

P

- parked fragments 313
- password obfuscation 112
- patch installation instructions 755

- perfmon 550
 - system metric counters 580
- Performance Center
 - firewall configuration 571
 - integration with Diagnostics 643
 - load tests, configuring to use
 - Diagnostics 645
 - offline analysis files 646
 - setting up to use Diagnostics 644
- performance counters
 - access rights 550
- performance history data 733
- performance monitor users group 550
- Permissions Page 665
- permissions, *See* user permissions
- persistence
 - configure 740
 - data file types 736, 737
 - data files 739
- persistence directory
 - history data 735
- persistence.purging.threshold 742
- PMI
 - EXPAND_PMI 609
- points
 - .NET 338
 - arguments defined for .NET 340
 - arguments defined for Java 264
 - edit 328
 - Java 262
 - specifying for Java applications 274
- points element 496
- points list 327
- portal layers 336
- preinstallation considerations 44
- priority 272
- privileges
 - user 659
- PRO product mode 404, 406
- probe administration UI 425
- probe aggregator 235
- probe level permissions 666
- probe metrics
 - additional .NET 492
 - collect additional 549
- probe metrics in offline analysis 639

- probe name
 - unique 409
 - probe settings
 - edit from the profiler 446
 - Probe, .NET
 - enabling 518
 - Probe, .NET advanced configuration
 - classes/methods 518
 - customizing instrumentation for ASP.NET applications 518
 - depth trimming 531
 - disabling logging 538
 - elements and attributes 455
 - latency trimming and throttling 527
 - light-weight memory Diagnostics (LWMD) 533
 - overriding default Probe host name 539
 - probe, *See* Java agent or .NET agent
 - probe.id 409
 - probe_config.xml 226
 - elements and attributes defined 455
 - probeconfig element 497
 - process element 498
 - product security 658
 - profiler
 - disabling 412
 - profiler element 500
 - profilers
 - authentication for standalone 428, 541
 - proxy
 - enabling communication through 555
 - proxy server
 - configuring for agent 418
 - purging
 - symbol table 742
- Q**
- query 652
 - queue size 397
- R**
- reflector 522
 - refresh rate
 - system health monitor 727
 - registrar 652
 - registrar view 713
 - regular expressions 782
 - regular expressions, backslash (\) 784
 - remote-backup.sh 746
 - remoting
 - instrumentation for 359
 - requirements, Diagnostics Server 44
 - REST services
 - configuring as web services 445
 - restore 748
 - resumeFragment 313
 - retention 739
 - reverse HTTP
 - configuring for SaaS 419
 - enabling and disabling 419
 - RMI instrumentation 311
 - roles 660
 - rootRenameTo 272
 - rum element 502
- S**
- SaaS
 - configuring reverse HTTP for agent 419
 - reverse HTTP 419
 - sample element 504
 - samples queue size 397
 - samples sent to BSM 628
 - sampling
 - thread stack trace 448
 - sampling
 - server requests 447
 - SAP NetWeaver application server startup
 - scripts
 - modifying 214
 - SAP NetWeaver, application server
 - configuration 214
 - SAP probe, configuring 100
 - scalability information 32
 - scope 268, 343
 - secure communications 689

Index

- security 653
- security permissions 659
- server request sampling 447
- Server, Diagnostics *See* Diagnostics Server
- ServiceGuard
 - high availability server 391
- setup menu 53
- signature 265, 344
- silent install
 - log file 62, 99, 159
 - specify temp dir 62, 99, 159
- SiteMinder JAAS authentication 688
- SiteMinder JAAS LoginModule
 - reverse proxy 684
- snapshot
 - system health monitor 728
- snapshot persistence 743
- SOAP Fault Data, configuration for .NET
 - probes 548
- SOAP fault data, configuring for Java probes 444
- SOAP faults
 - configuring capture 444, 548
- SOAP message handler 161, 219
 - disable 220
 - loading 221
- soapcapture element 505
- soaprequestforsoapfault element 507
- soaprule element 508
- soaprules element 509
- Solaris custom metrics 600
- SQL Server collector
 - Windows NT security 110
- SQL Server probe
 - configuring 108
- stack trace data
 - limiting 538
- stack trace sampling 448
 - examples 449
 - troubleshooting 451
- startup script
 - multiple probes 410
- summary files 737
- support collector 773
- symbol table backup
 - configuring 747
- symbol table files 737
- symbols element 510
- synchronize web service CIs 629
- synchronize with BAC 653
- System Health Monitor
 - accessing from browser 713
 - customizing display 724
 - filter by customer 726
 - icons 715
 - snapshots, exporting 728
- system health monitor 712
 - accessing 712
 - component configuration
 - information 721
 - component information, viewing
 - detailed 719
 - component log information 722
 - component map 715
 - component metric information 720
 - component status information 722
 - displaying legend 724
 - displaying load 725
 - legend 715
 - miscellaneous information 722
 - refreshing display 727
 - snapshots, importing 729
 - system log information 723
 - tooltips 717
 - view troubleshooting tips 723
 - viewing log history 723
- system metrics
 - .NET 575
 - about 593
 - add custom 580, 597
 - capture 575
 - customizing in Windows 597
 - customizing on Linux host 601
 - customizing on Solaris host 600
 - default metrics collected by .NET
 - metrics agent 576
 - default metrics collected by Java agent 594
 - Java 593
 - metrics collector 595
 - metrics collector entries 589
 - modifying captured metrics 592

- stopping capture 592
- system metrics collector 593
- system requirements 29
 - .NET agent 36
 - Diagnostics Server host 30
 - Java agent host 34
 - with .NET probes 32
 - with Java probes 31
- system/ 577

T

- temporary queues
 - grouping into a single node 445
- thread stack trace sampling 448
- throttling 530
- TIBCO ActiveMatrix configuration 219
- time synchronization 374
- time synchronizing between components 374
- timestamping 431
- timestamps
 - CPU 452
- topology element 511
- TransactionVision related instrumentation 271
- trend files 738
- trial license 74
- trim element 512
- trimming
 - controlling depth 418
 - controlling latency 417, 527
 - fragment name based 388
- trimming parameters 381
- Troubleshooting and Knowledge Base 20
- troubleshooting Diagnostics 769

U

- UI requirements
 - JRE version 30
- unique probe name 409
- UNIX installer
 - how to select an option 781
- updates, documentation 21
- upgrade paths

- general recommendations 756
- upgrade procedures 755
- upgrade recommendations 756
- upgrading earlier versions of Diagnostics 47
- use.cpu.timestamps 432
- user permissions
 - about 658
 - accessing Diagnostics, default users 661
 - assigning for Diagnostics users in Business Availability Center 626
 - managing user details 662, 669
- user roles 660
- users
 - list of 653, 679

V

- view
 - privilege level 659
- VMWare
 - time synchronization 518
- VMware
 - time synchronization 421
- VMware and CPU time metrics 432, 452
- vmware element 513

W

- WCF monitoring
 - requirements and limitations 234
- WCF.points 339
- WDEDelivery queue size 397
- WebLogic 8.1, application server configuration 195
- WebLogic 9.x and 10, application server configuration 200
- WebLogic application server startup scripts modifying 195
- weblogic over T3
 - instrumentation for cross VM 311
- WebLogic, application server configuration 195
- webservice element 514
- WebSphere 5.x and 6.0, application server configuration 182

Index

- WebSphere application server startup script
 - modifying 181
- WebSphere JAX-RPC
 - SOAP message handler 221
- WebSphere, application server configuration
 - 181
- weeks data 736
- windows credentials
 - SQL server collector 110
- Windows custom metrics 597
- ws element 515

X

- xml
 - look deeper in xml for consumer ID
 - 439
- xvm element 516

Y

- years data 736

Z

- z/OS
 - installing the Java agent 154
 - Java agent install 154
- zip files 739

