

# HP Diagnostics

for the Windows®, Unix and Linux operating systems

Software Version: 9.21

---

## Installation and Configuration Guide

Document Release Date: September 2013

Software Release Date: November 2012



# 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 - 2012 Hewlett-Packard Development Company, L.P.

## Trademark Notices

Java is a registered trademark of Oracle and/or its affiliates.

Adobe® and Acrobat® are trademarks of Adobe Systems Incorporated.

Microsoft®, Windows®, Windows® NT, Windows® XP and Windows Vista® are U.S. registered trademarks of Microsoft Corporation.

Oracle® is a registered trademark of Oracle and/or its affiliates.

UNIX® is a registered trademark of The Open Group.

## Acknowledgements

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

This product includes software developed by the Spice Group (<http://spice.codehaus.org>).

For information about open source and third-party license agreements, see the Documentation directory on the product installation media.

## 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](http://h20230.www2.hp.com/new_access_levels.jsp)**

---

# Table of Contents

<b>Welcome To This Guide .....</b>	<b>19</b>
How This Guide Is Organized .....	19
HP Diagnostics Online Documentation.....	21
Additional Online Resources.....	22
Documentation Updates .....	23

## **PART I: PREPARING TO INSTALL**

<b>Chapter 1: Preparing to Install HP Diagnostics .....</b>	<b>27</b>
HP Diagnostics Components and Data Flow .....	28
Supported Application Servers and Environments .....	30
System Requirements for the Diagnostics Components.....	31
Information Required for Installation .....	40
Pre-installation Considerations.....	47
Recommended Order of Installation.....	48
Licensing HP Diagnostics .....	50
Upgrading from Earlier Versions of Diagnostics.....	50

## **PART II: INSTALLATION OF THE DIAGNOSTICS SERVER AND COLLECTORS**

<b>Chapter 2: Installing the Diagnostics Server .....</b>	<b>53</b>
Installing Diagnostics Servers .....	54
Verifying the Diagnostics Server Installation .....	67
Silent Installation of the Diagnostics Server .....	68
Starting and Stopping the Diagnostics Server.....	70
Licensing Your Diagnostics Software .....	72
More Information on Configuring Diagnostics Servers.....	72
Determining the Version of the Diagnostics Server .....	72
Uninstalling the Diagnostics Server.....	73
Manual Installation of OM Agent and IAPA Components.....	74
Manual Uninstall of OM Agent and IAPA Components .....	76

<b>Chapter 3: Licensing HP Diagnostics .....</b>	<b>79</b>
About HP Diagnostics Licensing .....	80
Types of Licenses .....	80
Licensing the Diagnostics Server in Commander Mode.....	81
View License Information .....	84
Licensing the Other Diagnostics Components.....	88
<b>Chapter 4: Installing Diagnostics Collectors.....</b>	<b>91</b>
About Installing the Diagnostics Collector.....	92
Accessing the Collector Installer .....	93
Installing the Collector .....	94
Silent Installation of the Diagnostics Collector .....	103
Installing the Diagnostics Collector Using the Generic Installer .....	104
How to Manually Add Another Collection Type After Installing the Collector .....	105
Configuring the Active System Property Files .....	106
Configuration for SAP NetWeaver-ABAP.....	106
Configuration for Oracle.....	110
Configuration for SQL Server .....	113
Configuration for MQ.....	117
Configuration for TIBCO EMS .....	120
Configuration for webMethods Broker .....	121
Configuration for VMware.....	123
Password Obfuscation .....	125
Verifying the Diagnostics Collector Installation .....	127
Starting and Stopping the Diagnostics Collector.....	128
Determining the Version of the Diagnostics Collector .....	130
Uninstalling the Diagnostics Collector.....	130

## **PART III: INSTALLATION AND SETUP OF THE JAVA, .NET AND PYTHON AGENTS**

<b>Chapter 5: Installing Java Agents</b> .....	<b>133</b>
Overview of the Java Agent Installation .....	134
Accessing the Java Agent Installer.....	135
Installing the Java Agent .....	137
Running the Java Agent Setup Module.....	141
About Preparing the Application Server for Monitoring .....	150
Register the Agent with the Diagnostics Servers .....	150
Verifying the Java Agent Installation.....	151
About Additional Configuration and Custom Instrumentation .....	152
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 .....	157
Setting File Permissions (UNIX Only).....	159
Determining the Version of the Java Agent.....	160
Uninstalling the Java Agent .....	160
<b>Chapter 6: Preparing Application Servers for Monitoring with the Java Agent</b> .....	<b>161</b>
About Preparing Application Servers for Monitoring .....	162
Examples for Configuring Application Servers .....	163
About the JRE Instrumenter and Different Options to Invoke.....	219
Other Configuration Options .....	232
<b>Chapter 7: Preparing Application Servers for Client Monitoring with the Java Agent</b> .....	<b>245</b>
About Client Monitoring .....	245
Enabling Client Monitoring.....	246
Configuring and Disabling Client Monitoring .....	248
Manually Instrumenting HTML/JSP Pages for Client Monitoring: ..	249

<b>Chapter 8: Installing .NET Agents</b> .....	<b>251</b>
Overview of the .NET Agent Installation .....	252
Accessing the .NET Agent Installer .....	254
Installing the .NET Agent.....	255
Post Install Tasks .....	277
Verifying the .NET Agent Installation.....	278
About Configuration of the .NET Agent for Diagnostics.....	279
About Configuration of the .NET Agent for TransactionVision.....	279
Discovery and Standard Instrumentation.....	282
Probe Aggregator Service .....	286
Monitoring NET Applications Deployed in Azure Cloud.....	287
Determining the Version of the .NET Agent .....	288
Enabling and Disabling the Diagnostics Agent for .NET.....	288
Disabling Logging.....	289
Enabling and Disabling Standard Instrumentation for Applications .....	290
Troubleshooting .NET Web Applications Not Discovered .....	292
Other .NET Agent Troubleshooting Tips .....	294
Uninstalling the .NET Agent.....	294
<b>Chapter 9: Installing and Setting Up Python Agents</b> .....	<b>295</b>
Diagnostics Python Agent Overview .....	296
System Requirements for the Diagnostics Python Agent .....	296
Installing Python Agents.....	297
Instrumenting a Python Application .....	300
Configuring the Python Agent .....	309
Description of the Parameters in the Points File .....	316
Description of Custom Code.....	318
Available Out-of-the-box Configurations .....	328
Reconnect/Reinitialize Event Channel After Server Reboot .....	333
Troubleshooting .....	333
Removing the Python Agent.....	334



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

<b>Chapter 10: Custom Instrumentation for Java Applications .....</b>	<b>337</b>
About Instrumentation and Capture Points Files .....	338
Coding Points in the Capture Points File .....	340
Defining Points With Code Snippets .....	348
Controlling Class Map Capture.....	364
Instrumentation Examples .....	365
Understanding the Overhead of Custom Instrumentation .....	381
Instrumentation Control on a Per Layer Basis.....	382
Advanced Instrumentation Examples.....	383
Configuring Cross VM Correlations for New or Custom Technologies .....	398
Tutorial for Configuring Cross VM Correlation for Custom Technologies .....	403
Maintaining Instrumentation from the Java Profiler UI .....	412
Default Layers Defined for Typical Java Classes and Methods .....	423
<b>Chapter 11: Custom Instrumentation for .NET Applications .....</b>	<b>427</b>
About Instrumentation and Capture Points Files .....	428
Locating the .NET Capture Points Files.....	429
Coding Points in the Capture Points File .....	430
Instrumentation Examples .....	435
Understanding the Overhead of Custom Instrumentation .....	461
Default Layers for Typical .NET Applications .....	462

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

**Chapter 12: Advanced Diagnostics Server Configuration.....465**  
Synchronizing Time Between Diagnostics Components.....466  
Configuring the Diagnostics Server for a Large Installation.....470  
Overriding the Default Diagnostics Server Host Name .....476  
Changing the Default Diagnostics Server Port .....476  
Migrating Diagnostics Server from One Host to Another .....477  
Configuring the Diagnostics Server for Multi-Homed  
Environments .....479  
Reducing Diagnostics Server Memory Usage .....483  
Configuring Server Request Name Based Trimming.....484  
Automating Composite Application Discovery in HP Diagnostics ..485  
Preparing a High Availability Diagnostics Server.....488  
Configuring Diagnostics for HP ServiceGuard (HA solution).....489  
Diagnostics Server Assignments (LoadRunner/Performance  
Center Runs) .....491  
Configuring the Diagnostics Server for LoadRunner  
Offline Analysis File Size.....492  
Configuring Business Service Management Sample Queue  
Size and Web Services CI Frequency .....495  
Configuring Diagnostics Using the Diagnostics Server  
Configuration Pages.....496  
Optimizing the Diagnostics Server in Production to Handle  
More Probes .....496  
Configuring a Custom Context Root.....497

<b>Chapter 13: Advanced Java Agent and Application</b>	
<b>Server Configuration</b> .....	<b>499</b>
Advanced Configuration Overview.....	500
Disabling the Java Diagnostics Profiler .....	501
Controlling Probe Logging.....	502
Setting the Probe's Host Machine Name.....	503
Specifying a Different Probe IP Address .....	505
Set the Active Products Mode.....	505
Controlling Automatic Method Trimming on the Agent.....	508
Configuring URI Truncation, Mapping and Trimming.....	510
Configuring an Agent for a Proxy Server .....	511
Time Synchronization for Probes Running on VMware.....	512
Limiting Exception Tree Data .....	512
Diagnostics Probe Administration Page.....	515
Authentication and Authorization for Diagnostics Java Profilers .....	518
Configuring Collection of CPU Time Metrics.....	521
Configuring Consumer IDs .....	524
Configuring SOAP Fault Payload Data.....	535
Configuring REST Services .....	537
Customizing Grouping JMS Temporary Queue/Topics .....	537
Configuring SQL Query Parsing .....	537
Configuring Display of Application Name for Server Requests .....	538
Maintaining Probe Settings from the Java Profiler UI .....	539
Generating Performance Reports for JUnit Tests .....	547
<b>Chapter 14: Understanding the .NET Agent Configuration File</b> .....	<b>551</b>
Understanding the .NET Agent Configuration File .....	551
.NET Agent Configuration Elements.....	552

<b>Chapter 15: Advanced .NET Agent Configuration .....</b>	<b>627</b>
Time Synchronization for .NET Agents Running on VMware .....	628
Customizing the Instrumentation for ASP.NET Applications .....	628
Discovering the Classes and Methods in an Application .....	634
Controlling Which HP Software Products the Agent can Work With .....	637
Configuring Support for MSMQ Based Communication .....	641
Configuring Latency Trimming and Throttling .....	641
Configuring Depth Trimming.....	646
Configuring URI Truncation and Mapping .....	647
Configuring the .NET Agent for Lightweight Memory Diagnostics .....	649
Limiting Exception Stack Trace Data .....	652
Disabling Logging.....	655
Overriding the Default Probe Host Machine Name.....	656
Listing the Probes Running on a Host .....	657
Authentication and Authorization for .NET Profilers.....	658
Configuring Consumer IDs.....	660
Configuring SOAP Fault Data.....	665
Collecting Additional Probe Metrics or Modifying Probe Metrics ...	666

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

<b>Chapter 16: Configuring Diagnostics Servers and Agents for HTTP Proxy .....</b>	<b>671</b>
Enabling HTTP Proxy Communications for the Diagnostics Servers.....	672
Enabling HTTP Proxy Communications for the Java Agent.....	673
Enabling HTTP Proxy Communications for a .NET Agent .....	674
<b>Chapter 17: Configuring Diagnostics to Work in a Firewall Environment .....</b>	<b>675</b>
Overview of Configuring Diagnostics for a Firewall.....	676
Collating Offline Analysis Files over a Firewall .....	679
Installing and Configuring the MI Listener .....	680
Configuring the Diagnostics mediator server to Work with a Firewall .....	681
Configuring LoadRunner and Performance Center to Work with Diagnostics Firewalls .....	687

**PART VII: CONFIGURING DIAGNOSTICS METRICS COLLECTORS**

<b>Chapter 18: .NET System Metrics Agent - Systems</b>	
<b>Metrics Capture</b> .....	<b>691</b>
About the .NET System Metrics Agent .....	691
System Metrics Captured by Default .....	692
Configuring .NET System Metrics Capture .....	693
Adding System Metrics Using the Windows Performance	
Monitor .....	696
Default Entries in the .NET Agent metrics.config File .....	698
Keywords in the metrics.config File .....	699
<b>Chapter 19: Java Agent Metrics Collectors</b> .....	<b>703</b>
About Metrics Capture .....	703
What Metrics are Being Collected by the Java Agent .....	705
Understanding Metric Collector Entries .....	706
About Collecting Additional Probe Metrics .....	708
Modifying Probe Metrics Already Being Captured .....	708
Stopping Capture of a Metric .....	708
Using Customized metrics.config Files for Multiple JVM	
Applications on a System .....	709
<b>Chapter 20: Java Agent - System Metrics Capture</b> .....	<b>711</b>
About System Metrics .....	711
System Metrics Captured by Default .....	712
Configuring the System Metrics Collector .....	713
Capturing Additional Custom System Metrics .....	715
Enabling z/OS System Metrics Capture .....	721
<b>Chapter 21: Java Agent - JMX Metrics Capture</b> .....	<b>723</b>
About JMX Metrics .....	723
About Configuring JMX Metric Collectors .....	724
Additional Custom JMX Metrics .....	725
Getting a List of Available JMX or WebSphere PMI Metrics .....	725
Creating New JMX or WebSphere PMI Metrics Entries .....	728

**PART VIII: SETTING UP INTEGRATION WITH OTHER  
HP SOFTWARE PRODUCTS**

**Chapter 22: Setting Up the Integration Between Business Service Management and Diagnostics.....737**

About Setting Up the Integration Between Business Service Management and Diagnostics .....739

Registering the Diagnostics Server in Business Service Management .....740

Removing the Diagnostics Registration .....747

Understanding the Diagnostics Admin Page .....747

Assigning Permissions for Diagnostics Users in Business Service Management .....748

Password for Data Collectors to Access RTSM .....750

Accessing the Diagnostics Pages in Windows 2003 .....751

Accessing the Diagnostics Application from Business Service Management .....751

Data Samples Sent to Business Service Management.....752

Diagnostics Populates CIs and Models in Business Service Management .....753

Synchronize CIs Between Diagnostics and Business Service Management .....753

Diagnostics Provides KPI/HI Coloring to Business Service Management .....754

Enabling Diagnostics Integration with BSM’s Service Health Analyzer .....755

Integration with BSM’s Performance Graphing.....756

Diagnostics and OM Server Co-existence .....756

Configuration of Separate BSM Servers for DPS and Gateway .....761

Additional Information on Integration .....763

**Chapter 23: Installing the LoadRunner Diagnostics Add-in.....765**

Before Installing the LoadRunner Diagnostics Add-in .....766

Installing the LoadRunner Diagnostics Add-in.....766

**Chapter 24: Setting Up HP LoadRunner and HP Diagnostics Integration.....769**

How You Can Use HP Diagnostics with LoadRunner.....770

About Setting Up LoadRunner to Integrate with HP Diagnostics ....773

Configuring LoadRunner Scenarios to use HP Diagnostics .....774

Selecting Probe Metrics to Include in the Offline Analysis File.....774

Improving Transfer of Large Offline Analysis Files.....777

Out of Memory Issue in LoadRunner Controller’s Diagnostics UI...777

<b>Chapter 25: Setting Up Performance Center to Use Diagnostics ....</b>	<b>779</b>
How You Can Use HP Diagnostics with Performance Center .....	780
About Setting Up Performance Center to Use Diagnostics.....	782
Configuring Performance Center Load Tests to Use Diagnostics .....	783
Managing Performance Center Offline Files.....	784

## **PART IX: APPENDIXES**

<b>Appendix A: Diagnostics Administration UI .....</b>	<b>787</b>
Accessing the Diagnostics Administration UI.....	787
Using the Diagnostics Administration UI.....	790
<b>Appendix B: User Authentication and Authorization.....</b>	<b>797</b>
About User Authentication and Authorization .....	798
Understanding User Privileges .....	799
Understanding Roles .....	800
Accessing Diagnostics Using Default User Names .....	801
Understanding the Diagnostics Server Permissions Page .....	802
Creating, Editing and Deleting Users.....	810
Assigning Privileges Across the Diagnostics Deployment .....	812
Assigning Privileges for Probe Groups .....	813
Authentication and Authorization for Users of Integrated HP Software Products.....	816
Tracking User Administration Activity .....	818
List of Active Users .....	819
Configuring Diagnostics to use JAAS .....	820
<b>Appendix C: Enabling HTTPS Between Components.....</b>	<b>839</b>
About Configuring HTTPS Communications .....	840
Filtering Encryption Cipher Suites.....	840
HTTPS Checklist per Diagnostics Component.....	841
Enabling Incoming HTTPS Communication for Diagnostics Components.....	843
Generate Client Certificate.....	843
Enabling Outgoing HTTPS Communication from Diagnostics Components.....	853
Enabling HTTPS Communications for the Business Service Management Server .....	860
<b>Appendix D: Using System Views for Administrators .....</b>	<b>863</b>
System Views for Diagnostics' Administrators.....	863
System Health View Description.....	865
System Capacity View Description .....	866

<b>Appendix E: Diagnostics Data Management</b> .....	<b>867</b>
About Diagnostics Data.....	868
Custom View Data.....	868
Performance History Data.....	870
Data Retention .....	876
Disk Space Issues on the Server.....	882
Pre-Installation Data Management Considerations.....	882
Backing Up Diagnostics Data .....	883
Handling Diagnostics Data when Upgrading Diagnostics .....	888
<b>Appendix F: Diagnostics Technical Diagrams</b> .....	<b>889</b>
Communications with Business Service Management.....	890
Communications with LoadRunner and Performance Center.....	891
.NET Probe Aggregator Data Flow .....	892
<b>Appendix G: Upgrade and Patch Install Instructions</b> .....	<b>893</b>
Before You Begin .....	894
Diagnostics Compatibility with Earlier Diagnostics Versions .....	894
Upgrade or Patch Install Instructions for Diagnostics Components .....	894
Diagnostics Compatibility with Other HP Software Products.....	907
<b>Appendix H: Troubleshooting HP Diagnostics</b> .....	<b>909</b>
Component Installation Interrupted on a Solaris Machine .....	910
Diagnostics Installers Do Not Work on Some 64-bit Linux Systems.....	910
Error During Linux Install - Missing libstdc++.so.5 Shared Library ..	911
Java Agent Fails to Operate Properly.....	911
Error During WAS Startup with Diagnostics Profiler for Java.....	912
Missing Server-Side Transactions .....	913
Event Capture Buffer Full Warning.....	913
WebSphere Application Server Startup Issue .....	914
Java Agent Support Collector .....	915
Event Based Health Indicator Status Troubleshooting Flow.....	915
OM Agent Troubleshooting .....	919
Troubleshooting Registration of OMi Between the BSM Gateway Server and Data Processing Server .....	922
<b>Appendix I: General Reference Information</b> .....	<b>925</b>
Using UNIX Commands .....	925
Using Regular Expressions.....	926
Multi-Lingual User Interface Support .....	934



<b>Appendix J: Data Exporting .....</b>	<b>937</b>
Task 1: Prepare the target database .....	938
Task 2: Determine which metrics you want to export .....	939
Task 3: Determine the frequency and the recovery period .....	942
Task 4: Modify the data export configuration file .....	943
Task 5: Monitor the data export operation.....	947
Task 6: Verify the results .....	949
Task 7: Select the data from the target database .....	950
Sample Queries .....	950
<b>Index.....</b>	<b>953</b>

Table of Contents

---

# 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    Preparing to Install**

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 Servers and the HP Diagnostics Collectors.

### **Part III    Installation and Setup of the Java, .NET and Python Agents**

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 HP Diagnostics applies to the classes and methods of monitored applications to enable it to gather 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 the following and provides technical data flow diagrams:

- ▶ 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
- ▶ Managing data as well as doing backup and recovery
- ▶ Upgrading Diagnostics and installing patch updates
- ▶ Using the Data Export feature
- ▶ Troubleshooting and finding 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 the Diagnostics UI or from the help menu in the integrated HP Software product. You access the PDF version of the User's Guide from the Diagnostics online help Home page, 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 the PDF of this guide from the Diagnostics online help Home page, 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**).
- ▶ **Diagnostics Frequently Asked Questions.** Gives answers to frequently asked questions. You can access the pdf from the Diagnostics online help.
- ▶ **Diagnostics Data Model and Query API.** Describes the Diagnostics data model and the query API you can use to access the data. You can access the pdf from the Diagnostics online help.
- ▶ **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 an Upgrade and Patch Install Instructions document with details for installing an upgrade or 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 the PDF of this guide on the agent system in the \docs directory, or from the Java Diagnostics Profiler UI online 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 the PDF of this guide on the agent system in the \docs directory, or from the .NET Diagnostics Profiler UI online 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

**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](http://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](http://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](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

---

## Preparing to Install

This section includes:

- ▶ Preparing to Install HP Diagnostics



# 1

---

## Preparing to Install HP Diagnostics

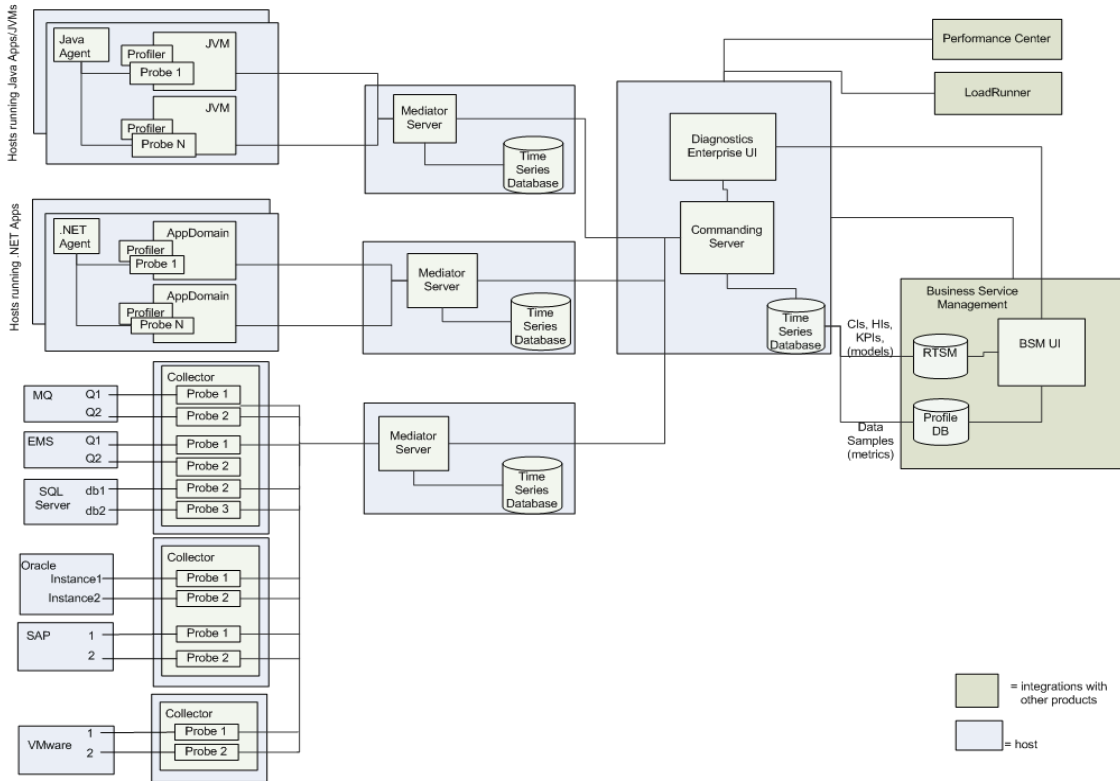
Before you install HP Diagnostics, read the following information and instructions that will help you plan and prepare for the installation and configuration of the Diagnostics components.

**This chapter includes:**

- HP Diagnostics Components and Data Flow on page 28
- Supported Application Servers and Environments on page 30
- System Requirements for the Diagnostics Components on page 31
- Information Required for Installation on page 40
- Pre-installation Considerations on page 47
- Recommended Order of Installation on page 48
- Licensing HP Diagnostics on page 50
- Upgrading from Earlier Versions of Diagnostics on page 50

## 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.** Capture events from your J2EE and .NET applications such as method invocations, the beginning and end of business transactions and server requests and then aggregates performance metrics to be sent to a Diagnostics Server. There is also now a Python Agent for monitoring Python applications.

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.

- ▶ **Diagnostics Collectors.** Responsible for collecting data from external environments including Oracle Databases, SQL Server systems, IBM WebSphere MQ messaging systems, TIBCO Enterprise Message Service, Software AG webMethods Broker, VMware vCenter or VMware ESX servers and SAP NetWeaver - ABAP systems. 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 user interface.
- ▶ **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 commander 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 commander server, with all the rest running as (distributed) mediators.

In a typical deployment there is a Diagnostics commander 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 commander server, which displays the processed metrics in the UI.

The Diagnostics commander 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.

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

The commander server is also responsible for displaying the performance information for the monitored applications in the Diagnostics user interfaces.

**User Interfaces.** The main Diagnostics user interface (Diagnostics Enterprise UI) displays performance data in charts and graphs for use in monitoring performance, isolating problems and analyzing causes to solve complex performance problems.

If you are using Diagnostics with other HP Software products you can also access the Diagnostics Enterprise UI from the user interface of the other products. For example you can access the Diagnostics Enterprise UI from HP Business Service Management. And in pre-production, during a load test, you can access the Diagnostics Enterprise UI from HP LoadRunner or HP Performance Center.

Diagnostics also provides Java and .NET profilers displayed in separate user interfaces (Diagnostics Profiler UIs) available directly on the agent systems or as a drill down from the main Diagnostics user interface.

- ▶ **Integrations.** Diagnostics has integrations with the following other HP Software products. See Part VIII, “Setting Up Integration with Other HP Software Products” for more information. Also see the Online Help or User’s Guide section on "Integrations with Other HP Software Products".
  - ▶ HP Business Service Management
  - ▶ HP LoadRunner
  - ▶ HP Performance Center
  - ▶ HP Sitescope
  - ▶ HP Continuous Delivery Automation (CDA)

## 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.

- Python applications.
- SAP NetWeaver–ABAP systems.
- Oracle databases.
- SQL Server databases.
- IBM WebSphere MQ systems.
- TIBCO Enterprise Message Service (EMS) systems.
- VMware vCenter or VMware ESX servers.
- Software AG webMethods Broker

For the most recent information on supported environments, see the Diagnostics Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](http://support.openview.hp.com/sc/support_matrices.jsp).

## 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 32
- “Requirements for the Diagnostics Enterprise UI” on page 32
- “Requirements for the Diagnostics Server Host” on page 32
- “Scalability Information” on page 34
- “Requirements for the Diagnostics Java Agent Host” on page 36
- “Requirements for the Host of the Diagnostics Java Profiler User Interface” on page 37

- “Requirements for the Diagnostics .NET Agent Host” on page 38
- “Requirements for the Host of the Diagnostics .NET Profiler User Interface” on page 39
- “Requirements for the Diagnostics Collector Host” on page 39
- “Requirements for the Python Agent” on page 39

## **Supported Environments for the Diagnostics Components**

For the most recent information on supported environments for the Diagnostics components, see the Diagnostics Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](http://support.openview.hp.com/sc/support_matrices.jsp).

Diagnostics Servers and Diagnostics Collectors use the Java 1.6 JVM.

---

**Important:** For Diagnostics Linux installers (both 32 and 64 bit for servers, agents and collectors) the 64-bit Linux system must have the patch libXtst-1.0.1-3.1 installed in order to run the installers in graphical mode.

---

## **Requirements for the Diagnostics Enterprise UI**

The Diagnostics Enterprise UI is presented in a web browser using a Java applet that requires JRE 1.6, minimally to be installed on the client systems that access the UI. Supported browsers include Microsoft Internet Explorer 7, 8, 9 and Mozilla Firefox 3.5, 3.6, 5, 6. For the most recent information on supported browsers, see the Diagnostics Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](http://support.openview.hp.com/sc/support_matrices.jsp).

## **Requirements for the Diagnostics Server Host**

The system configuration requirements for the host of a Diagnostics Server depend upon the number of probes and mediator servers that are reporting to it. When a server is designated as the Diagnostics commander server, the probe data is generally stored on each mediator server that reports to it.



If installing the Diagnostics Server on a SAN storage device the SAN must have adequate read and write speed comparable with a mid to high end drive (see “Scalability Information” on page 34).

---

**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 (typically a mediator 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
All	Heap Size	512 M	750 M	1280 M

Platform	Item	Up to 50 Java Probes	Up to 100 Java Probes	Up to 200 Java Probes
All	Disk	4 GB per probe		
<b>Notes regarding the test environment</b>				
<ul style="list-style-type: none"> <li>▶ Call profile (depth of method calls) for each Server Request: 5</li> <li>▶ Number of unique Server Requests per probe: 23</li> </ul>				

The following table lists the desired system requirements for the host of a Diagnostics server (typically a mediator 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
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)

<b>Memory:</b>	8 GB
<b>Disk I/O:</b>	Smart Array P400i, 2SCSI drives in RAID 0 (136 GB) [130 MB/S, sequential read and write]
<b>Java Heap:</b>	5.9 GB (-Xmx6096m); 64-bit JVM
<b>Disk Space:</b>	2-4 GB per probe (overall disk space can be adjusted by changing retention intervals)
<b>Network:</b>	1 Gbps

---

**Note:** A 64-bit OS and JVM is recommended for use with Diagnostics for optimal performance.

---

Scalability numbers for the previously referenced hardware.

<b>Up to 100 Java probes:</b>	100 Server Requests per probe, 78 methods per Call Profile pulled every 45s (default)
<b>Up to 400 Java probes:</b>	25 Server Requests per probe, 78 methods per Call Profile pulled every 45s (default)
<b>Up to 150 Java probes:</b>	150 Server Requests per probe, 25 methods per Call Profile pulled every 240s
<b>Up to 230 Java probes:</b>	100 Server Requests per probe, 25 methods per Call Profile pulled every 240s
<b>Up to 40 Java probes:</b>	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 470.

---

**Notes:**

- ▶ For environments with many probes, better performance can be achieved by having two or more instances of the Diagnostics server 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 commander server, see “Pre-Installation Data Management Considerations” on page 882.
  - ▶ 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 496.
- 

### **Requirements for the Diagnostics Java Agent 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:

<b>Platform:</b>	All Platforms
<b>Memory:</b>	50MB Additional RAM
<b>Free Hard Disk Space:</b>	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.

---

**Adjusting heap size.** For information on setting the max heap for the Java Agent, see “Adjusting the Heap Size for the Java Agent in the Application Server” on page 237.

**Adjusting permgen size.** Typically any increase in permgen size as a result of adding the Diagnostics agent is small. However, in some cases the application, without the agent, uses almost all its permgen limit. In such cases you will need to adjust it. For example you could increase it by the existing limit \* 1.05 +5MB. To adjust permgen for Hotspot JVM, use `-XX:MaxPermSize` option, for example: `-XX:MaxPermSize=240m`.

## **Requirements for the Host of the Diagnostics Java Profiler User Interface**

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 client system that accesses the UI. This machine must be able to access the Diagnostics Profiler URL: [http://<probe\\_host>:<probeport>/profiler](http://<probe_host>:<probeport>/profiler). By default, the probes are assigned to the first available port beginning at 35000. Supported browsers include Microsoft Internet Explorer 7, 8, 9 and Mozilla Firefox 3.5, 3.6, 5, 6. For the most recent information on supported browsers, see the Diagnostics Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](http://support.openview.hp.com/sc/support_matrices.jsp).

## Requirements for the Diagnostics .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:

<b>Platform</b>	All Supported Platforms
<b>Memory</b>	60 MB Additional RAM
<b>Free Hard Disk Space</b>	200 MB Additional Space
<b>.NET Framework</b>	2.0 or later

---

**Important:** If you must support .NET Framework 1.1, use an earlier version of the .NET Agent (8.x) which will continue to be supported and updated via patches.

---

**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.

## **Requirements for the Host of the Diagnostics .NET Profiler User Interface**

The user interface for the .NET Diagnostics Profiler is presented using DHTML/XML/XSLT/JScript technology that requires Internet Explorer 7 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, SQL Server, IBM WebSphere MQ, TIBCO EMS, Software AG webMethods Broker or VMware application from which it is collecting data.

350MB free disk space is required for the Collector install. More space might be required during runtime due to the creation of logfiles or for large environments.

For the most recent information on supported environments for the Diagnostics components, see the Diagnostics Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](http://support.openview.hp.com/sc/support_matrices.jsp).

## **Requirements for the Python Agent**

See “Installing and Setting Up Python Agents” on page 295.

## 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	Description
Will you have a commander server and one or more mediators?	During planning for the Diagnostics deployment this will be determined based on the size and complexity of the monitored environment.



Information Required	Description
<b>For a Diagnostics commander server, the location of the HP Diagnostics license that was generated for the machine that will host the server</b>	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.
<b>For a Diagnostics mediator server, the URL for the Diagnostics commander server</b>	After the Diagnostics commander server has been installed, the URL is available to open Diagnostics views is available.
<b>Will Diagnostics be used in a SaaS environment?</b>	If you are deploying Diagnostics in a SaaS (HP hosted) environment different installer options will be presented.
<b>Will Diagnostics be integrated in a Business Service Management environment?</b>	If you are deploying Diagnostics into a Business Service Management environment you'll need to select this option in the installer.

## Java Agent

### ► HP Software Product and Diagnostics Server Information

Information Required	Where to find it	Value
Mode for installing the agent	Choose according to product license.	<ul style="list-style-type: none"> <li>► Profiler only (no connection to server)</li> <li>► Used only with LoadRunner/ Performance Center (AD license)</li> <li>► Enterprise mode (AM license) for use with one of the following or both:                             <ul style="list-style-type: none"> <li>► Diagnostics</li> <li>► TransactionVision</li> </ul> </li> </ul>
Diagnostics Server Name	<p>Fully qualified host name or IP address of the host of the Diagnostics Server.</p> <p>This is not required for using the Java 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 commander server.</p> <p>In a distributed environment with a commander server and mediator servers, this is the Diagnostics mediator server that is to receive events from the agent.</p>
Diagnostics Server Port	<p>Use the default 2006 or the port you configure for accessing Diagnostics.</p> <p>This is not required for using the Java Diagnostics Profiler in a standalone mode.</p>	<b>Default value:</b> 2006

► Instrumented Application Server and Agent Information

Information Required	Where to find it	Value
<b>Java agent name</b>	<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>	For example: WebLogic_MedRec_java
<b>Java agent group</b>	<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>	<b>Default value:</b> Default
<b>Type of application server that will be instrumented for monitoring</b>	The host system administrator.	

Information Required	Where to find it	Value
<b>Application Server configuration properties</b>	The host system administrator.  The details vary according to the application server you are monitoring.	
<b>Location of the JRE executable</b>	The host system administrator.  Depends on the type of application server you plan to monitor. See “Preparing Application Servers for Monitoring with the Java Agent” on page 161.	

## .NET Agent

### ► Diagnostics Server Information

Information Required	Where to find it	Value
Mode for installing the agent	Choose according to product license.	<ul style="list-style-type: none"> <li>► Profiler only (no connection to server)</li> <li>► Used only with LoadRunner/ Performance Center (AD license)</li> <li>► Enterprise mode (AM license) for use with one of the following or both:               <ul style="list-style-type: none"> <li>► Diagnostics</li> <li>► TransactionVision</li> </ul> </li> </ul>
Diagnostics Server Name	Fully qualified host name or IP address of the host of the Diagnostics Server.  This is not required for using the .NET 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 commander server.  In a distributed environment with a commander server and mediator servers, this is the Diagnostics mediator server that is to receive the events from the agent.
Diagnostics Server Port	Use the default 2006 or the port you configure for accessing Diagnostics.  This is not required for using the .NET Diagnostics Profiler in a standalone mode.	<b>Default value:</b> 2612

► Agent and Port Information

Information Required	Where to find it	Value
<p><b>agent group</b></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><b>Default value:</b> Default</p>
<p><b>Web Port Min</b></p>	<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><b>Default value:</b> 35000</p>
<p><b>Web Port Max</b></p>	<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><b>Default value:</b> 35100</p>

## Pre-installation Considerations

---

**Note:** Before you install any of the Diagnostics components on a Windows machine, make sure that the **Services** window, accessible from Administrative Tools, is not open.

---

### Diagnostics Server

- The performance metrics for HP Diagnostics cannot be displayed until the Diagnostics commander 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 commander server that is properly licensed. For more information on obtaining a license and other licensing issues, see Chapter 3, “Licensing HP Diagnostics.”
- Diagnostics does not support localization of agent names.

### **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 commander server that is properly licensed. For more information on obtaining a license and other licensing issues, see Chapter 3, “Licensing HP Diagnostics.”
- ▶ Diagnostics does not support localization of agent names.

### **Diagnostics Python Agent**

- ▶ The Python Agent must be installed on the same system as the Python application under test.

### **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 31.

**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 a Python environment, see Chapter 9, “Installing and Setting Up Python Agents”.

For Oracle Database, SAP NetWeaver-ABAP, SQL Server Database, VMware vCenter or VMware ESX servers, WebSphere MQ, TIBCO EMS and Software AG webMethods Broker 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, “Preparing Application Servers for Monitoring with the Java Agent.”

For more information, see Chapter 7, “Preparing Application Servers for Client Monitoring with 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, each of these products requires configuration in order to use HP Diagnostics.**

For Business Service Management, see Chapter 22, “Setting Up the Integration Between Business Service Management and Diagnostics.”

For LoadRunner integration, install the LoadRunner Diagnostics Add-in (see Chapter 23, “Installing the LoadRunner Diagnostics Add-in”) and set up LoadRunner to use Diagnostics (see Chapter 24, “Setting Up HP LoadRunner and HP Diagnostics Integration”).

For Performance Center, see Chapter 25, “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 commander server. 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 Diagnostics Servers on page 54
- ▶ Verifying the Diagnostics Server Installation on page 67
- ▶ Silent Installation of the Diagnostics Server on page 68
- ▶ Starting and Stopping the Diagnostics Server on page 70
- ▶ Licensing Your Diagnostics Software on page 72
- ▶ More Information on Configuring Diagnostics Servers on page 72
- ▶ Determining the Version of the Diagnostics Server on page 72
- ▶ Uninstalling the Diagnostics Server on page 73
- ▶ Manual Installation of OM Agent and IAPA Components on page 74
- ▶ Manual Uninstall of OM Agent and IAPA Components on page 76

## Installing Diagnostics Servers

This chapter provides detailed instructions for installing the Diagnostics Server and applies to:

- ▶ A Windows environment
- ▶ Most UNIX environments using a graphical installer. You can also install on UNIX using the console mode command line interface.

For the most recent information on supported platforms, see the Diagnostics Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](http://support.openview.hp.com/sc/support_matrices.jsp). Contact HP Support for installation assistance for additional platforms not listed in the support matrix.

---

**Note:** If an earlier version of the Diagnostics Server is installed on your machine, see Appendix G, “Upgrade and Patch Install Instructions.”

---

**Root Access Requirement.** If the Diagnostics commander server will be integrated with Business Service Management 9.00 or later, root access is required during the Diagnostics Server installation. Root access is required for the installation of the OM Agent and IAPA component.

If you need to install the Diagnostics Server without root access you can chose to not install the OM Agent and the IAPA component and install them later manually. When you see the dialog box: OM Agent and IAPA component installations leave the box unchecked and install later (see “Manual Installation of OM Agent and IAPA Components” on page 74).

Root privileges are also required on Solaris and Linux systems to setup the server to auto-start at boot.

**HP Software-as-a-Service (SaaS).** HP Diagnostics can be deployed into an HP Software-as-a-Service (SaaS) environment. In a SaaS deployment the Diagnostics Java Agents are installed in your company's IT environment and the Diagnostics Commander Server and Mediator Servers are installed by HP on a SaaS system on-premises at HP. So for customers using Diagnostics in a SaaS environment you would typically not install any Diagnostics servers on your company's systems and can ignore this chapter. In a SaaS deployment, customers would just be installing Java Agents that connect to the Servers set up by HP SaaS administrators on HP premises. Contact your SaaS administrator for more information.

This section includes:

- “Launching the Diagnostics Server Installer” on page 55
- “Running the Installation” on page 58

## 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 68.

---

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

---

### To access the Windows installer:

- 1** From the Diagnostics DVD (Autorun.exe) the installation menu page is displayed. From the menu, select **Diagnostics Server 32-bit** to install the 32-bit Windows version of the Diagnostics Server. And select **Diagnostics Server 64-bit** to install the 64-bit version of the Diagnostics Server.
- 2** Or you can run the appropriate installer directly by double-clicking 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) in the **Diagnostics\_Servers** directory.

Continue with “Running the Installation” on page 58.

**To access the UNIX installer:**

- 1** From the Diagnostics installation location access the **Diagnostics\_Servers** directory. Copy the appropriate 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_linux.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_linux.bin -console
```

Continue with “Running the Installation” on page 58.

**To download the installer from the HP Software Download Center:**

- 1** Go to the HP Software web site’s Software Download Center.
- 2** Locate the **Diagnostics** information and choose the appropriate link for downloading the Diagnostics Server software.
- 3** Follow the download instructions on the web site.

Continue with “Running the Installation” on page 58.



**To download 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 download the appropriate Diagnostics Server installer.

---

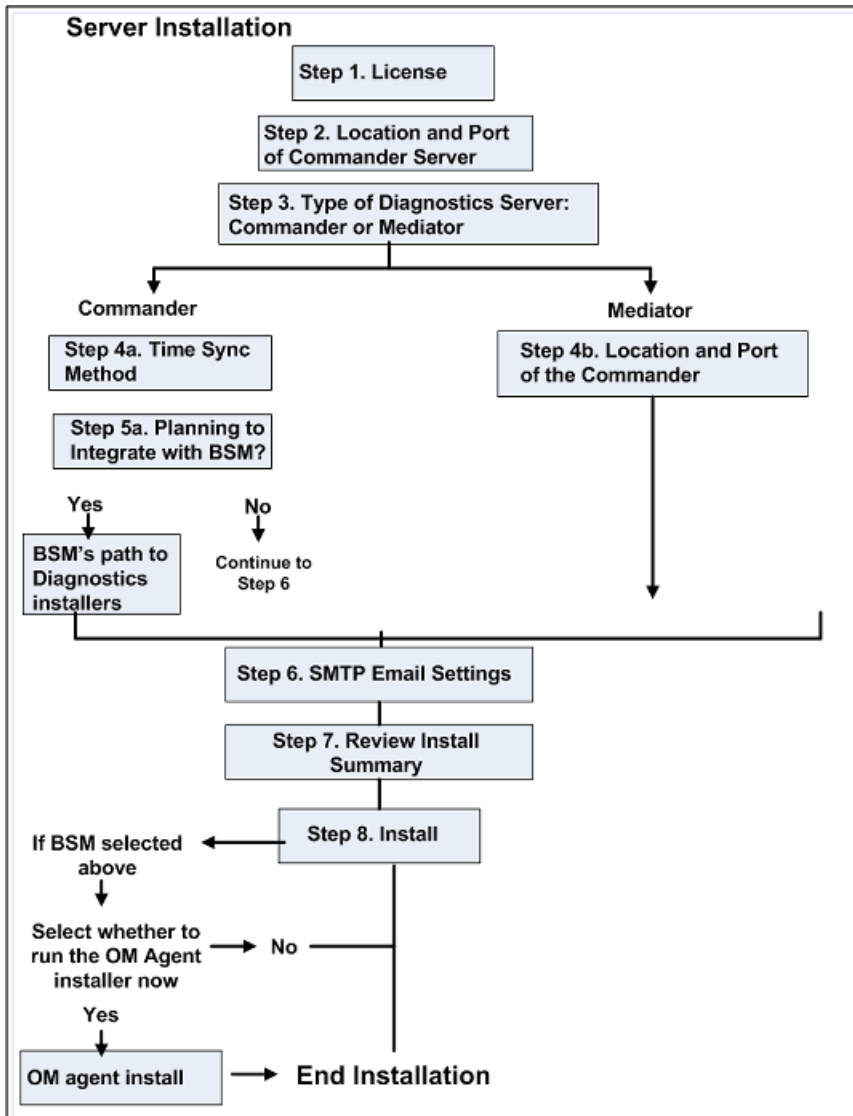
**Note:** The Java Agent installers are available in Business Service Management only if they are placed into a directory that Business Service Management can access. You can enable this during the installation of the Diagnostic Server, or you can copy the server installers manually from the installation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

---

Continue with “Running the Installation” on page 58

## Running the Installation

An overview of the Diagnostics Server installation steps is shown in the diagram below; refer to the rest of this section for details on each step.



Note that additional steps are required by the HP SaaS administrator for a SaaS deployment of servers on HP premises (HP internal documentation).

**Important:** If there is a pre-existing installation of the Diagnostics Server on the host machine, you must follow the instructions for upgrading the server system instead of these install instructions see “Upgrade and Patch Install Instructions” on page 893.

---

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

---

**Note:** On some 64-bit Linux systems, if the installer program terminates without showing the license agreement, you need to install the 32-bit glibc library and then try to run the installer again (see “Diagnostics Installers Do Not Work on Some 64-bit Linux Systems” on page 910 for details).

---

**To begin the installation and select the installation location and mode:**

**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 commander and mediator roles. When there is more than one Diagnostics Server in a deployment, one is configured in Commander mode and all the rest in Mediator mode reporting to the Commander Server.

- ▶ 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**.

Ignore the **This Server is to be used in an HP Software-as-a-Service (SaaS) environment** checkbox as this is to be used by an HP SaaS administrator installing a Diagnostics Server (either Commander or Mediator) on HP premises.

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

- ▶ To install the Diagnostics commander server, continue with “Installing the Diagnostics Server in Commander Mode” on page 61.
- ▶ To install a Diagnostics mediator server, continue with “Installing the Diagnostics Server in Mediator Mode” on page 65.

## Installing the Diagnostics Server in Commander Mode

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

### 4 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.

### 5 Select optional configurations for the Diagnostics Server.

**This Server is to be used with HP Business Service Management (BSM).**

Check this box if the Diagnostics commander server will be integrated 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. IAPA is the Integration Adapter Policy Activation component of the OMi agent that Diagnostics uses to communicate with Business Service Management.

You will be prompted towards the end of this Diagnostics Commander Server installation to confirm if you want to install these components.

See “Setting Up the Integration Between Business Service Management and Diagnostics” on page 737 for additional post install configuration required to integrate with BSM. Also, if you need to set up reporting to an OM Server as well as BSM Servers see “Diagnostics and OM Server Co-existence” on page 756 for instructions.

Select the option that applies to this Diagnostics Server, and then select **Next** to continue.

If you selected the HP Business Service Management option then an additional dialog is displayed where you **Provide the path to the directory on the HP Diagnostics Installation DVD where the Diagnostics installers are located.**

---

**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.

---

## 6 Indicate the SMTP setting for email alerts (optional).

These SMTP settings are *optional* during the installation. You configure SMTP settings in order to have email alerts sent when there are problems with the Diagnostics servers. If you want to configure this later (or modify these settings later) you can skip this dialog and configure these setting using the Diagnostics Server's alert properties page (see the *HP Diagnostics User's Guide* section on Alerts for more information).

- **SMTP Server.** Host name or IP address of the SMTP server.
- **SMTP Port.** Port number for the SMTP server.
- **From Email Address.** The email address to send the email messages from.
- **Admin Alert Email Addresses.** If you want the Diagnostics administrator to receive email alerts when there are problems with this Diagnostics server then specify a comma-separated list of email addresses for the administrator. Alerts can be issued for problems such as probes generating large number of server requests to the server, disk space issues on the server or from the Commander Server - license checking alerts.

The thresholds that determine these types of alerts for the Diagnostics administrator are factory configured in the server's **server.properties** file. For more details see the comments in this file for the various **watchdog** properties. Also see "Disk Space Issues on the Server" on page 882.

## 7 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**.

## 8 Installation begins.

The server installation is started. When the installation is complete you will see the post-installation summary information or, if you selected integration with Business Service Management in step 5a above, an additional dialog is displayed.

Check the post-installation summary and select **Finish** to exit the installation or continue on to the next step if integrating with BSM.

## 9 OM Agent and IAPA component installations checkbox.

If you selected that the Diagnostics commander server was to be used with Business Service Management you will see an additional dialog box for OM Agent and IAPA component installations. Check the box to install these components. Note you can leave the box unchecked and install these components later manually. See “Manual Installation of OM Agent and IAPA Components” on page 74 for details.

When integrating Diagnostics with Business Service Management 9.00 or later, 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 you are integrating with an earlier version of Business Availability Center you do not need to install these components.

Errors are reported in the `<Diagnostics_install_dir>/server/log.txt` file. See “OM Agent Troubleshooting” on page 919 if you have any problems with the installation.

**Root access** is required for the installation of the OM Agent and IAPA component. If you need to install the Diagnostics Server without root access you can chose to not install these two components and install them later manually.

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

On Windows systems these components take a while to install.

You can leave the box unchecked to skip the installation of the OM Agent and IAPA component and them install later.



The installer bits are always laid down whether you check this box or not - so you can install the components later when needed from `<Diagnostics_install_dir>/server/setup/ovo-agent` and `/ovoiaipa`. See “Manual Installation of OM Agent and IAPA Components” on page 74 for details.

Additional configuration steps are required in Business Service Management after installing these components. See “Registering the Diagnostics Server in Business Service Management” on page 740 for more information.

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 70.

---

## **Installing the Diagnostics Server in Mediator Mode**

If you are installing the Diagnostics mediator server, continue the installation as follows.

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

### **4 Provide the location of the Diagnostics Server in Commander mode.**

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

- Enter the host name or IP address for the Diagnostics commander server.
- Enter the port for the Diagnostics commander server.

The default port for the Diagnostics commander 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 476.

- 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.

#### **5 Indicate the SMTP setting for email alerts (optional).**

These SMTP settings are *optional* during the installation. You configure SMTP settings in order to have email alerts sent when there are problems with the Diagnostics servers. If you want to configure this later (or modify these settings later) you can skip this dialog and configure these setting using the Diagnostics Server’s alert properties page (see the *HP Diagnostics User’s Guide* section on Alerts for more information).

- **SMTP Server.** Host name or IP address of the SMTP server.
- **SMTP Port.** Port number for the SMTP server.
- **From Email Address.** The email address to send the email messages from.
- **Admin Alert Email Addresses.** If you want the Diagnostics administrator to receive email alerts when there are problems with this Diagnostics server then specify a comma-separated list of email addresses for the administrator. Alerts can be issued for problems such as probes generating large number of server requests to the server, disk space issues on the server or from the Commander Server - license checking alerts.

The thresholds that determine these types of alerts for the Diagnostics administrator are factory configured in the server’s **server.properties** file. For more details see the comments in this file for the various **watchdog** properties. Also see “Disk Space Issues on the Server” on page 882.

## 6 Review the pre-installation summary information.

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**.

## 7 Installation begins.

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 70.

---

## Verifying the Diagnostics Server Installation

To verify that a Diagnostics Server was installed correctly you can check the `<Diagnostics_server_install_dir>/log/server.log` file for errors and warnings.

You can also launch the Diagnostics Enterprise UI to verify that the server is running. Go to `http://<Diagnostics_commander_server>:2006/`. For now you can use the default user/password of **admin/admin** or the login you were given if a different one has been set up for you. But note that as Diagnostics is deployed within your enterprise the admin/admin default login should be changed (see Appendix B, “User Authentication and Authorization” when you are ready to configure users, roles, permissions and authentication).

Note that in order to see data from agents in the UI you will also need to install and configure Diagnostics agent and/or collector software to collect and report performance data to the server for display in the UI.

You can also check the System Health view to find information about the Diagnostics servers and the machines that host them.

**To access the System Views:**

- 1** Open the Diagnostics UI as the Mercury System customer from [http://<Diagnostics\\_Commanding\\_Server\\_Name>:2006/query/](http://<Diagnostics_Commanding_Server_Name>:2006/query/).
- 2** In the query page locate the Mercury System customer in the list and select the link to Open Diagnostics.
- 3** Log in to Diagnostics and on the Applications window select Entire Enterprise and select any link to open the Diagnostics Views.
- 4** In the Views pane you'll see the System Views view group. Open the view group and select either the System Health view or System Capacity view.

## **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.

---

**Important:** With each new release of Diagnostics you should re-record the Diagnostics Server silent install response files prior to performing silent installation on multiple machines.

---

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. Note that for Windows installers the options must be preceded with **-a**. For example: `HPDiagServer_9.20_win32.exe -a -options-record myfile`.

```
<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. Note that for Windows installers the options must be preceded with **-a**. For example: `HPDiagServer_9.20_win32.exe -a -silent -options myfile`.

```
<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

Diagnostics servers are started automatically. But if you need to manually start or stop a Diagnostics server follow the instructions below.

### Instructions for a Windows Machine

To start the Diagnostics Server on a Windows machine:

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

To stop the Diagnostics Server on a Windows machine:

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

### Instructions for Solaris 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.

But note that, the `m_daemon_setup` script does not configure the server to restart automatically after a system boot. To support this the startup will need to be integrated with the boot sequence or manually executed after each system boot.

To start the Diagnostics Server on a Solaris 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 some Linux systems, if you encounter an error saying that the libstdc++.so.5 shared library is missing, you may need to install it. For example, on CentOS, enter the following command to install the library:

```
yum install compat-libstdc++-33
```

**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 Solaris 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 Solaris or Linux machine:**

Run **<diagnostics\_server\_install\_dir>/bin/server.sh**.

**To stop the Diagnostics Server on a Solaris or Linux machine:**

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

## Licensing Your Diagnostics Software

Your Diagnostics software comes with an instant-on license so you can start using it right away. But eventually you will need to install your permanent license key; which is done on the Diagnostics Commander Server. For instructions on requesting a license file and uploading it, see Chapter 3, “Licensing HP Diagnostics.”

## More Information on Configuring Diagnostics Servers

The Diagnostics Server is installed with a default configuration that enables it to begin working right away.

However you could encounter situations where changing the configuration enables better Diagnostics Server performance or allows it to work in unusual or complex situations. For information about advanced configuration of the Diagnostics Servers, see Chapter 12, “Advanced Diagnostics Server Configuration.”

See “Setting Up Integration with Other HP Software Products” on page 735 for additional post install configuration required to integrate with BSM. Also, if you need to set up reporting to an OM Server as well as BSM Servers see “Diagnostics and OM Server Co-existence” on page 756 for instructions.

## Determining the Version of the Diagnostics Server



When you request support, you must know the version of the Diagnostics Server. In the Diagnostics Enterprise UI the About dialog box shows you the version of the Diagnostics server. Access the About dialog box by selecting **About HP Diagnostics** from the Help menu in the Diagnostics Enterprise UI toolbar.



## 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 70.
- 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 commander 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 Availability Center you do not need to install these components).

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

See “OM Agent Troubleshooting” on page 919 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 commander 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>/server/bin/install_ovo_iapa.vbs /i HPOprIAPA-09.00.111-WinNT4.0-release.msi <log file>`

Where `<log file>` is a file where the results of the install are logged, path is optional.

Or from the command line in the win64 directory execute

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

Where `<log file>` is a file where the results of the install are logged, path is optional.

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 executes  
**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:

- 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” on page 740 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>\server\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>\server\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 commander server.

**This chapter includes:**

- ▶ About HP Diagnostics Licensing on page 80
- ▶ Types of Licenses on page 80
- ▶ Licensing the Diagnostics Server in Commander Mode on page 81
- ▶ View License Information on page 84
- ▶ Licensing the Other Diagnostics Components on page 88

## About HP Diagnostics Licensing

Diagnostics is licensed using a file that you upload to the Diagnostics commander server. You request this license file from your HP Software Customer Support representative.

When the Diagnostics agents and Diagnostics mediator server first connect with the Diagnostics commander server they are licensed based on the license installed on the Diagnostics commander server.

If you want the Diagnostics administrator to receive license checking alerts then when installing the Commander Server specify a comma-separated list of Admin Alert Email Addresses in the SMTP Settings installation dialog. Or the Admin address can be setup after installation using the Commander Server's Alert Properties page.

## Types of Licenses

At installation you are given an **Instant-On license** which is packaged with the product. With the Instant-On license you can install Diagnostics components, begin to monitor applications, and process the performance metrics. The Instant-On license is valid for a fixed period of time from the time of installation or first use of the product.

Within this time period you must obtain a **Permanent license** or request an Evaluation license to extend the evaluation period. Evaluation licenses are available for Diagnostics to provide license keys that are meant to extend a customer's evaluation of the product. The Evaluation license is valid for a fixed period of time.

If the Instant-On license (or the extended Evaluation license) expires before you obtain a permanent license, the Diagnostics Server will issue reminder messages.



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

---

Your permanent license will typically be for a specific capacity (see “License Information Based on Currently Connected Probes” on page 85). Once you install the license key, Diagnostics will count usage against this capacity.

For Diagnostics there are two types of LTUs (License to use):

- ▶ AM License - For use when using the product in an application management/enterprise mode, typically in a production environment. AM licensed agents can also be used with LoadRunner/Performance Center.
- ▶ AD License - For use when using the product in Diagnostics mode for LoadRunner/Performance Center runs in a pre-production load testing environment.

The Instant-On licenses you receive with Diagnostics have the following time and capacity limits: AM - 60 days and capacity of 50, AD - 14 days and capacity of 50.

You will see reminder messages when limits are exceeded. See “License Information Based on Currently Connected Probes” on page 85 for details on AD and AM licenses.

## **Licensing the Diagnostics Server in Commander Mode**

Obtain your Diagnostics license from your HP Software Customer Support representative. The License Management page described below contains useful information for 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.

You will receive a license certificate from HP verifying the terms of the license purchase. License Keys/Passwords are issued after you enter the Sales Order Number associated with their software product purchase, which is unique for every order. This number appears on the license redemption form, as well as on all paperwork associated with the shipment and packaging of the order.

Store the license file in a directory that can be accessed from the License Management page for the Diagnostics commander server. Then upload it to the Diagnostics Commander Server as described in the steps below.

---

**Important:** For customers with licenses for versions prior to Diagnostics 9.10 your old licenses will still work with 9.10 or later versions. However the following section describes how to use the new licensing process for new purchases of Diagnostics 9.10 or later.

---

**To license your Diagnostics deployment:**

- 1** Access the License Management page for the Diagnostics commander server by accessing the Diagnostics Enterprise UI ([http://<Diagnostics\\_Server>:2006](http://<Diagnostics_Server>:2006))
- 2** Enter the login and password. Either use the default or whatever has been created and assigned to you. Default login is **admin** and default password is **admin**.
- 3** Select **Configure Diagnostics**.
- 4** Select the **license** link. The License Management page opens providing the following:
  - ▶ Information about current licenses.
  - ▶ A utility to upload a license received from HP Software Support.
  - ▶ Information on operating system instance totals as well as application server/probe instances in your monitored environment. You can also find information on usage against Diagnostics AD and AM license capacity.

## License Management

## Default Client AD License Information

Attribute	Value
License:	HP Diagnostics AD Implicit Feature
Start Date:	Thursday, October 25, 2012
Expiration:	Tuesday, January 21, 2014
Days Remaining:	434
Capacity:	100

## Default Client AM License Information

Attribute	Value
License:	HP Diagnostics AM Implicit Feature
Start Date:	Thursday, October 25, 2012
Expiration:	Tuesday, January 21, 2014
Days Remaining:	434
Capacity:	100

## Autopass License Upload

**Note:** The uploaded file will be added to "DiagnosticsLicFile.txt".

License File:

## License information based on currently connected probes

Customer Name: Default Client

Attribute	Value
Total Operating System Instances:	39
Application Management/Enterprise Mode (AM License) OS instances:	39
Load Runner/Performance Center (AD License) OS instances:	0
Total Application Server Instances:	69
Application Management/Enterprise Mode (AM License) probe instances:	69
Load Runner/Performance Center (AD License) probe instances:	0
.NET processes:	11
Java probes:	58
Old .NET probes:	0
Unknown probes:	0
Collector instances:	5

[Details](#)

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

The **Server License Upload (Obsolete)** section is obsolete and will only appear when the type of license key Diagnostics previously used (.lic file) is installed or only the Instant-On license is installed on the server. This upload is provided for existing customers who already have a license from a Diagnostics version prior to 9.10 allowing you to upload your old license.

---

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

---

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

If successful (the keys in the license file are valid and are not expired), the licenses are added to **DiagnosticsLicFile.txt** by the upload process and stored in the **<Diagnostics\_Install\_Dir>/etc** directory of the Diagnostics Commander Server. With AutoPass licensing you can upload incremental licenses which are added to the license file (you can't do this when mixed with the old licenses).

## 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 license capacity.

## License Information Based on Currently Connected Probes

In the License information section you will see counts based on currently connected probes. Counts are shown for operating system instances (see example below). 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.

License information based on currently connected probes	
Customer Name: Default Client	
Attribute	Value
Total Operating System Instances:	39
Application Management/Enterprise Mode (AM License) OS instances:	39
Load Runner/Performance Center (AD License) OS instances:	0
Total Application Server Instances:	69
Application Management/Enterprise Mode (AM License) probe instances:	69
Load Runner/Performance Center (AD License) probe instances:	0
.NET processes:	11
Java probes:	58
Old .NET probes:	0
Unknown probes:	0
Collector instances:	5
<input type="button" value="Refresh"/>	
<a href="#">Details</a>	

The following counts are based on the number of operating system instances running an agent:

- ▶ **Total Operating System Instances.** Total number of operating system instances running an agent (not a collector). This is the sum of your AM and AD Operating System Instances. Your license capacity must cover this total.
- ▶ **Application Management/Enterprise Mode (AM License) OS instances.** The number of OS instances that host Enterprise/AM mode agent instances in your production environment. These are counted against your HP Diagnostics AM license capacity.

When you install an agent, you are prompted to specify if the agent will be configured in Application Management/Enterprise mode (AM License) to work with a Diagnostics Server in a production environment. If you select this mode then the following values are set in Diagnostics:

For a Java agent - the value of the **active.products** property in the **etc/probe.properties** file is set to **Enterprise** mode at the time you install the Java Agent (see “Set the Active Products Mode” on page 505). You can change the mode value after installation by modifying this property.

For a .NET agent - the value of the **probe\_config.xml** **<modes>** element is set to **enterprise** mode at the time you install the .NET Agent (see “<modes> element” on page 592). You can change the mode value after installation by modifying this element.

For a Python agent - the mode is always set automatically to AM (cannot be set to AD mode).

For agents with Enterprise mode set, the agent hosts will be counted against your HP Diagnostics AM license capacity.

- **LoadRunner/Performance Center (AD License) OS instances.** The number of OS instances that host active LoadRunner or Performance Center AD mode application instances (does not include Enterprise/AM mode agent instances). Only active AD mode agents are counted against your HP Diagnostics AD license capacity. Those not in a run are not counted.

When you install an agent, you are prompted to specify if the agent will be configured in AD mode for LoadRunner and Performance Center runs. If you select the AD license option then the following values are set in Diagnostics:

For a Java agent - the value of the **active.products** property in the **etc/probe.properties** file is set to **AD** mode at the time you install the Java Agent (see “Set the Active Products Mode” on page 505). You can change the mode value after installation by modifying this property.

For a .NET agent - the value of the **probe\_config.xml** **<modes>** element is set to **ad** mode at the time you install the .NET Agent (see “<modes> element” on page 592). You can change the mode value after installation by modifying this element.

The advantage of running a probe in AD mode is that you only need license capacity for the number of hosts that are currently in a LoadRunner or Performance Center test run. So for example if you have agents installed on 100 test systems but you will only have probes running on 10 hosts at any one time then you would only need an AD license capacity of 10 hosts.

The following is for information only (these counts are not used as license counts) and relates to probe instances rather than OS instances (you can have more than one probe running on an OS instance).

- ▶ **Total Application Server Instances.** An application server instance is a Java Agent instance (a probe) or a .NET Agent instance (.NET worker process) or a Python Agent instance. This value is the total of Application Management/Enterprise Mode (AM License) probe instances and Load Runner/Performance Center (AD License) probe instances.
- ▶ **.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.
- ▶ **Python processes.** Any processes instrumented for monitoring by one or more Python probes.
- ▶ **Java probes.** Monitored java or javaw processes or any other processes embedding the JVM. This is equivalent to a Java probe.
- ▶ **Collector instances.** Collector instances include 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 included.
  - ▶ 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 included.
  - ▶ WebSphere MQ - Instances configured for monitoring with a <mqInstance> entry in **mq-config.xml** are included.
  - ▶ TIBCO EMS - Instances configured for monitoring with a <emsInstance> entry in **tibco-ems-config.xml** are included.

- ▶ WebMethods Broker - Instances configured for monitoring an <WmBrokerInstance> entry in **wm-broker-config.xml** are included.
- ▶ SAP/ABAP - Each discovered Dialog instance (SAP ABAP probes) is included.
- ▶ VMware - The number of vSphere servers as specified in the **vmware-config.xml** file are included.
- ▶ Any probes prior to 8.0x will be listed under Old probes.

## License Details

Selecting the Details link at the bottom of the License page displays detailed information for each host with Diagnostics probes or collectors. Details include HostName, Probe Name, port or PID, Run ID (for probes in a LoadRunner/Performance Center load testing run), probe version and product mode.

Following is an example showing part of the License Management Details page:

### License Management

---

Details for Default Client					
.NET Probes					
Host Name	PID	Probe Name	Mode	Run ID	Version
OVRNTT209.ovrtest.adapps.hp.com	:12084	L81_1ROOTCallChain2_0_DefaultWebSite.NET_OVRNTT209_W2k3	Enterprise,PRO	1	9.20.116.470
OVRNTT209.ovrtest.adapps.hp.com	:12084	L81_1ROOTJavaTrader2.WebClient_DefaultWebSite.NET_OVRNTT209_W2k3	Enterprise,PRO	1	9.20.116.470
OVRNTT209.ovrtest.adapps.hp.com	:12084	L81_1ROOTTestService2.WebClient_DefaultWebSite.NET_OVRNTT209_W2k3	Enterprise,PRO	1	9.20.116.470
OVRNTT209.ovrtest.adapps.hp.com	:12084	L81_1ROOTTestService2.WebService_DefaultWebSite.NET_OVRNTT209_W2k3	Enterprise,PRO	1	9.20.116.470

## 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 commander server. The first time these components connect to a licensed Diagnostics commander 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 commander 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 92
- Accessing the Collector Installer on page 93
- Installing the Collector on page 94
- Silent Installation of the Diagnostics Collector on page 103
- Installing the Diagnostics Collector Using the Generic Installer on page 104
- How to Manually Add Another Collection Type After Installing the Collector on page 105
- Configuring the Active System Property Files on page 106
- Configuration for SAP NetWeaver–ABAP on page 106
- Configuration for Oracle on page 110
- Configuration for SQL Server on page 113
- Configuration for MQ on page 117
- Configuration for TIBCO EMS on page 120
- Configuration for webMethods Broker on page 121
- Configuration for VMware on page 123
- Password Obfuscation on page 125
- Verifying the Diagnostics Collector Installation on page 127
- Starting and Stopping the Diagnostics Collector on page 128

- ▶ Determining the Version of the Diagnostics Collector on page 130
- ▶ Uninstalling the Diagnostics Collector on page 130

## 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 Databases (including Oracle RAC)
- ▶ IBM WebSphere MQ
- ▶ TIBCO Enterprise Message Service (EMS)
- ▶ Software AG webMethods Broker
- ▶ SQL Server Databases
- ▶ VMware vCenter or VMware ESX Servers

During the installation of the Collector, you can choose to monitor any of these active systems. After the installation, you define instances of Oracle Databases, SQL Server systems, VMware vCenter or VMware ESX servers, IBM WebSphere MQ messaging systems, TIBCO EMS systems, Software AG webMethods Broker and SAP NetWeaver–ABAP systems to be monitored. Each monitored instance is represented by a probe entity. Multiple probes can be configured for each Collector.

---

**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, Tibco EMS, webMethods Broker, VMware or SQL Server application. For Collector host requirements, see “Requirements for the Diagnostics Collector Host” on page 39.

---

## Accessing the Collector Installer

The installation can be launched from the Diagnostics installation disk, or copy the executable installation file to another location and run it, or select it from the Diagnostics Downloads page in Business Service Management.

### To access the installer from the Diagnostics installation media:

- ▶ For Windows, from the Diagnostics installation DVD (Autorun.exe) the installation menu page is displayed. From the menu, select **Diagnostics Collector** to launch the installer.
- ▶ Or you can run the appropriate installer by locating the **HPDiagCollector\_<release number>\_win.exe** file for Windows or the **HPDiagCollector\_<release number>\_<platform>.bin** files for Unix on the installation media and copying the file to the new installation location.

Continue with “Installing the Collector” on page 94.

### To download the installer from the HP Software Download Center:

- 1** Go to the HP Software web site’s Software Download Center.
- 2** Locate the **Diagnostics** downloads and choose the appropriate link for downloading the Diagnostics Collector software.
- 3** Follow the download instructions on the web site to download the installer and save it to a local disk.

Continue with “Installing the Collector” on page 94.

**To download 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 appropriate Collector installer.

---

**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 this during the installation of the Diagnostic Server by providing the path to the Diagnostics Agent and Collector installers, or you can manually copy files from the installation disk to the `<diag_server_install_dir>/html/opal/downloads` folder of the Diagnostics Server installation directory. See Step on page 62 of Chapter 2, “Installing the Diagnostics Server.”

---

Continue with “Installing the Collector” that follows.

## Installing the Collector

The following steps provide detailed instructions for installing the Collector on Windows or Unix systems.

For information on other types of installation see the following:

- ▶ 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 104
- ▶ For information on silent installation see “Silent Installation of the Diagnostics Collector” on page 103.

---

**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, you must follow the instructions for upgrading the Collector instead of these install instruction, see “Upgrade and Patch Install Instructions” on page 893

---

The Windows installer and the Unix installers in graphical mode display the same screens. If you run the Unix installer in console mode, prompts are displayed instead of screens but the flow is the same as documented in this section.

For Unix installers, where necessary, change the mode of the installer file to make it executable.

- To run the Unix 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.

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

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 925. The installer runs in console mode displaying a series of prompts.

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**. In console mode press **Enter** to continue through the license agreement and when prompted, enter **1** to accept the agreement.

---

**Note:** On some 64-bit Linux systems, if the installer program terminates without showing the license agreement, you need to install the 32-bit glibc library and then try to run the installer again (see “Diagnostics Installers Do Not Work on Some 64-bit Linux Systems” on page 910 for details).

---

Select **Next** to continue.

**2 Specify the location to install the Collector.**

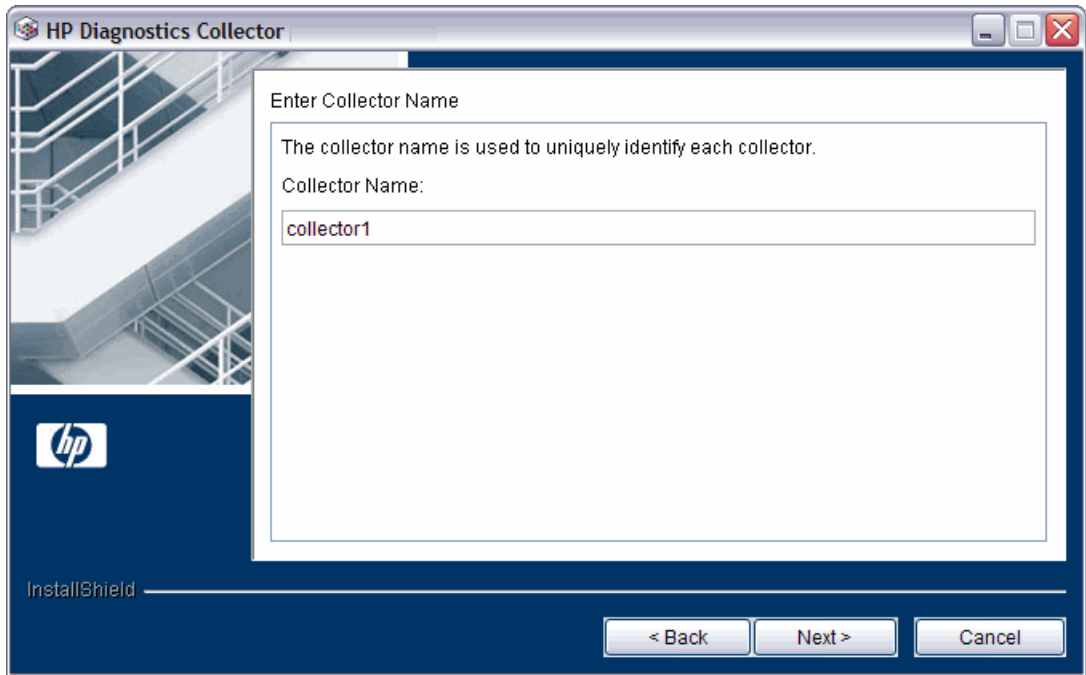
In the **Installation Directory Name** box, accept the default directory, **C:\MercuryDiagnostics\Collector** or type the name of the directory where you want to install the Collector. Or click **Browse** to navigate to another directory. In this documentation this is referred to as the `<collector_install_dir>`.

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.”

Select **Next** to continue.



### 3 Assign a unique name to the Collector.

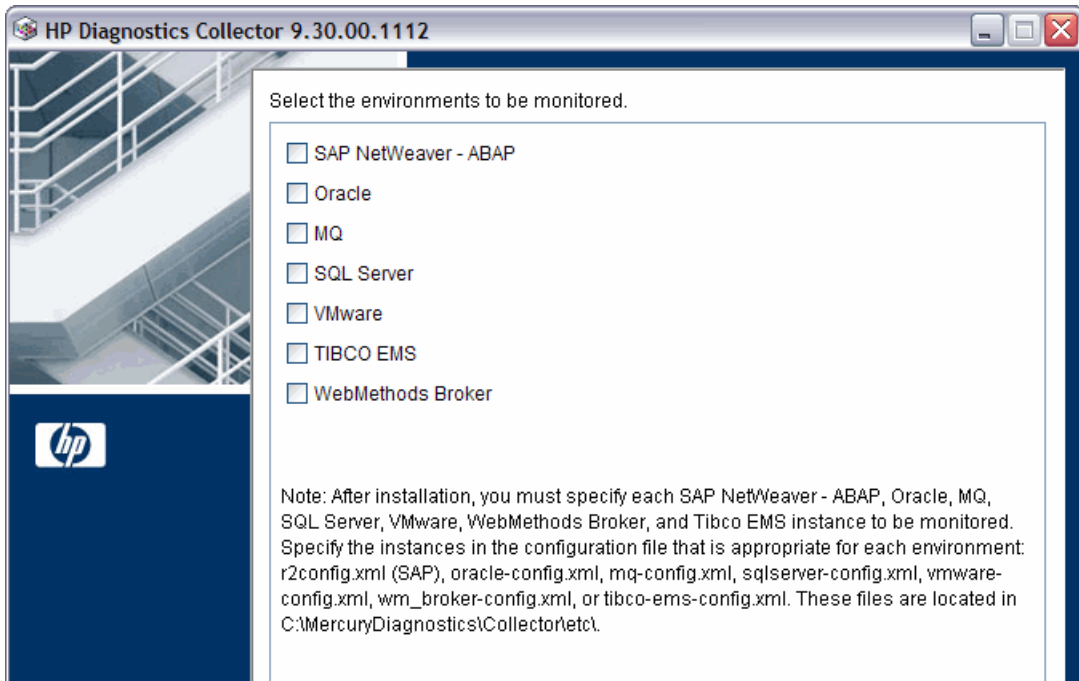


Assign a name to the Collector that uniquely identifies this specific Collector.

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

Select **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 from an Oracle 10g database server, select **Oracle**.
- ▶ To collect data in an MQ series environment, select **MQ**.
- ▶ To collect data from an SQL Server database, select **SQL Server**.
- ▶ To collect data from either a VMware vCenter or a VMware ESX server, select **VMware**.
- ▶ To collect data from a TIBCO EMS environment, select **TIBCO EMS**.
- ▶ To collect data from a webMethods Broker system, select **WebMethods Broker**.

---

**Important:** After installation, specify each of the SAP NetWeaver–ABAP, Oracle, MQ, TIBCO EMS, SQL Server, webMethods Broker and VMware 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 106.

---

Select **Next** to continue.

### 5 Provide information about the Diagnostics mediator server.

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

HP Diagnostics Collector

Provide the location of the Diagnostics Server in Mediator mode.

Diagnostics Server Mediator Host (Name or IP address):

Diagnostics Server Mediator Port:

2006

The default value is 2006.

Check the connectivity to the Diagnostics Server Mediator Host and Port.

InstallShield

< Back   Next >   Cancel

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.

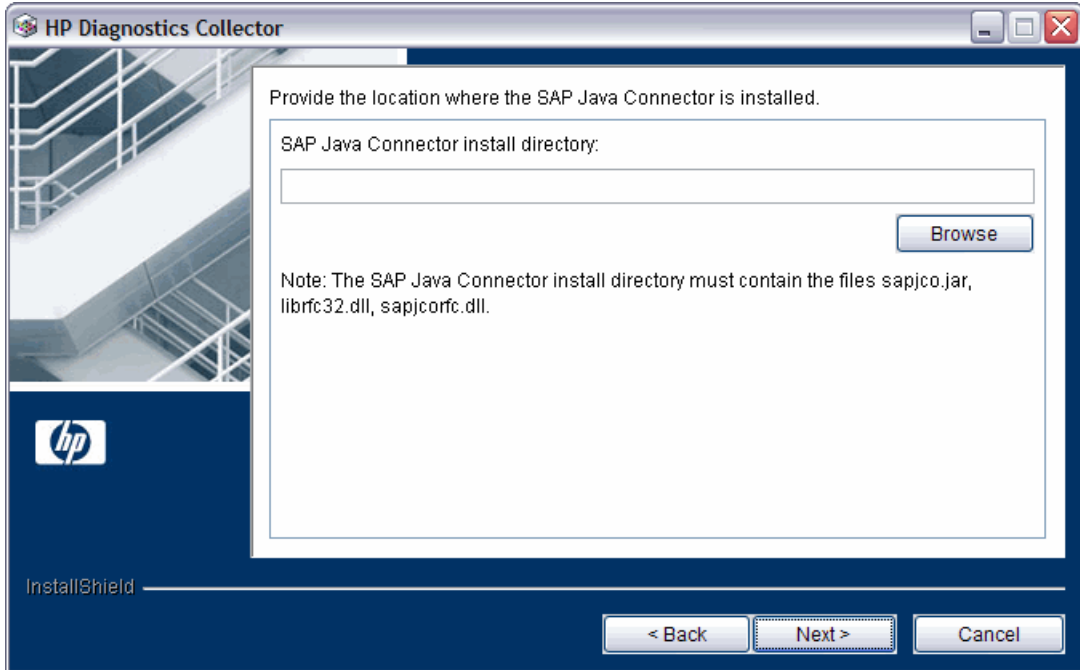
---

- 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 **2006**. 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**.

Select **Next** to continue.

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 path to the directory where the SAP Java Connector is installed. The installer will copy the necessary files to the `<collector_install_dir>\lib` directory on the system where the collector is installed.

This directory must contain the following files:

- sapjco.jar
- librfc.dll or librfc32.dll or librfccm.so
- sapjcorfc.dll or libsapjcorfc.so

If you do not know the SAP Java Connector directory name or if any of these files are missing from the directory, contact your SAP representative.

**7 Remember to copy required files after installation to the Collector system.**

If you selected Tibco EMS you will see a reminder to copy the following third party jars after installation: **tibjms.jar**, **tibjmsadmin.jar**. The files are typically found in your TIBCO EMS installation in the `<Tibco_EMS>/ems/<version>/lib` directory and you copy them to the `<collector_install_dir>/lib` directory on the system where the Collector is installed.

If you selected webMethods Broker you will see a reminder to copy the following third party jars after installation: **wm-brokerclient.jar**, **wm-g11nutils.jar**. The files are typically found in your Software AG installation in the `<SoftwareAG>/common/lib` directory and you copy them to the `<collector_install_dir>/lib` directory on the system where the Collector is installed.

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

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

To select different installation settings, click **Back** (or in console mode select Previous).

To begin installation, select **Next**.

**9 The installation completes.**

When the installation completes, a message is displayed confirming that the Collector is successfully installed. Select **Finish** to exit the installer.

**10 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 106.

**11 Verify that the Collector was installed properly and is running.**

The Collector starts running automatically when the installation is complete. You can verify the Collector installation by checking the **collector.log** file for errors. For details see, “Verifying the Diagnostics Collector Installation” on page 127.

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

## 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.

---

**Important:** With each new release of Diagnostics you should re-record the Diagnostics Collector silent install response files prior to performing silent installation on multiple machines.

---

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. Note that for Windows installers the options must be preceded with `-a`. For example: `HPDiagServer_9.20_win32.exe -a -options-record myfile`.

```
<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. Note that for Windows installers the options must be preceded with `-a`. For example: `HPDiagServer_9.20_win32.exe -a -silent -options myfile`.

```
<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 106.

## **How to Manually Add Another Collection Type After Installing the Collector**

During the initial installation of the Collector you select the different collection types or types of active systems you want to monitor such as SAP or Oracle. After installing the Collector you can add another collection type or active system type manually.

**To manually add another type of active system:**

- 1** Manually copy any required files to the `<collector_install_dir>\lib` directory. These files are required for SAP, Tibco EMS and webMethods Broker active systems. See the installation instructions for details on what files are required. Other types of collection such as Oracle or SQL Server do not require this step.
- 2** On the system where you installed the Collector, in the `<collector_install_dir>\etc\collector.properties` file edit the `active.systems` property to add the additional collection type. Valid values (case insensitive) are SAP\_R3, Oracle, MQ, SQL\_Server, VMWARE, EMS, WM\_BROKER.

## Configuring the Active System Property Files

When you install the Collector, you are asked to indicate the types of collection (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 Collector installation. An instance definition in the XML file is viewed as a probe entity of the active system. Refer to the following sections for configuration instructions:

- ▶ “Configuration for SAP NetWeaver–ABAP” on page 106
- ▶ “Configuration for Oracle” on page 110
- ▶ “Configuration for SQL Server” on page 113
- ▶ “Configuration for MQ” on page 117
- ▶ “Configuration for TIBCO EMS” on page 120
- ▶ “Configuration for webMethods Broker” on page 121
- ▶ “Configuration for VMware” on page 123

## 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\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
<b>r3system name</b>	A logical name for the probe group under which this SAP NetWeaver–ABAP probe entity appears in the Diagnostics UI.	User-defined.
<b>systemId</b>	The ID of the SAP NetWeaver–ABAP system. Consists of 3 characters only.	Format: [XXX] Obtainable from the SAP system administrator.

Property	Description	Value
<b>client</b>	The client name for the SAP NetWeaver–ABAP system.	Obtainable from the SAP system administrator.
<b>user</b>	<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 in order to query the Dialog info. The user on the target system must have this object in their authorization profile to be able to use RFC to connect to the target system.</p> <p>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\etc\r3.properties</code>).</p>	Obtainable from the SAP system administrator.
<b>password</b>	The password (plaintext) of the user connecting to the SAP NetWeaver–ABAP system.	Obtainable from the SAP system administrator.
<b>encrypted-password</b>	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 125) to encrypt the password.
<b>messageServerHost</b> (Message Server connection only)	The name of the SAP Message Server host machine.	Obtainable from the SAP system administrator.

Property	Description	Value
<b>r3Name</b> (Message Server connection only)	Consists of 3 characters only.	Format: [XXX]  Obtainable from the SAP system administrator.
<b>group</b> (Message Server connection only)	The group of the SAP application servers.	Obtainable from the SAP system administrator.
<b>dialogInstance</b>	Specify a list of Dialog Instances to be monitored.  By default all Dialog Instances within the ABAP system (cluster) are automatically discovered and monitored.  However, if the Dialog Instances are too many (and too busy) for a single Collector to handle (it may run out of memory), you can use this property to monitor only some of the Dialog Instances and monitor the rest by different Collectors.	SAP Dialog Instances

## 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 enclosed in the comment tags (`<!--` and `-->`) and paste it at the end of the file.

Use the **oracleInstance** element from the template to collect from Oracle 10g and 11g instances. If you want to collect from multiple instances add separate entries of the oracleInstance element.

To collect from Oracle RAC (Real Application Clusters) specify the **oracleRac** element. The oracleRac configurations must come after the oracleInstance configurations in the `oracle-config.xml` file.

- 3** In the copied code, alter the value of each property as described in the following table and save the file.

Properties	Description	Value
<b>hostName</b>	The name of the Oracle database server host machine. You must use the fully qualified host name.  In an oracleRAC configuration, this will be the cluster alias.	Obtainable from the Oracle administrator.
<b>portNumber</b>	Port where the Oracle database server listens for requests.	Default value: <b>1521</b>
<b>instanceName</b>	Use for the oracleInstance element (does not apply to oracleRAC element). The name given to the Oracle instance during installation of the Oracle database server.	Default value: <b>Orcl</b>  Obtainable from the Oracle administrator.

Properties	Description	Value
<b>serviceName</b>	Use for the oracleRac element (does not apply to oracleInstance element). serviceName along with the cluster alias hostName, isolates clients from changes in the RAC installation.	Obtainable from the Oracle administrator.
<b>userId</b>	The ID of the user connecting to the Oracle database server. <b>Note:</b> The user needs at least <b>CREATE SESSION</b> and <b>SELECT ANY DICTIONARY</b> to collect performance metrics.	Obtainable from the Oracle administrator.
<b>password</b>	The password (plaintext) of the user connecting to the Oracle database server.	Obtainable from the Oracle administrator.
<b>encrypted-password</b>	The password (encrypted) of the user connecting to the Oracle database server.	Use the EncryptPassword.jsp utility (see “Password Obfuscation” on page 125) to encrypt the password.

Properties	Description	Value
<b>probeName</b>	<p>For an oracleInstance. The logical name to represent this Oracle instance in the Diagnostics UI. This name must be unique.</p> <p>In an oracleRac configuration there will be multiple probes so you don't enter a probeName.</p>	<p>User-defined. If this value is not defined, the same value given for <b>instanceName</b> is used.</p> <p>The probe name for each Oracle instance in an Oracle RAC configuration is retrieved at run-time from the INSTANCE_NAME column in the GV\$INSTANCE view.</p>
<b>probeGroupName</b>	<p>The logical name of the probe group under which the probe appears in the Diagnostics UI. It can be an existing probe group, or you can define a new one.</p> <p>Optional for the oracleRac element, if omitted, probe group is set to serviceName.</p>	<p>User-defined; for example:</p> <p>Existing: <b>Default</b></p> <p>New: <b>Oracle</b></p>

**To enable collection of additional metrics:**

- 1 If the Collector encounters a metric it is not configured to collect, a warning containing the unrecognized metric ID and name is logged. If the metric is a count, percent, byte, or centisecond metric, you can optionally collect the metric by adding the metric ID to `<collector_install_dir\etc\oracle.properties`.
- 2 Locate the property name that corresponds to the type of metric you want the Collector to collect and add the metric. The property names are:
  - oracle.metrics.count
  - oracle.metrics.percent



- oracle.metrics.bytes
  - oracle.metrics.centiseconds (the Collector converts to milliseconds)
- 3** Restart the Collector.

## 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
<b>hostName</b>	The name of the SQL Server database host machine. You must use the fully qualified host name.	Obtainable from the SQL Server administrator.
<b>portNumber</b>	The number of the port where the SQL Server database listens for requests.	Default value: <b>1433</b>

Properties	Description	Value
<b>instanceName</b>	<p>The name given to the SQL Server instance during installation of the SQL Server database.</p> <p>When you specify an instance name, Diagnostics <b>automatically discovers all SQL Server databases in the instance</b>. To exclude some of these databases from collection (for example, system databases), specify a comma-separated list in the <b>exclude.db.list</b> property in the <code>&lt;collector_install_dir&gt;\etc\sqlserver.properties</code> file.</p>	<p>Default value: <b>Default</b></p> <p>Obtainable from the SQL Server administrator.</p>

Properties	Description	Value
<b>integratedSecurity</b>	<p>If set to <b>true</b>, 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 <b>HP Diagnostics Collector</b>, the Windows user credentials used to connect to SQL Server must be set as the <b>logon</b> property for the service. To do this, run the Windows Services Manager (<b>services.msc</b> from the run dialog, or <b>My Computer &gt; Manage &gt; Services and Applications &gt; Services</b>). Open the Properties dialog for service <b>HP Diagnostics Collector</b>, select the Log On tab, and set <b>Log on as:</b> 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 <b>false</b>, the username and password must be supplied. If this is not specified, its default value is false.</p>	Default value: <b>false</b>

Properties	Description	Value
<p><b>userId</b></p>	<p>The ID of the user connecting to the SQL Server database.</p> <p><b>Note:</b> The user needs at least <b>VIEW SERVER STATE</b> to collect performance metrics.</p> <p>Create the user as follows for VIEW SERVER STATE:</p> <ul style="list-style-type: none"> <li>➤ CREATE LOGIN diag WITH PASSWORD = '&lt;pwd&gt;';</li> <li>➤ USE master;</li> <li>➤ GRANT VIEW SERVER STATE TO diag;</li> <li>➤ GO</li> </ul> <p>From the SQL Server Management Studio GUI, you can right-click on the Instance name and select the properties.</p>	<p>Obtainable from the SQL Server administrator.</p>
<p><b>password</b></p>	<p>The password (plaintext) of the user connecting to the SQL Server database.</p>	<p>Obtainable from the SQL Server administrator.</p>
<p><b>encrypted-password</b></p>	<p>The password (encrypted) of the user connecting to the SQL Server database.</p>	<p>Use the EncryptPassword.jsp utility (see “Password Obfuscation” on page 125) to encrypt the password.</p>

Properties	Description	Value
<b>probeName</b>	<p>The name to be used to represent this instance as a probe in the HP Diagnostics UI.</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 <code>probeName</code>. The probes for each database are shown as <code>probeName_databaseName</code>.</p>	<p>User-defined.</p> <p>If this value is not defined, the same value given for <b>instanceName</b> is used.</p>
<b>probeGroupName</b>	<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: <b>Default</b></p> <p>New: <b>SQL Server</b></p>

## 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 <queue_manager_name> -n ** -t queue -g <OS_group_name> +dsp +get
```

```
setmqaut -m <queue_manager_name> -n SYSTEM.ADMIN.COMMAND.QUEUE -t queue -g <OS_group_name> +dsp +get +put
```

```
setmqaut -m <queue_manager_name> -n ** -t channel -g <OS_group_name> +dsp
setmqaut -m <queue_manager_name> -t qmgr -g <OS_group_name> +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 only from predefined (or non-dynamic) queues. You specify the queue types to collect or to ignore by setting properties in the `<collector_install_dir>\etc\mq.properties` file.

**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.queues</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** The `collect.predefined.queues` property is set to true by default. The other three properties are set to false by default. Specify **false** for any type for which you do not want the Collector to gather metrics and then save the `mq.properties` file.

---

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

---

MQ jar files are included with the Diagnostics Collector but if you must overwrite these files you can. The MQ jar files provided with the Collector are located in the `<collector_install_dir>\lib` directory and can be overwritten by the MQ jar files provided in your local MQ installation. You can typically find the jar files in you local WebSphere's MQ installation's `\lib` directory containing the `com.ibm.mq.jar` file. If you have difficulty locating these files contact your WebSphere MQ administrator.

**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
<b>hostName</b>	The hostName.	Obtainable from the MQ administrator.
<b>portNumber</b>	The number of the port	(optional).
<b>queueManagerName</b>	The MQ Manager to connect to.	Obtainable from the MQ administrator.
<b>channelName</b>	The channel through which to connect to the Queue Manager.	Obtainable from the MQ administrator.
<b>securityExit</b>	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 ensure your security Exit class is available on the classpath.	
<b>probeName</b>	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 not defined, it defaults to the Queue Manager name.
<b>probeGroupName</b>	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: <b>Default</b> New: <b>MQ</b>

## Configuration for TIBCO EMS

You configure the Collector to collect data for each instance of an active TIBCO Enterprise Message Service (EMS) system to be monitored.

You configure TIBCO EMS monitoring in the `<collector_install_dir>\etc\tibco-ems-config.xml` file. The layout, elements, and attributes of the xml file are described in `<collector_install_dir>\etc\tibco-ems-config.xsd`.

In addition to the configuration described below, the following TIBCO EMS jar files must be copied from your TIBCO EMS installation's `Tibco_EMS/<version>/lib` directory to the `<collector_install_dir>\lib` directory on the system where the Collector is installed:

- `tibjms.jar`
- `tibjmsadmin.jar`

**To configure TIBCO EMS monitoring:**

- 1** Open `<collector_install_dir>\etc\tibco-ems-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
<code>emsServerUrl</code>	The EMS Server URL	Default is <code>tcp://localhost:7222</code>
<code>username</code>	EMS server username. The user must have the following privileges: "view-destination" and "view-server"	
<code>password</code>	EMS server password, plain text	



Properties	Description	Value
<b>obfuscated-password</b>	EMS server password obfuscated (optional). This property takes precedence over the plain test password if both are defined. If neither are defined blank is used for the password.	Use the EncryptPassword.jsp utility (see “Password Obfuscation” on page 125) to encrypt the password.
<b>probeName</b>	The name to be used to represent this instance as a probe in the HP Diagnostics UI. This name must be unique.	User-defined.
<b>probeGroupName</b>	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: <b>Default</b> New: <b>TIBCO</b>

You can customize TIBCO data collection by setting properties in the `<collector_install_dir>\etc\tibco-ems.properties` file.

- How often to collect data
- How often to attempt to reconnect when a connection is not established
- Enable or disable server-level, queue-level and topic-level metric collection
- Include or exclude Global, Static or Temporary Queues and Topics
- Select individual metrics

## Configuration for webMethods Broker

You can configure the Collector to collect data for the webMethods Broker system to be monitored.

You configure webMethods Broker monitoring in the `<collector_install_dir>\etc\wm-broker-config.xml` file. The layout, elements and attributes of the xml file are described in `<collector_install_dir>\etc\wm-broker-config.xsd`.

In addition to the configuration described below, the following webMethods Broker jar files must be copied from your webMethods Broker installation's `<SoftwareAG>/common/lib` directory to the `<collector_install_dir>\lib` directory on the system where the Collector is installed:

- `wm-brokerclient.jar`
- `wm-g11nutils.jar`

**To configure webMethods Broker monitoring:**

- 1** Open `<collector_install_dir>\etc\wm-broker-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
<b>hostname</b>	Broker server hostname.	Required. For example: localhost.
<b>brokerName</b>	The name of the broker to connect to. If omitted, connects to the default broker as defined within the broker server.	Optional

Properties	Description	Value
<b>clientGroup</b>	The name of the client group to use. If omitted, connects to the 'admin' client group.	Optional
<b>probeGroupName</b>	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.	Optional, defaults to <b>Default</b> .  You can enter a user defined name.

You can customize webMethods data collection by setting properties in the `<collector_install_dir>\etc\wm-broker.properties` file.

- How often to collect data
- How often to attempt to reconnect when a connection is not established
- Enable or disable server-level, queue-level metric collection
- Select individual metrics

## Configuration for VMware

You configure the Collector to collect data for each VMware node to be monitored. You configure VMware monitoring in the `<collector_install_dir>\etc\vmware-config.xml` file. The layout, elements, and attributes of the xml file are described in `<collector_install_dir>\etc\vmware-config.xsd`. Changes to the `vmware-config.xml` file are picked up dynamically.

The Collector requires a patch be installed on the vCenter server: (see [http://kb.vmware.com/selfservice/microsites/search.do?cmd=displayKC&docType=kc&externalId=1024596&sliceId=1&docTypeId=DT\\_KB\\_1\\_1&dialogID=139216791&stateId=1 0 139218894](http://kb.vmware.com/selfservice/microsites/search.do?cmd=displayKC&docType=kc&externalId=1024596&sliceId=1&docTypeId=DT_KB_1_1&dialogID=139216791&stateId=1 0 139218894) for more information).

You should have the latest VMware Tools installed on the VMware Guest. These tools can be installed using the vSphere Client. The latest tools are required for you to drill down from VMware Guest to Hosts in the Diagnostics UI because the VMware tools make the Guest's FQDN available to the VMware Collector via the vCenter.

The VMware Host/Guests associations are created when the Collector is started. New VMware Hosts and VMware Guests may take up to 15 minutes to show up in Diagnostics. Deleted or migrating VMware Guests may take up to five minutes to show up in Diagnostics.

**To configure VMware monitoring:**

- 1** Open `<collector_install_dir>\etc\vmware-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
<b>serverURL</b>	The URL used to connect to the VMware ESX or vCenter server via the VMware infrastructure vSphere Web Services API.  For example: <code>https://&lt;myVM.myCo.com&gt;/sdk</code>	
<b>userId</b>	The VMware ESX or vCenter user id.  At a minimum the user must have ReadOnly access and be placed in the Users group.	

Properties	Description	Value
<b>encrypted-password</b>	The encrypted VMware password corresponding to userId. First checks for an encrypted password and use it if it is non-blank; otherwise use the plaintext password. the plaintext password does not exist or is blank then uses blank for the password.	Optional
<b>password</b>	The plaintext VMware password corresponding to userId.	Optional

You can customize VMware data collection by setting properties in the `<collector_install_dir>\etc\vmware.properties` file for the following:

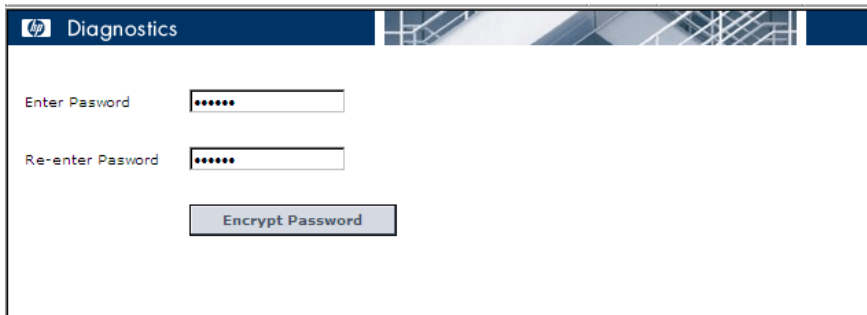
- ▶ You can limit the query interval and reconnection time. The query interval is just a hint to the Collector, because the sampling interval must actually be a multiple of the interval configured on the VMware Server.
- ▶ You can also filter by VMware host (ESX Server) and VMware guest (Virtual Machine). If the VMware Collector is unable to handle the load of an entire vCenter, then the host and guest filters may allow you to use the VMware Collector on the part of the vCenter that is most important to you or to partition the vCenter among multiple VMware Collectors. To use these filters, modify **vmware.properties** as described in the property file for **host.filters** and **guest.filters** properties.

## Password Obfuscation

Create an obfuscated password using the web application included with Diagnostics. Access the Security page (<http://<host name>:2006/security>) and select **Encrypt Password** at the bottom of the page. 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 following xml files for the different collection types:

- **r3config.xml** file used to configure the SAP NetWeaver–ABAP collector
- **oracle-config.xml** file used to configure the Oracle collector
- **vmware-config.xml** file used to configure the VMware collector
- **tibco-ems-config.xml** file used to configure the TIBCO EMS collector
- **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 password input fields: "Enter Password" and "Re-enter Password", both with masked characters (dots). Below the 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**, **vmware-config.xml**, **tibco-ems-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 and select Encrypt Password to encrypt the
# password.
# Plaintext passwords can be used by replacing the security.encrypted-password
# with security.password. You must also change the encrypted password in the
# <install-dir>/etc/.htaccess file, as well as all the Diagnostics probe, and
# servers, that communicate with each other in your enterprise.
#
security.username=mercury
security.encrypted-password=OBF:1c431jg81hv41k1d1l161wu81z0d1pyl1wmt1n6h1y
m71n511wnd1pw11z0h1wu61kxw1jyl1hse1jd21c2z
```

## Verifying the Diagnostics Collector Installation

The Collector starts running automatically when the installation is complete. You can verify the Collector installation by checking the `collector.log` file for errors.

Once a collector probe instance is started you can launch the Diagnostics Enterprise UI to verify that the probe is working. Go to `http://<Diagnostics_commander_server>:2006/`. For now you can use the default user/password of `admin/admin` or the login you were given if a different one has been set up for you.

You can also check the System Health view to find information about the Collector deployment and the machine that hosts the collector.

**To access the System Views:**

- 1** Open the Diagnostics UI as the Mercury System customer from [http://<Diagnostics\\_Commanding\\_Server\\_Name>:2006/query/](http://<Diagnostics_Commanding_Server_Name>:2006/query/).
- 2** In the query page locate the Mercury System customer in the list and select the link to Open Diagnostics.
- 3** Log in to Diagnostics and on the Applications window select Entire Enterprise and select any link to open the Diagnostics Views.
- 4** In the Views pane you'll see the System Views view group. Open the view group and select either the System Health view or System Capacity view.

## Starting and Stopping the Diagnostics Collector

### Instructions for a Windows Machine

**To start the Collector on a Windows machine:**

- Select **Start > All 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 > All 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.

## 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, .NET and Python Agents**

This section includes:

- Installing Java Agents
- Preparing Application Servers for Monitoring with the Java Agent
- Preparing Application Servers for Client Monitoring with the Java Agent
- Installing .NET Agents
- Installing and Setting Up Python Agents



# 5

---

## Installing Java Agents

This section describes how to install a Java Agent and give you information about the setup and configuration of the Java Agent

**This chapter includes:**

- ▶ Overview of the Java Agent Installation on page 134
- ▶ Accessing the Java Agent Installer on page 135
- ▶ Installing the Java Agent on page 137
- ▶ Running the Java Agent Setup Module on page 141
- ▶ About Preparing the Application Server for Monitoring on page 150
- ▶ Register the Agent with the Diagnostics Servers on page 150
- ▶ Verifying the Java Agent Installation on page 151
- ▶ About Additional Configuration and Custom Instrumentation on page 152
- ▶ 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 157
- ▶ Setting File Permissions (UNIX Only) on page 159
- ▶ Determining the Version of the Java Agent on page 160
- ▶ Uninstalling the Java Agent on page 160

## Overview of the Java Agent Installation

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 setting up 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.
- ▶ Run the Java Agent setup module which starts automatically after the installer.
- ▶ The next steps are to instrument the JRE used by your application server and configure your application server JVM parameters to invoke the Java Agent. Depending on your application server, you may use the automatic JRE instrumentation options instead of manually running the JRE Intrumenter utility.

Allow approximately 400MB of free space in the temp directory. For information about the recommended system configurations for hosting the Java Agent, see “Requirements for the Diagnostics Java Agent Host” on page 36. Java Agent installers are provided for Windows and for several UNIX platforms. You can run the installers in graphical mode (which displays install screens like the Windows installer) or using the console mode command line interface. If you are not able to use the regular UNIX installers, you can use the Generic installer as described in “Installing the Java Agent Using the Generic Installer” on page 156.

---

**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
```

---

---

**Note:** If there is a pre-existing installation of the Java Agent on the host machine, see “Upgrade and Patch Install Instructions” on page 893 for important instructions on how to upgrade the agent systems.

---

**HP Software-as-a-Service (SaaS).** HP Diagnostics can be deployed into an HP Software-as-a-Service (SaaS) environment. In a SaaS deployment the Diagnostics Java Agents are installed in your company's IT environment and the Diagnostics Commander Server and Mediator Servers are installed by HP on a SaaS system on-premise at HP. During the setup of the Java Agent you select the option for configuring the agent for - Diagnostics with SaaS-hosted mediator installed on HP premises.

See Accessing the Java Agent Installer to begin.

## Accessing the Java Agent Installer

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 access the Installer from a Diagnostics installation media:**

- ▶ For Windows, from the Diagnostics installation DVD (Autorun.exe) the installation menu page is displayed. From the menu, select **Diagnostics Agent for Java** to launch the installer.

OR

- ▶ You could run the appropriate installer directly by locating the **HPDiagTVJavaAgt\_<release number>\_<platform>.bin** files for Unix or **HPDiagTVJavaAgt\_<release number>.win.exe** files for Windows on the installation media and copying the file to the new installation location.

Continue with “Installing the Java Agent” on page 137.

**To download the installer from the HP Software Download Center:**

- 1** Go to the HP Software web site’s Software Download Center.
- 2** Locate the **Diagnostics** (or TransactionVision) downloads and choose the appropriate link for downloading the Diagnostics Agent software. Note that you could also use the download center to get the Diagnostics profiler trial/evaluation software.
- 3** Follow the download instructions on the web site to download the installer and save it to a local disk.

Continue with “Installing the Java Agent” on page 137.

**To download the installer from the Business Service Management’s Diagnostics downloads page:**

- 1** In Business Service Management select **Admin > Diagnostics** from the main menu and click the **Downloads** tab.
- 2** On the Downloads page, click the link to download the appropriate Java Agent installer.



---

**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 Installing the Java Agent.

## Installing the Java Agent

This section provides detailed instructions for a *first time* installation of the Java Agent on Windows or UNIX systems.

---

**Important:** If there is a pre-existing installation of the Java Agent on the host machine, you must follow the instructions for upgrading the agent system instead of these install instructions see “Upgrade and Patch Install Instructions” on page 893.

---

### Tips for using the installer on UNIX systems:

Where necessary when using the installer, change the mode of the UNIX installer file to make it executable. For more information about UNIX commands, see “Using UNIX Commands” on page 925.

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

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

The console mode UNIX installer displays installation prompts rather than a UI.

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

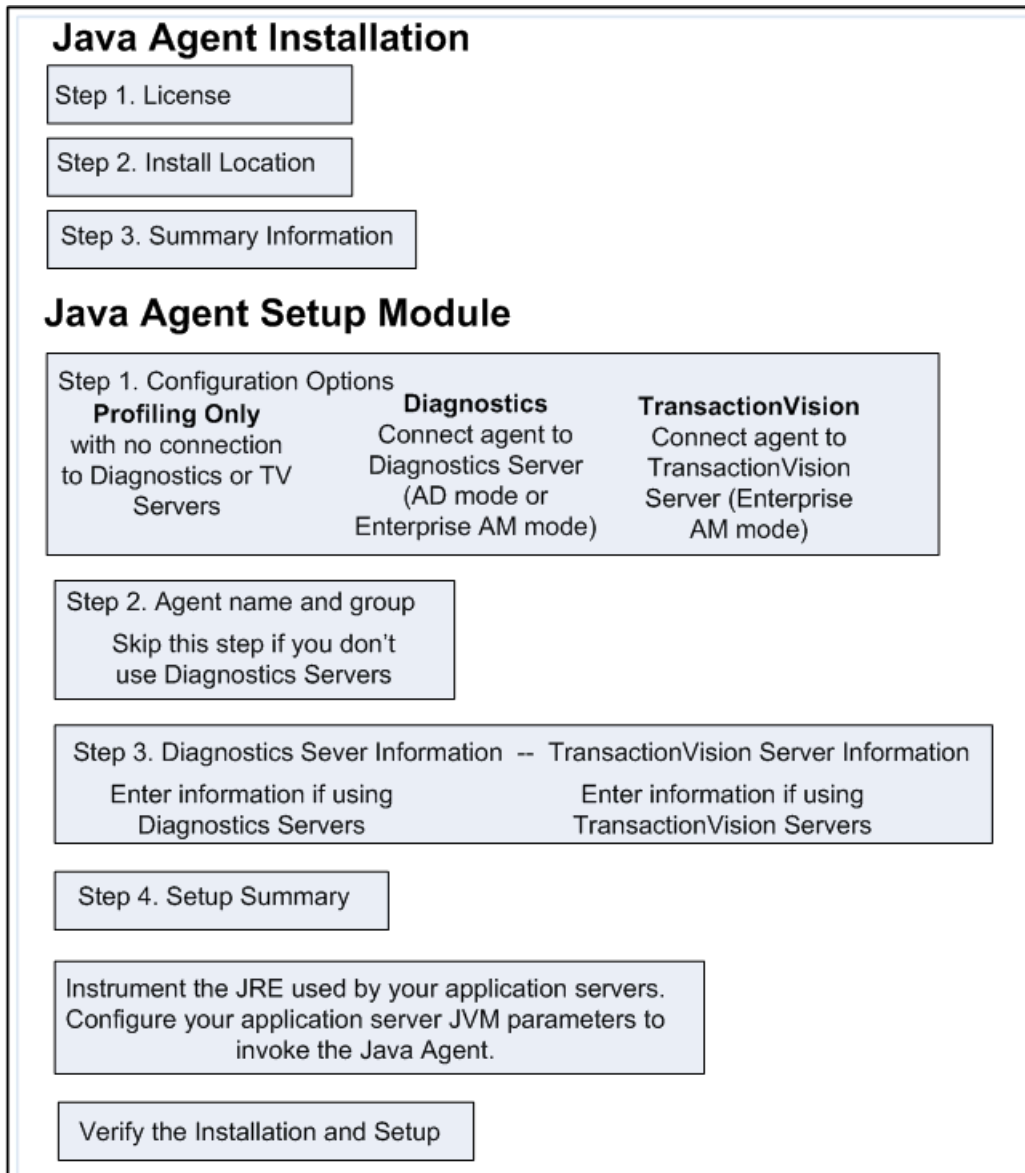
```
./<installer>
```

The graphical mode UNIX installer displays the same screens that are displayed for the Windows installer.

**For information on other types of installation see the following:**

- ▶ For z/OS see “Installing the Java Agent on a z/OS Mainframe” on page 154.
- ▶ For information on installation with a generic installer see “Installing the Java Agent Using the Generic Installer” on page 156.
- ▶ For information on silent installation see “Silent Installation of the Java Agent” on page 157

An overview of the Java Agent installation steps is shown in the diagram below; refer to the rest of this section for details on each step.



Begin the Java Agent installation with “Step 1. End User License Agreement” on page 140.

## Step 1. End User License Agreement

Accept the end user license agreement.

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

In the console mode interface, press **Enter** to move to the next page of text instead of selecting Next, or type **q** to jump to the end of the license agreement.

---

**Note:** On some 64-bit Linux systems, if the installer program terminates without showing the license agreement, you need to install the 32-bit glibc library and then try to run the installer again (see “Diagnostics Installers Do Not Work on Some 64-bit Linux Systems” on page 910 for details).

---

Select **Next** (in console mode **Enter**) to proceed and continue to the next step.

## Step 2. Specify Install Location

Specify the location where you want to install the agent.

Accept the default install directory or specify 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.

In the console mode interface, at the **Installation Directory Name** prompt, accept the default installation location shown in brackets, or enter the path to a different location.

---

**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.

---

If you encounter an error, check to see if the JavaAgent directory already exists and remove it. This could happen if you installed and uninstalled the Java Agent previously without also removing the JavaAgent sub-directory.

Select **Next** (in console mode **Enter**) to proceed and continue to the next step.

### **Step 3. Review Pre Installation Summary Information**

Review the installation summary information.

The installation directory and size requirement are listed.

If these are acceptable, select **Next** (in console mode **Enter**) to start the installation. The installation can take a few minutes.

When the installation is complete the Java Agent Setup Module starts. Continue on to the next section on Running the Java Agent Setup Module.

## **Running the Java Agent Setup Module**

The Java Agent can be configured as a Profiler without any connection to a Diagnostics Server (or as an agent that works with a Diagnostics Server and/or TransactionVision Server). When the agent is initially configured as a Profiler only, you can, at a later time, configure the agent to work with a Diagnostics server by re-running the Java Agent Setup Module.

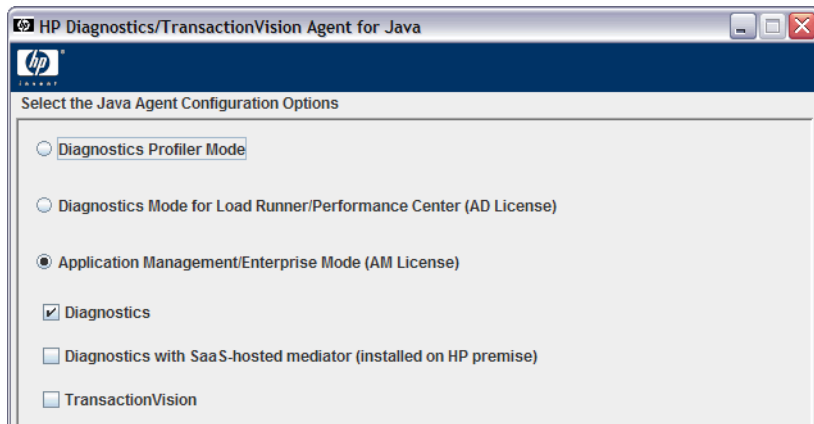
Configure the Java Agent by using the Java Agent Setup Module which starts automatically at the end of the Java Agent installation or you can start it at any time by choosing **Start > All Programs > HP Java Agent > Setup Module**. Or for UNIX you can start it at any time by running <probe\_install\_dir>/bin/setupModule.sh.

The Java Agent Setup Module includes the following steps, select Step 1. Configuration Options to begin:

- ▶ “Step 1. Configuration Options” on page 142
- ▶ “Step 2. Agent Name and Group” on page 144
- ▶ “Step 3. Diagnostics Server Information” on page 145
- ▶ “Step 4. Post Setup Summary” on page 149

## Step 1. Configuration Options

Indicate if the Java Agent is to be installed as a standalone Profiler without any connection to a server (for example if you are installing the Diagnostics Java Profiler trial software), or if you are installing the agent to work with LoadRunner/Performance Center or to work with a Diagnostics and/or TransactionVision Server.



Make the selection that is appropriate for the environment where you will be using the agent.

- ▶ **Application Management/Enterprise Mode (AM License).** Select this option to install the agent for use with a Diagnostics Server and/or a TransactionVision Server in an enterprise (or production) environment.

Then indicate which of the following the agent will be configured for:

- ▶ Either a Diagnostics Server (installed locally) OR a Diagnostics Server hosted on an HP SaaS system on-premise at HP

- A TransactionVision server
- Both a Diagnostics Server installed locally and a TransactionVision Server

If you select HP SaaS mode then an HP SaaS administrator will provide you with information on connecting the Java agent to an HP SaaS hosted Diagnostics mediator server.

If you select TransactionVision, see the *HP TransactionVision Deployment Guide* in the Business Service Management documentation library for details on setup options specific to TransactionVision.

With the Application Management/Enterprise Mode (AM License) option, the value of the **active.properties** property in the **etc/probe.properties** file is set to **Enterprise** mode if you select the Diagnostics Server. It is set to **TV** mode if you select the TransactionVision server at the time you install the Java Agent (see “Set the Active Products Mode” on page 505).

For those agents with Enterprise mode set, the agent will be counted against your HP Diagnostics AM license capacity.

- Select **Diagnostics Profiler Mode** to configure the agent as a Diagnostics Java Profiler without any connection to a Diagnostics server. Diagnostics Profiler mode is typically used when installing the Diagnostics Java Profiler trial software prior to purchasing the HP Diagnostics product.

If you select Diagnostics Profiler Mode the value of the **active.products** property in the **etc/probe.properties** file is set to **PRO** mode at the time you install the Java Agent (see “Set the Active Products Mode” on page 505).

When you select Diagnostics Profiler Mode there are no other configuration options so you can select **Finish** to complete the configuration and the Post Setup Summary dialog is displayed.

- **Diagnostics Mode for LoadRunner/Performance Center (AD License)**. Select this option to install the agent for use with a Diagnostics Server in a load testing (or pre-production) environment where probes are used only in LoadRunner or Performance Center runs.

The agent will be installed in AD license mode which means the agent will only be counted against your HP Diagnostics AD license capacity when the agent is in a LoadRunner or Performance Center testing run. See “License Information Based on Currently Connected Probes” on page 85 for more information on AD license capacity.

In AD mode the agent will ONLY capture data during a LoadRunner or Performance Center run and the results will be stored in a specific Diagnostics database for that run, for example, Default Client:21. When the agent is in AD mode it will NOT send any data to the server unless the probe is part of a LoadRunner/Performance Center run.

If you select this AD License option, the value of the **active.properties property** in the **etc/probe.properties** file is set to **AD** mode at the time you install the Java Agent (see “Set the Active Products Mode” on page 505).

The advantage of running a probe in AD mode is that probes in AD mode are only counted against license capacity if they are in a LoadRunner or Performance Center test run. For example if 20 probes are installed in LoadRunner/Performance Center AD mode but only have 5 are in a run at any one time then you would only need an AD license capacity of 5 probes.

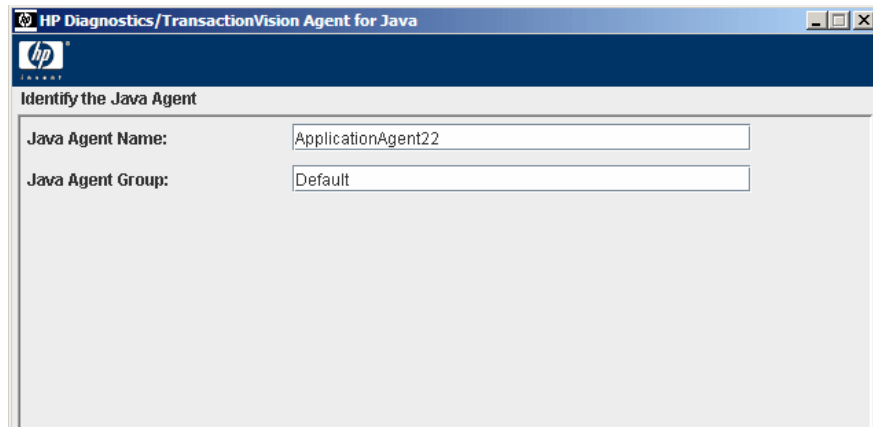
In the console mode interface enter an X to select the mode for installation.

Select **Next** (in console mode **Enter**) to proceed and continue to the next step.

## Step 2. Agent Name and Group

Skip this step if the agent won't be reporting to a Diagnostics Server.

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 and the system the agent is installed on (for example if installing on the system ovrserver130 with a WebLogic application server you could use the agent name WL10\_MedRec\_ovrserver130).

- 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.

Select **Next** (in console mode **Enter**) to proceed and continue with the next step.

### **Step 3. Diagnostics Server Information**

Skip this step if the agent won't be reporting to a Diagnostics Server.

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

In the console mode interface for each option enter an **X** for Yes and **O** for No.

The screenshot shows a Windows-style dialog box titled "HP Diagnostics/TransactionVision Agent for Java". The main heading is "Configure the Diagnostics Java Agent".

**Diagnostics Server Connectivity**

Diagnostics Server Name: localhost  
Diagnostics Server Port: 2006

**Additional Options**

- Tune Diagnostics Java Agent for use in an SAP NetWeaver Application Server
- Enable gzip compression (Recommended for HP SaaS deployments)
- Enable SSL
- Use Proxy Server to connect to Diagnostics Server

**Proxy Server Options**

Proxy Server Name:   
Proxy Server Port: 80  
Proxy Server Username (optional):   
Proxy Server Password (optional):

**Local Profiler Password (Recommended to change the default "admin" password)**

Password: admin

**Notes:**  
The default server port is 2006. When SSL is enabled, the default server port or 8443. When SSL is enabled AND the mediator is SaaS hosted, the default server port is 443.

Buttons: Back, Next, Finish, Cancel

Footer: Wed May 09 14:41:37 PDT 2012: This is the last dialog...please click the Finish button to save and exit

In the **Diagnostics Server Name** box, enter the host name or IP address of the host of the Diagnostics Server this agent should connect to. 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.

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

**Mediator Server.** In a distributed environment with a commander server and mediator servers, enter the information for the Diagnostics Mediator Server that is to receive data from the agent.

If you are using **HP Software-as-a-Service (SaaS)** then the Diagnostics Mediator is installed by HP on an HP SaaS system on-premise at HP. An HP SaaS administrator will provide you with the information on the host name and port to use. Also note that for an HP SaaS environment the Enable gzip option will be checked automatically for you and you will not see the Enable SSL option because it is configured on the Diagnostics Commander/Mediator on HP premises.

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

The default port for the Diagnostics Server is **2006**. For SSL communications with the server the port is typically set to **8443** for a locally installed server.

The default port if you are installing the agent for a SaaS environment is **443** (the SaaS administrator will provide you with details).

If the port was changed since the Diagnostics Server was installed, you should specify the new port number here instead of the default.

- ▶ 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.
- ▶ If you need to compress the data between the Java Agent and the mediator, set the **Enable gzip compression** check box. This is a tradeoff between bandwidth and probe performance overhead. In an HP SaaS environment you are typically asked to enable gzip compression, see your SaaS administrator for more information.

- ▶ The Java Agent connects to the Diagnostics server via SSL when Enable SSL is checked or the Diagnostics server is SaaS-hosted on HP premises. Checking the **Enable SSL** checkbox instructs the agent to connect to the Diagnostics Server in SSL mode and to attempt to download the required certificate chain from the server. As a result the **server.properties** trusted certificate will then include the certificate. For more information on secure communications see “Enabling HTTPS Between Components” on page 839.
- ▶ If a proxy server is used to communicate with the Diagnostics Mediator Server select **Use Proxy Server to connect to Diagnostics Server** check box and enter the appropriate options. In an HP SaaS environment if your company requires a proxy to communicate to outside servers then you would select this option. These options can also be set in the **dispatcher.properties** file on the agent system by setting proxy.enabled to true and entering the other options. See “Configuring Diagnostics Servers and Agents for HTTP Proxy” on page 671.

Proxy Server Options:

- ▶ **Proxy Server Name.** Host name of the proxy server.
  - ▶ **Proxy Server Port.** Port of the proxy server.
  - ▶ **Proxy Server Username (optional).** The user used to authenticate the proxy server.
  - ▶ **Proxy Server Password (optional).** The password used to authenticate the proxy server.
- ▶ It is recommended that you change the **Local Profiler Password** from the default (admin) password.

## TransactionVision Information

If you selected to configure this agent for TransactionVision then you will see additional dialog boxes to configure the agent for TransactionVision. See the *HP TransactionVision Deployment Guide* for details on these installation options.

## Setup Process Begins

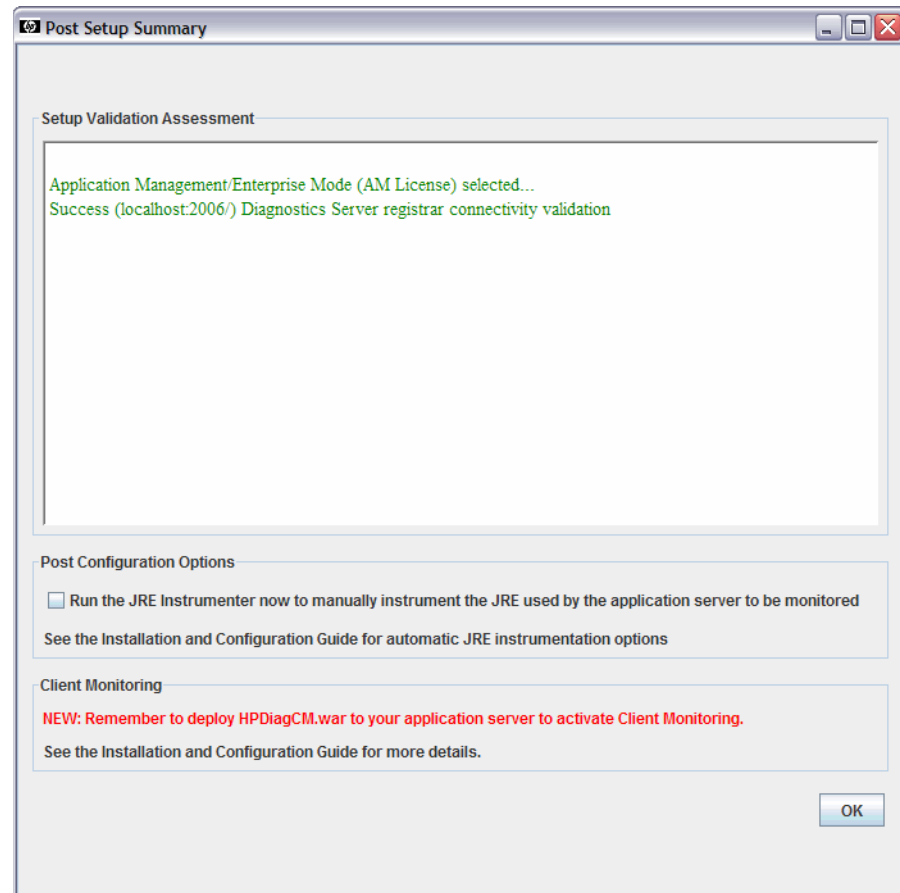
The Java Agent Setup process begins. In graphical mode a progress bar indicates how the configuration is proceeding.

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.

Continue with the next step.

## Step 4. Post Setup Summary

Review the Post Setup Summary and click **OK**.



You should use the automatic JRE instrumentation options instead of manually running the JRE Instrumenter utility. Therefore the checkbox for running the JRE Instrumenter utility is left blank by default. See the About Preparing the Application Server for Monitoring below to continue.

## About Preparing the Application Server for Monitoring

The next steps are to instrument the JRE used by your application server and configure your application server JVM parameters to invoke the Java Agent.

Follow the instructions in “Preparing Application Servers for Monitoring with the Java Agent” on page 161 for how to instrument the JRE used by your application server and how to configure the JVM parameters for specific application servers to invoke the Java Agent.

After you prepare the application server for monitoring by the Java Agent then you restart the application server and the Java Agent will be invoked to begin monitoring the application.

For more information on client monitoring see Chapter 7, “Preparing Application Servers for Client Monitoring with the Java Agent”.

## Register the Agent with the Diagnostics Servers

Configure the agent to connect to the Diagnostics Commander Server.

One of the functions of the Diagnostics commander 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/
```

## Verifying the Java Agent Installation

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 check that the agent is working properly until you instrument the JRE used by your application server and configure your application server JVM parameters to invoke the Java Agent.

Once a Java probe instance is started you can launch the Diagnostics Enterprise UI to verify that the probe is working. Go to [http://<Diagnostics\\_commander\\_server>:2006/](http://<Diagnostics_commander_server>:2006/). For now you can use the default user/password of **admin/admin** or the login you were given if a different one has been set up for you.

You can also check the System Health view to find information about the Java agent deployments and the machines that host them.

### To access the System Views:

- 1** Open the Diagnostics UI as the Mercury System customer from [http://<Diagnostics\\_Commanding\\_Server\\_Name>:2006/query/](http://<Diagnostics_Commanding_Server_Name>:2006/query/).
- 2** In the query page locate the Mercury System customer in the list and select the link to Open Diagnostics.
- 3** Log in to Diagnostics and on the Applications window select Entire Enterprise and select any link to open the Diagnostics Views.
- 4** In the Views pane you'll see the System Views view group. Open the view group and select either the System Health view or System Capacity view.

You can also 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 instrument the JRE or did not enter the Java parameter such as **Xbootclasspath** correctly. In the **probe.log** file look for errors and look for an entry that says "Successfully downloaded first command" which indicates that the communication between the probe and the server has been established.

Continue on to the next section for post installation/setup tasks.

## About Additional Configuration and Custom Instrumentation

There is some additional configuration and optional custom instrumentation you can do. See the following:

- “Configure SOAP Message Handlers” on page 152
- “Specify Probe Properties as Java System Properties” on page 153
- “Optional Advanced Configurations” on page 153
- “Optional Custom Instrumentation” on page 153
- For information on configuration for environments with proxies see “Configuring Diagnostics Servers and Agents for HTTP Proxy” on page 671, firewalls see “Configuring Diagnostics to Work in a Firewall Environment” on page 675 and for enabling HTTPS see “Enabling HTTPS Between Components” on page 839.

### Configure 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 239.



## Specify 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=Enterprise,TV
```

- ▶ 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
```

## Optional Advanced Configurations

Determine which advanced configurations of the probe apply to your environment. See Chapter 13, “Advanced Java Agent and Application Server Configuration.”

## Optional Custom Instrumentation

You can also configure custom instrumentation if needed. See Chapter 10, “Custom Instrumentation for Java Applications,” for details.

## Installing the Java Agent on a z/OS Mainframe

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

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

- ▶ The Diagnostics Java Agent is installed in and makes extensive use of the Unix System Services environment (USS) on z/OS.
- ▶ When installed in a z/OS environment, the Java Agent expects the Diagnostics property files to be in EBCDIC format rather than in ASCII. Use an EBCDIC editor to update the property files and store the updates in that same format.
- ▶ System metrics are not captured for z/OS. The Diagnostics Java Agent 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 721.

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

A .tgz file containing the Java Agent files is included on the Diagnostics installation disk and is used to install the Java Agent on a z/OS mainframe.

**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 system where you wish to unzip the installer.
- 2** Unzip **HPDiagTVJavaAgt\_<release number>\_zos.tgz** using gzip as shown in the following example:

```
gzip -d HPDiagTVJavaAgt_9.10_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_9.10_zos.tar
```

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

- 4 Ensure that you have a Java executable on your path, and then run the Java Agent Setup Module to configure the Java Agent as a Profiler only or as a Java Agent to work with a Diagnostics Server and/or a TransactionVision Processing Server. Refer to “Running the Java Agent Setup Module” on page 141 for details. For example (or as appropriate for your shell):

```
setenv PATH /u/Java6_31/J6.0/bin:/bin
```

And then:

```
<probe_install_dir>/bin/setupModule.sh
```

- 5 After you install the agent and run the Setup Module you must then instrument the JRE used by the application server and configure your application server JVM parameters to invoke the Java Agent see Chapter 6, “Preparing Application Servers for Monitoring with the Java Agent.”
- 6 Verify the agent installation as described in “Verifying the Java Agent Installation” on page 151.
- 7 Complete post installation configuration as required. See “About Additional Configuration and Custom Instrumentation” on page 152.

## 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_9.10_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 -xzf HPDiagTVJavaAgt_9.10_unix.tar
```

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

- 4 Run the Java Agent Setup Module to configure the Java Agent as a Profiler only or as a Java Agent to work with a Diagnostics Server and/or a TransactionVision Processing Server. Refer to “Running the Java Agent Setup Module” on page 141 for details.

```
<probe_install_dir>/bin/setupModule.sh
```

- 5 After you install the agent and run the Setup Module then you must instrument the JRE used by your application servers and configure your application server JVM parameters to invoke the Java Agent, see Chapter 6, “Preparing Application Servers for Monitoring with the Java Agent.”
- 6 Verify the agent installation as described in “Verifying the Java Agent Installation” on page 151.
- 7 For additional information see “About Additional Configuration and Custom Instrumentation” on page 152.

## Silent Installation of the Java Agent

Silent installation of the Java Agent is supported. 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.

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.

---

**Important:** With each new release of Diagnostics you should re-record the response files prior to performing silent installation on multiple machines.

---

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. Note that for Windows installers the options must be preceded with -a. For example: HPDiagServer\_9.20\_win32.exe -a -options-record myfile.

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

Where <installResponseFileName> is the fully qualified file. 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  
<JASMRresponseFileName>
```

On UNIX:

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

Where <JASMRresponseFileName> is the fully qualified file. 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. Note that for Windows installers the options must be preceded with -a. For example:  
HPDiagServer\_9.20\_win32.exe -a -silent -options myfile.

```
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 <JASMRResponseFileName>
```

On UNIX systems, use empty quotes to unset the environment variable.

```
set HP_JAVA_AGENT_SETUP=""
```

#### 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.

## Setting File Permissions (UNIX Only)

On UNIX only, after installing the Java Agent, make the agent's 'group' the same as the application server's 'group'. Then assign the following permissions to files in the probe install directory for the group:

- ▶ 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.

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
```

## 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`).
- ▶ And the version number is available in the System Health view in the Diagnostics UI (see Appendix D, “Using System Views for Administrators”).

## 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 `uninstaller.bin`, which is located in the `<probe_install_dir>/_uninst` directory.
- ▶ On other UNIX machines, choose a 1.4 or later JVM and run `java -jar <probe_install_dir>/_uninst/uninstall.jar` to uninstall the Java Agent.

Also remember to remove the Java Agent parameter you added to your application server startup.



# 6

---

## Preparing Application Servers for Monitoring with the Java Agent

This chapter describes how to prepare your application servers to allow the HP Diagnostics Java Agent to monitor your applications.

**This chapter includes:**

- About Preparing Application Servers for Monitoring on page 162
- Examples for Configuring Application Servers on page 163
- About the JRE Instrumenter and Different Options to Invoke on page 219
- Other Configuration Options on page 232

## About Preparing Application Servers for Monitoring

Once you install the HP Diagnostics Java Agent you must prepare (instrument) your application servers to allow the Java Agent to monitor your applications. This preparation usually involves **instrumenting the JRE** used by your application servers and **configuring your application server JVM parameters** to invoke the Java Agent.

Diagnostics' JRE instrumentation does not modify the installed JRE, but rather places copies of instrumented classes under the Java Agent installation directory. Then with the proper JVM parameters these instrumented classes will be loaded into the JVM that runs your application server. The instrumentation is done using the Diagnostics JRE Instrumenter utility which can be invoked automatically using various options or manually.

There are two-levels of instrumentation:

► Basic instrumentation.

By adding the Java Agent to your application server start up, your application server will be instrumented and monitored. This is done by adding the `-javaagent` option to your application server JVM parameters.

► Recommended instrumentation.

In addition to the basic instrumentation, we recommend that you also instrument the JRE (Java Runtime Environment) used by your application server using the JRE Instrumenter utility provided by the Java Agent. This extra instrumentation will enable the Java Agent to provide advanced features such as the patent-pending Collection Leak Pinpointing (CLP). CLP automatically detects leaking collections and provides a stack trace of where the leak occurs. This helps identify issues early, while there is time to mitigate the issue (such as an eventual out of memory error/server crash), as well as saves developers time by avoiding the tedious task of analyzing heap dumps (see “Configuring Collection Leak Pinpointing” on page 377). And this extra instrumentation also has performance benefits on certain application servers such as WebSphere 6.1.

For general instructions on using the different JRE instrumentation modes see “About the JRE Instrumenter and Different Options to Invoke” on page 219.

For older application servers that use JRE 1.4, such as WebLogic 8.1 and WebSphere 5.1/6.0, the basic instrumentation is not available; you must use the recommended instrumentation on them.

To continue, find your application server in the list below and follow the instructions for instrumenting and configuring.

## Examples for Configuring Application Servers

This section provides examples of how to configure various commonly used application servers for monitoring. See the section “About the JRE Instrumenter and Different Options to Invoke” on page 219 for a description of the different ways you can invoke the JRE Instrumenter.

---

**Important:** 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 application server configuration changes. Always create a backup copy of any file that you plan to update before making the changes.

---

“Example 1: Configuring GlassFish” on page 164

“Example 2: Configuring JBoss” on page 168

“Example 3: Configuring Oracle” on page 172

“Example 4: Configuring SAP NetWeaver” on page 178

“Example 5: Configuring TIBCO ActiveMatrix/BusinessWorks” on page 183

“Example 6: Configuring Tomcat” on page 187

“Example 7: Configuring WebLogic” on page 192

“Example 8: Configuring WebSphere” on page 196

“Example 9: Configuring webMethods” on page 212

For the most recent information on what application server versions are supported on what platforms, see the HP Diagnostics Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](http://support.openview.hp.com/sc/support_matrices.jsp) or contact HP customer support.

---

**Note:**

- ▶ The script examples shown in this chapter may have line breaks to make it easier to read. The actual scripts do not have line breaks. The text of the commands will wrap on your screen as necessary.
  - ▶ Use quotes if there are spaces in the path that you specify.
- 

### **Example 1: Configuring GlassFish**

Configuring a GlassFish application server involves modifying its configuration files to add JVM parameters. Below are the instructions for a generic GlassFish 3.x or 9.1 application server 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.

For GlassFish application servers you configure JRE instrumentation using implicit mode (see “Using the JRE Instrumenter in Automatic Implicit Mode” on page 224).

#### **To configure a GlassFish application server:**

- 1** For GlassFish 3.x application servers locate the property **org.osgi.framework.bootdelegation** in the GlassFish configuration files and append **com.mercury.opal.capture.proxy** to its end (also need a comma as a separator).

For GlassFish 9.1 skip this step.

In GlassFish 3.1.2, this property is located in **<GlassFish\_install\_dir>/glassfish/config/osgi.properties**.

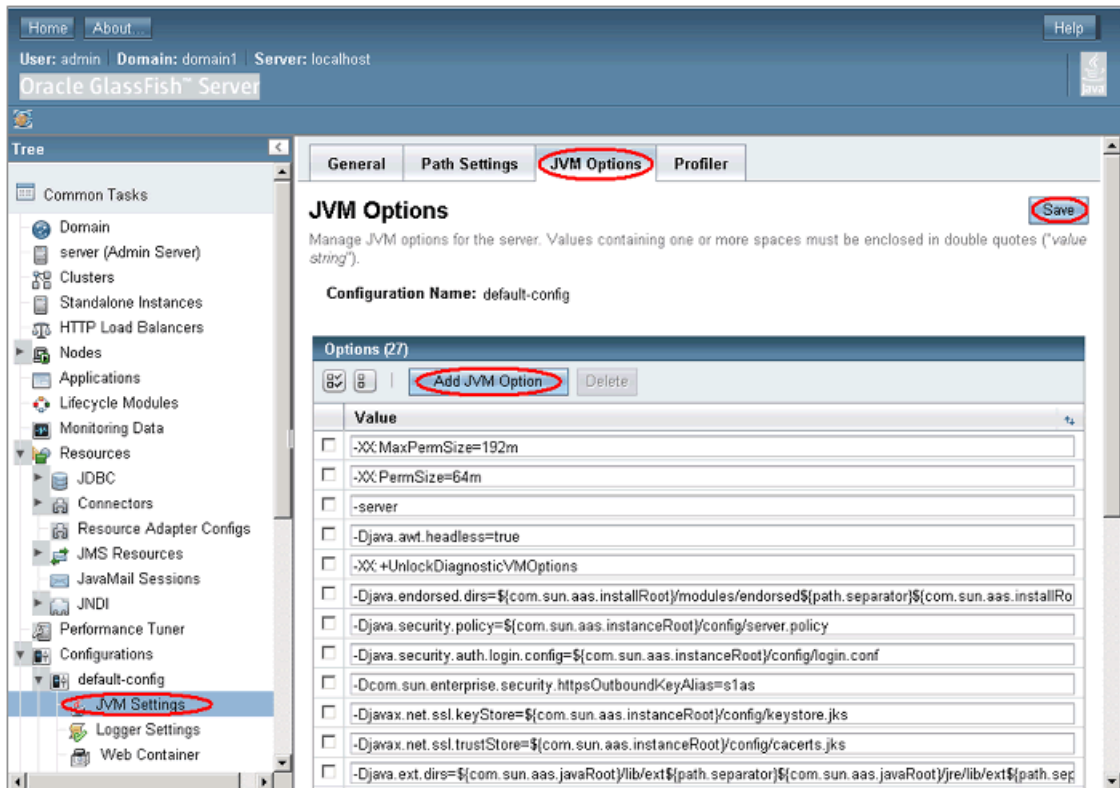
In an earlier version of GlassFish, this property may reside in the following two files:

< **GlassFish\_install\_dir** >/osgi/equinox/configuration/config.ini

< **GlassFish\_install\_dir** >/osgi/felix/conf/config.properties

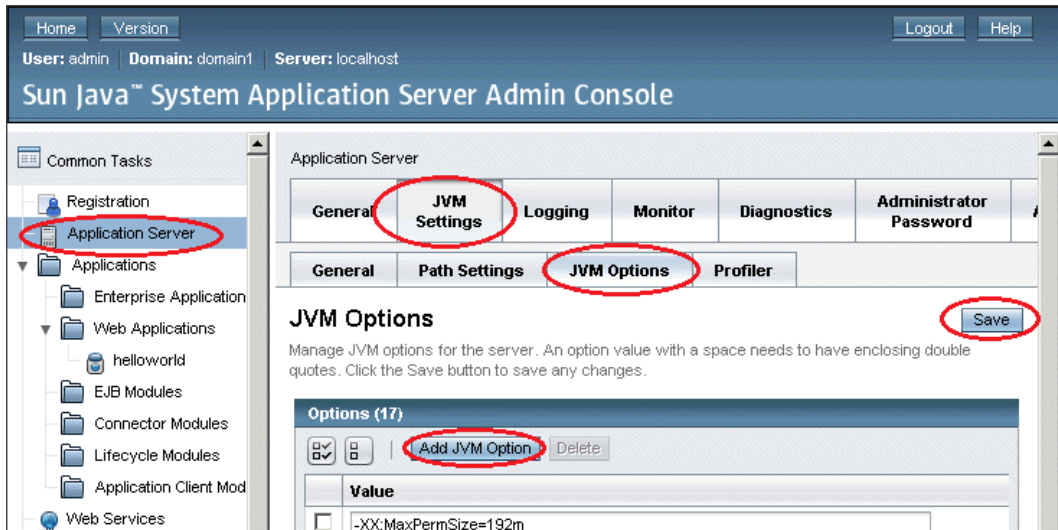
- 2 Log in to the GlassFish Administration Console and go to the **JVM Options** page, by using the following steps:

For GlassFish 3.1.2, in the left-hand tree go to **Configurations** > **{config\_name}** > **JVM Settings**, where {config\_name} is the name of your server configuration (such as, **server-config**). Then select the **JVM Options** tab. See the screenshot below as a reference.



If you are working with an earlier version of GlassFish, click **Application Server** in the left-hand tree and then select the **JVM Settings** tab at the top. Then select the **JVM Options** tab.

For GlassFish 9.1, in the left-hand tree click Application Server and then select the JVM Settings tab at the top. Then select the **JVM Options** tab. See the screenshot below as a reference.



- 3 Click the **Add JVM Option** button to add two JVM parameters, one at a time. The first parameter (-javaagent) causes the application server JVM to invoke the Java Agent at start-up. On the first invocation, the second parameter (-Xbootclasspath) causes the application server JRE to be instrumented. In the -Xbootclasspath parameter enter a name to specify the name of the directory for storing the instrumented classes. In the following examples you would substitute a name you choose for **MyServer**.

Below is an example for a Windows environment:

```
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\auto\MyServer\instr.jre
```

Below is an example for a UNIX environment:

```
-javaagent:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar
-Xbootclasspath/p:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/classes/auto/MyServer/instr.jre
```

**Note:** In case of cluster setup, also add the following JVM parameter  
`-Dprobe.id=<ProbeID>_%0`  
where `<ProbeID>` is the probe name that you want to assign to this application server cluster. The `"%0"` string will be replaced with a unique ID so that you can differentiate different probe instances.

---

**4** Restart the GlassFish application server.

The Java Agent will be invoked and implicitly run the JRE Instrumenter to instrument the JRE.

If the GlassFish application server does not start, you can check and change the JVM parameters in the `<GlassFish_install_dir>/glassfish/domains/<domain_name>/config/domain.xml` file to resolve the issue, where `<domain_name>` is the name of your domain (such as, `domain1`).

If GlassFish fails to start with class loader exceptions, then check whether `AS_JAVA` is set correctly in `<GlassFish_install_dir>/config/asenv.bat`.

If the GlassFish application server takes a long time to initialize and fails to start due to the following error:

Could not load Logmanager "com.sun.enterprise.server.logging.ServerLogManager"

add the following JVM option

`-Ddiag.agent.init.delay.ms=<delay_ms>`

where `<delay_ms>` is the number of milliseconds (for example, 6000). You can increase the `<delay_ms>` until the error is gone.

**5** To verify that the probe was configured correctly, check for entries in the `<JavaAgent_install_dir>/DiagnosticsAgent/log/<probe_id>/probe.log` file. If there are no entries in the file, you may not have set the JVM parameters correctly. Look for error messages in the GlassFish server log.

**6** Optionally, restart the application server again so that it will use the instrumented JRE.

**Important:** If you update the JRE used by your application server in the future (such as applying an application server patch), before you start your application server again you must delete the `<JavaAgent_install_dir>/DiagnosticsAgent/classes/auto/MyServer` directory (use your directory name for MyServer) so that the new JRE will be instrumented. Otherwise, your application server may not start. For details see “Using the JRE Instrumenter in Automatic Implicit Mode” on page 224.

---

## Example 2: Configuring JBoss

JBoss application servers are started by shell or command scripts. Therefore, we recommend that you modify the startup scripts to configure the JBoss application server. 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 with specific examples for 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.

For JBoss application servers you configure JRE instrumentation using explicit mode (see “Using the JRE Instrumenter in Automatic Explicit Mode” on page 221).

To use Automatic Explicit mode, you need to accomplish two tasks:

- ▶ Modify your application server startup script to run the JRE Instrumenter using the same JRE used by your application server. The output from the JRE Instrumenter will give you the JVM parameters you will need in the next task.
- ▶ Configure your application server JVM parameters found in the output from the JRE Instrumenter.

### To configure a JBoss application server:

- 1 Locate the startup script that is used to start JBoss for the application. For example:
  - ▶ On JBoss versions earlier than 7.0:



The startup script file is typically located in a path similar to the following:

```
<JBOSS_HOME>\bin\run.[bat|sh]
```

where <JBOSS\_HOME> is the path to your JBoss installation directory, such as `C:\jboss-4.2.3.GA` or `C:\jboss-6.0.0.Final`.

- On JBoss 7.0 or higher:

The startup script file is typically located in a path similar to one of the following:

```
<JBOSS_HOME>\bin\domain.[bat|sh]
```

```
<JBOSS_HOME>\bin\standalone.[bat|sh]
```

where <JBOSS\_HOME> is the path to your JBoss installation directory, such as `C:\jboss-as-7.1.0.Final`.

---

**Note:** If your JBoss application server is started as a Windows service, before you continue with the following steps, make sure that you can start the application server using the startup script, as the startup script will be invoked by the Windows service behind the scene.

---

- 2 Create a backup copy of the startup script and use your editor to open the startup script.
- 3 Locate the java command line (or code block) that starts the application server.

Below is an example showing the java command line from a .bat file:

```
"%JAVA%" %JAVA_OPTS% ^  
-Djava.endorsed.dirs="%JBOSS_ENDORSED_DIRS%" ^  
-classpath "%JBOSS_CLASSPATH%" ^ org.jboss.Main %*
```

Below is an example showing the java command line from a .sh file:

```
if [ "x$LAUNCH_JBOSS_IN_BACKGROUND" = "x" ]; then
  # Execute the JVM in the foreground
  "$JAVA" $JAVA_OPTS \
    -Djava.endorsed.dirs="$JBOSS_ENDORSED_DIRS" \
    -classpath "$JBOSS_CLASSPATH" \
    org.jboss.Main "$@"
  JBOSS_STATUS=$?
else
  # Execute the JVM in the background
  "$JAVA" $JAVA_OPTS \
    -Djava.endorsed.dirs="$JBOSS_ENDORSED_DIRS" \
    -classpath "$JBOSS_CLASSPATH" \
    org.jboss.Main "$@" &
  JBOSS_PID=$!
  . . . . .
```

- 4 Add two lines directly above the java command line (or code block). The first line invokes the java command to run the JRE Instrumenter; the second line adds the required JVM parameters. If you do not know what JVM parameters to use, you can add them later in step 6. In these two lines, you enter a name to specify the directory for storing the automatically instrumented JRE classes. In the following examples you would substitute a name you choose for **MyServer**.

Below is an example showing the added two lines in a .bat file of a JBoss application server using JRE 5 or higher:

```
"%JAVA%" -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\reinstrumenter.jar -f MyServer
set JAVA_OPTS=%JAVA_OPTS% -Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

Below is an example showing the added two lines in a .sh file of a JBoss application server that uses JRE 5 or higher:

```
"$JAVA" -jar /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/reinstrumenter.jar -f MyServer
JAVA_OPTS="$JAVA_OPTS -Xbootclasspath/p:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/classes/
MyServer/instr.jre -javaagent:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar"
```

Below is an example showing the added two lines in a .bat file of a JBoss application server using JRE 1.4:

```
"%JAVA%" -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f MyServer
set JAVA_OPTS=%JAVA_OPTS% -Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre;C:\MercuryDiagnostics\JavaA
gent\DiagnosticsAgent\classes\boot
```

---

**Note:** If your java command line does not use the JAVA\_OPTS variable to define the JVM parameters, you need to change the variable name JAVA\_OPTS shown in these examples to the correct name.

---

---

**Important:**

- ▶ For JBoss 6 or higher, add the following JVM parameter to JAVA-OPTS:  
**-Djava.util.logging.manager=org.jboss.logmanager.LogManager**
  - ▶ For JBoss 7 or higher, add the following JVM parameter to JAVA-OPTS:  
**-Djboss.modules.system.pkgs=org.jboss.byteman,com.mercury.opal**
- 

- 5** Save the changes to the startup script and restart the application server using the modified script.
- 6** To help catch errors or typos, execute the startup script, find the output from the JRE Instrumenter (search for Xbootclasspath). If you have added the second line (setting JVM parameters) to the startup script in Step 4, compare it with the JVM parameters from the JRE Instrumenter output. If they are not the same, update the startup script with the correct JVM parameters provided by the JRE Instrumenter and restart the application server. If you have not added the second line (setting JVM parameters) to the startup script in Step 4, add it now and restart the application server. See Step 4 for examples.

- 7 To verify that the probe was configured correctly, check for entries in the `<javaAgent_install_dir>\DiagnosticsAgent\log\<probe_id>\probe.log` file. If there are no entries in the file, either the JRE instrumentation did not succeed or you did not configure the JVM parameters correctly. For details, see “Using the JRE Instrumenter in Automatic Explicit Mode” on page 221.

### Example 3: Configuring Oracle

This section provides instructions for configuring an Oracle 9i or 10g application server.

#### About configuring an Oracle9i application server:

Oracle9i application servers are configured by adding the JVM parameters provided by the JRE Instrumenter to the XML file used to start the application server.

#### 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 and open the `opmn.xml` file to be edited using your editor.
- 3 Run the JRE Instrumenter manually to instrument the JRE used by your Oracle application server. (See “Using the JRE Instrumenter in Manual Mode” on page 226 for instructions on how to run the JRE Instrumenter manually.)
- 4 Add the Java parameters that you copied from the JRE Instrumenter results (such as the `Xbootclasspath` property) to the `java-option value` property.

The following is an example of the `Xbootclasspath` parameter:

```
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\Sun\1.4.2_04\instr.jre;  
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\boot
```

**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" mmi="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" 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/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:\MercuryDiagnostics\JavaAgent\
  DiagnosticsAgent\classes\SunM_42_07\instrjre;C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\boot" />
  <oc4j-option value="-properties" />
  <port ajp="3000-3100" mi="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" mi="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>
</process-manager>
</ias-instance>

```

- 5 Save the changes to the XML file and restart the Oracle application server.
- 6 To verify that the probe was configured correctly, check for entries in the `<JavaAgent_install_dir>\DiagnosticsAgent\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 Java parameter such as Xbootclasspath correctly.

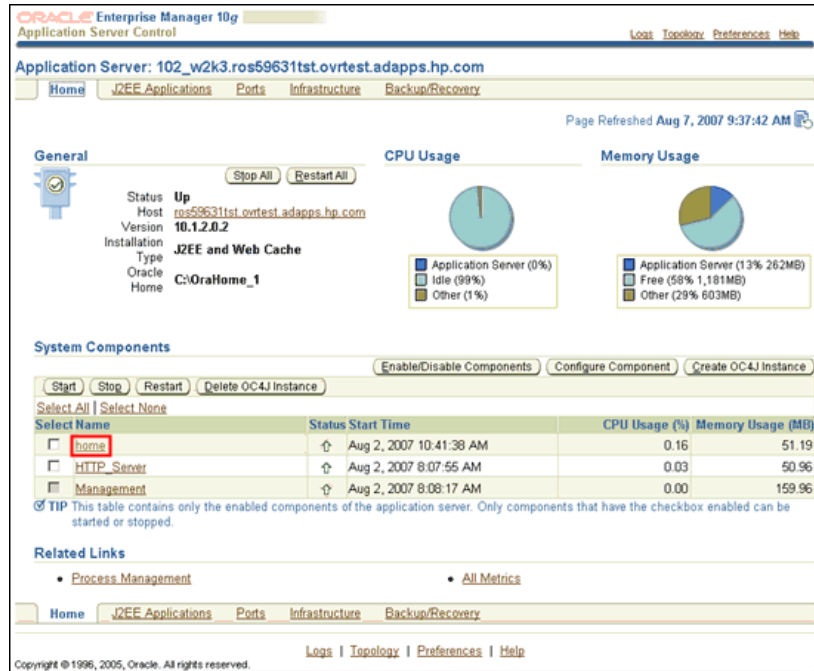
---

**Important:** If you update the JRE used by your application server in the future (such as applying an application server patch), you must run the JRE Instrumenter again to instrument the new JRE and change the JVM parameters accordingly. Otherwise, your application server may not start. For details, see “Using the JRE Instrumenter in Manual Mode” on page 226.

---

To configure an Oracle 10g application server:

- 1 Run the JRE Instrumenter manually to instrument the JRE used by your Oracle application server. (See “Using the JRE Instrumenter in Manual Mode” on page 226 for details.) The JRE Instrumenter will provide the JVM parameters (such as the -Xbootclasspath parameter) to be used later.
- 2 Open Oracle's **Application Server Control Console**,



- 3 Click the **home (or MyOC4J)** System Component.
- 4 On the **OC4J: home** page, select **Administration**.

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



6 In the Server Properties window, under **Command Line Options**, add the JVM parameters you copied from the JRE Instrumenter results to the **Java Options** box.

---

**Note:** It is required to add a (^) prior to the /p switch or Oracle will change the (/) switch option to a (\).

---

The following is an example of the Xbootclasspath parameter with the (^) inserted.

```
-Xbootclasspath^/ p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\Sun\1.4.2_07\instr.jre;  
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\boot
```



**ORACLE Enterprise Manager 10g**  
Application Server Control

Application Server: 102\_w2k3\_ros596311st\_ovrtst.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:   
 Application Directory:   
 Deployment Directory:

**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	<a href="#">Virtual Machine</a> <a href="#">Links</a> <a href="#">Metrics</a>

**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:   
 JMS Ports:   
 AJP Ports:

**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](#)

**7** Apply the changes and restart the Oracle server.

**8** To verify that the probe was configured correctly, check for entries in the `<JavaAgent_install_dir>\DiagnosticsAgent\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 Java parameter such as `Xbootclasspath` correctly.

Some of the Web Services deployed on Oracle OC4J application server, due to their non-compliance to the JAX-WS standard, may not be recognized by Diagnostics. To work around this issue you can try setting `annotation.inheritance.allow=true` in `inst.properties`.

**Important:** If you update the JRE used by your application server in the future (such as applying an application server patch), you must run the JRE Instrumenter again to instrument the new JRE and change the JVM parameters accordingly. Otherwise, your application server may not start. For details, see “Using the JRE Instrumenter in Manual Mode” on page 226.

---

### **Example 4: Configuring SAP NetWeaver**

Depending on the JRE version used by your SAP NetWeaver application server, the instructions for configuration are different. The following two sections provide instructions for a generic NetWeaver implementation with JRE version 5 and higher and a generic NetWeaver implementation with JRE version 1.4. Your site administrator should be able to use these instructions to guide you through making the changes that are appropriate to your specific environment.

For SAP NetWeaver application servers with JRE version 5 or higher you configure JRE instrumentation using implicit mode (see “Using the JRE Instrumenter in Automatic Implicit Mode” on page 224). For SAP NetWeaver application servers with JRE version 1.4 you run the JRE Instrumenter manually (see “Using the JRE Instrumenter in Manual Mode” on page 226).

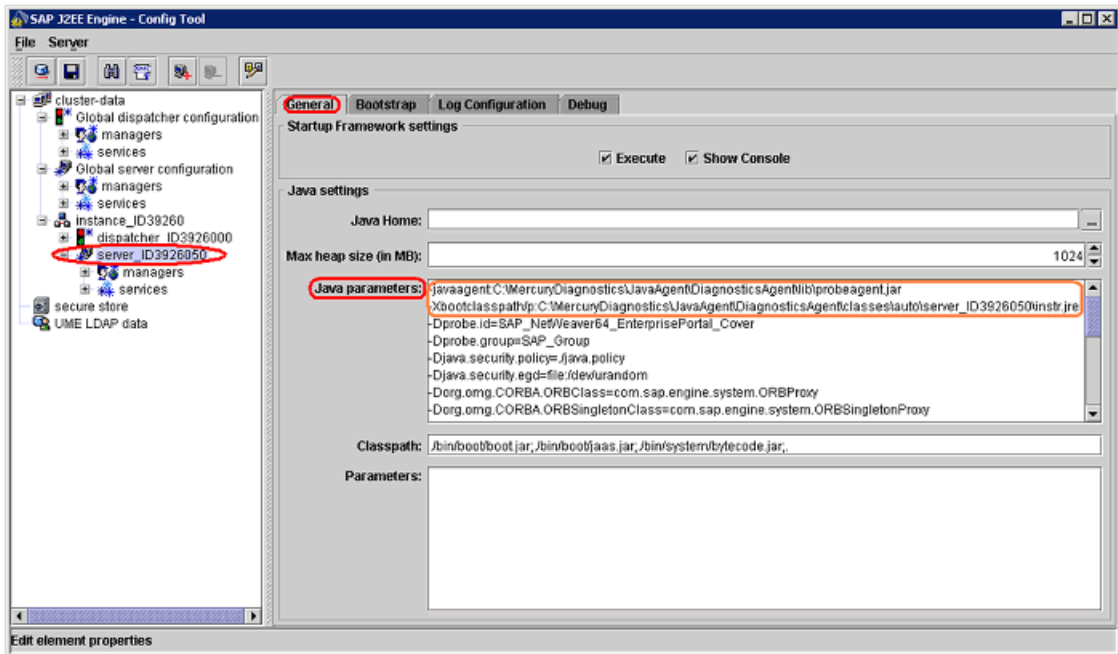
**To configure a SAP NetWeaver application server that uses JRE version 5 or higher:**

- 1** Run the NetWeaver J2EE Engine configuration tool. The script to run this tool is called **configtool.bat** and is located in the **usr\sap\CR2\JC00\j2ee\configtool** directory, where CR2 is an example of the name of your SAP system.
- 2** In the configuration tool UI, in the left-hand tree, select the server that you want to monitor. For example in the screenshot below, select **cluster-data > instance\_ID39260 > server\_ID3926050**. Then, at the right-hand side select the **General** tab and add two JVM parameters into the **Java parameters** text window.

The first parameter (**-javaagent**) causes the application server JVM to invoke the Java Agent at start-up. On the first invocation, the second parameter (**-Xbootclasspath**) causes the application server JRE to be instrumented. In the **-Xbootclasspath** parameter enter a name to specify the directory for storing the instrumented classes. In the example below the name `server_ID3926050` is used. You would substitute a name you choose for `server_ID3926050`.

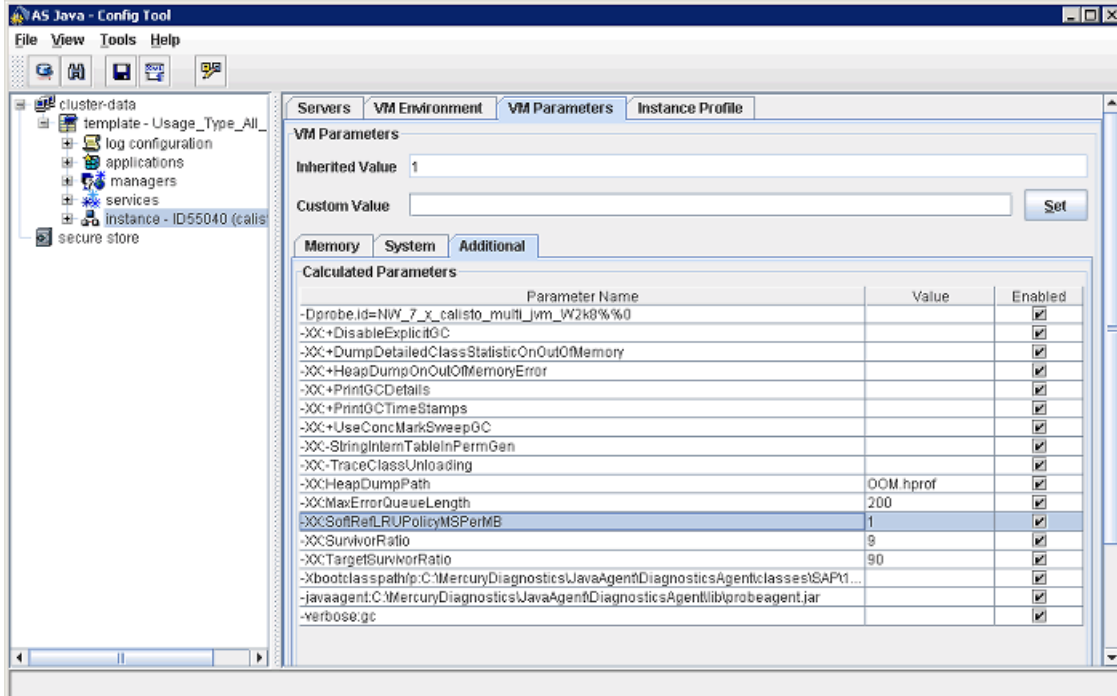
```
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\auto\server_ID3926050\instr.jre
```

Below is an example screen for SAP NetWeaver versions 7.1 or earlier of the JVM parameters.



The following is an example screen for SAP NetWeaver version 7.3. You enter the JVM parameters in the Custom parameters box and you must enter each parameter separately (**-javaagent** and **-Xbootclasspath**).

In a clustered environment where a single startup script is used to start multiple probed application server instances you also enter the Custom parameter `-Dprobe.id=<probeName>%0`. This will generate a custom probe identifier for each probe. On Windows, use `%%0`. Use the first `%` to escape the second `%`.



- 3 Save your changes and exit the configuration tool.
- 4 Edit the `<JavaAgent_install_dir>\etc\capture.properties` file and assign the following values to these properties:
  - `event_buffer.size = 10000`
  - `event_buffer.flush.level = 1000`
- 5 Restart the SAP NetWeaver application server.

- 6 To verify that the probe was configured correctly, check for entries in the `<JavaAgent_install_dir>/DiagnosticsAgent/log/<probe_id>/probe.log` file. If there are no entries in the file, the JRE Instrumentation may have failed or you did not enter the JVM parameters correctly. Check the NetWeaver bootstrap log for error messages.

---

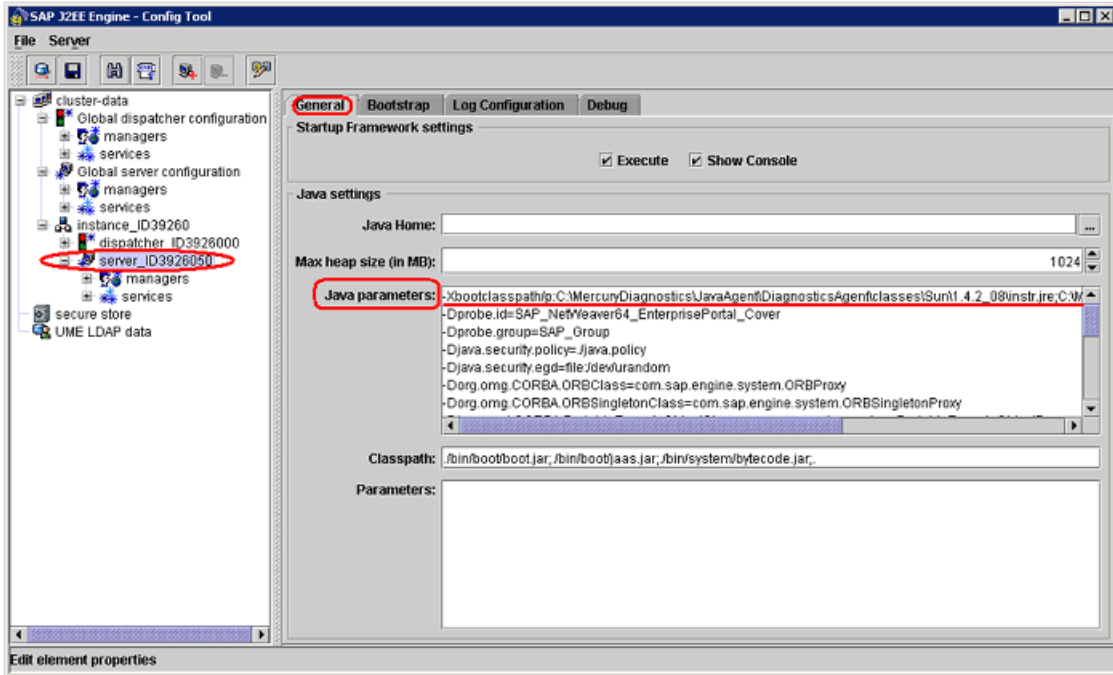
**Important:** If you update the JRE used by your application server in the future (such as applying an application server patch), before you start your application server again you must delete the `<JavaAgent_install_dir>/DiagnosticsAgent/classes/auto/server_ID3926050` directory (use your directory name for server\_ID3926050) so that the new JRE will be instrumented. Otherwise, your application server may not start. For details, see “Using the JRE Instrumenter in Automatic Implicit Mode” on page 224.

---

**To configure a SAP NetWeaver application server that uses JRE version 1.4:**

- 1 Locate the JRE that is used to run the NetWeaver application server.
- 2 Run the JRE Instrumenter to instrument this JRE. The JRE Instrumenter will provide the JVM parameters (such as the `-Xbootclasspath` parameter) to be used later. See “Using the JRE Instrumenter in Manual Mode” on page 226” for how to run the JRE Instrumenter and copy the resulting JVM parameters to the clipboard for use in step 4 below.
- 3 Run the NetWeaver 2EE Engine configuration tool. The script to run this tool is called `configtool.bat` and is located in the `usr\sap\CR2\JC00\j2ee\configtool` directory, where CR2 is an example of the name of your SAP system.

- 4 On the configuration tool UI, in the left-hand tree, select the server that you want to monitor, for example, **cluster-data > instance\_ID39260 > server\_ID3926050**. In the General tab, you can find the Java parameters text window. Add the JVM parameter provided by the JRE Instrumenter into the text window. See the screenshot below as a reference.



- 5 Save your changes and exit the configuration tool.
- 6 Edit the `<JavaAgent_install_dir>\etc\capture.properties` file and assign the following values to these properties
 

```
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 `<JavaAgent_install_dir>/DiagnosticsAgent/log/<probe_id>/probe.log` file. If there are no entries in the file, you may not have set the JVM parameters correctly. Look for error messages in the NetWeaver bootstrap log.

## Example 5: Configuring TIBCO ActiveMatrix/ BusinessWorks

The following sections describe the steps to configure TIBCO ActiveMatrix and BusinessWorks so that the applications can be monitored.

For TIBCO ActiveMatrix and BusinessWorks application servers you configure JRE instrumentation using implicit mode (see “Using the JRE Instrumenter in Automatic Implicit Mode” on page 224).

---

**Important:** If you update the JRE used by your application server in the future (such as applying an application server patch), before you start your application server again you must delete the `<JavaAgent_install_dir>/DiagnosticsAgent/classes/auto/Tibco_Node1` directory (use your directory name for `Tibco_Node1`) so that the new JRE will be instrumented. Otherwise, your application server may not start.

---

To instrument TIBCO servers you add two JVM parameters as shown in the following procedures for different TIBCO versions. The first parameter (`-javaagent`) causes the application server JVM to invoke the Java Agent at start-up. On the first invocation, the second parameter (`-Xbootclasspath`) causes the application server JRE to be instrumented. In the `-Xbootclasspath` parameter you enter a name to identify the server. In the following procedures you substitute a name you choose for the example name shown in bold.

### To configure TIBCO BusinessWorks:

For TIBCO BusinessWorks, append the two JVM parameters like the following example to the **`java.extended.properties`** in one of the following TIBCO BusinessWorks files depending on how you deployed the application and which applications you want to monitor. If **`java.extended.properties`** is not present, add it. One way of doing this is shown in the example below:

```
java.extended.properties=-javaagent:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar
-Xbootclasspath/p:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/classes/auto/TIBCO_BW1/instr.jre
```

If you want to monitor a previously deployed application or monitor only certain applications in TIBCO BusinessWorks, update `<BusinessWorks_install_dir>\tra\domain\<Domain_Name>\application\<Application_Name>\<Application_Name>.tra` with the JVM parameters. The JRE to instrument is specified in the .tra file user property `tibco.env.TIB_JAVA_HOME`.

- ▶ If you deployed the application using TIBCO Administrator. Update `bwengine.tra` with the JVM parameters.
- ▶ If you deployed the application using TIBCO Designer. Update `designer.tra` with the JVM parameters.

#### To configure TIBCO BusinessWorks for JMX Metrics Collection:

For TIBCO BusinessWorks JMX metrics collection, enable JMX access to the Business Works process. This is done by adding the following property to the same Business Works .tra file where the Java Agent instrumentation is configured.

**JmxEnabled=true**

Additional JMX metrics are exposed by certain components used by TIBCO, such as Apache Tomcat and Pramati J2EE server. Some of these metrics are not collected by default (these metrics are commented out in the `metric.config` file). These metrics can be activated by uncommenting them out in the `metrics.config` file. The reason to make them inactive is that the metrics represent mainly performance tuning configuration parameters and should rarely change during the lifetime of an application.

See “Java Agent - JMX Metrics Capture” on page 723 for general information on JMX metrics collection.

#### To configure TIBCO ActiveMatrix Service Bus 2.0 and 2.3:

- 1 For TIBCO ActiveMatrix Service Bus (AMSB) 2.0 and 2.3 locate the AMSB .tra file and append JVM parameters to `java.extended.properties`. If `java.extended.properties` does not exist, add it.

Below is an example on Windows that uses `TIBCO_Node1` as the name. Note the use of slash (/) instead of backward slash (\).

```
java.extended.properties=-javaagent:jarC:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar
-Xbootclasspath/p:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/classes/auto/TIBCO_Node1/instr:jre
```



- 2** In addition, in order to see outbound JMS web service operations in AMSB, you may need to update **details.conditional.properties** in **etc\inst.properties** with the correct version. Currently, there are 2 versions of AMSB supported, 2.0 and 2.3. Make sure only one version is enabled by setting it to 'true' and setting the other to 'false'. The example below shows the default version, AMSB version 2.3, enabled:

```
details.conditional.properties= \
mercury.enable.SOAPHandler=true, \
mercury.enable.autoLoadSOAPHandler=true, \
mercury.enable.resourcemonitor.jdbcConnection=false, \
mercury.enable.resourcemonitor.jdbcStatement=false, \
mercury.enable.resourcemonitor.jdbcResultSet=false, \
mercury.enable.tibco.amsb2.0=false, \
mercury.enable.tibco.amsb2.3=true
```

### To configure TIBCO ActiveMatrix Service Bus 3.1.2 Production Environment:

- 1** For TIBCO ActiveMatrix Service Bus (AMSB) 3.1.2 in a production environment, locate the following AMSB file:

```
<tibco_amx_configuration_dir>\data\tibcohost\<EnterpriseName_ServerName>\
tools\machinemodel\machine.xml
```

- 2** Update the **runtimes** section of the file **for each node** you want to monitor. For example:

```
<runtimes xsi:type="machinemodel:OSGiRuntime" name="Node1"
```

In the runtimes section for each node locate the **frameworkProperties** key **org.osgi.framework.bootdelegation** and append **com.mercury.\*** to the value of the property.

For example:

```
<frameworkProperties key="org.osgi.framework.bootdelegation"
value="com.ibm.*, ...,sun.*,com.mercury.*"/>
```

- 3** Then **for each node** locate the .tra file.

```
<tibco_amx_configuration_dir>\data\tibcohost\<EnterpriseName_ServerName>\
nodes\<NodeName>\bin\tibams_<NodeName>.tra
```

Append the two JVM parameters to the **java.extended.properties** in each file. Below is an example that uses **AMSB\_Node1** as the name.

```
java.extended.properties=-javaagent:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar
-Xbootclasspath/p:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/classes/auto/AMSB_Node1/instr.jre
```

**To configure TIBCO ActiveMatrix Service Bus 3.1.2 Development Environment:**

- 1 For TIBCO ActiveMatrix Service Bus (AMSB) 3.1.2 in a development environment you would update the boot delegation:

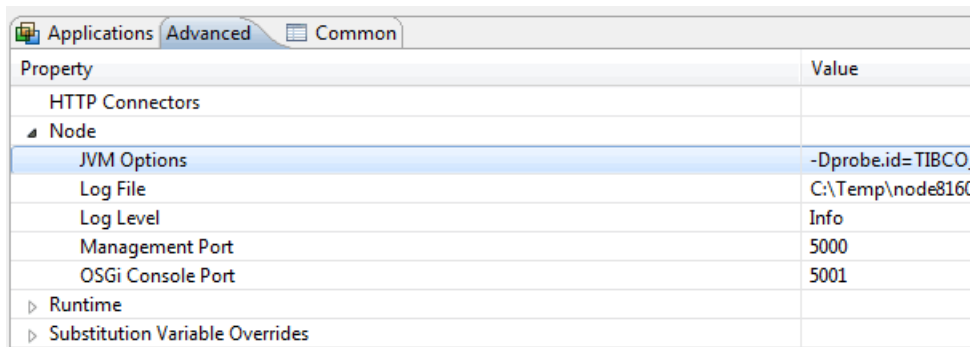
```
<tibco_amx_configuration_dir>\components\shared\1.0.0\plugins\com.tibco.metadata.h
pa.tibcohost.nodetype.integration_3.1.200.000\META-INF\com.tibco.amf.node.types\co
m.tibco.amf.hpa.tibcohost.node.integration.feature\3.1.200\provisioning.properties
```

And append **com.mercury.\*** to the end of **org.osgi.framework.bootdelegation**.

For example:

```
# OSGi framework properties
org.osgi.framework.bootdelegation=\
...
org.xml.*,\
sun.*,\
com.mercury.*
```

- 2 Then update the JVM Options in the **Advance** tab of Run Configuration to add the two JVM parameters.



Following is an example, which uses **AMSB\_DevNode** as the name.

```
-javaagent:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar
-Xbootclasspath/p:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/classes/auto/AMSB_DevNode/instr.jre
```

## Example 6: Configuring Tomcat

Apache Tomcat is frequently embedded into other applications or servers. As a result, the way to instrument it may vary. The following sections provide instructions on how to configure a Tomcat server in simple scenarios, but it is generic enough to guide you in your particular situation.

If your Tomcat server is started by a shell or Windows command script, it is recommended that you modify the startup script to instrument it, see “Configuring Tomcat Server with a Startup Script” on page 187.

In a Windows environment, if Tomcat is installed as a Windows service and has no scripts, we recommend that you modify its Java Options to instrument it. See “Configuring Tomcat Server without a Startup Script” on page 191.

## Configuring Tomcat Server with a Startup Script

Because the startup scripts that Tomcat provides are frequently customized by other applications or 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 with specific examples for a generic Tomcat server implementation. Your site administrator should be able to use these instructions to guide you to make these changes in your customized environment.

You use Automatic Explicit instrumentation mode to configure Tomcat server with a startup script (see “Using the JRE Instrumenter in Automatic Explicit Mode” on page 221). This involves two tasks:

- ▶ Modify your application server startup script to run the JRE Instrumenter using the same JRE used by your application server. The output from the JRE Instrumenter will give you the JVM parameters you will need in the next task.
- ▶ Configure your application server JVM parameters found in the output from the JRE Instrumenter.

**To configure a Tomcat server with a startup script:**

- 1 Locate the startup script that is used to start Tomcat for your application.

In some scenarios, the startup script will end up calling the following script to start Tomcat:

```
<Tomcat_install_dir>/bin/catalina.[bat|sh]
```

where **<Tomcat\_install\_dir>** is the path to your Tomcat installation directory, such as **C:\apache-tomcat-7.0.8**.

- 2 Create a backup copy of the startup script and use your editor to open the startup script.
- 3 Locate the **java** command line (or code block) that starts the Tomcat server.

Below is an example from the **catalina.bat** file:

```
rem Execute Java with the applicable properties
if not "%JPDA%" == "" goto doJpda
if not "%SECURITY_POLICY_FILE%" == "" goto doSecurity
%_EXECJAVA% %JAVA_OPTS% %CATALINA_OPTS% %DEBUG_OPTS%
-Djava.endorsed.dirs="%JAVA_ENDORSED_DIRS%" -classpath "%CLASSPATH%"
-Dcatalina.base="%CATALINA_BASE%" -Dcatalina.home="%CATALINA_HOME%"
-Djava.io.tmpdir="%CATALINA_TMPDIR%" %MAINCLASS% %CMD_LINE_ARGS%
%ACTION%
.....
```

Below is an example from the `catalina.sh` file:

```
if [ "$1" = "-security" ] ; then
  if [ $have_tty -eq 1 ]; then
    echo "Using Security Manager"
  fi
  shift
  eval \"$_RUNJAVA\" \"\$LOGGING_CONFIG\" $JAVA_OPTS $CATALINA_OPTS \
  -Djava.endorsed.dirs=\"\$JAVA_ENDORSED_DIRS\" -classpath \"\$CLASSPATH\" \
  -Djava.security.manager \
  -Djava.security.policy=\"\$CATALINA_BASE/conf/catalina.policy\" \
  -Dcatalina.base=\"\$CATALINA_BASE\" \
  -Dcatalina.home=\"\$CATALINA_HOME\" \
  -Djava.io.tmpdir=\"\$CATALINA_TMPDIR\" \
  org.apache.catalina.startup.Bootstrap \"$@" start
else
  eval \"$_RUNJAVA\" \"\$LOGGING_CONFIG\" $JAVA_OPTS $CATALINA_OPTS \
  -Djava.endorsed.dirs=\"\$JAVA_ENDORSED_DIRS\" -classpath \"\$CLASSPATH\" \
  -Dcatalina.base=\"\$CATALINA_BASE\" \
  -Dcatalina.home=\"\$CATALINA_HOME\" \
  -Djava.io.tmpdir=\"\$CATALINA_TMPDIR\" \
  org.apache.catalina.startup.Bootstrap \"$@" start
fi
```

- 4 Add two lines directly above the `java` command line (or code block). The first line invokes the `java` command to run the JRE Instrumenter; the second line adds the required JVM parameters. If you do not know what JVM parameters to use, you can add them later in step 6. In these two lines, you enter a name to specify the directory for storing the automatically instrumented JRE classes. In the following examples you would substitute a name you choose for **MyServer**.

Below is an example showing the added two lines in the `catalina.bat` file of a Tomcat server that uses JRE version 5 or higher:

```
%_EXECJAVA% -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f MyServer
set JAVA_OPTS=%JAVA_OPTS% -Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

Below is an example of the **catalina.sh** file of a Tomcat server that uses JRE version 5 or higher:

```
"$ _RUNJAVA" -jar /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/ jreinstrumenter.jar -f MyServer
JAVA_OPTS="$JAVA_OPTS -Xbootclasspath/p:/opt/MercuryDiagnostics/JavaAgent/ DiagnosticsAgent/classes/
MyServer/instr.jre -javaagent:/opt/MercuryDiagnostics/ JavaAgent/DiagnosticsAgent/lib/probeagent.jar"
```

Below is an example of the **catalina.bat** file of a Tomcat server that uses JRE 1.4:

```
%_EXECJAVA% -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f MyServer

set JAVA_OPTS=%JAVA_OPTS% -Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre;C:\Mercur
yDiagnostics\JavaAgent\DiagnosticsAgent\classes\boot
```

---

**Note:** If your java command line does not use the JAVA\_OPTS variable to define the JVM parameters, you need to change the variable name JAVA\_OPTS shown in these examples to the correct name.

---

- 5 Save the changes and restart the application server.
- 6 In the output from running the startup script, find the output from the JRE Instrumenter (search for Xbootclasspath). If you have added the second line (setting JVM parameters) to the startup script in Step 4, compare it with the JVM parameters from the JRE Instrumenter output. If they are not the same, update the startup script with the correct JVM parameters provided by the JRE Instrumenter and restart the application server. If you have not added the second line (setting JVM parameters) to the startup script in Step 4, add it now and restart the application server. See Step 4 for examples.
- 7 To verify that the probe was configured correctly, check for entries in the `<JavaAgent_install_dir>\log\<probe_id>\probe.log` file. If there are no entries in the file, either the JRE instrumentation did not succeed or you did not configure the JVM parameters correctly. For details, see “Using the JRE Instrumenter in Automatic Explicit Mode” on page 221.

## Configuring Tomcat Server without a Startup Script

The following instructions describe how to configure a Tomcat server running as a Windows service:

### To configure a Tomcat server without a startup script:

- 1** From the Windows Task bar, right-click on the Apache Tomcat service icon and then select **Configure**. Alternatively, you can navigate from the Start menu. For example, **Programs > Apache Tomcat 7.0 > Configure Tomcat**.
- 2** In the Apache Tomcat Properties dialog box, select the **Java** tab.
- 3** In the **Java Options** box, add two JVM parameters like the following. The first parameter (-javaagent) causes the application server JVM to invoke the Java Agent at startup. On the first invocation, the second parameter (-Xbootclasspath) causes the application server JRE to be instrumented. In the -Xbootclasspath parameter enter a name to specify the name of the directory for storing the instrumented classes. In the following example you would substitute a name you've chosen instead of **MyServer**.

```
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\auto\MyServer\instr.jre
```

---

**Important:** Each JVM parameter must be on its own line.

---

- 4** Restart the Tomcat service.

The Java Agent will be invoked and implicitly run the JRE Instrumenter to instrument the JRE.

- 5** To verify that the probe was configured correctly, check for entries in the `<JavaAgent_install_dir>/DiagnosticsAgent/log/<probe_id>/probe.log` file. If there are no entries in the file, you may not have set the JVM parameters correctly. Look for error messages in the `<Tomcat_install_dir>/logs/catalina.<date>.log` file, where `<date>` is today's date.
- 6** Optionally, restart the application server again so that it will use the instrumented JRE.

**Important:** If you update the JRE used by your Tomcat server in the future, before you start the Tomcat server again, you must delete the `<JavaAgent_install_dir>/DiagnosticsAgent/classes/auto/MyServer` directory (use your directory name for MyServer) so that the new JRE will be instrumented. Otherwise, your application server may not start. For general information on the instrumentation mode used see “Using the JRE Instrumenter in Automatic Implicit Mode” on page 224.

---

### **Example 7: Configuring WebLogic**

WebLogic application servers are started by shell or command scripts. Therefore, we recommend that you modify the startup scripts to instrument them.

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 section provides general instructions with specific examples for 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.

You use Automatic Explicit instrumentation mode to configure WebLogic server (see “Using the JRE Instrumenter in Automatic Explicit Mode” on page 221). This involves two tasks:

- ▶ Modify your application server startup script to run the JRE Instrumenter using the same JRE used by your application server. The output from the JRE Instrumenter will give you the JVM parameters you will need in the next task.
- ▶ Configure your application server JVM parameters found in the output from the JRE Instrumenter.

**To configure a WebLogic application server:**

- 1** Locate the startup script used to start WebLogic for your domain.



- On WebLogic 9.0 or higher:

The startup script file is typically located in a path similar to the following:

```
<DOMAIN_HOME>\bin\startWebLogic.[cmd|sh]
```

where <DOMAIN\_HOME> is the path to your domain directory, such as **C:\bea\user\_projects\domains\<Domain\_Name>**; or **C:\bea\wlserver\_10.0\samples\domains\<Domain\_Name>** where <Domain\_Name> is the name of your domain.

For example, if your domain name is **MedRec**, the path would look like the following:

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

- On WebLogic 8.1:

The startup script file is typically located in a path similar to one of the following:

```
<WLS_HOME>\server\bin\startWLS.[cmd|sh]
```

where <WLS\_HOME> is the path to your WebLogic installation directory such as **C:\bea\weblogic81**

```
<DOMAIN_HOME>\start<Domain_Name>Server.[cmd|sh]
```

where <DOMAIN\_HOME> is the path to your domain directory, such as **C:\bea\user\_projects\domains\<Domain\_Name>** or **C:\bea\weblogic81\samples\domains\<Domain\_Name>**, where <Domain\_Name> is the name of your domain.

For example, if your domain name is **MedRec**, the path would look like the following:

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

- 2 Create a backup copy of the startup script and use your editor to open the startup script.

- 3 Locate the **java** command line that starts the application server. Below is an example from a `.cmd` file:

```
%JAVA_HOME%\bin\java %JAVA_VM% %MEM_ARGS% %JAVA_OPTIONS%  
-Dweblogic.Name=%SERVER_NAME%  
-Djava.security.policy=%WL_HOME%\server\lib\weblogic.policy  
%PROXY_SETTINGS% %SERVER_CLASS%
```

- 4 Add two lines directly above the `java` command line (or code block). The first line invokes the `java` command to run the JRE Instrumenter; the second line adds the required JVM parameters. If you do not know what JVM parameters to use, you can add them later in step 6. In these two lines, you enter a name to specify the directory for storing the automatically instrumented JRE classes. In the following examples you would substitute a name you choose for **MedRec**.

Below is an example showing the added two lines in a `.cmd` file in WebLogic 9.x or higher:

```
%JAVA_HOME%\bin\java -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f  
MedRec  
  
set JAVA_OPTIONS=%JAVA_OPTIONS% -Xbootclasspath/  
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MedRec\instr.jre  
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

Below is an example showing the added two lines in a `.sh` file in WebLogic 9.x or higher:

```
`${JAVA_HOME}/bin/java -jar /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/jreinstrumenter.jar -f  
MedRec  
  
JAVA_OPTIONS="${JAVA_OPTIONS} -Xbootclasspath/p:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/  
classes/MedRec/instr.jre -javaagent:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar"
```

Below is an example showing the added two lines in a `.cmd` file for WebLogic 8.1:

```
%JAVA_HOME%\bin\java -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f  
MedRec  
  
set JAVA_OPTIONS=%JAVA_OPTIONS% -Xbootclasspath/  
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MedRec\instr.jre;C:\MercuryDiagnostics\JavaAg  
ent\DiagnosticsAgent\classes\boot
```

**Note:**

- ▶ If your java command line does not use the JAVA\_OPTIONS variable to define the JVM parameters, you need to change the variable name JAVA\_OPTIONS shown in these examples to the correct name.
  - ▶ On WebLogic 8.1 with the JRockit JRE, add the following JVM parameter to the end of the second line:  
-Xmanagement:class=com.mercury.opal.capture.proxy.JRockitManagement
- 

- 5 Save the changes to the startup script and restart the application server using the modified script.
  - 6 In the output from running the startup script, find the output from the JRE Instrumenter (search for Xbootclasspath). If you have added the second line (setting JVM parameters) to the startup script in Step 4, compare it with the JVM parameters from the JRE Instrumenter output. If they are not the same, update the startup script with the correct JVM parameters provided by the JRE Instrumenter and restart the application server. If you have not added the second line (setting JVM parameters) to the startup script in Step 4, add it now and restart the application server. See Step 4 for examples.
- 

**Note:** If you cannot find the JRE Instrumenter output, or if there are error messages, the JRE may not be properly instrumented. You should check the startup script and resolve any issues.

---

- 7 To verify that the probe was configured correctly, check for entries in the `<JavaAgent_install_dir>\DiagnosticsAgent\log\<probe_id>\probe.log` file. If there are no entries in the file, either the JRE instrumentation did not succeed or you did not configure the JVM parameters correctly. For details, see “Using the JRE Instrumenter in Automatic Explicit Mode” on page 221.

## Example 8: Configuring WebSphere

WebSphere application servers are started by a shell script or by a Node Agent in a UNIX environment. In a Windows environment, the application server may be installed as a Window service, but can also be started by a Windows command script. In either case, we recommend that you modify the startup script to run the JRE Instrumenter in the Automatic Explicit mode to instrument the JRE used by the application server.

---

**Note:** If you do not modify the startup script, or if your WebSphere application server is running on z/OS, you have to choose one of the following choices:

- ▶ You manually run the JRE Instrumenter to instrument the WebSphere JRE and add the JVM parameters (provided by running the JRE Instrumenter) into the application server JVM configuration. See “Using the JRE Instrumenter in Manual Mode” on page 226 for details. In addition, if your WebSphere is version 6.0, which uses a 1.4.2 JRE, you also need to add the `-Xj9` parameter in addition to the JVM parameters provided by the JRE Instrumenter.
- ▶ Alternatively, if your WebSphere version is 7.0 or higher, you can use the JRE Instrumenter in the Automatic Implicit mode. In this approach, you only need to add JVM parameters to the application server JVM configuration. See “Using the JRE Instrumenter in Automatic Implicit Mode” on page 224 for details.

---

Because the WebSphere Application Server JVM parameters are not controlled by the startup scripts, but by configuration files, we recommend that you use the Integrated Solutions Console (also called WebSphere Application Server Administrative Console in older versions) to add the JVM parameters required to invoke the Java Agent and use the instrumented JRE.

The appearance of the Console can differ for different versions of WebSphere. As a result, the following example might not correspond exactly to your WebSphere version but does provide the information needed. Enter the required parameters for monitoring by Diagnostics in the appropriate location in the Console.

Procedures are provided for WebSphere 5.1/6.0 (shown below) and for WebSphere 6.1 or higher (see “To configure WebSphere 6.1 or higher:” on page 205). Also see “To configure WebSphere 6.1/7.0 server for JMX metrics collection:” on page 211.

### To configure WebSphere 5.1 or 6.0:

- 1 Locate the script that is used to start the WebSphere application server.

For example, `<WAS_install_dir>\bin\startServer.bat`

where `<WAS_install_dir>` is the path to your WebSphere installation directory, such as `C:\Program Files\IBM\WebSphere\AppServer`.

---

**Note:** On some systems, you may use the `startServer.[bat|sh]` script in a profile's bin directory to start a server. However, this script is usually a simple wrapper that calls the `startServer.[bat|sh]` script in the `<WAS_install_dir>/bin` directory.

---

- 2 Create a backup copy of the startup script and use your editor to open the startup script.
- 3 For WebSphere 5.1 or 6.0, locate the java command line that runs the application server launcher. Below is an example from a `startServer.bat` file:

```

"%JAVA_HOME%\bin\java" -Dcmd.properties.file=%TMPJAVAPROFILE%
%WAS_TRACE% %WAS_DEBUG% %CONSOLE_ENCODING% "%CLIENTSAS%"
"%CLIENTSSL%" %USER_INSTALL_PROP% "-Dwas.install.root=%WAS_HOME%"
com.ibm.ws.bootstrap.WSLauncher
com.ibm.ws.management.tools.WsServerLauncher "%CONFIG_ROOT%"
"%WAS_CELL%" "%WAS_NODE%" %* %WORKSPACE_ROOT_PROP%

```

Above the identified java command line, add a line to invoke the JRE Instrumenter. In this line, you need to specify the name of the directory for storing the instrumented classes. Below is an example for WebSphere 6.0 on Windows. You would substitute a name you've chosen instead of **MyServer**.

```

"%JAVA_HOME%\bin\java" -jar
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f MyServer

```

- 4 Save the changes and restart the application server using the modified script.
- 5 In the output from running the startup script, find the output from the JRE Instrumenter (search for Xbootclasspath).

Below is an example JRE Instrumenter output (-Xbootclasspath) for WebSphere 6.0 (or 5.1) on Windows using **-f MyServer** to specify the directory for storing the instrumented classes - see step 3 above. You would substitute a name you choose for MyServer.

```
-Xbootclasspath/  
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre;C:\MercuryDiagnostics\JavaA  
gent\DiagnosticsAgent\classes\boot
```

If you cannot find the JRE Instrumenter output, or if there are error messages, the JRE may not be properly instrumented. You should check the startup script and resolve issues. For details, see “Using the JRE Instrumenter in Automatic Explicit Mode” on page 221.

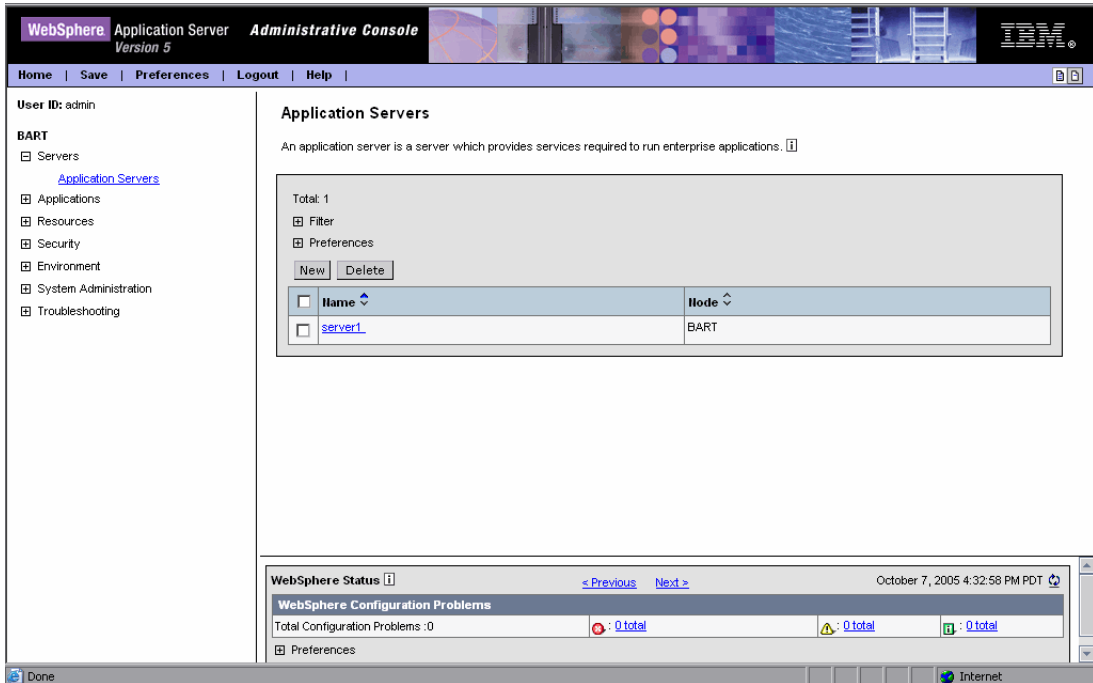
After you get the output from the JRE Instrumenter, you need to add it to the application server JVM parameters.

- 6 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 and possibly 9090 with the correct administrative port number (such as 9060, 9061, and so on).

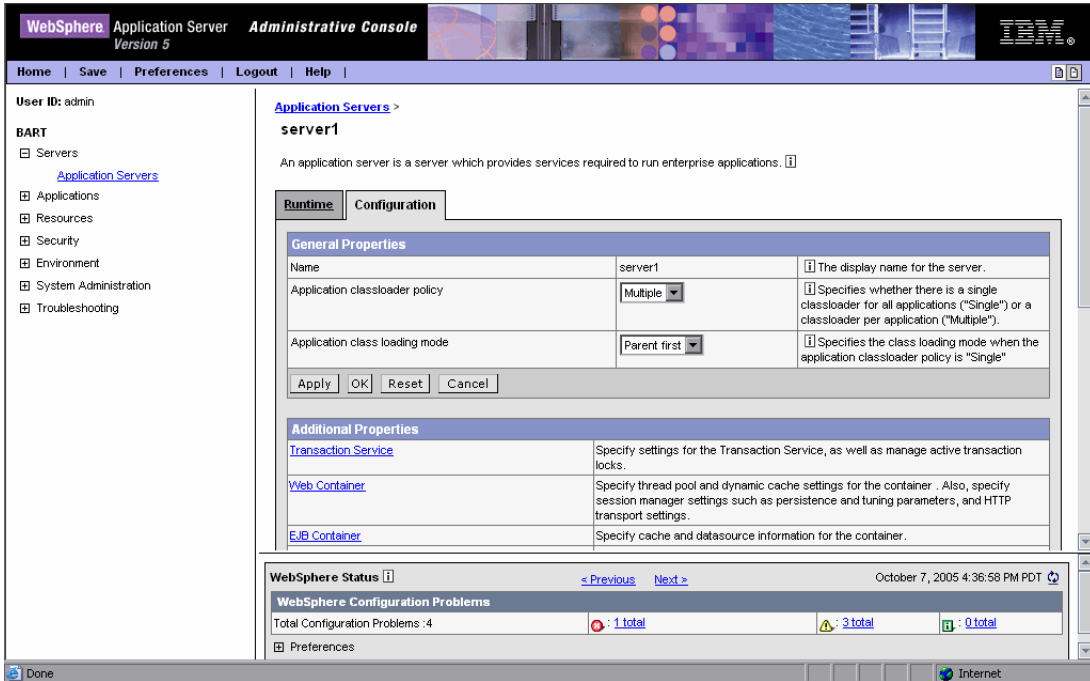
The WebSphere Application Server Administrative Console opens.



**7** In the left panel, select **Servers > Application Servers**.

**8** From the list of application servers in the right panel, select the name of the server that you want to configure to be monitored by the probe.

The Configuration tab for the selected application server is displayed.



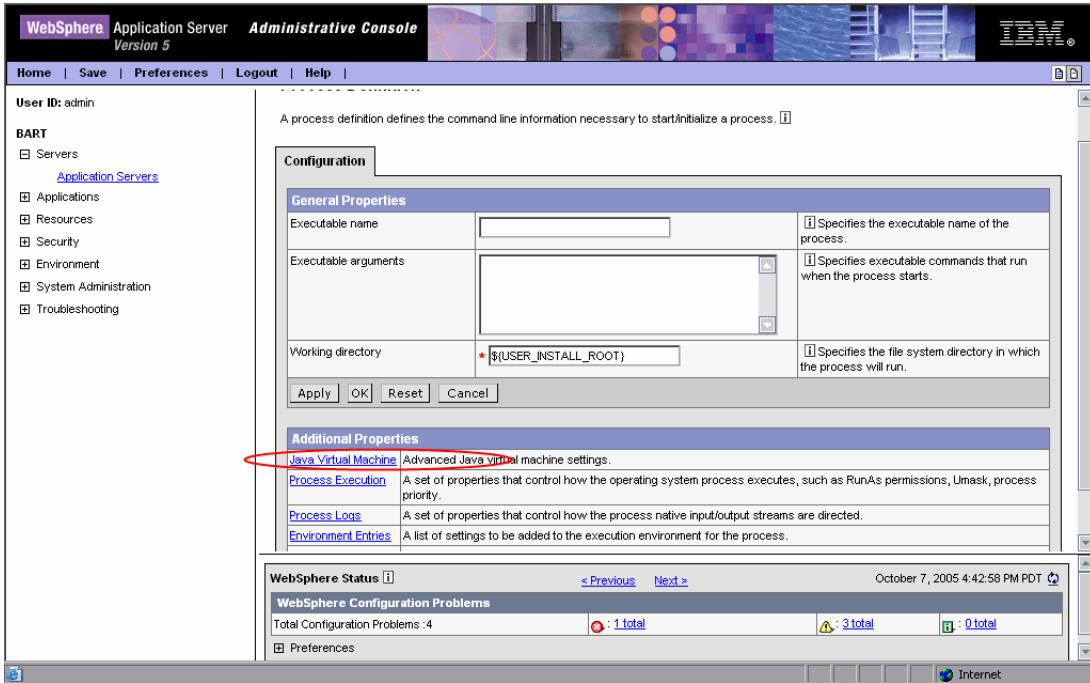


9 Scroll down in the Configuration tab and, in the **General Properties** column, look for the **Process Definition** property.



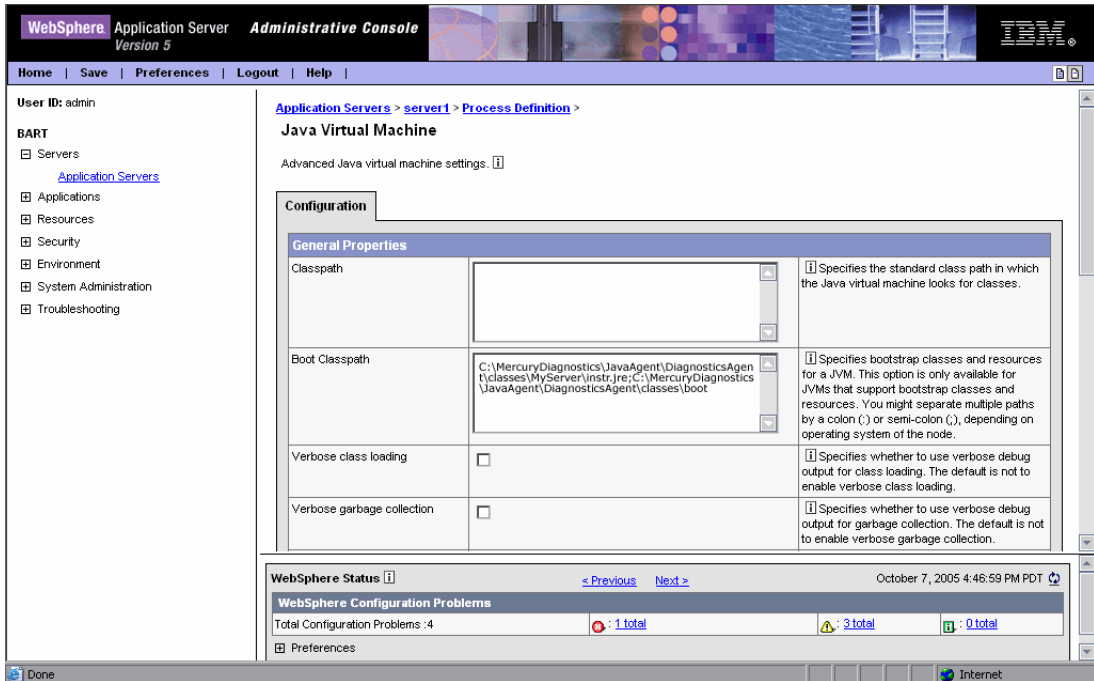
10 Click **Process Definition**.

**11** Scroll down in the right panel, and look for **Java Virtual Machine**.



**12** Click **Java Virtual Machine**.

### 13 The Configuration tab for the Java Virtual Machine is displayed.

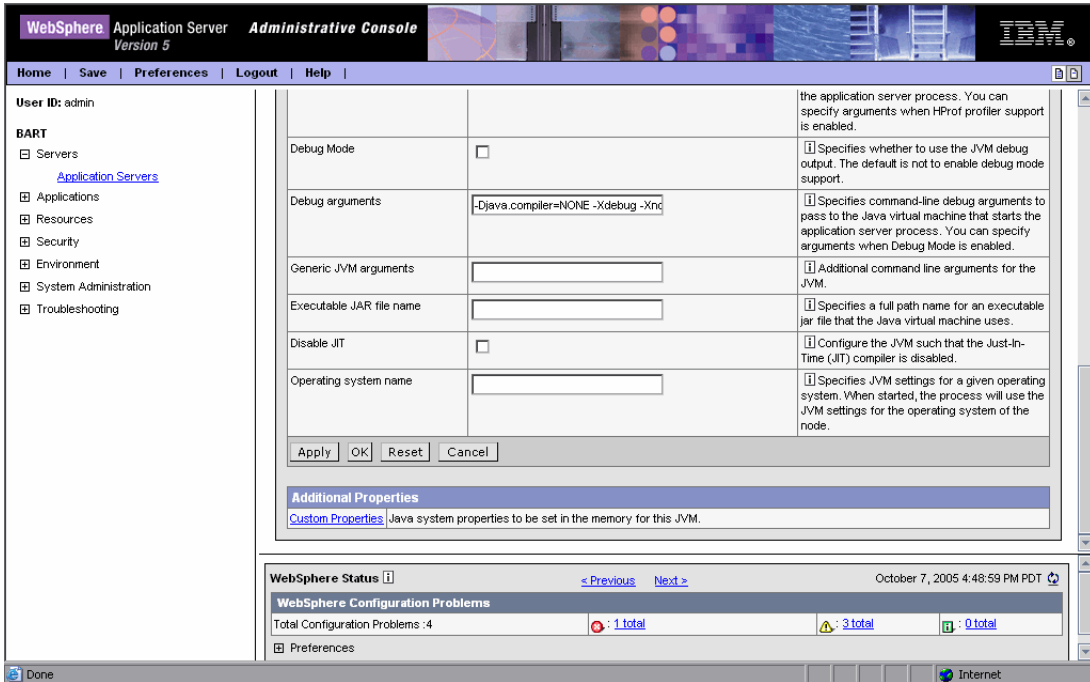


- 14** In the **Boot Classpath** box, enter the boot class path from the JVM parameters provided by the JRE Instrumenter (that is, the string after the `-Xbootclasspath/p:` parameter). The JVM parameters output from the JRE Instrumenter were generated earlier in step 3.

Below is an example for WebSphere 6.0 (or 5.1) that uses `-f MyServer` to specify the name of the directory for storing the instrumented classes:

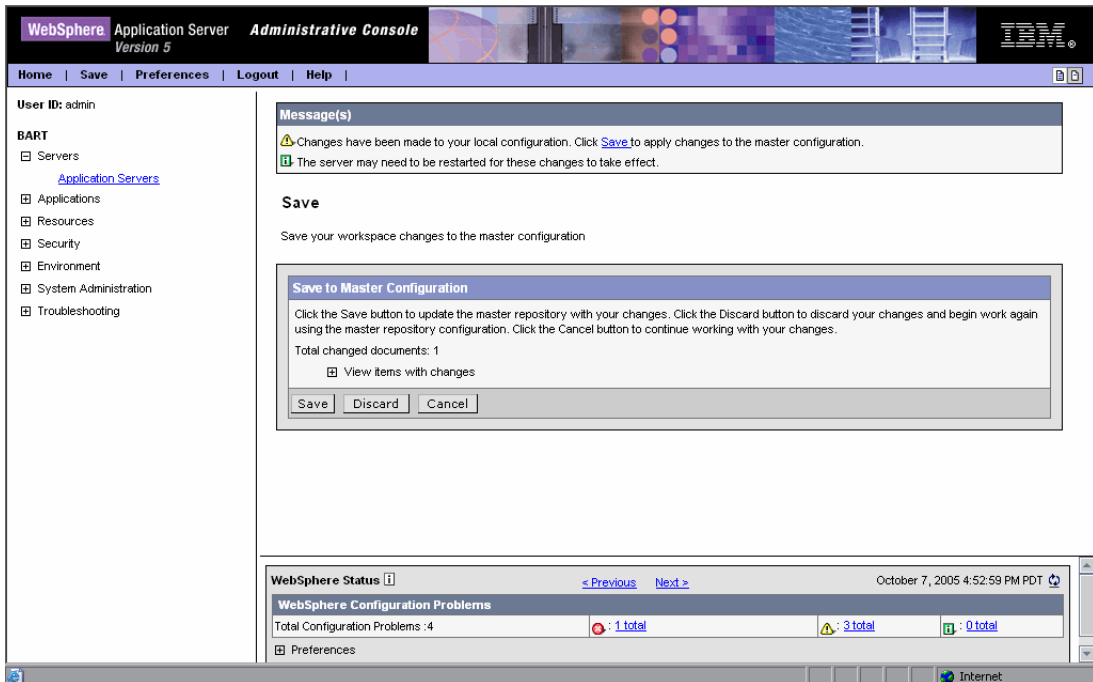
```
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre;C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\boot
```

**15** Scroll to the bottom of the Configuration tab until the command buttons are visible.



Click **Apply**.

- 16** A message confirms that your changes were applied. In the **Save to Master Configuration** area, click **Save**.



- 17** Click **Save** to apply the changes to the master configuration. If you are prompted for confirmation, click **Save** again.
- 18** Restart the WebSphere application server.
- 19** To verify that the probe was configured correctly, check for entries in the `<JavaAgent_install_dir>\DiagnosticsAgent\log\<probe_id>\probe.log` file. If there are no entries in the file, the Java Agent was not started correctly or you did not run the JRE Instrumenter or you did not enter the Java parameter such as `Xbootclasspath` correctly. For details, see “About the JRE Instrumenter and Different Options to Invoke” on page 219.

#### To configure WebSphere 6.1 or higher:

- 1** Locate the script that is used to start the WebSphere application server. For example, `<WAS_install_dir>\bin\startServer.bat`

where `<WAS_install_dir>` is the path to your WebSphere installation directory, such as `C:\Program Files\IBM\WebSphere\AppServer`.

---

**Note:** On some systems, you may use the `startServer.[bat|sh]` script in a profile's bin directory to start a server. However, this script is usually a simple wrapper that calls the `startServer.[bat|sh]` script in the `<WAS_install_dir>/bin` directory.

---

- 2 Create a backup copy of the startup script and use your editor to open the startup script.
- 3 For WebSphere 6.1 or higher, locate the code block that defines the `JAVA_EXE` variable.

```
if exist "%JAVA_HOME%\bin\java.exe" (  
    set JAVA_EXE="%JAVA_HOME%\bin\java"  
) else (  
    set JAVA_EXE="%JAVA_HOME%\jre\bin\java"  
)
```

Below the above code block, add a line to invoke the JRE Instrumenter. In this line, you need to specify the name of the directory for storing the instrumented classes. Below is an example for WebSphere 6.1 on Windows. You would substitute a name you've chosen instead of **MyServer**.

```
%JAVA_EXE% -jar  
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f MyServer
```

- 4 Save the changes and restart the application server using the modified script.
- 5 In the output from running the startup script, find the output from the JRE Instrumenter (search for `Xbootclasspath`).

**WebSphere 6.1.** Below is an example JRE Instrumenter output (-Xbootclasspath) for WebSphere 6.1 on Windows (using **-f MyServer** to specify the directory for storing the instrumented classes - see step 3 above. You would substitute a name you choose for MyServer.

```
-Xbootclasspath/  
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre;C:\MercuryDiagnostics\JavaA  
gent\DiagnosticsAgent\classes\boot -Xshareclasses:none
```

**WebSphere 7.0 or 8.0.** Below is an example JRE Instrumenter output (-Xbootclasspath) for WebSphere 7.0 (or 8.0) on Windows (using **-f MyServer** to specify the directory for storing the instrumented classes - see step 3 above. You would substitute a name you choose for MyServer.

```
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre  
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar -Xshareclasses:none
```

If you cannot find the JRE Instrumenter output, or if there are error messages, the JRE may not be properly instrumented. You should check the startup script and resolve issues. For details, see “Using the JRE Instrumenter in Automatic Explicit Mode” on page 221.

After you get the output from the JRE Instrumenter, you need to add it to the application server JVM parameters.

- 6 Open the WebSphere Application Server Administrative Console. For example:

[http://<App\\_Server\\_Host>:9060/ibm/console](http://<App_Server_Host>:9060/ibm/console)

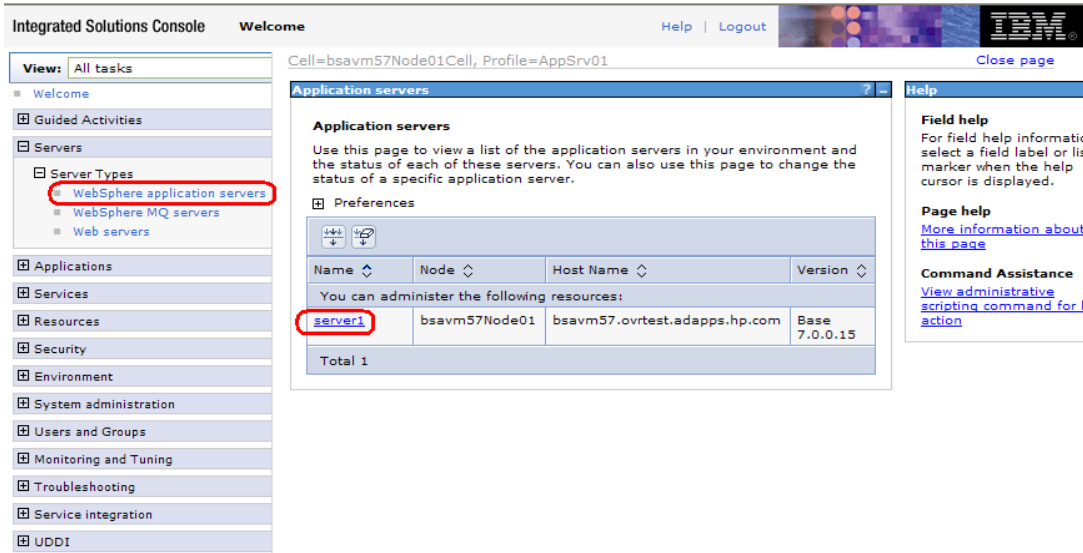
Replace **<App\_Server\_Host>** with the machine name for the application server host and 9060 with the correct administrative port number (such as 9060, 9061, and so on).

- 7 Navigate to the Java Virtual Machine page. For example:

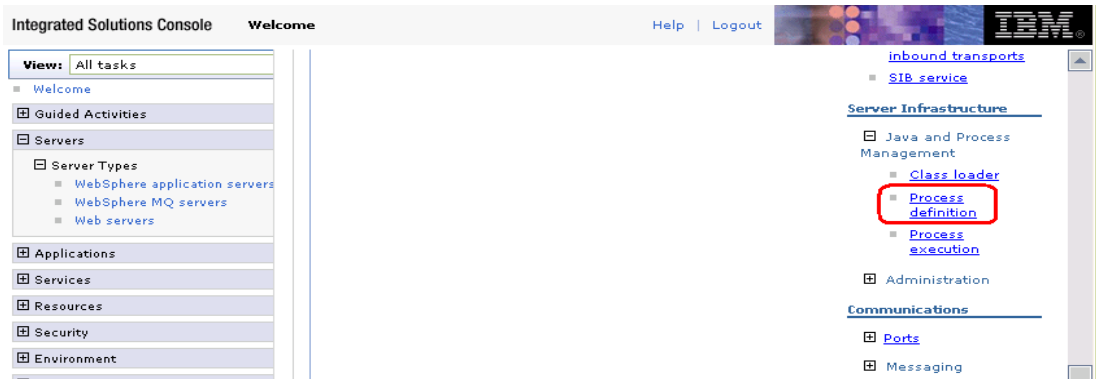
For WebSphere 6.1, navigate to: **Servers > Application servers**

For WebSphere 7.0, navigate to: **Servers > Server Types > WebSphere Application servers**

Then click the application server instance name (such as **server1**).

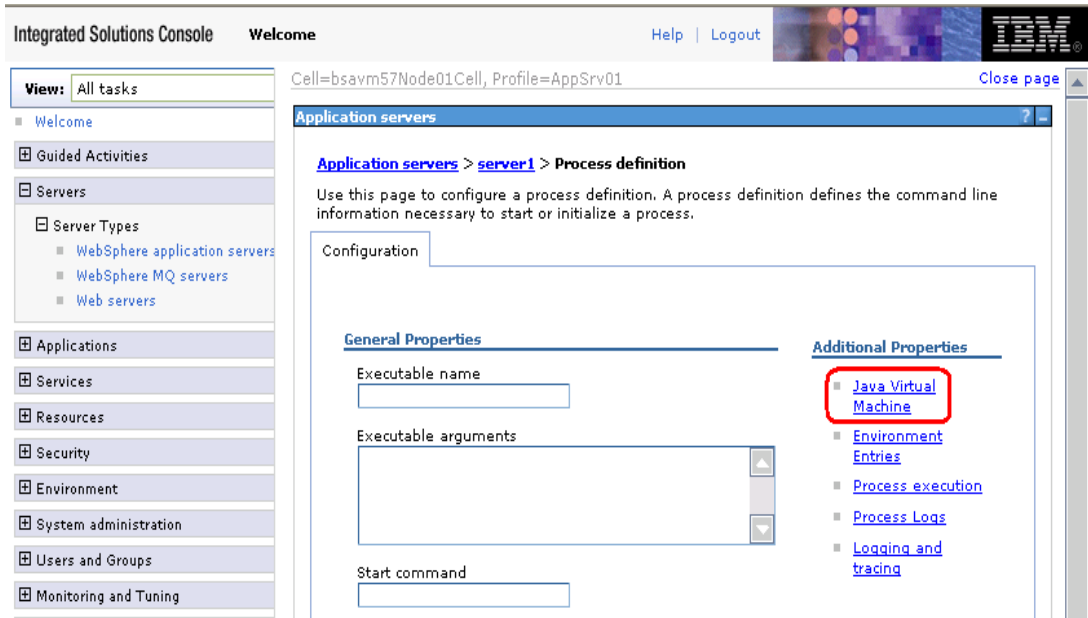


Then, under **Server Infrastructure > Java and Process Management**, click **Process Definition > Java Virtual Machine**.





Then, under **Additional Properties**, click **Java Virtual Machine**.



- 8** On the Java Virtual Machine page, in the **Generic JVM Arguments** box, enter the JVM parameter from the JRE instrumenter. The JVM parameters output from the JRE Instrumenter were generated earlier in step 3.

Below is an example for WebSphere 6.1 that uses **-f MyServer** to specify the name of the directory for storing the instrumented classes:

```
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre;  
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\boot -Xshareclasses:none
```

Below is an example for WebSphere 6.1 that does not modify the startup script but manually uses the JRE Instrumenter to instrument the JRE.

```
-Xbootclasspath/  
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\IBM\1.5.0\instr.jre;C:\MercuryDiagnostics\JavaA  
gent\DiagnosticsAgent\classes\boot -Xshareclasses:none
```

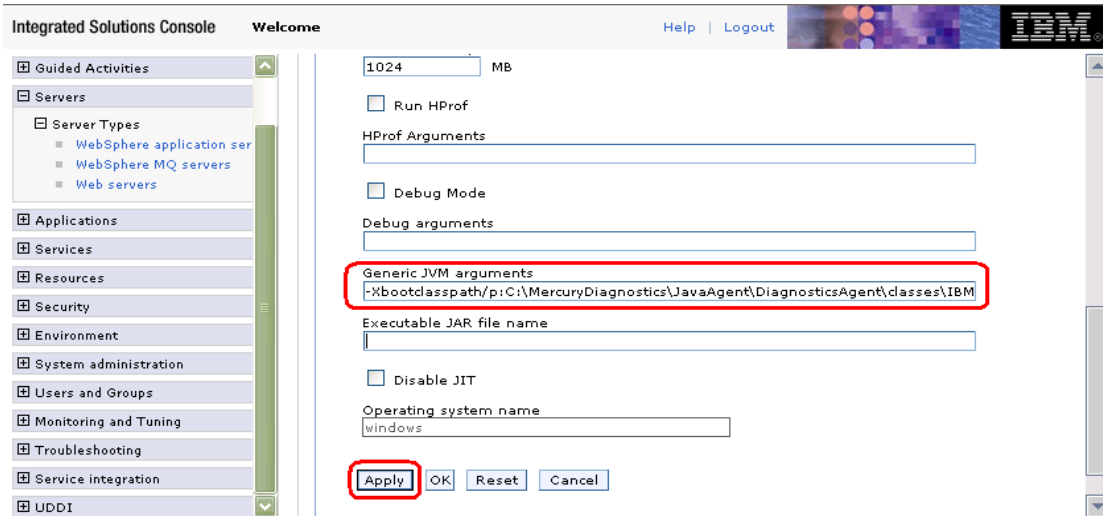
Below is an example for WebSphere 7 (or 8) that uses **-f MyServer** as the command-line option to the JRE Instrumenter in the startup script:

```
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre  
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar -Xshareclasses:none
```

Below is an example for WebSphere 7 (or 8) that does not modify the startup script or manually run the JRE Instrumenter (using the JRE Instrumenter in the Automatic Implicit mode):

```
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\auto\server1\instr.jre  
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar -Xshareclasses:none
```

9 Apply and save your changes.



10 Restart the WebSphere application server.

11 To verify that the probe was configured correctly, check for entries in the `<JavaAgent_install_dir>/DiagnosticsAgent/log/<probe_id>/probe.log` file. If there are no entries in the file, the Java Agent was not started correctly or you did not run the JRE Instrumenter or you did not enter the Java parameter such as Xbootclasspath correctly. For details, see “About the JRE Instrumenter and Different Options to Invoke” on page 219.

## 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 different java.exe executables to choose from. You must instrument the one that is used to run WebSphere.

### To instrument the correct java.exe:

- 1 Determine the version of WebSphere you are using.
- 2 Determine the location of the appropriate java.exe.
- 3 Run the JRE Instrumenter and add the correct JVM. See “About the JRE Instrumenter and Different Options to Invoke” on page 219 for details.

## Configuring WebSphere for JMX Metric Collection

You might need to configure the Performance Monitoring Infrastructure (PMI) service 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.

---

### To configure WebSphere 6.1/7.0 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;
};
```

**8** Restart the application server.

### **Example 9: Configuring webMethods**

There are two types of webMethods servers discussed in this example:

- webMethods Integration Server
- My webMethods Server

Because the startup scripts that webMethods 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 section provides general instructions with specific examples for webMethods Integration Server and My webMethods Server. Your site administrator should be able to use these instructions to show you how to make these changes in your customized environment.

**To configure a webMethods Integration Server:**

The webMethods Integration Server is started by shell or command scripts. Therefore, we recommend that you modify the startup scripts to instrument the server.

- 1 Locate the startup script used to start the webMethods Integration Server. There are two options based on how the server is started:

```
... \IntegrationServer\bin\server.bat
```

```
... \profiles\IS\bin\runtime.bat
```

- 2 Create a backup copy of the startup script and use your editor to open the startup script.
- 3 Update the file as described below.

- a For the **server.bat** file locate the following section where the server is started:

```
if "%1"=="1-service" (
    if exist LOCKFILE del LOCKFILE
    "%JAVA_EXEC%" -classpath %IS_PROXY_JAR%
    com.wm.app.server.CustomServiceUpdater -isdir "%IS_DIR%" -wrapperdir
    "%IS_DIR%\..\profiles\IS\configuration" -binpath "%PATH%" -jvmargs
    "%SERVER_VM_OPT% %JAVA2_MEMSET% %JAVA_OPTS%" -progargs
    %3#%4#%5#%6#%7#%8#%9
    goto :EOF
)

call "%PROFILES_DIR%\bin\start_runtime.bat" %1 %2 %3 %4 %5 %6 %7 %8 %9
```

And directly above this section add the following:

```
%JAVA_HOME%\bin\java -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\reinstrumenter.jar -f
MyServer

set JAVA_OPTIONS=%JAVA_OPTIONS%
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

The first line invokes the java command to run the JRE Instrumenter; the second line adds the required JVM parameters. If you do not know what JVM parameters to use for the second line, you can add them later in step 5. In these two lines, you enter a name to specify the directory for storing the automatically instrumented JRE classes. In the example above you would substitute a name you choose for **MyServer**.

- b** For the **runtime.bat** file locate the following section where the server is started:

```
%JAVA_RUN% -Xbootclasspath/a:"%OSGI_CLASSPATH%" %JAVA_OPTS%
%JAVA_SYSPROPS% -cp "%OSGI_FRAMEWORK_JAR%"
org.eclipse.equinox.launcher.Main -configuration %OSGI_CONFIGURATION_AREA%
%CMD_ARGS%
goto end_start_cmd
```

And directly above this section add the two lines as shown in the example :

```
%JAVA_HOME%\bin\java -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f
MyServer

set JAVA_OPTIONS=%JAVA_OPTIONS%
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

The first line invokes the java command to run the JRE Instrumenter; the second line adds the required JVM parameters. If you do not know what JVM parameters to use for the second line, you can add them later in step 5. In these two lines, you enter a name to specify the directory for storing the automatically instrumented JRE classes. In the example above you would substitute a name you choose for **MyServer**.

- 4** Save the changes to the startup script and restart the application server using the modified script.
- 5** In the output from running the startup script, find the output from the JRE Instrumenter (search for Xbootclasspath). If you have added the second line (setting JVM parameters) to the startup script in Step 3, compare it with the JVM parameters from the JRE Instrumenter output. If they are not the same, update the startup script with the correct JVM parameters provided by the JRE Instrumenter and restart the application server. If you have not added the second line (setting JVM parameters) to the startup script in Step 3, add it now and restart the application server.

---

**Note:** If you cannot find the JRE Instrumenter output, or if there are error messages, the JRE may not be properly instrumented. You should check the startup script and resolve any issues.

---

- 6 To verify that the probe was configured correctly, check for entries in the `<JavaAgent_install_dir>\DiagnosticsAgent\log\<probe_id>\probe.log` file. If there are no entries in the file, either the JRE instrumentation did not succeed or you did not configure the JVM parameters correctly. For details, see “Using the JRE Instrumenter in Automatic Explicit Mode” on page 221.

**To configure the My webMethods Server startup script:**

The My webMethods Server is started either by script or by a wrapper configuration. Therefore you will either modify the startup script, as in this example, or edit the wrapper configuration file to instrument the server as described in the next procedure.

- 1 Locate the startup script used to start your My webMethods Server. The script file is: `... \MWS\server\bin\mws.bat`.
- 2 Create a backup copy of the file and use your editor to open the file.
- 3 Update the file as described below.

For the **mws.bat** file, you locate the definition of `RUN_CMD` as highlighted in the following example:

```
set JAVA_OPTIONS=%JAVA_OPTIONS% -Dserver.name=%SERVER_NAME%
-Djava.awt.headless=true
set PARAMS=
set MAIN_CLASS=com.webmethods.portal.system.PortalSystem
set RUN_CMD=%JAVA% -cp %CLASSPATH% %JAVA_ARGS% %JAVA_OPTIONS%
%ACTION_PARAMS% -Dmain.class=%MAIN_CLASS%
7 %8 %9
```

And above this section add the two lines as shown in the following example:

```
%JAVA_HOME%\bin\java -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f
MyServer

set JAVA_OPTIONS=%JAVA_OPTIONS%
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

The first line invokes the java command to run the JRE Instrumenter; the second line adds the required JVM parameters. If you do not know what JVM parameters to use for the second line, you can add them later in step 5. In these two lines, you enter a name to specify the directory for storing the automatically instrumented JRE classes. In the example above you would substitute a name you choose for **MyServer**.

- 4 Save the changes to the startup script and restart the application server using the modified script.
- 5 In the output from running the startup script, find the output from the JRE Instrumenter (search for Xbootclasspath). If you have added the second line (setting JVM parameters) to the startup script in Step 3, compare it with the JVM parameters from the JRE Instrumenter output. If they are not the same, update the startup script with the correct JVM parameters provided by the JRE Instrumenter and restart the application server. If you have not added the second line (setting JVM parameters) to the startup script in Step 3, add it now and restart the application server.

---

**Note:** If you cannot find the JRE Instrumenter output, or if there are error messages, the JRE may not be properly instrumented. You should check the startup script and resolve any issues.

---

- 6 To verify that the probe was configured correctly, check for entries in the `<JavaAgent_install_dir>\DiagnosticsAgent\log\<probe_id>\probe.log` file. If there are no entries in the file, either the JRE instrumentation did not succeed or you did not configure the JVM parameters correctly. For details, see “Using the JRE Instrumenter in Automatic Explicit Mode” on page 221.



**To configure the My webMethods Server configuration wrapper:**

The My webMethods Server is started either by script or by a wrapper configuration. Therefore you will either modify the startup script or edit the wrapper configuration file, as in this example, to instrument the server.

- 1 Locate the configuration wrapper used to start your My webMethods Server. The configuration file is:  
...MWS\server\\config\wrapper.conf
- 2 Create a backup copy of the file and use your editor to open the file.
- 3 Update the file as described below.

For the **wrapper.conf** file add the following (changing the numbers 270 and 280 depending on your configuration file::

```
wrapper.java.additional.270=-Xbootclasspath/p:  
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\auto\MyServer\instr.jre  
  
wrapper.java.additional.280=-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

The first line invokes the java command to run the JRE Instrumenter; the second line adds the required JVM parameters. On the first invocation, the second parameter (-Xbootclasspath) causes the application server JRE to be instrumented. In the -Xbootclasspath parameter enter a name to specify the name of the directory for storing the instrumented classes. In the example above you would substitute a name you choose for **MyServer**.

- 4 Save the changes to the configuration wrapper and restart the application server using the modified wrapper.

The Java Agent will be invoked and implicitly run the JRE Instrumenter to instrument the JRE.

- 5 To verify that the probe was configured correctly, check for entries in the <JavaAgent\_install\_dir>/DiagnosticsAgent/log/<probe\_id>/probe.log file. If there are no entries in the file, you may not set the JVM parameters correctly.
- 6 Optionally, restart the application server again so that it will use the instrumented JRE.

**Important:** If you update the JRE used by your My webMethods Server when started with the configuration wrapper in the future, before you start the My webMethods Server again, you must delete the `<JavaAgent_install_dir>/DiagnosticsAgent/classes/auto/MyServer` directory (use your directory name for MyServer) so that the new JRE will be instrumented. Otherwise, your application server may not start. For general information on the instrumentation mode used see “Using the JRE Instrumenter in Automatic Implicit Mode” on page 224.

---

## About the JRE Instrumenter and Different Options to Invoke

The JRE Instrumenter is a utility to instrument a JRE so that the Java Agent can provide advanced features such as the patent-pending Collection Leak Pinpointing (CLP). It does not modify the installed JRE in any way, but rather places copies of instrumented classes somewhere under the `<JavaAgent_install_dir>/DiagnosticsAgent/classes` directory. You can use the JRE Instrumenter to instrument multiple JREs if they are installed on your system.

The JRE Instrumenter instruments some standard Java classes used by the application server JVM and applications running on it. 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.

With different command-line options, the JRE Instrumenter can be invoked and used in three different ways, each of which has its own advantages and limitations. You will use one of these methods according to the characteristics of your application servers (see “Examples for Configuring Application Servers” on page 163 for examples).

- **Automatic Explicit Mode.** If your application server is or can be started by a script, it is recommended that you use this mode. To use this mode, you add a line to your application server startup script to explicitly and non-interactively run the JRE Instrumenter to instrument the JRE. Your script will continue to start the application server JVM (with additional parameters) using the freshly instrumented JRE. See “Using the JRE Instrumenter in Automatic Explicit Mode” on page 221.

- ▶ **Automatic Implicit Mode.** With this mode, you do not need to explicitly run the JRE Instrumenter — you only need to modify your application server JVM parameters to invoke the Java Agent and ask it to run the JRE Instrumenter as needed. When the Java Agent is used for the first time, it implicitly runs the JRE Instrumenter to instrument the JRE. However, the first time this instrumented JRE will not be used; your application server will be using an uninstrumented JRE. The next time your application server is started, it will use the instrumented JRE. Therefore, if you want to use the full monitoring features of the Java Agent, you need to restart your application server twice after you enable the Java Agent. See “Using the JRE Instrumenter in Automatic Implicit Mode” on page 224.
- ▶ **Manual Mode.** With this mode, you need to manually and interactively run the JRE Instrumenter, either at the end of the Java Agent installation or at a later time, to instrument the JRE. You then modify your application server JVM parameters according to the parameters provided by the JRE Instrumenter. This method is how the JRE Instrumenter works in earlier versions of HP Diagnostics. See “Using the JRE Instrumenter in Manual Mode” on page 226.

If your JRE is updated (such as, applying an application server patch) or if you update the Java Agent, you may need to instrument the JRE again. This issue will be discussed in each mode.

Below is a table that summarizes the requirements of each of the four different methods of doing instrumentation:

		Recommended Instrumentation (Using the JRE Instrumenter)		
	Basic Instrumen- tation	In Automatic Explicit Mode	In Automatic Implicit Mode	In Manual Mode
Minimum required JRE version	1.5	1.4	1.5	1.4
Requires the application server being started by a script	No	Yes	No	No

		Recommended Instrumentation (Using the JRE Instrumenter)		
		Basic Instrumentation	In Automatic Explicit Mode	In Automatic Implicit Mode
Requires knowing where the JRE is installed	No	No	No	Yes
Requires manually running the JRE Instrumenter	No	No	No	Yes
Requires knowing where the JVM parameters can be configured	Yes*	Yes*	Yes*	Yes*
Requires restarting the application server after enabling Java Agent	Yes, once	Yes, once or twice	Yes, twice	Yes, once
Requires maintenance after JRE upgrade/patch	No	No	Yes	Yes

\* If you cannot find where the JRE invocation parameters can be defined, you may still have the option of using an environment variable such as **JAVA\_OPTIONS** to do that.

### Using the JRE Instrumenter in Automatic Explicit Mode

Using the JRE Instrumenter in the Automatic Explicit Mode is recommended when an application server is started by a script, such as WebLogic and JBoss application servers. It is also recommended for WebSphere application servers if they are or can be started by a script - this is the case for most platforms except z/OS. It is also recommended for Tomcat if it is not installed as a Windows service.

To use Automatic Explicit mode, you need to accomplish two tasks:

- ▶ Modify your application server startup script to run the JRE Instrumenter using the same JRE used by your application server. The output from the JRE Instrumenter will give you the JVM parameters you will need in the next task.
- ▶ Configure your application server JVM parameters found in the output from the JRE Instrumenter.

---

**Note:** Make sure you understand the structure of the startup script, how the property values are set, and how to use environment variables before you make any configuration changes. Always create a backup copy of any file you plan to modify before making the changes.

---

In modifying the application server startup script, you first need to identify the line (or lines) in which the JRE is invoked to start the application server JVM. Then, right above this line, you add a line like the following to invoke the JRE Instrumenter using the same JRE used by your application server:

```
<java_command> -jar <JavaAgent_install_dir>/DiagnosticsAgent/lib/jreinstrumenter.jar
-f <pathname>
```

The **<java\_command>** must be **exactly** the same java command that is used to start your application server JVM, since it is the JRE that is instrumented by the JRE Instrumenter. You can usually get this java command by copying the beginning portion of the line that starts your application server JVM.

Below is a table showing the java command used by the original startup script of some commonly used application servers. (Note that this table is provided as helpful tips only; your application server startup script may use a different java command.)

Application Server	Shell Scripts (.sh)	Windows Command Scripts (.bat or .cmd)
JBoss	"\$JAVA"	"%JAVA%"
Tomcat	\${_RUNJAVA}	%_RUNJAVA%.

Application Server	Shell Scripts (.sh)	Windows Command Scripts (.bat or .cmd)
WebLogic	<code>\${JAVA_HOME}/bin/java</code>	<code>%JAVA_HOME%\bin\java</code>
WebSphere	<code>\${JAVA_EXE}</code>	<code>%JAVA_EXE%</code>

The `<JavaAgent_install_dir>` indicates the directory where the Java Agent is installed.

The `<pathname>` must be relative. The JRE Instrumenter will put the instrumented classes in the `<JavaAgent_install_dir>/DiagnosticsAgent/classes/<pathname>/instr.jre` directory. If you run multiple application servers with Diagnostics, you should give each application server a unique `<pathname>` (such as the probe name) so that the multiple instances of the JRE Instrumenter do not interfere each other. See also “Configure Monitoring of Multiple Java Processes on an Application Server” on page 233 for details.

After you add the line as described above to the startup script, every time you start your application server using the startup script, the JRE Instrumenter is invoked and instruments the current JRE. It also prints out the JVM parameters that you should use in the next task. You can usually find the output of the JRE Instrumenter among the output from running the startup script.

Below is an example output from the JRE Instrumenter that instruments a typical JRE version 5.0 or higher:

```
-Xbootclasspath/p:<JavaAgent_install_dir>/DiagnosticsAgent/classes/<pathname>/instr.jre
-javaagent:<JavaAgent_install_dir>/DiagnosticsAgent/lib/probeagent.jar
```

Below is an example output from the JRE Instrumenter that instruments a typical JRE version 1.4.x:

```
-Xbootclasspath/p:<JavaAgent_install_dir>/DiagnosticsAgent/classes/<pathname>/instr.jre;
<JavaAgent_install_dir>/DiagnosticsAgent/classes/boot
```

The second task for using the Automatic Explicit JRE instrumentation is to modify your application server JVM parameters according to the output of the JRE Instrumenter. In many cases, you just need to modify the java command-line options in the startup script to include the JVM parameters provided by the JRE Instrumenter. However, in some scenarios (such as for WebSphere application servers), you may need to modify a configuration file or use an administration console to add these JVM parameters.

Note: To get the output from the JRE Instrumenter, you need to modify the startup script as described in the first task and restart the application server. Then, after you make changes to the application server JVM parameters, you need to restart the application server again (causing you to restart the application server twice). However, for most of the JREs, the actual JVM parameters provided by the JRE Instrumenter will be the same as or will include what is provided in the examples above. Therefore, you can safely add these "default" JVM parameters even before you run the modified script. This approach is used in the instructions for specific application servers. Refer to the example for your application server (JBoss, WebLogic, WebSphere, Tomcat) to see detailed instructions for how to configure using automatic explicit mode.

Alternatively, you can redirect (or pipe) the output from the JRE Instrumenter to the java command-line options, or get the JVM parameters from a difference source to avoid restarting twice.

### **Using the JRE Instrumenter in Automatic Implicit Mode**

Using the JRE Instrumenter in the Automatic Implicit Mode is recommended when an application server cannot be started by a script, such as GlassFish, NetWeaver, Tomcat installed as a Windows service (no scripts), WebSphere installed on z/OS, and TIBCO ActiveMatrix and BusinessWorks.



To use this mode, you do not need to explicitly invoke the JRE Instrumenter; it is implicitly invoked by the Java Agent. You just configure your application server JVM parameters to invoke the Java Agent and, when the Java Agent sees that the JVM boot class path contains a path pointing to a location matching the following pattern, it enters the automatic instrumentation mode to create the instrumented classes and populates the specified directories with copies of the instrumented classes:

```
<JavaAgent_install_dir>/DiagnosticsAgent/classes/auto/<name>/instr.jre
```

For example if you add the following JVM parameters:

```
-Xbootclasspath/p:<JavaAgent_install_dir>/DiagnosticsAgent/classes/auto/ServerOne/instr.jre  
-javaagent: <JavaAgent_install_dir>/lib/probeagent.jar
```

Then during the first execution of the application server, the directory **<JavaAgent\_install\_dir>/DiagnosticsAgent/classes/auto/ServerOne/instr.jre** may not even exist. The Java Agent will create and populate the specified directory with the instrumented classes. And it will use the exact (uninstrumented) JRE that it runs on.

The first execution of the application server will not benefit from the instrumented JRE, but all subsequent executions will use the instrumented classes prepared in the first run.

---

**Important:** If you update the JRE used by your application server (such as applying an application server patch) or if you update the Java Agent, before you start the application server again you must delete the **<JavaAgent\_install\_dir>/DiagnosticsAgent/classes/auto/ServerOne** directory (use your directory name for ServerOne) so that the new JRE will be instrumented. Otherwise, your application server may not start. You can also manually delete this directory when you want the Java Agent to instrument the JRE again.

---

## Using the JRE Instrumenter in Manual Mode

You can manually run the JRE Instrumenter and copy the provided JVM parameters into your application server startup settings. Using the JRE Instrumenter in the Manual Mode is recommended for Oracle application servers.

The JRE Instrumenter performs the following functions:

- ▶ Identifies JREs that are available to be instrumented.
- ▶ Searches for additional JREs in directories you specify.
- ▶ Instruments the JREs you specify and provides the parameter you must add to the startup script for the JRE to point to the location of the instrumented classes.
- ▶ When the Instrumenter is run using the graphical interface or console mode in a Windows or UNIX environment, the Instrumenter places the instrumented classes in a folder under the `<JavaAgent_install_dir>/DiagnosticsAgent/classes/<JRE_vendor>/<JRE_version>` directory.

---

**Important:** If you update the JRE used by your application server (such as applying an application server patch) or if you update the Java Agent, you must run the JRE Instrumenter again to instrument the new JRE and change the JVM parameters accordingly. Otherwise, your application server may not start.

---

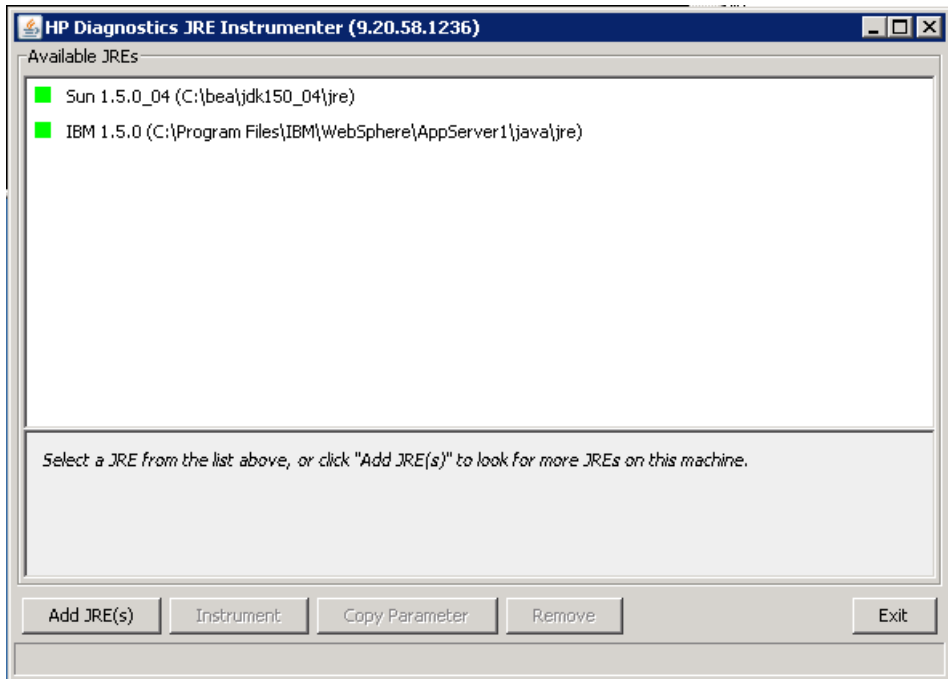
## Running the JRE Instrumenter Utility in UI Mode

When the JRE Instrumenter is run without any options the Instrumenter displays the dialogs of its graphical user interface.

To start the JRE Instrumenter utility on a Windows system run the `<probe_install_dir>\bin\jreinstrumenter.cmd` command.

To start the JRE Instrumenter utility on UNIX or Linux run the `<probe_install_dir>\bin\jreinstrumenter.sh` command.

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.

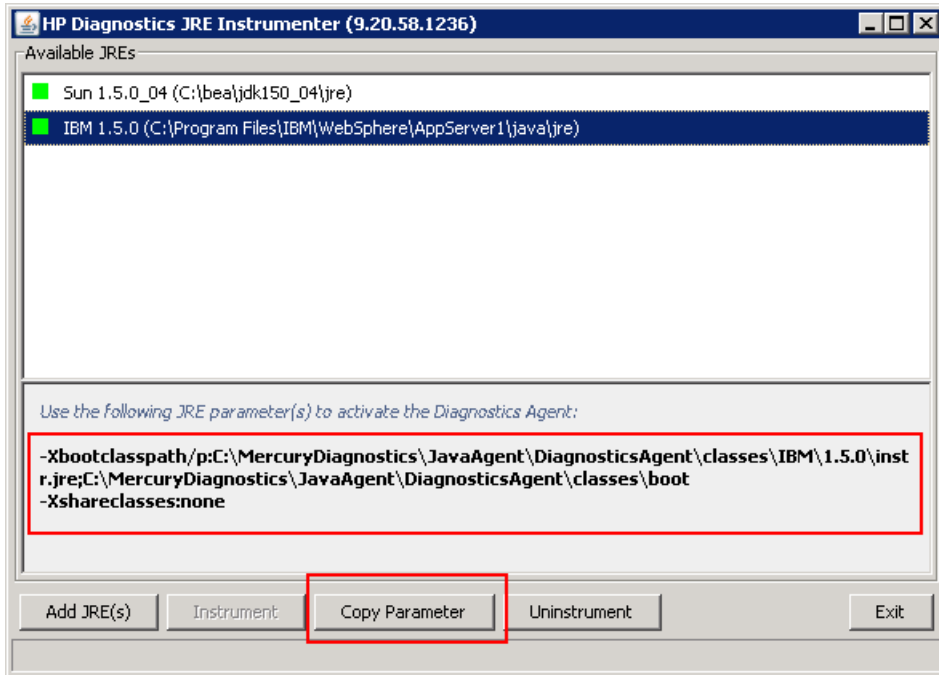


If the **JRE Directory** is not listed on the dialog, click the **Add JRE(s)** button to browse to the JRE. Navigate to the directory location where you want to begin searching for JVMs and then select the file where you want to begin the search and click Search from here. The Instrumenter searches and then lists the JVMs found in the Available JREs list.

Select the JRE to be instrumented and then 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 JREs list.

When the JRE Instrumenter instruments a JRE, it also creates the JVM parameters 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 JRE from the Available JREs list, the JVM parameters are displayed in the box below the list.



Click **Copy Parameter** to place the corresponding parameter on the clipboard. The JVM parameter is copied to the clipboard so that you can use the JVM parameters in configuring your application server to activate monitoring by the Java Agent.

---

**Important:** You will use the clipboard contents later in configuring you application server, so be careful to not overwrite the clipboard contents.

---

Click **Exit** to close the JRE Instrumenter window and continue with configuring your application server JVM parameters.

For general instructions for how to insert the JVM parameter into application server startup scripts see “Including the JVM Parameter in the Application Server’s Startup Script” on page 230. For specific examples of how to insert the JVM parameter into startup scripts for different application servers such as JBoss, WebLogic and Tomcat see “Examples for Configuring Application Servers” on page 163.

### Running the JRE Instrumenter in Console Mode

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

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

When the Instrumenter runs, it displays a list of the processing options that are available. The following table directs you to the documentation for each of the processing options:

Instrumenter Function	Description
<code>jreinstrumenter -l</code>	Display a list of the JVMs that are known to the JRE Instrumenter. Displays the JVM vendor, JRE version, and the location where the JRE is located.
<code>jreinstrumenter -i &lt;jre_directory&gt;</code>	Select a JRE in a specific directory for instrumentation. Replace <code>&lt;jre_directory&gt;</code> with the path to the folder where the JRE you selected from the Available JVM list is found.  This command instructs the JRE Instrumenter to instrument the classes for the selected JVM and to place the instrumented classes in a folder under the <code>&lt;probe_install_dir&gt; /classes/ &lt;JVM_vendor&gt;/&lt;JRE_version&gt;</code> directory.

Instrumenter Function	Description
<pre>jreinstrumenter -a &lt;directory&gt;</pre>	<p>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. Replace &lt;directory&gt; with the path to the location where you would like the Instrumenter to begin searching.</p> <p>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.</p>

Copy the JVM parameter from the output of the JRE Instrumenter so that you can paste it into the location that allows it to be picked up when your application server starts in order to activate monitoring by the Java Agent.

Exit the JRE Instrumenter and continue with configuring your application server JVM parameters.

For General instructions for how to insert the JVM parameter into application server startup scripts see “Including the JVM Parameter in the Application Server’s Startup Script” on page 230. For specific examples of how to insert the JVM parameter into startup scripts for different application servers such as JBoss, WebLogic and Tomcat see “Examples for Configuring Application Servers” on page 163.

### **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. General instructions are provided below.

See “Examples for Configuring Application Servers” on page 163 for specific examples of how to insert the JVM parameter for application servers such as WebLogic, WebSphere, JBoss and others.

**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 Java parameter from the JRE Instrumenter to the java command line that starts the application server, for example:

```
-Xbootclasspath/p:<probe_install_dir>\classes\Sun\1.4.2_04\instr.jre;  
<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 Java parameter:

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m -classpath "%CLASSPATH%"  
-Dweblogic.Domain=petstore -Dweblogic.Name=petstoreServer -Dbea.home="C:\bea"  
-Dweblogic.management.password=%WLS_PW%  
-Dweblogic.ProductionModeEnabled=%STARTMODE%  
-Dcloudscape.system.home=./samples/eval/cloudscape/data  
-Djava.security.policy="C:\bea\wlserver6.1/lib/weblogic.policy" weblogic.Server
```

The following is an example of a WebLogic java command line in a startup script after adding the Java parameter:

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m
-Xbootclasspath/p:<probe_install_dir>\classes\Sun\1.5.0_17\instr.jre;
-javaagent:<probe_install_dir>\lib\probeagent.jar
-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.
- 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 instrument the JRE used by the application server or did not configure your application server JVM parameters to invoke the Java Agent (see the instructions in this chapter for your application server).

## Other Configuration Options

The following sections give you other configuration options:

- “Configure Monitoring of Multiple Java Processes on an Application Server” on page 233
- “Adjusting the Heap Size for the Java Agent in the Application Server” on page 237
- “Configuring the SOAP Message Handler” on page 237
- “Configuring the Discovery of a New J2EE Server for CI Population” on page 241
- “Special Considerations for Applications Based on the OSGi Framework” on page 242
- “Special Considerations for Azul Users” on page 243



## **Configure Monitoring of Multiple Java Processes on an Application Server**

When your application server is using multiple Java processes, or when you want to collect performance data for multiple Java processes, you must perform additional agent configuration steps. You have two options. You can configure a separate Java Agent installation for each JVM on a host, or you can configure a single Java Agent installation to be shared by all of the JVMs.

This section includes:

- ▶ “Configure a Single Java Agent Installation to be Shared by Multiple JVMs” on page 233
- ▶ “Configure a Separate Java Agent Installation For Each JVM” on page 236

### **Configure a Single Java Agent Installation to be Shared by Multiple JVMs**

To allow multiple JVMs to share a single Java Agent installation, you must configure a separate probe instance for each JVM. This configuration enables the following:

- ▶ Establishment of communication between the Diagnostics Server and the probe
- ▶ Identification of the probe by the Diagnostics Server
- ▶ Instrumentation of the JRE used by the JVM

#### **To configure a single Java Agent installation to be shared by multiple JVMs:**

When a single Java Agent installation is used to monitor multiple JVMs, you must configure application server JVM parameters accordingly to invoke the Java Agent. Each JVM can use a different JRE instrumentation mode (see Chapter 6, “Preparing Application Servers for Monitoring with the Java Agent” for details on JRE instrumentation modes.

- 1** If you did not instrument each of the JRE versions used, do so now. See Chapter 6, “Preparing Application Servers for Monitoring with the Java Agent.”

- 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 (Min/Max) 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 **webserver.properties** file in the folder `<probe_install_dir>/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 line 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 using the same script, 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 each probe will use. By default, the points file name is `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 JRE versions on the same machine using a single agent installation, and one or more of the JREs needs specific methods and classes included in a layer to support custom instrumentation.

```
-Dprobe.points.file.name="<Custom_AutoDetect_Points_File>"
```

### Configure a Separate Java Agent Installation For Each JVM

When there are multiple JVMs on a single host, you can configure a separate Java Agent installation for each JVM instance. You install the agent multiple times and define an instance of a probe by setting the probe's `id` property in the `probe.properties` file in each agent's installation directory.

**To configure a separate installed agent for each JVM:**

- 1 If you did not instrument the JRE, do so now see Chapter 6, "Preparing Application Servers for Monitoring with the Java Agent."
- 2 Locate the `probe.properties` file in the `<probe_install_dir>/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>/log` directory where the log messages for the probe are stored.

## Adjusting the Heap Size for the Java Agent in the Application Server

The size of the heap can impact the performance of the Java Agent and the application server. The default value for the heap size is 64 MB, but an application server usually increases it to a larger amount. When you add the Java Agent to an application server, you may need to increase the heap size to accommodate the memory used by the Java Agent. See “Requirements for the Diagnostics Java Agent Host” on page 36 for details.

The heap size is set in the application server JVM configuration using the following JVM argument:

```
-Xmx<size>
```

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

## Configuring the SOAP Message Handler

The Diagnostics SOAP message handler is required for Java 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 239.

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 Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](http://support.openview.hp.com/sc/support_matrices.jsp).

---

This section includes the following:

- ▶ “Disabling the SOAP Message Handler” on page 238
- ▶ “Loading the Diagnostics SOAP Message Handler” on page 239
  - ▶ “WebSphere 5.1 JAX-RPC” on page 239
  - ▶ “Oracle 10g JAX-RPC” on page 240

### **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>\br/><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 you 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>
```

## Special Considerations for Applications Based on the OSGi Framework

If your application is based on the OSGi framework, you may need to set some additional properties. If not already the default value, set the `org.osgi.java.profile.bootdelegation` property to the default value "ignore". Then append `com.mercury.*` to the end of the `org.osgi.framework.bootdelegation` property in your `org.osgi.java.profile`. For example:

```
org.osgi.framework.bootdelegation= <existing packages>,com.mercury.*
```

## Special Considerations for Azul Users

Azul provides two highly scalable and highly performing solutions for enterprise Java users: Vega and Zing. Vega is a special hardware appliance which connects to the user local network. Zing is a virtual equivalent of Vega, provided in a form of a guest image for VMware or KVM. A major advantage of the Azul appliances is its innovative pauseless garbage collector, which runs continuously and can handle heaps up to tens of gigabytes. Both appliances are supported by Diagnostics equally, although we tested only Zing in the lab.

The Java SDK or JRE provided by Azul installs on a traditional system, such as Linux or Solaris, but when it is invoked, it delegates the execution of any Java code to the appliance. Thus, although the Java application seems to be running where it was invoked, it actually runs on a different system. This is done seamlessly, so the application interacts with its environment just as if it was running on a local system. If the application makes a JNI call, it is made across the network to be executed on the originating host.

This execution model creates a number of issues for Diagnostics users. The JNI calls made by the probe are costly, but what is more important, they do not provide the results the user might expect.

- ▶ The CPU timestamps do not work correctly. They measure the CPU time used on the originating server, and therefore are useless.
- ▶ Process metrics are useless, too, because they measure the front-end process.
- ▶ In most cases, all system metrics are useless as well. They measure the originating system and are irrelevant to the application running on the appliance.
- ▶ Garbage collection metrics are confusing. Since Azul uses continuous garbage collector, seeing garbage collection percentages over 100% is normal.
- ▶ Heap Breakdown and Heap Walker do not work.
- ▶ VMware special timers do not work (even if using virtual appliance on VMware)

## Configuring Diagnostics for Azul VM

Invoking Azul java command requires adding parameters that properly identify the appliance to be used for running the application. This creates a difficulty for JREinstrumenter (unless run in Automatic Implicit mode), which needs to run the JRE to be instrumented in order to determine its version and vendor, but is not capable of adding the required parameters.

The solution is to edit the file **azul.properties** found in the Azul JRE installation and define the required parameters. The settings are needed while the JREinstrumenter runs and can be removed for running the application with Diagnostics.

To eliminate possible confusion and pointless overhead, we recommend to use the following settings while using Diagnostics Agent:

- In **metrics.config**, comment out all metrics for "system" and "ProcessMetrics" collectors, and Garbage Collection metrics for the "Java Platform" collector.
- In **capture.properties** set `use.cpu.timestamps=false`.

# 7

---

## Preparing Application Servers for Client Monitoring with the Java Agent

This section explains how to prepare application servers for client monitoring with the Java Agent.

### **This chapter includes:**

- About Client Monitoring on page 245
- Enabling Client Monitoring on page 246
- Configuring and Disabling Client Monitoring on page 248
- Manually Instrumenting HTML/JSP Pages for Client Monitoring: on page 249

### **About Client Monitoring**

Client Monitoring measures web page performance as seen by the user's browser and correlates these measurements with the back end server request.

Three important metrics are measured: the back-end time, the front-end time, and the total time.

The back-end time is the amount of time it takes from when a web page request is sent until the first byte of the response is received.

The front-end time is the amount of time it takes from when the first byte of the response is received until the page is loaded.

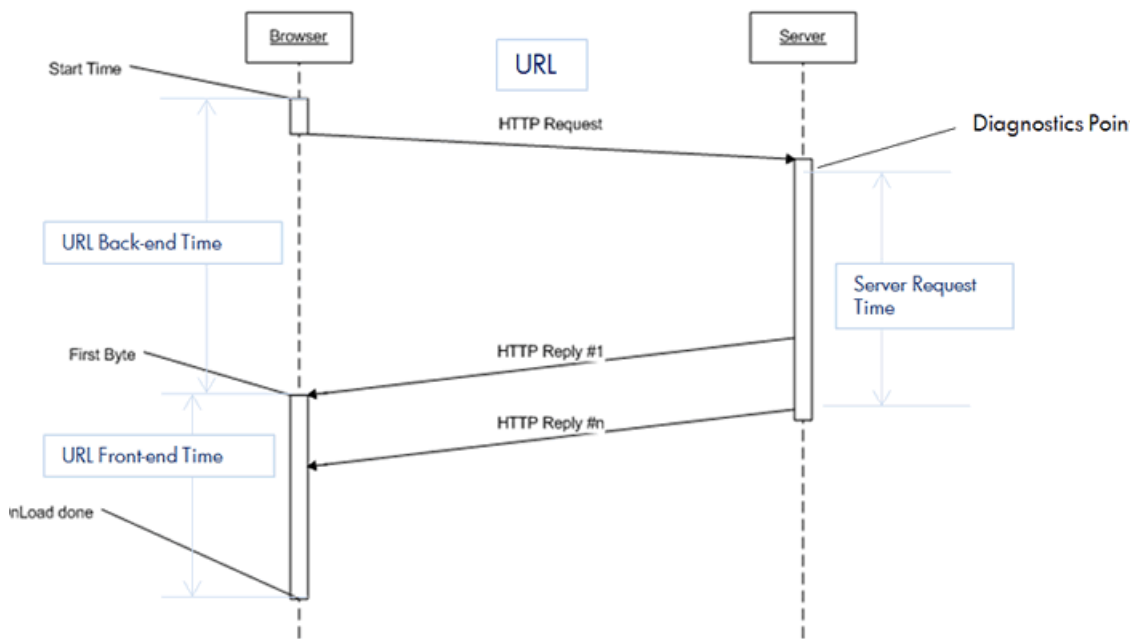
The total-time is the sum of the front and back end times.

Client Monitoring aggregates these measures and presents them by URL, Location, and Browser-OS combination.

By monitoring web page performance, application owners can quickly identify performance issues, characterizing them by tier (front or back-end), location, and browser.

When the issue is on the back-end, client monitoring correlates the URL to the associated server request and its call-profile.

An example showing client monitoring is shown below:



## Enabling Client Monitoring

Enabling client monitoring requires you to deploy a .war file on the application server and in some cases to configure the web server. Client Monitoring views are available in the Diagnostics Enterprise UI.

**To enable Client Monitoring:**

When client monitoring is enabled, most JSP pages served via JBoss, Tomcat, WebSphere and WebLogic will be automatically modified to include additional Java Script calls near the <head> tag. You can see which pages are instrumented by opening the page in your browser and selecting view source.

Other application servers may require manual page instrumentation for client monitoring. See “Manually Instrumenting HTML/JSP Pages for Client Monitoring:” on page 249.

Client monitoring, including **automatic JSP instrumentation**, will remain disabled until this .war file is deployed.

**1 Deploy HPDiagCM.war file.**

Use the application server’s Administrative Console to deploy the <probe\_install\_dir>\contrib\HPDiagCM.war as an application.

Client monitoring will remain disabled until this .war file is deployed.

For WebSphere application servers, be sure to set the context root to /HPDiagCM instead of the default (/).

- 2** If you have configured a web server as the front-end of your application, then you also need to add the following context root to your Web Server's configuration:  
/HPDiagCM/\*

---

**Tip:** You can verify the web server is correctly configured if your browser can access this link: (it will return a blank page)  
<http://hostname:port/HPDiagCM/B/>.

---

## Example - Setting up an Apache HTTP Server Reverse Proxy for Client Monitoring

---

**Important:** These are very basic instructions. These configuration files are highly customized in each customer's environment. Please consult the Apache HTTP Server documentation for more details.

---

In order for client monitoring JavaScript file to be successfully downloaded by browsers and for client-side metrics to be received by the probe, it is necessary to configure the web server to correctly forward those requests to the application server. This is typically achieved by setting up a reverse proxy or gateway.

- 1 Update the `conf/httpd.conf` file by adding the following lines, replacing `<HostName>` and `<HostPort>` with the host name and port of the application server, and restart the web server.

```
ProxyPass      /HPDiagCM http://<HostName>:<HostPort>/HPDiagCM
ProxyPassReverse /HPDiagCM http://<HostName>:<HostPort>/HPDiagCM
```

- 2 Check if your changes are successful by driving traffic to your web application via the web server and checking the web server's log messages in the `log/access.log` file. Error messages will have an http response code in the 400-500 range such as "GET /HPDiagCM/boomerang-min.js HTTP/1.1" 404. When successful, you should see log messages such as "GET /HPDiagCM/boomerang-min.js HTTP/1.1" 200.

If you don't see either of these messages, then client monitoring is not correctly set up in your environment.

## Configuring and Disabling Client Monitoring

If desired, Client Monitoring can be dynamically controlled by updating several properties in `<probe_install_dir>\etc\dynamic.properties`.



The **client.monitoring.enable** property provides a master switch to dynamically enable and disable the client monitoring feature. When set to false, all client monitoring data events received are dropped, JSP page auto-instrumentation will be disabled, and **client.monitoring.sampling.percent** is set to 0.0 (to disable manually instrumented JSP pages' client monitoring Java Script code).

You can reduce the client monitoring load on your server by adjusting the **client.monitoring.sampling.percent** property in **dynamic.properties**.

You can also specify that you want a strict check on the referrer by setting **client.monitoring.strict.referrer** to **true**. This will help ensure that only events that originate from a web page instrumented with client monitoring are used. The default value is false but the recommended value is true if this setting works in your environment.

You can also stop or uninstall/undeploy the HPDiagM.war using your application server management console.

## Manually Instrumenting HTML/JSP Pages for Client Monitoring:

Add the following code to your HTML/JSP pages immediately after the <head> tag:

```
<!-- HP Client Monitoring -->
<script>
if (window.t_firstbyte === undefined) {
    var t_firstbyte = Number(new Date());
}
</script>
<script type='text/javascript' src='/HPDiagCM/boomerang-min.js'>
</script>
<script>
BOOMR.init({beacon_url:"/HPDiagCM/B",
    RT:{cookie:"X-HP-CM-RT",cookie_exp:600,expandFrames:true,hashURLs:true},
    HP:{cookie:"X-HP-CM-GUID"}});
</script>
```

If you prefer to manually instrument HTML/JSP pages you can permanently disable auto-instrumentation by setting the following properties in `inst.properties` to `false`. These changes require a restart of the application server.

**<probe\_install\_dir>\etc\inst.properties:**

```
details.conditional.properties= \  
mercury.enable.clientmonitoring.JspWriterImpl.autoinstrumentation=false,\  
mercury.enable.clientmonitoring.CoyoteWriter.autoinstrumentation=false,\  
mercury.enable.clientmonitoring.BodyContentImpl.autoinstrumentation=false,\  

```

# 8

---

## Installing .NET Agents

This section describes how to install a .NET Agent and gives you information about the setup and configuration of the .NET Agent.

**This chapter includes:**

- ▶ Overview of the .NET Agent Installation on page 252
- ▶ Accessing the .NET Agent Installer on page 254
- ▶ Installing the .NET Agent on page 255
- ▶ Post Install Tasks on page 277
- ▶ Verifying the .NET Agent Installation on page 278
- ▶ About Configuration of the .NET Agent for Diagnostics on page 279
- ▶ About Configuration of the .NET Agent for TransactionVision on page 279
- ▶ Discovery and Standard Instrumentation on page 282
- ▶ Probe Aggregator Service on page 286
- ▶ Monitoring NET Applications Deployed in Azure Cloud on page 287
- ▶ Determining the Version of the .NET Agent on page 288
- ▶ Enabling and Disabling the Diagnostics Agent for .NET on page 288
- ▶ Disabling Logging on page 289
- ▶ Enabling and Disabling Standard Instrumentation for Applications on page 290
- ▶ Troubleshooting .NET Web Applications Not Discovered on page 292
- ▶ Other .NET Agent Troubleshooting Tips on page 294
- ▶ Uninstalling the .NET Agent on page 294

## Overview of the .NET Agent Installation

The .NET Agent software is installed on the machine hosting the application you want to monitor. With the .NET Agent you instrument the application domains for monitoring.

See Chapter 1, “Preparing to Install HP Diagnostics,” for .NET Agent requirements.

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.

---

**Important:** If you need to 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. WCF bindings using the following transports are supported:

- Http
- TCP

If your application uses a transport 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 cross VM correlation.

The HP Diagnostics/TransactionVision .NET Agent installer installs a .NET Agent to collect data for either Diagnostics or TransactionVision or both.

The .NET Agent installer automatically detects the ASP.NET applications on the system where the agent is installed. See “Discovery and Standard Instrumentation” on page 282

The installer configures the agent to capture basic workload and events for each of the ASP.NET applications detected. The agent configuration is controlled using the **probe\_config.xml** file. See “Automatic Instrumentation and Configuration for Discovered ASP.NET Applications” on page 283.

The .NET agent uses **points files** to provide standard instrumentation to enable you to start monitoring applications. The points files control the workload the agent captures for the application. See Chapter 11, “Custom Instrumentation for .NET Applications”. See “Enabling and Disabling Standard Instrumentation for Applications” on page 290.

The following points files are installed and enabled to provide instrumentation for monitoring ASP.NET applications:

- ASP.NET.points
- ADO.points
- WCF.points
- The following points files can be used for instrumenting applications that use other Microsoft technologies:
- Remoting.points (for .NET remoting environments)
- msmq.points (for MSMQ environments)
- LWMD.points (for analysis of memory used by collections in applications)

Separate 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. And each appdomain section contains an instrumentation points file reference. The .NET Agent uses this runtime instrumentation to capture method latency information from specified applications.

**HP Software-as-a-Service (SaaS).** HP Diagnostics can be deployed into an HP Software-as-a-Service (SaaS) environment. In a SaaS deployment the Diagnostics .NET Agents are installed in your company’s IT environment and the Diagnostics Commander Server and Mediator Servers are installed by HP on a SaaS system on-premise at HP. During the setup of the .NET Agent you select the following option for configuring the agent: Diagnostics Mode with SaaS-hosted mediator installed on HP premises.

See Accessing the .NET Agent Installer to begin.

## Accessing the .NET Agent Installer

You can launch the .NET Agent installer a number of different ways. You can install the .NET Agent from the Diagnostics installation disk or the BSM installation disk or from the Downloads page in Business Service Management. You can install the software from the SSO Portal. And if you want to install a trial version of the HP Diagnostics Profiler for .NET you can launch the installer from the HP Software Web site download center.

### To access the Installer from a Diagnostics installation location:

- ▶ From the Diagnostics Installation DVD (Autorun.exe) the installation menu page is displayed. From the menu, select **Diagnostics Agent for .NET 32-bit** to launch the install for the 32-bit Windows version of the .NET agent. And select **Diagnostics Agent for .NET 64-bit** to launch the install for the 64-bit version of the .NET agent.
- ▶ You could run the appropriate installer directly by locating the executable file **HPDiagTV.NETAg<release number>\_win32.msi** (32-bit) or **HPDiagTV.NETAg<release number>\_win64.msi** (64-bit) in the location you install from and copying the file to the new installation location and then double-clicking it to run the installer.

Continue with “Installing the .NET Agent” on page 255.

### To download the installer from the HP Software Download Center:

- 1** Access the SSO portal at <http://support.openview.hp.com/selfsolve> using your HP Passport login.
- 2** Locate the Diagnostics (or TransactionVision) downloads and choose the appropriate link for downloading the Diagnostics .NET Agent software. Note that you could also use the download center in order to get the Diagnostics .NET profiler trial/evaluation software.
- 3** Follow the download instructions on the web site.

Continue with “Installing the .NET Agent” on page 255.

To download the Installer from Business Service Management's Diagnostics downloads page:

- 1 In **Business Service Management**, either select **Admin > Diagnostics** from the main menu and click the **Downloads** tab. Or select **Admin > Platform** from the main menu and click the **Setup and Maintenance** 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 "Installing the .NET Agent" on page 255.

To launch the installer for HP Diagnostics Profiler for .NET trial software 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 **Trial Software**, select the appropriate link.
- 4 Follow the download instructions on the web site.

Continue with "Installing the .NET Agent" on page 255.

## Installing the .NET Agent

This section provides detailed instructions for a *first time* installation of the .NET Agent.

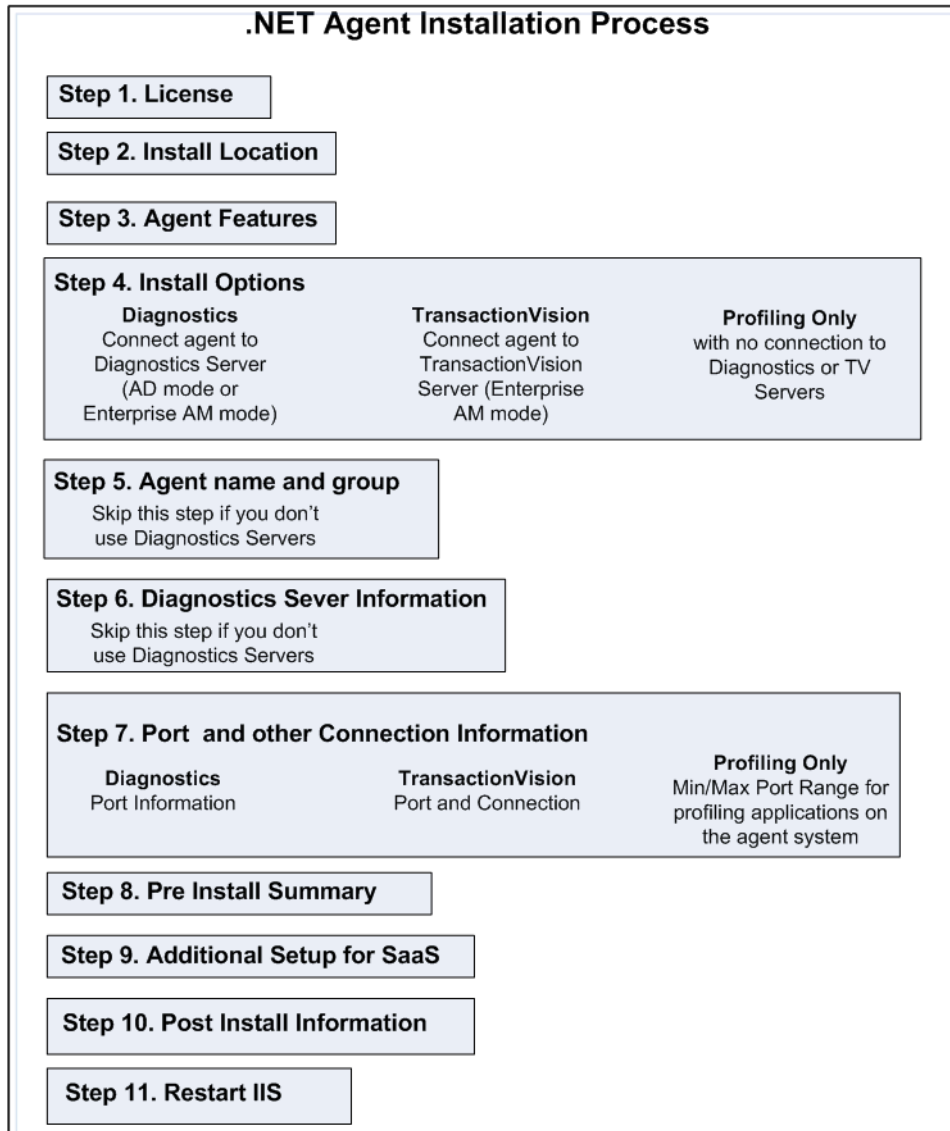
---

**Important:** If there is a pre-existing installation of the .NET Agent on the host machine, you must follow the instructions for upgrading the agent system instead of these install instructions see “Upgrade and Patch Install Instructions” on page 893.

---



An overview of the .NET Agent installation steps is shown in the diagram below, refer to the rest of this section for details on each step:



The .NET Agent installation process includes the following steps, select Step 1. End user license agreement to begin:

- “Step 1. End user license agreement” on page 258
- “Step 2. Specify install location” on page 258
- “Step 4. Select agent features to install” on page 261
- “Step 3. Select installation options” on page 259
- “Step 5. Agent name and group” on page 262
- “Step 6. Diagnostics server information” on page 264
- “Step 7. Port and connection information” on page 266
- “Step 8. Pre-install summary” on page 272
- “Step 9. Additional Setup for Agents Working in an HP SaaS Environment” on page 273
- “Step 10. Post Install Information” on page 275
- “Step 11. Restart IIS” on page 276

### **Step 1. End user license agreement**

Accept the end user license agreement.

Read the agreement and select **I accept the terms of the License Agreement**.

Click **Next** to proceed and continue to the next step.

### **Step 2. Specify install location**

Provide the location where you want the Agent installed.

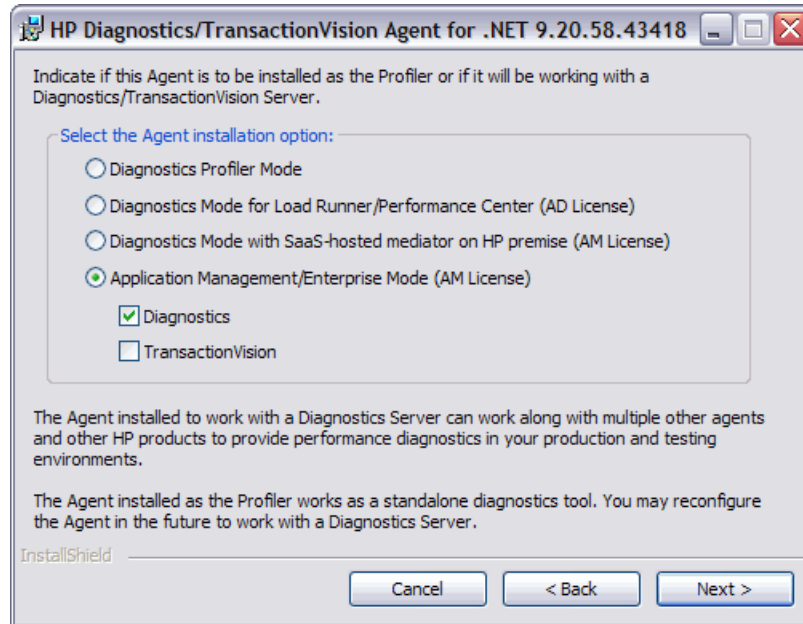
By default, the Agent is installed in **C:\MercuryDiagnostics\.NET Probe**. This location becomes the <probe\_install\_dir>.

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 proceed and continue to the next step.

### Step 3. Select installation options

Indicate if the .NET Agent is to be installed as a standalone Profiler without any connection to a server (for example if you are installing the Diagnostics .NET Profiler trial software), or if you are installing the agent to work LoadRunner/Performance or to work with a Diagnostics and/or TransactionVision Server.



Make the selection that is appropriate for the environment where you will be using the agent.

- **Diagnostics Profiler Mode.** Select this option to install the agent as a Diagnostics .NET Profiler without any connection to a Diagnostics server. This is typically selected when installing the Diagnostics .NET Profiler trial software prior to purchasing the HP Diagnostics product.

If you select Diagnostics Profiler Mode option, the value of the **probe\_config.xml** `<modes>` element is set to **pro** mode at the time you install the .NET Agent (see “`<modes>` element” on page 592).

- **Diagnostics Mode for LoadRunner/Performance Center (AD License).** Select this option to install the agent for use with a Diagnostics Server in a load testing (or pre-production) environment where probes are used only in LoadRunner or Performance Center runs.

The advantage of running a probe in AD mode is that probes in AD mode are only counted against your HP Diagnostics AD license capacity when in a LoadRunner or Performance Center run. For example if you have 20 probes installed in LoadRunner/Performance Center AD mode but only 5 in a run, then only 5 are counted against your AD license capacity. See “License Information Based on Currently Connected Probes” on page 85 for more information on AD license capacity.

In AD mode the agent will ONLY capture data during a LoadRunner or Performance Center run and the results will be stored in a specific Diagnostics database for that run, for example, Default Client:21. When the agent is in AD mode it will not use resources or send any data to the server unless the probe is part of a LoadRunner/Performance Center run.

If you select this AD License option, the value of the **probe\_config.xml** `<modes>` element is set to **ad** mode at the time you install the .NET Agent (see “`<modes>` element” on page 592).

- **Diagnostics Mode with SaaS-hosted mediator on HP premise (AM License).** Select this option to install the agent to work in a SaaS environment where the .NET agent will connect to an HP SaaS server on-premise at HP. An HP SaaS administrator will provide you with information on connecting the .NET agent to an HP SaaS hosted Diagnostics mediator server.
- **Application Management/Enterprise Mode (AM License).** Select this option to install the agent for use with a Diagnostics Server and/or a TransactionVision Server in an enterprise (or production) environment.

Then indicate which of the following the agent will be configured for:

- A Diagnostics Server (installed locally)
- A TransactionVision server
- Both a Diagnostics Server installed locally and a TransactionVision Server

If you select TransactionVision, see the *HP TransactionVision Deployment Guide* in the Business Service Management documentation library for details on setup options specific to TransactionVision.

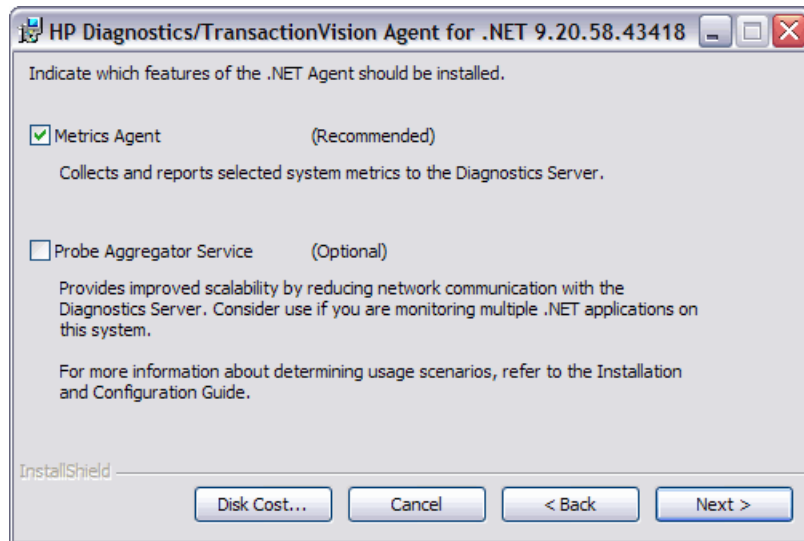
With this option, the value of the **probe\_config.xml** <modes> element is set to **enterprise** mode if you select the Diagnostics Server and **tv** mode if you select the TransactionVision server at the time you install the .NET Agent (see “<modes> element” on page 592).

For those agents with Enterprise mode set, the agent will be counted against your HP Diagnostics AM license capacity.

Click **Next** to proceed and continue to the next step.

## Step 4. Select agent features to install

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 18, “.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. If you are installing the agent to work in an HP SaaS environment the Probe Aggregator box will be checked for you since this option is required for SaaS and cannot be changed.

This Probe Aggregator service aggregates .NET Agent data to 5 second intervals before sending the performance data to the Diagnostics mediator server. This can improve scalability by reducing network communications with the server but the aggregator will also increase probe system overhead. See “Probe Aggregator Service” on page 286 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 proceed and continue to the next step.

### Step 5. Agent name and group

Skip this step if the agent won't be reporting to a Diagnostics Server.

Enter the Agent Name and Agent Group Name.

The Agent Name uniquely identifies each agent. The default is the name of the application which loads the agent.

Agent Name (Leave blank to accept default based on application name):

An Agent Group is a logical collection of agents that are monitored by the same Diagnostics Server. The default value is "Default".

Agent Group Name:

Default

Enter the admin user password used to connect to the profiler. If left blank, the default password ("admin") is set.

Profiler Admin Password:

InstallShield

Cancel < Back Next >

- **Agent Name.** The name that identifies the agent within HP Diagnostics. If you leave this field blank, the .NET Agent will auto-generate an agent name based on the application domain name of the monitored application. The agent name is assigned as the probe entity name.

---

**Note:** 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.
- \$(COMMANDLINE:n) Where n is the command line parameter number.

For example:

```
<id probeid="ILTEST_$(COMMANDLINE:3)_rest" probegroup="Default"/>
```

with a command line of `iltest "heart and lung" -abc server` results in a probeid of `ILTEST_server_rest`.

Note that n=0 indicates the executable/command name.

---

**Note:**

- For applications that 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.
  - 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.

- **Profiler Admin Password.** Enter the admin user password used to connect to the .NET Diagnostics Profiler. If left blank, the default password (admin) is set.

Click **Next** to proceed and continue to the next step.

## **Step 6. Diagnostics server information**

Skip this step if the agent won't be reporting to a Diagnostics Server or if you are installing the agent to work in an HP SaaS environment. Your HP SaaS administrator will provide details for configuring communication between the agent and the SaaS-hosted Diagnostics Server.

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.

- ▶ 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 mediator 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 mediator 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 and continue to the next step.

### **Step 7. Port and connection information**

You will see different port and connection configuration dialogs depending on what install options you selected. Select from the following and proceed with the configuration:

- ▶ Port connection information for Diagnostics Servers
- ▶ Port and connection information for TransactionVision Server
- ▶ Profiler mode with no connection to a Diagnostics or TransactionVision Server

**If you are installing the Agent to work with a Diagnostics Server, you will see the following dialog box.**

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:**

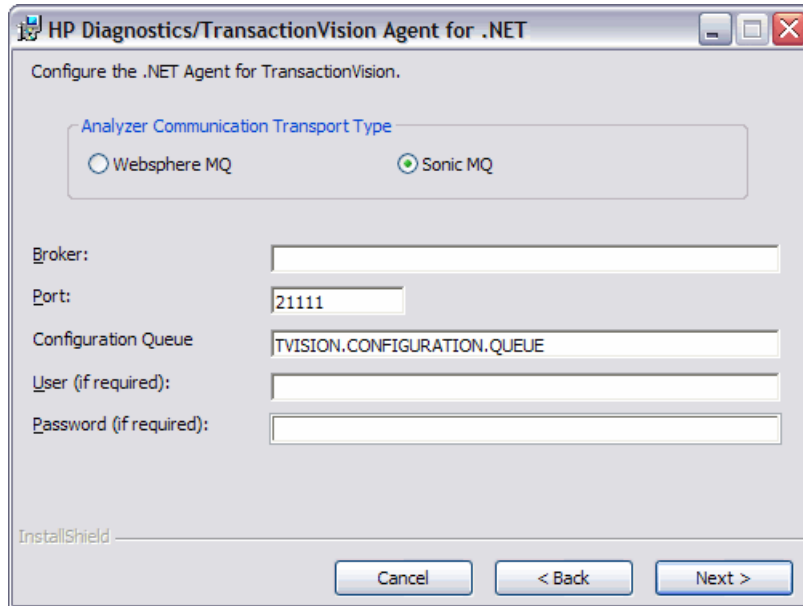
- ▶ If the Agents are working with ASP.NET applications, double the number of ports to account for ASP.NET's appdomain recycling.
- ▶ If you have a firewall between the Agent and a component that will be communicating with the Agent, open the firewall for the ports within the range. Adjust the range to be just big enough.

Click **Next** to proceed and continue to the next step. If you also selected the option to have the agent work in a TransactionVision Environment see the following section for additional configuration.

**If you are installing the Agent to work in a TransactionVision environment, you will see the following dialog box.**

If you selected to install the agent to work with a TransactionVision Server you will see additional screens in the installation. See the *TransactionVision Deployment Guide* for information on using the agent in a TransactionVision environment.

The Configure the .NET Agent for TransactionVision dialog box appears.



Choose the Messaging Middleware Provider. Options are: WebSphere MQ and SonicMQ.

SonicMQ is included with the .NET Agent. If you specify this option, the Sonic MQ .NET client (Sonic.Client.dll - Progress SonicMQ .NET Client, version 7.6.0.112) is installed as part of the Agent installation.

A third-party WebSphere MQ installation can be used instead. In this case, you must install the MQ series .NET client (amqmdnet.dll - WebSphere MQ Classes for .NET, version 1.0.0.3) on the host being monitored.

By default, SonicMQ is selected.

- For SonicMQ, enter the following:

**Broker.** Host name on which the Sonic broker is running. Typically this will be the Analyzer hostname.

**Port.** The port on which the broker communicates. By default, 21111.

**Configuration Queue.** Name of the configuration queue. By default, TVISION.CONFIGURATION.QUEUE.

**User.** User id if required by SonicMQ installation for connection. By default, no username is required.

**Password.** Password if required by SonicMQ installation for connection. This is in the obfuscated form created by using the **PassGen** utility. By default, no password is required. For more information about **PassGen**, see "Command-Line Utilities" in *Using Transaction Management*.

- For WebSphere MQ, enter the following:

**Host.** The host on which the WebSphere MQ queue manager resides.

**Port.** Port number for WebSphere MQ queue manager.

**Configuration Queue.** Name of the configuration queue.

**User.** User id if required by WebSphere installation for connection.

**Password.** Password if required by the WebSphere MQ installation for connection. This is in the obfuscated form created by using the PassGen utility. For more information about **PassGen**, see "Command-Line Utilities" in *Using Transaction Management*.

**Websphere MQ channel.** Channel name for WebSphere MQ queue manager.

**Websphere MQ Q Manager.** CCSID for WebSphere.

Click **Next** to proceed and continue to the next step.

**If you are installing the Agent in Profiler mode, you will see the following dialog box:**

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.

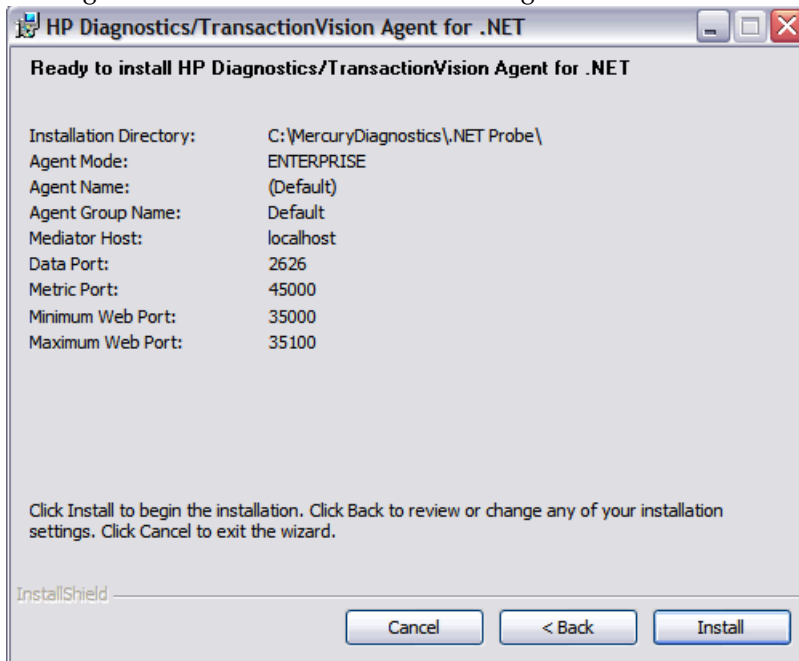
**Considerations when setting the Web Port Range:**

- ▶ 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 proceed and continue to the next step.

**Step 8. Pre-install summary**

The pre-installation summary screen opens. Click **Back** to make any changes. Click **Install** to start the .NET Agent installation.





**Note:** When installing the agent for use as a Profiler only, there is no test for Metric Port connectivity.

---

If you are installing the agent to work in an HP SaaS environment continue to Step 9 otherwise skip the next step and continue to Step 10.

### **Step 9. Additional Setup for Agents Working in an HP SaaS Environment**

If you are installing the agent to work in an HP SaaS environment then the SaaS Setup module starts automatically or you can run the SaaS Setup module anytime by selecting **Start > All Programs > HP Diagnostics .NET Probe > SaaS Setup**.

In the SaaS Setup module the following dialog is displayed. If you are not setting up the agent for an HP SaaS environment then you will not see this dialog.

The screenshot shows a Windows-style dialog box titled "HP Diagnostics/TransactionVision Agent for Java". The window has a dark blue header with the HP logo and the text "Configure the Diagnostics Java Agent".

The main content area is divided into several sections:

- Diagnostics Server Connectivity:** Contains two text input fields. "Diagnostics Server Name:" has the value "localhost" and "Diagnostics Server Port:" has the value "443".
- Additional Options:** Contains a checkbox labeled "Use Proxy Server to connect to Diagnostics Server" which is currently unchecked.
- Proxy Server Options:** A sub-section containing four text input fields: "Proxy Server Name:" (empty), "Proxy Server Port:" (value "80"), "Proxy Server Username (optional):" (empty), and "Proxy Server Password (optional):" (empty).
- Probe Aggregator Admin password (Used for Support purposes only):** Contains a text input field with the value "admin".

At the bottom of the window, there is a "Notes:" section with the following text: "The default server port is 2006. When SSL is enabled, the default server port or 8443. When SSL is enabled AND the mediator is SaaS hosted, the default server port is 443." Below the notes are four buttons: "Back", "Next", "Finish", and "Cancel".

The status bar at the very bottom of the window displays the text "Wed May 09 15:06:32 PDT 2012".

- ▶ **Diagnostics Server Connectivity.** In an HP SaaS environment the Diagnostics Server is setup by HP on an a system on-premise at HP. The default port for a SaaS environment is **443**. An HP SaaS administrator will provide you with the information on the Diagnostics Server host name and port to use.
- ▶ If a proxy server is used to communicate with the Diagnostics Mediator Server select **Use Proxy Server to connect to Diagnostics Server** check box and enter the appropriate options. In an HP SaaS environment if your company requires a proxy to communicate to outside servers then you would select this option.

Proxy Server Options:

- ▶ **Proxy Server Name.** Host name of the proxy server.
- ▶ **Proxy Server Port.** Port of the proxy server.
- ▶ **Proxy Server Username (optional).** The user used to authenticate the proxy server.
- ▶ **Proxy Server Password (optional).** The password used to authenticate the proxy server.
- ▶ **Probe Aggregator Admin password.** The password is automatically set to the same password as the .NET Profiler Admin password (entered in step 5), so for an initial agent setup for SaaS you will not see this field. If you want to subsequently change the Probe Aggregator Admin password, you can run the SaaS Setup module again and this field will be displayed.

Continue on to the next step to finish the installation.

## Step 10. Post Install Information

On the final installation screen, you can select the Show the Windows Installer Log checkbox to view the log file and check for errors.

Click **Finish** to exit the installer.

For information on post installation tasks see “Post Install Tasks” on page 277.

When you are ready you must restart IIS, see the next step.

## Step 11. Restart IIS

After you finish installing and setting up the agent you must restart either the IIS or the Web publishing service before you can use the .NET agent with ASP.NET applications.

To restart IIS from the command line or from the **Start > Run** menu, type `iisreset` and press **Enter**.

For Diagnostics 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.

For TransactionVision 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 reads the Configuration Queue Messages from the Analyzer.

---

**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.

---

Continue with the next section to learn more about post installation tasks.

For information on verifying the installation see “Verifying the .NET Agent Installation” on page 278.

## Post Install Tasks

See the following topics for information about additional configuration for the .NET Agent:

- For information on how the .NET Agent automatically discovers applications and configures standard instrumentation to allow monitoring see “Discovery and Standard Instrumentation” on page 282.
- For information on configuring the .NET Agent for Diagnostics and for links to more advanced topics see “About Configuration of the .NET Agent for Diagnostics” on page 279.
- For information on configuring the .NET Agent for TransactionVision and to see the types of events TransactionVision can trace with the .NET Agent see “About Configuration of the .NET Agent for TransactionVision” on page 279.
- “Enabling and Disabling Standard Instrumentation for Applications” on page 290 for more information.
- For information on configuration for environments with proxies see “Configuring Diagnostics Servers and Agents for HTTP Proxy” on page 671, firewalls see “Configuring Diagnostics to Work in a Firewall Environment” on page 675 and for enabling HTTPS see “Enabling HTTPS Between Components” on page 839.

## Verifying the .NET Agent Installation

On the final installation screen you can select the **Show the Windows Installer Log** checkbox to view the log file and check for errors.

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

Once a .NET probe instance is started you can launch the Diagnostics Enterprise UI to verify that the probe is working. Go to **http://<Diagnostics\_commander\_server>:2006/**. For now you can use the default user/password of **admin/admin** or the login you were given if a different one has been set up for you.

You can also check the System Health view to find information about the .NET agent deployments and the machines that host them.

### To access the System Views:

- 1** Open the Diagnostics UI as the Mercury System customer from **http://<Diagnostics\_Commanding\_Server\_Name>:2006/query/**.
- 2** In the query page locate the Mercury System customer in the list and select the link to Open Diagnostics.
- 3** Log in to Diagnostics and on the Applications window select Entire Enterprise and select any link to open the Diagnostics Views.
- 4** In the Views pane you'll see the System Views view group. Open the view group and select either the System Health view or System Capacity view.

## About Configuration of the .NET Agent for Diagnostics

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 14, “Understanding the .NET Agent Configuration File.” For instructions on advanced .NET Agent configuration, see Chapter 15, “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 11, “Custom Instrumentation for .NET Applications.”

## About Configuration of the .NET Agent for TransactionVision

When used with TransactionVision the .NET Agent captures events from .NET applications and sends the events to the TransactionVision Analyzer. See the *Business Service Management Documentation Library* for more information about TransactionVision.

## .NET Agent Configuration for TransactionVision

The default configuration of the .NET Agent allows you to begin tracing certain .NET events in a monitored application. You can customize the .NET Agent configuration to control what .NET events are traced and sent to the TransactionVision Analyzer.

To override the default configuration, access the `<agent_install_dir>/etc/probe_config.xml` file. See “Understanding the .NET Agent Configuration File” on page 551 for details on the elements you can configure for both Diagnostics and TransactionVision.

The `<modes>` element in the `probe_config.xml` file is set at installation for both Diagnostics and TransactionVision (see “`<modes>` element” on page 592).

When you select to install the .NET Agent to work in a TransactionVision environment the `<modes>` element in the `probe_config.xml` file is set to `tv`. When this is the only mode selected the agent will work in a TV only mode where the Profiler and the Diagnostics probe is disabled and only TV events are generated. When you select to install the .NET Agent to work in other modes such as with Diagnostics then both TV events and Diagnostics data collection will be enabled.

In order to specify TransactionVision specific and TransactionVision transport specific configuration the following elements in the `probe_config.xml` file are used exclusively for TransactionVision:

- ▶ `<tv>` element (see “`<tv>` element” on page 619 for details)
- ▶ `<timeskew>` element (see “`<timeskew>` element” on page 614 for details)
- ▶ `<transport>` element (see “`<transport>` element” on page 616 for details)
- ▶ `<gentvhttpeventforwcf>` element (see “`<gentvhttpeventforwcf>` element” on page 570 for details)

If the .NET Agent is using SonicMQ transport to communicate with the TransactionVision Analyzer, SSL can be enabled. See the *HP Business Service Management Hardening Guide* PDF for details.

By default, .NET Events are not correlated. To enable correlation refer to the HP TransactionVision documentation.



## Types of Events TransactionVision Can Trace with the .NET Agent

When used with TransactionVision the .NET Agent traces the following types of .NET events:

### 1 Web Services

#### a ASP.NET (\*.asmx) - Client and Server

To generate events, use the **ASP.NET.points** file.

#### b WCF (\*.svc) - Client and Server

To generate events, use the **wcf.points** file.

#### c REST style services - Server

To generate events, use the **wcf.points** file and set up the instrumentation of the application as described in “Configure WCF REST Services for Monitoring” on page 444.

### 2 Database calls executed using ADO.NET

To generate events, use the **ADO.points** file.

### 3 .NET Remoting - Client and Server

To generate .NET remoting events, use the **Remoting.points** file and setup the application for instrumentation as described in “How to Configure Instrumentation for .NET Remoting” on page 451.

### 4 MSMQ - Send and Receive (asynchronous)

To generate events, use the **Msmq.points** file.

### 5 HTTP

#### a Client outbound - includes calls to REST services

To generate events, use the **ASP.NET.points** file.

#### b ASP.NET inbound/server (POST, GET, PUT) (\*.aspx)

To generate events for HTTP, use **ASP.NET.points** file.

## 6 User defined events

Use the detail argument **tv:user\_event** (see “Optional Point Entries” on page 432)

To turn off event generation remove the relevant points file from scope.

## Discovery and Standard Instrumentation

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 Instrumentation and 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 14, “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 289.

---

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

With Diagnostics 9.0x or later, Diagnostics populates CIs and model relationships in the Business Service Management 9.0 or later Run-time Service Model (RTSM) 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 or later

---

► 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 Run-time Service Model CIs for .NET Application. See Chapter 22, “Setting Up the Integration Between Business Service Management and Diagnostics” for a discussion of the integration with the Business Service Management 9.0 Run-time Service Model 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 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 11, “Custom Instrumentation for .NET Applications” for more details.

## 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 mediator 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 892 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 mediator 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. For information on how to disable and enable the Probe Aggregator Service see “Enabling and Disabling the Diagnostics Agent for .NET” on page 288.

## Monitoring NET Applications Deployed in Azure Cloud

Microsoft provides Windows Azure SDK for developers to create and deploy Azure applications to the Microsoft Windows Azure Cloud Infrastructure. The Diagnostics .NET Agent leverages the Azure SDK to provide seamless deployment of the .NET Agent into the Azure Infrastructure. Once deployed the .NET Agent monitors applications running in the Azure Cloud, collecting performance data and reporting to an HP Diagnostics Server for analysis and problem detection. See the [AzurePackReadMe.pdf](#) in the .NET Agent AzurePack zip file for details on installing and configuring the .NET Agent for monitoring applications in the Windows Azure Cloud.

## 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 view in the Diagnostics UI Appendix D, “Using System Views for Administrators”).

## 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 > All Programs > HP Diagnostics .NET Probe > Disable HP .NET Probe**.

**To enable a .NET Agent that was disabled:**

- ▶ Select **Start > All 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 > All 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>
```

## 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="1/ROOT/myApplication" website="Default Web Site"
enabled="true">
    <points file="DefaultWebsite-myApplication.points" />
  </appdomain>
  <appdomain name="1/ROOT/myApplicationTwo" website="Default Web Site"
enabled="false">
    <points file="DefaultWebsite-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.

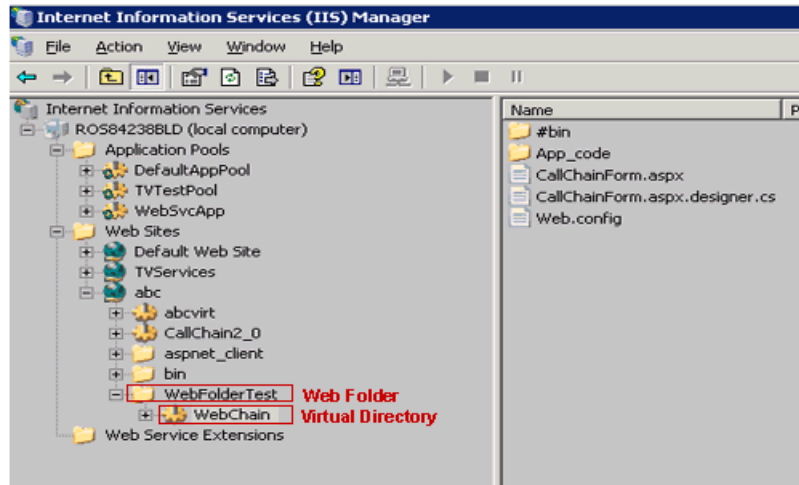
The following example shows instrumentation enabled for all applications:

```
<process name="ASP.NET" enablealldomains="true">
  <logging level=""/>
  <points file="ASP.NET.points"/>
  <appdomain name="1/ROOT/myApplication" website="Default Web Site"
enabled="false">
    <points file="DefaultWebsite-myApplication.points"/>
  </appdomain>
  <appdomain name="1/ROOT/myApplicationTwo" website="Default Web Site"
enabled="false">
    <points file="DefaultWebsite-myApplicationTwo.points"/>
  </appdomain>
</process>
```

## Troubleshooting .NET Web Applications Not Discovered

In a Microsoft Windows 2003 server and IIS 6 environment, if your web site has a virtual directory under a web folder .NET Agent may fail to discover the virtual directory. This is because of an issue with the Microsoft WMI provider used by Diagnostics to walk down the web site tree. The WMI provider does not properly recognize the web folder as an IIS web directory and so Diagnostics can't discover the virtual directory under the folder. See the example described below.

The example shows web folder WebFolderTest under the web site abc. Under this web folder there is a virtual directory WebChain.



Because of an issue with the WMI provider, the listing in WMI for this web site would not show the WebFolderTest/WebChain virtual directory. The .NET Agent uses the listing from the WMI provider to discover web applications. So in situations like this, the .NET Agent may not be able to discover virtual directories under a web folder.

Microsoft recommends modifying the metabase directly or using a simple script like the following to set the folder style using ADSI:

```
Set objRoot = GetObject("IIS://localhost/W3SVC/1/Root/WebFolderTest")
objRoot.KeyType = "IIsWebDirectory"
objRoot.SetInfo()
```

Instead of using a script you can manually configure the web folder as an application in IIS. Once this is done it can be reverted to a non-application but the property would now be set and Diagnostics would be able to discover the web application.

Another option is to manually add the excluded APPDOMAIN in the ASP.NET appdomain list in the **probe\_config.xml** file.

## Other .NET Agent Troubleshooting Tips

If you have problems getting the agent started properly here are some things to check:

- ▶ Make sure you restarted the web server and that a URL in the application was accessed, this triggers the agent to begin collecting data.
- ▶ Check if a `probe_config.xml` file was created and is formatted correctly (that is, no missing tag closers, etc.). This can be done by opening the file in a web browser.
- ▶ Look for any message in the Windows Event Log named “HP Diagnostics”. This log is used exclusively by the .NET Agent. There should be a message for each attempt to instrument an application.

## 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**.

# 9

---

## Installing and Setting Up Python Agents

This section describes how to install a Python Agent and gives you information about the setup and configuration of the Python Agent.

**This chapter includes:**

- ▶ Diagnostics Python Agent Overview on page 296
- ▶ System Requirements for the Diagnostics Python Agent on page 296
- ▶ Installing Python Agents on page 297
- ▶ Instrumenting a Python Application on page 300
- ▶ Configuring the Python Agent on page 309
- ▶ Description of the Parameters in the Points File on page 316
- ▶ Description of Custom Code on page 318
- ▶ Available Out-of-the-box Configurations on page 328
- ▶ Reconnect/Reinitialize Event Channel After Server Reboot on page 333
- ▶ Troubleshooting on page 333
- ▶ Removing the Python Agent on page 334

## Diagnostics Python Agent Overview

The HP Diagnostics Python Agent install package includes the software necessary to capture events such as method invocations, server requests, and system metrics from Python applications. The Python Agent package must be installed on the systems to be monitored that are running Python applications. Each instrumented application results in a unique probe entity which can be independently configured for data collection. The Diagnostics Python Agent is distributed for all platforms in a single zip file named **HPDiagPythonAgt\_<release number>.zip**.

## System Requirements for the Diagnostics Python Agent

The following sections describe the system configurations that are recommended for hosting the Diagnostics Python Agent.

### Platform Support

The Python Agent has been tested on the following operating systems:

- Win2008 64-bit, Win7 64-bit
- Ubuntu 11.10, 12.04
- RedHat 6.2
- SuSE 12.1

The Python Agent has been tested under the following environments:

Platform/Technology	Versions Certified
Python	2.6.5, 2.6.6, 2.6.8, and 2.7
OpenStack	Diablo, Essex
Django	1.3, 1.4



The Python Agent may work under additional platform versions as well. Please review the Diagnostics Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](http://support.openview.hp.com/sc/support_matrices.jsp) to assess known platform compatibility, or contact HP Support to determine the feasibility of using the Python Agent in your environment.

## **Diagnostics Server Compatibility**

The Diagnostics 9.21 Python Agent requires the Diagnostics 9.20 Server as a prerequisite. Customers running older versions of the server must upgrade to the 9.20 version in order to use the Python Agent. If you are running a later version of the Diagnostics Server, please review the Diagnostics Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](http://support.openview.hp.com/sc/support_matrices.jsp) to assess potential compatibility.

## **Installing Python Agents**

The Python Agent is installed on the same machine as the Python application under test.

This section describes how to install a Python Agent and gives you information about the setup and configuration of the Python Agent.

### **Overview of the Python Agent Installation**

The Python Agent must be installed on all systems running Python application that you wish to monitor. Installation involves simply unzipping the install package and running the `setup.py` script on each system. After this is completed, it is then necessary to define the points that you wish to monitor within your applications. If you wish to monitor OpenStack or Django, configuration files and scripts have been supplied that will allow you monitor these applications.

### **Installing the Python Agent**

The Python Agent installation process includes the following steps:

- 1** Unzip the `HPDiagPythonAgt_<release number>.zip` file. This creates a directory named `HPDiagPythonAgt_<version>`.

- 2** Change directory to **HPDiagPythonAgt\_<version>/pythonprobe-<version>**.
- 3** Execute the **probe\_setup.py** script using the Python interpreter that is used for the monitored application:
  - For Linux  
/`<path to python>/python probe_setup.py`
  - For Windows  
`<path to python>\python.exe probe_setup.py`

## The probe\_setup.py Script

The script **probe\_setup.py** is used to install, upgrade or remove the HP Diagnostics Python Agent.

### Usage:

`probe_setup.py [-h|--help] [-u|--update] [-r|--remove] [-d|--dont_ask]`

### Options:

Option	Description
-h, --help	Show this help message and exit.
-u, --update	Update or upgrade the Python Probe.
-r, --remove	Remove the Python Probe.
-d, --dont_ask	Install or remove the Python Probe without asking.

The **probe\_setup.py** script accomplishes the following steps during the installation:

- 1** Install the **hpdiag modules** in the **site-packages** or **dist-packages** directory of the Python installation (see "Directory Structure" on page 299 for details on where files are installed).
- 2** Install the **hpdiag scripts** in the Python **bin** (Linux) or **Scripts** (Windows) directory.
- 3** Install the PythonProbe configuration files to the **hpdiag/etc** directory.

- 4 Install the systemmetrics binary to the **hpdiag/bin** directory.
- 5 Create the PythonProbe log directory **/var/log/hpdiag** (Linux) or **%PROGRAMDATA%\Hewlett-Packard\hpdiag\log** (Windows).
- 6 Store a list of installed files in **hpdiag/backups/installed\_files**.

## Directory Structure

The Python Agent uses the following directory structure.

### Python Modules

The **hpdiag** Python modules are stored in the Python **site-packages** or **dist-packages** directory as follows:

- On Linux: **/path/to/lib/python[python\_version]/site-packages/hpdiag**
- On Windows:  
**\path\to\lib\python[python\_version]\site-packages\hpdiag**

### Scripts

- On Linux, the **hpdiag** Python scripts are copied into the **bin** directory, where the Python executable also resides.
- On Windows the scripts are installed into the **Scripts** directory under the Python installation directory.

### hpdiag Directory

The HP Diagnostics Python Agent requires a dedicated directory for its configuration and binary files. The location of this directory differs based on the platform and in the case of Windows, is based on the Windows version as well.

- On Linux: **/opt/hpdiag**
- On Windows XP/2k3: **C:\ProgramData\Hewlett-Packard\hpdiag**
- On Windows Vista/7/2k8: **%PROGRAMDATA%\Hewlett-Packard\hpdiag**

### Binaries

The binaries are stored in: **<hpdiag\_dir>/bin**

### **Configuration Files**

The configuration files are stored in: `<hpdiag_dir>/etc`

### **Log Files**

The HP Diagnostics Python Agent creates the following directories for the Python Agent to place its log files:

- On Linux: `/var/log/hpdiag`
- On Windows XP/2k3: `C:\ProgramData\Hewlett-Packard\hpdiag\log`
- On Windows Vista/7/2k8:  
`%PROGRAMDATA%\Hewlett-Packard\hpdiag\log`

## **Instrumenting a Python Application**

There are multiple ways to instrument a Python application, and each is explained below.

- "Using the `hpdiag_instrument.py` Wrapper Script" on page 300
- "Instrument the Main Script of the Monitored Application" on page 303
- "Decorate the Functions and Classes of the Monitored Application" on page 305
- "In Code Creation of Capture Points" on page 306
- "Instrumenting a Single Script" on page 308

### **Using the `hpdiag_instrument.py` Wrapper Script**

The HP Diagnostics Python Agent provides a script to instrument and start an application: `hpdiag_instrument.py`.

No source code change is required in the Python application using this approach. If the main script of the monitored Python application is called "`app_main.py`", for example, then the instrumented application is run by the following command:

```
hpdiag_instrument.py --config app_main.conf --point app_main.point  
app_main.py
```

The script **hpdiag\_instrument.py** initializes the Python probe and reads the capture points from the given point file. Afterwards, it starts the main script of the application via Python's **execfile** function. When the monitored application exits, this script closes all resources of the running probe.

The modules used by the python application are instrumented at runtime when they are imported. The probe uses the custom import hook **sys.meta\_path** as described in the PEP 302 of the Python language. This might conflict with applications that also use this import hook. See "Decorate the Functions and Classes of the Monitored Application" on page 305 for an alternative.

### Usage:

```
hpdiag_instrument.py [--config_dir <config dir>] [--bin_dir <bin_dir>] \
    [--config <config_file>] --point <point_file> \
    [--single] <target_script> [<target_script_args>]
```

### Options:

Option	Description
-h, --help	Show this help message and exit.
-d CONFIGDIR, --config_dir=CONFIGDIR	Configuration directory of the Python Agent.
-b BINDIR, --bin_dir=BINDIR	Binary directory of the Python Agent.
-c FILE, --config=FILE	Python probe configuration file [default = probe.conf]
-p FILE, --point=FILE	Configuration of methods to measure.
-s, --single	Instrument the target_script as well as any modules it loads.  By default, only modules referenced in the target_script are instrumented.

**Parameters:**

The parameters `--config_dir`, `--bin_dir` and `--config` are optional and are only needed when it is desired to use different settings than the defaults.

Option	Default	Environment Variable	Description
<code>-d, --config_dir</code>	<code>/opt/hpdiag/etc;</code> <code>%PROGRAMDATA%\Hewlett-Packard\hpdiag\etc</code>	<code>\$PYPROBE_CONFIG_DIR</code>	Directory containing the configuration files.
<code>-b, --bin_dir</code>	<code>/opt/hpdiag/bin;</code> <code>%PROGRAMDATA%\Hewlett-Packard\hpdiag\bin</code>	<code>\$PYPROBE_BIN_DIR</code>	Directory containing the binary files like 'systemmetrics'
<code>-c, --config</code>	<code>probe.conf</code>	N/A	File containing the probe configuration.

---

**Note:** The specification of the directories as parameter for `hpdiag_instrument.py` has a higher priority than the environment variable settings. The environment variable settings have a higher priority than the defaults.

---

Several examples for starting your application are shown below:

**Example 1:**

```
hpdiag_instrument.py --point webapp.point webapp.py
```

**Example 2:**

```
hpdiag_instrument.py --config my_probe.conf --point webapp.point webapp.py
```

**Example 3:**

```
hpdiag_instrument.py -d /path/to/my/config/data \
```

```
-p other_weapp.point \
webapp.py
```

On Windows, the `path_to_python\python.exe` must be added in front of `hpdiag_instrument.py`.

## Instrument the Main Script of the Monitored Application

It is also possible to initialize and shutdown the Python probe directly from the main script of the Python application (similar to what `hpdiag_instrument.py` does). The code below shows this approach:

```
try:
    from hpdiag import pyprobe
except ImportError:
    class PyProbeDummy(object):
        @staticmethod
        def init(*args, **kws):
            print "Warning: Cannot initialize HP Diagnostics Python Agent. Failed to import
'hpdiag.pyprobe' in file '%s'" % __file__
        @staticmethod
        def shutdown():
            print "Warning: Cannot shutdown HP Diagnostics Python Agent. Failed to import
'hpdiag.pyprobe' in file '%s'" % __file__
    pyprobe = PyProbeDummy

pyprobe.init(config_file = "app_probe.conf", point_file = "app.point")

try:
    def main():
        # call the application entry point here

    if __name__ == '__main__':
        main()

finally:
    pyprobe.shutdown()
```

There are only a couple of lines to be added into the main script:

- 1 The statement to import the module "hpdiag.pyprobe"

- 2** The initialize function "pyprobe.init()" at the beginning
- 3** The shutdown function at the end
- 4** The try-finally block around the original code. This is optional, but highly recommended.

---

**Note:** In WSGI scripts, only the first two lines are needed. Adding the shutdown function at the end will cause the probe to not function properly. See below for more details.

---

The initialize function takes up to four parameters:

- 1 config\_file:** The configuration file for the probe.
- 2 point\_file:** The point file containing the capture points for the instrumented Python application.
- 3 config\_dir:** The directory where the configuration files (probe configuration and point file) are located, if different from the default location.
- 4 bin\_dir:** The directory where the executables (systemmetrics, ...) are located, if different from the default location.

---

**Note:** Please be sure to always specify the parameter for the pyprobe.init() function using keywords like "point\_file = app.point". This allows the parameters to be listed in any order, and also allows for parameters to remain unset so that they will get the default values.

---

All APIs of the python probe are in the module **hpdiag.pyprobe**. Only the functions and classes defined in this module should be used to instrument the monitored application! Functions and classes in all other modules of the Python probe may change without notice at any time!



## Decorate the Functions and Classes of the Monitored Application

It is also possible to create the capture points in the Python source at run-time by using the following decorator functions from the module `hpdiag.pyprobe`: `func_point`, `method_point`, and `class_point`. They are used as decorators directly in the Python source above the instrumented function, method, or class. The supported arguments for these decorators are exactly the same arguments as those for the capture points in the point file. For example:

```
from hpdiag import pyprobe

@pyprobe.class_point(method = "^fib$|rfib ", layer = "fiboLayer")
class Fibo(object):
    # the implementation of the Fibo class
```

This decorator creates one capture point for the class **Fibo**. The method's argument specifies that the method `fib` and the method `rfib` should be instrumented. Please note that Python regular expressions are used here. The regular expression `^fib$` means that only the method `fib` is instrumented whereas the regular expression `rfib` means that any method that has a sub-string `rfib` in its name will be instrumented (for example, also `rfib_seq`).

The argument **layer** defines the layer for all instrumented methods of this class. The other mandatory arguments **class** and **module** are automatically determined by the decorator.

It is also possible to decorate a single function or method using the **func\_point** and **method\_point** decorator. For example:

```
class Fibo(object):
    @pyprobe.method_point(clazz = "Fibo", layer = "fiboLayer", args = "0")
    def fib_seq2(self, n):
        # the implementation of the method
```

Even though the decorator is executed in the context of the method, it is necessary to specify the name of the class because the class is not yet defined (and so cannot be automatically determined) at the time the decorator is executed.

---

**Note:** Please also note that the argument name is `clazz` because `class` is a Python key word which cannot be used.

---

If the instrumented function or method already has other decorators (for example, it is a `@staticmethod` or `@classmethod`), then the decorator that creates the capture point for the probe must be written directly above the function or method (if not it might cause problems). For example:

```
@_DecoMemoize
@pyprobe.func_point(layer = "fiboLayer", args = 0)
def mfib(n):
    # the implementation of the function
```

Please note that all three decorator functions are executed at import time of the module and create just the capture point. The automatic instrumentation of the module via the above described import hook is performed after the module was loaded. Thus, the decorated functions, methods, and classes are treated like any capture point read from the point file.

## In Code Creation of Capture Points

If you do not want to add the decorators in all the source files of the monitored application (or if the sources are not available at all), it is also possible to create all the capture points in one place within your application.

```
from hpdiag import pyprobe
pyprobe.init(config_file = "probe.conf", point_file = "app.point")
pl = pyprobe.PointList()
pl.create_method_point("func_name", "module_name", <point arguments>)
pl.create_method_point("method_name", "module_name", "class_name", <point arguments>)
pl.create_class_point(<class instance>, <point arguments>)
pl.create_point(<point arguments>)
```

The point arguments are the same as the options of a point in the point file, for example, `layer="Database"`, `detail="is_sql_statement"`.

Once all points are created, it is possible to trigger the instrumentation by calling:

```
pyprobe.instrument(pl)

# call the actual application entry point

pyprobe.shutdown()
```

Passing the point list to the instrument function ensures that only the newly created points are instrumented. Capture points that were read from the point file (passed as second parameter to the init function) are, by default, automatically instrumented at import time of the module (using the custom import hook describe above).

Please note that the point file that is passed to the init function is optional! If not specified, only the capture points created by the decorator functions are used by default, that is, if the automatic instrumentation at import time is enabled.

It is possible to disable the automatic instrumentation at import time. Use the following argument in the probe section of the probe config file to do this:

```
[Probe]
auto_instrument = True
```

If this argument is set to False (the default is True), the modules imported by the monitored application are not automatically instrumented. Instead, it is possible to trigger the instrumentation any time at runtime by calling:

```
pyprobe.instrument()
```

Because no point list is passed as parameter to the function, it will use all capture points that were created so far at runtime and/or read by the init function from the point file to instrument the currently loaded modules.

## Instrumenting a Single Script

A single script is characterized by the fact that it is not imported by another script. Thus it is more difficult to instrument such a script. If a script can be instrumented or not depends on the availability of classes and methods inside the script.

### Prerequisites

The `hpdiag_instrument.py` tool allows the execution of instrumented single scripts by loading it as a module and calling its `main()` method. This means that the existence of a `main()` function in the script is a prerequisite. Simple Python scripts often have no `main()` method, but look like this:

```
if __name__ == '__main__':  
    instance = MyClass()  
    :  
    xyz = helper_function()
```

In most cases this can be easily changed to:

```
if __name__ == '__main__':  
    instance = MyClass()  
    :  
    xyz = helper_function()
```

In most cases this can be easily changed to:  
`def main():`

```
    instance = MyClass()  
    :  
    xyz = helper_function()  
  
if __name__ == '__main__':  
    main()
```

This allows access to the `main()` method by the `hpdiag_instrument.py` tool, and thus to instrument this single script.

## Point Definitions

The script is imported with its file name as module name, so that its name is referenced in the point file as module name to define the instrumentation points. For example when the script name is myScript.py then this is imported as 'myScript' and might be referenced in the point file as follows:

```
[myScript_1]
module = myScript
class = MyClass
method = class_method
layer = myscript
```

---

**Note:** Because single scripts are imported as module, the file name must not contain any dots ('.'). For example myScript-0.2.py does not work because dots are not allowed in module names. Correct is myScript.py or my\_script.py.

---

## Calling hpdiag\_instrument.py

The hpdiag\_instrument.py tool has the parameter '-s | --single' to indicate that the called Python script is a single script:

```
hpdiag_instrument.py --config myScript.conf --point myScript.point \
    --single /path/to/myScript.py --script_par1 ...
```

## Configuring the Python Agent

The file <hpdiag\_dir>/etc/probe.conf drives the basic agent behavior. The probe.conf file has section/namespaces. Configuration parameters are defined within these namespaces.

The following sections give detailed descriptions of the configuration parameters in the probe.conf file. Also included are two sections that give details on some specific URI replace pattern configurations in the probe.conf file.

## Namespace [Mediator]

- **hostname:** The Diagnostics mediator host name.
- **port:** The Diagnostics mediator port number.
- **channeltype:** One of synchronous, threaded, or multiprocessing. This value configures how events are sent to the mediator. Python has a very peculiar threading behavior, so testing may be necessary to determine the optimal settings for your application.
  - **synchronous:** The events are sent as part of the business application thread. This might slow down the business application.
  - **threaded:** The events are sent in a separate thread, but in the same process as the business application. This is the default.
  - **multiprocess:** The events are sent in a separate process.
- **reconnect\_timeout:** The timeout in seconds before the next reconnect, in case the connection to the mediator has been lost. Server requests that complete while the mediator connection is unavailable are dropped silently.
- **keep\_alive\_interval:** Interval in seconds at which the probe will send keep alive messages to the registrar on the mediator.

## Namespace [Logging]

- **class:** Specify the logging (handler) type. There are two types supported:
  - **TimedRotatingFileHandler:** It supports rotation of disk log files at certain timed intervals.
  - **RotatingFileHandler:** It supports rotation of disk log files based on file size limits.
- **file:** The absolute path to the log file.
- **level:** The default logging level: CRITICAL, ERROR, WARNING, INFO or DEBUG.

- **level\_exceptions:** Specify exceptions to the default logging level of the Python probe. These exceptions are specified as Python dictionary with a Python pattern as key and the logging level as value (in the form of a string). The probe iterates through all keys (patterns) of the dictionary and will use the first one that matches. The order is not defined, however.

The example below sets the DEBUG level to all loggers in modules that start with `hpdiag.location`. Likewise, it sets the INFO level to all loggers in modules that start with `hpdiag.importhook`:

```
level_exceptions = {'hpdiag.location.*' : 'DEBUG', 'hpdiag.importhook.*' : 'INFO'}
```

- **backup\_count:** If nonzero, at most `backup_count` files will be kept. If more would be created when roll-over occurs, the oldest one is deleted.
- **max\_file\_size:** For `RotatingFileHandler`: The maximum size of the log file in MB.
- **when:** For `TimedRotatingFileHandler`: Rotating happens based on the product of `when` and `interval`. Possible values are:
  - 'S' Seconds, 'M' Minutes, 'H' Hours, 'D' Days, 'W#' Week day (# = 0 - 6 with 0 = Monday), or 'midnight' Roll over at midnight.
- **interval:** For `TimedRotatingFileHandler`: The roll-over interval. Example: If `when` is set to '1#' (= Tuesday) and `interval` is set to '2', then the log file will be rolled over every second Tuesday.
- **utc:** Use times in UTC (default is local time).

## Namespace [Probe]

- **probe\_id:** The name of the probe instance. Add %0 to the `probe_id` to get a unique probe name if several instances of the same probe are running on the same system.
- **registered\_hostname:** The hostname to be used if DNS/IP lookups don't work reliably.
- **probe\_group:** Probe Group name (used in the same manner as in the Java and .NET probes).
- **system\_group:** System Group name (used in the same manner as in the Java and .NET probes).

- ▶ **auto\_instrument:** Enable/disable automatic instrumentation at import time (default: True).
- ▶ **instrument\_loaded\_modules:** Instrument modules that have been loaded before `pyprobe.init()` is called (default: False).
- ▶ **instrument\_pyprobe\_threads:** Instrument points found in the probe threads, e.g. monitor the probe itself (default: False).
- ▶ **error\_on\_duplicate\_location:** An exception is thrown whenever the same location is instrumented multiple times (default: False).
- ▶ **sql\_parsing\_mode:** Parsing mode of SQL queries.
  - ▶ 1 = just methods, no SQL queries
  - ▶ 2 = main categories for SQL queries (select/update/insert/delete/...)
  - ▶ 3 = a measurement per whole SQL query aggregating similar statements into a single measurements
  - ▶ 4 = a measurement per whole SQL query aggregating only identical statements
- ▶ Agent side trimming:
  - ▶ **maximum\_stack\_depth:** Don't capture any data about methods called at a depth greater than this. For example, if `maximum_stack_depth` is 3, and `"/login.do"` calls `a()` calls `b()` calls `c()`, only `login.do`, `a`, and `b` will be captured. Setting a low `maximum_stack_depth` can somewhat reduce the overhead of capture. Setting this to a very high value disables depth trimming. This is dangerous if potentially recursive methods are instrumented as it can lead to nearly infinite call-trees. This will consume a lot of memory. Setting this value above 100 is strongly discouraged. The default is 25.
  - ▶ **minimum\_method\_latency:** Latency trimming - never capture any data about regular methods that execute faster than this number of milliseconds. Depending upon your platform & whether hi-res time stamps are being used, it may not be useful to specify this value in increments of less than 10ms. It defaults to 51 milliseconds.
  - ▶ **minimum\_fragment\_latency:** If an entire server request takes less than this number of milliseconds, it will not be captured, unless a threshold has been set on that server request. The default value is 51ms.



- ▶ **maximum\_method\_calls:** Tree size trimming - never capture more than this number of methods per instance tree. This is regardless of latency and depth trimming. It defaults to 1000. Note that this applies to all methods, including outbound calls.
- ▶ **minimum\_sql\_latency:** If an SQL statement takes less than this amount of time, it will not be trended, until it does exceed this time. It defaults to 1000 milliseconds (one second).
- ▶ **httpserver\_port:** Port to use for python probe http server.
- ▶ **http\_client\_show\_url:** Enables/disables the inclusion of the URL as part of the identity of an outbound call. The value should be set to false for REST service client applications.
- ▶ **uri\_replace\_pattern:** A comma-separated list of pattern replacement operations to attempt on each URI (see "URI Truncation and Mapping" on page 314).
- ▶ **uri\_pathsegments:** Number of URI path segments to allow (see "URI Path Segment Trimming" on page 316).
- ▶ **username:** User name used to authenticate the mediator with the probe http server. If it is empty, a default user name will be used.
- ▶ **password:** Password used to authenticate the mediator with the probe http server. Use the utility **hpdiag\_encodepassword.py** to encode your password before adding it there. If it is empty, a default password will be used.

## Namespace [SystemMetricsCollector]

- ▶ **enabled:** True or False, decides whether the system metrics collector is active.
- ▶ **sampling\_rate:** How fast should a metric be locally sampled. Uses time string values, like 5s.
- ▶ **metrics\_group:** What group should system metrics be associated with? This value may be the same as an existing probe group, or completely independent.
- ▶ **udp\_port:** Port to use for system metrics UDP control port. Do NOT modify this unless there is a conflict with another application. All Diagnostics agents on a system MUST be configured to use the same port.

- ▶ **mediator\_port**: Port on the mediator used to deliver metrics to.
- ▶ **udp\_retry\_interval**: How often should the metrics collector try to open the UDP port in case it is in use by another program. Uses time string values, like 10min.
- ▶ **username**: User name used to authenticate the system metrics collector with the mediator.
- ▶ **password**: Password used to authenticate the system metrics collector with the mediator. Use the utility `hpdiag_encodepassword.py` to encode your password before entering it here.

## Namespace [SystemMetrics]

This namespace contains the system metrics to collect.

These system metrics collector entries use the same layout as the ones for the Java Agent (see Chapter 20, "Java Agent - System Metrics Capture") with the exception that the collector name is not available in the Python agent.

## URI Truncation and Mapping

It is possible to truncate or change the URI of a request using Python regular expressions. This is specified in the `probe.conf` file in the option **uri\_replace\_pattern**. This is a comma-separated list of pattern replacement operations to attempt on each URI. This is useful to replace many server request URIs with one simplified server request URI that aggregates them. The truncation or mapping of URIs is done using the `'s/pattern/replace/'` syntax, which is the only supported syntax for the URI replacement patterns.

## How and Where are the Patterns Used

This functionality is applied after `before:code` custom functions, `args:name` or `args:n` were applied. The output of `before:code` or `args:x` is used as input for the URI replacement patterns.

If more than one pattern is specified, all patterns will be applied. The patterns are applied in order. The output of a previous matched pattern will be used as input for the next pattern. The resulting string is used in the Diagnostics GUI for the request name.

## Characteristics

Because `s/pattern/replace` is not Python syntax, it is necessary to use `'#'` instead of `'/'` in the configuration file

`s/pattern/replace/`

must be written as

`s#pattern#replace#`

`s/pattern/replace/` is used to be comparable with the syntax in Perl or on the Unix shell. It is also possible to omit the `s` and write `#pattern#replace#`.

## Examples

Truncate before a string, match the string and any characters that follow it and leave `replace` empty. In this example `$` matches end-of-line.

```
uri_replace_pattern = s#string.*$##
```

Truncate after a string. Match the string in a grouping and use `\group-number` to put the string into the replacement.

```
uri_replace_pattern = s#(string).*#\1#
```

Use a comma separated list to perform multiple operations. The operations will all be performed in order. This would change every `foo` to `bar` and then change every `bar` back to `foo`.

```
uri_replace_pattern = s#foo#bar#,s#bar#foo#
```

Truncate before any semicolon.

```
uri_replace_pattern = s#;. *$##
```

Truncate before any `/!` or `!.` This uses `?` to say that the slash is optional.

```
uri_replace_pattern = s#/?!\. *$##
```

Truncate before any `';` or `'/!` or `'!`.

```
uri_replace_pattern = s#(;|/?!\.)*$##
```

Map `/django/portal/` and `/django/myportal/` to Django Portal.

```
uri_replace_pattern = s#^/django/(my)?portal/#Django Portal#
```

Other examples:

```
uri_replace_pattern = s#(;/  
?!).*###,s#.*\.(js|css|jpg|gif|png|pdf|html|jar|class)$#Static Content#
```

```
uri_replace_pattern = s#.*([a-zA-Z0-9_]*).py#\1#
```

## URI Path Segment Trimming

The URI path can be trimmed by the definition of **uri\_pathsegments** in the **probe.conf** file. **uri\_pathsegments** is set to the number of URI path segments to allow - everything after this point will be trimmed. For example, with a setting of 2, URLs like `/foo/bar/1`, `/foo/bar/2` will be trimmed to `/foo/bar`. A value of -1 or 0 will disable the path trimming.

## Description of the Parameters in the Points File

The points file specifies which Python classes, methods and functions are monitored.

The syntax of the points file is the same as for the Java probe. Therefore see the Java probe documentation for details.

The following arguments are supported:

Argument	Description	Mandatory
module	A Python regular expression	yes
class	A Python regular expression	no
method	A Python regular expression	yes
layer	The name of the layer	yes

Argument	Description	Mandatory
layer_type	One of the following 4 values: <ul style="list-style-type: none"> <li>➤ method (the default)</li> <li>➤ trended_method</li> <li>➤ portlet</li> <li>➤ sql</li> </ul>	no
detail	Specifies more specific capture instructions. It is a comma-separated list of the following: <ul style="list-style-type: none"> <li>➤ before:code:&lt;name&gt;: execute the custom code with filename &lt;name&gt; before the instrumented method/function</li> <li>➤ after:code:&lt;name&gt;: execute the custom code with filename &lt;name&gt; after the instrumented method/function</li> <li>➤ args:name: uses the string representation of the instance on which the instrumented method was called as call argument</li> <li>➤ args:n: uses the string representation of the argument on index 'n' as call argument in the GUI (see more details below)</li> <li>➤ is_sql_statement: marks methods/functions that execute SQL statements</li> <li>➤ inbound: marks a method/function as inbound call that is used to track cross-VM transactions</li> <li>➤ outbound: marks a method/function as outbound call that is used to track cross-VM transactions</li> <li>➤ 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.</li> <li>➤ method_no_trim: indicates that no latency-based trimming should take place when a method instrumented by this point is executed.</li> <li>➤ no_layer_recurse: prohibits recording of any methods called from the method instrumented by this point, unless the callee belongs to a different layer.</li> </ul>	no

For example:

```
[httplibHTTPConnectionOutbound]
module = httplib
class = HTTPConnection
method = request
layer = Sending
detail = outbound,before:code:httpconnection_outbound
```

To distinguish a method of a class from a function within a module, the Python agent introduces the additional argument “module” and considers the class argument as optional. Thus, a point describes either a set of module functions or a set of class methods. If both functions as well as class methods within the same module should be captured, it is necessary to specify two different points.

### Including Points Files

The point file referenced during the instrumentation can include other point files. This is done by using the special point IncludePoints. The file references have to be relative to the location of the main point file.

For example:

```
[IncludePoints]
1 = ../etc/httprequest.point
2 = httpserver.point
3 = others/database.point
```

## Description of Custom Code

Custom code are Python functions that can be executed before or after the monitored method or function is executed. These functions are stored in files in the Python agent custom\_code directory. The custom code functions used are defined in the points file and are specified separately for each monitored function. The custom code functions are referenced by file name.

The following sections gives details about custom code.

- "The Purpose of Custom Code" on page 319

- "Custom Functions" on page 320
- "Returning HTTP Request Status Codes" on page 324
- "Cross VM Server Requests" on page 324
- "Argument Extraction" on page 328
- "Available Out-of-the-box Configurations" on page 328
- "URI Path Segment Trimming" on page 316

## The Purpose of Custom Code

Custom code can be used in the Python Probe to extract data from the arguments passed into an instrumented function or method. If this data is returned by the custom code, it will be displayed as an argument of the method in the Diagnostics GUI (in the call profile). With custom code, it is even possible to modify the arguments of an instrumented function or method. Custom code is also used to track calls between multiple probe installations (cross VM calls).

Custom code can be called two times: before the instrumented method is called (before:code) and after it was called (after:code).

### before:code

The **before:code** is used to extract data from the argument list of the instrumented method. If this extracted data (for example, a URI) is returned by the custom function, it will be displayed in the Diagnostics GUI as call argument.

The custom code functions are also used to intercept information that is needed for correct display in the Diagnostics UI. The custom code function can return a string (used as the argument of the call, as explained above), a dictionary, or a tuple of both. In the dictionary, the following entries are used by the probe to report data to the server:

Key	Meaning
uri	URI of an incoming http service request.
inbound_coloring	Coloring token of an inbound call used to track cross-VM transactions.

Key	Meaning
remote_ip	Caller IP address of an inbound call.
diag_arg	The diag argument required for both incoming and outgoing calls.

Server requests are reported as inbound to the Diagnostics server if a coloring token has been reported by any method in the call stack.

If any method reported an URI, the server request type is reported as 'HTTP'; otherwise it will show up as "Pseudo"

Check the files in **etc/custom\_code** for syntax and usage examples of custom code, especially the way the coloring tokens are injected and retrieved from the calls.

### **after:code**

The **after:code** can be used to do any processing that might be useful after the instrumented method was called.

### **Custom Functions**

All custom code needs to be written as a function with the name `custom_fct_before(...)` or `custom_fct_after(...)`. The custom function that is used for `before:code` takes the following argument list:

`custom_fct_before(instance, location, args, kws)`

- ▶ **instance:** the class instance on which the instrumented method is called. It is None for instrumented module functions.
- ▶ **location:** the python probe location object that identifies the instrumented function/method.
- ▶ **args:** the tuple of positional arguments passed to the instrumented function/method.
- ▶ **kws:** the dictionary of keyword arguments passed to the instrumented function/method.

It can return the following values:



- ▶ **method argument:** the argument string for the instrumented method as displayed by the Diagnostics GUI in the call profile.
- ▶ **a dictionary:** a dictionary of key value pairs. This dictionary is passed to the before:after function after the instrumented method got called. There are two special keys in this dictionary, however. If custom\_fct\_before adds the keys "method\_args" and/or "method\_kws" to this dictionary, it is assumed that they represent the modified argument list of the instrumented function/method being called. The value for key "method\_args" must be of type 'tuple' and the value of key "method\_kws" must be of type 'dict'. If the instrumented method is an outbound call, then this dictionary has to contain the key "diag\_arg". If it is an inbound call, it has to contain the keys "diag\_arg", "inbound\_coloring" and "remote\_ip".
- ▶ **a tuple:** both values described above wrapped into a tuple.
- ▶ **None:** the custom code function may also return None.

The custom function for after:code takes the following parameters:  
custom\_fct\_after(instance, location, method\_return\_value, code\_dict).

- ▶ **instance:** the class instance on which the instrumented method is called. It is None for instrumented module functions.
- ▶ **location:** the python probe location object that identifies the instrumented function/method.
- ▶ **return value:** the return value of the instrumented function or method.
- ▶ **a dictionary:** the dictionary that was returned by the before:code function.

Example for before:code in the file custom\_code/cust\_example\_before.py:

```
# Used by [DiagShop]

from urlparse import urlparse

def custom_fct_before(instance, location, args, kws):

    ret_val = None
    purl = urlparse(str(args[0]))
    if len(purl.scheme) == 0:
        ret_val = ""
    else:
        ret_val = purl.path

    return ret_val
```

The file name cust\_example\_before is used as reference of the custom code to be used in the point file. The function name is always custom\_fct\_before(instance, location, args, kws). This code would be referenced in the point file via the following:

```
detail = before:code:cust_example_before
```

Example for after:code in the file custom\_code/cust\_example\_after.py:

```
def custom_fct_after(instance, location, method_return_value, code_dict):

    print "CustomAfter: Custom code executed - does not return anything."
```

It is possible to define a custom\_fct\_before(...) function and a custom\_fct\_after(...) function in the same file and reference it using the same name. Which function is be used is defined in the detail section in the point file.

---

**Note:** The Python Probe imports the custom code files as Python modules. This means that all limitations regarding the file names for Python modules also apply to the custom code files. For example the characters (<space>) or ('-') are not allowed in file names

---

## Using Sub-directories

Because the custom code files are handled as Python modules by the Python Probe, it is also possible to categorize custom code files in sub-directories (modules). If this is desired, each sub-directory needs to have a Python special file in it - this is the file `_init_.py`. This file can be empty, but must be there to be able to import custom code from a sub-directory. Example:

```
pyapp_code
|- get_http_request.py
|- get_request_2.py
|- pyapp_controller
|  |- __init__.py
|  |- get_details.py
|  |- do_something.py
|- pyapp_scheduler
|  |- __init__.py
|  |- get_request_from_queue.py
|  |- get_service_request.py
```

With these files in place, the files of this structure can be referenced in the point file for example via

```
detail = before:code:get_http_request
```

from the custom code base directory or for the pyapp controller from the `pyapp_controller` sub-directory:

```
detail = before:code:pyapp_controller.get_details
```

---

**Note:** A `_init_.py` file is not needed in the custom code base directory, because the files in this directory are not regarded as Python modules.

---

## Returning HTTP Request Status Codes

For each HTTP request the HTTP server returns a status code. The custom code can be used to report this status code to the HP Diagnostics server. To do this, the location object, passed to the before and after functions, implements the method `add_request_attribute`. It takes the attribute name and the attribute value as parameters. At the moment, only the following four attributes are supported by the HP Diagnostics server:

- `WS_consumer_id`
- `HTTP_status_code`
- `HTTP_status_desc`
- `tcp_server_port`

The following example shows how to extract the HTTP status code of requests to django applications and have it sent to the HP Diagnostics server:

```
def custom_fct_after(instance, location, method_return_value, custom_code_dict):
    from django.core.handlers import wsgi

    try:
        status_text = wsgi.STATUS_CODE_TEXT[method_return_value.status_code]
    except KeyError:
        status_text = 'UNKNOWN STATUS CODE'

    if method_return_value.status_code >= 500 and method_return_value.status_code
    <=699:
        location.add_request_attribute("HTTP_status_code",
        str(method_return_value.status_code))
        location.add_request_attribute("HTTP_status_desc", status_text)

    return None
```

## Cross VM Server Requests

### Outbound Calls

To enable HP Diagnostics to connect calls made from one instrumented application to another, a unique identifier (coloring) needs to be added to the data sent to the called application. This can be done with custom code.

The following example is used to instrument the request method of the python `httplib.HTTPConnectionOutbound` class. It shows how to get the coloring from the probe using the `location.get_outbound_coloring` call which takes the called target as parameter. The next step is to add it to the data which will get sent to the called application.

`location.create_diag_envelope` will either add it to the data to be sent (passed as second parameters) or will return an encoded version of the coloring if no data is passed. In the latter case, you have to add the coloring to the request yourself. The data to be sent has to be a str for the enveloping to work! This example adds it as an additional HTTP header called `X-Mercury-Diag-HTTP-Color`.

Then a string called `diag_arg` needs to get constructed which must be passed back to the Python probe via a dictionary (using the dictionary key `"diag_arg"`). In case the arguments of the instrumented methods are modified within the custom code, they also have to be passed back to the probe via the returned dictionary (using the keys `"method_args"` for the positional arguments and `"method_kws"` for the keyword arguments).

File `httpconnection_outbound.py`:

```
# Used by [httplibHTTPConnectionOutbound]

import httplib

def custom_fct_before(instance, location, args, kws):

    if isinstance(instance, httplib.HTTPSConnection):
        url = "https://%s:%s/%s"
    elif isinstance(instance, httplib.HTTPConnection):
        url = "http://%s:%s/%s"
    else:
        url = "request://%s:%s/%s"

    outbound_coloring = location.get_outbound_coloring(url % (instance.host,
instance.port, args[1]))
    outbound_coloring = location.create_diag_envelope(outbound_coloring, "")

    if (args[3]):
        args[3]['X-Mercury-Diag-HTTP-Color'] = outbound_coloring
    else:
        args[3] = {'X-Mercury-Diag-HTTP-Color' : outbound_coloring}

    param_dict = {'name': '{0};{1}'.format(instance.host, instance.port),
                  'target': '{0};{1}'.format(instance.host, instance.port)}
    diag_arg = location.create_diag_arg('http', param_dict)
    result = {}
    result['diag_arg'] = diag_arg

    result['method_kws'] = kws
    result['method_args'] = args

    return result
```

## Inbound Calls

In inbound calls, the custom code is used to remove the coloring from the request received and pass it to the python probe.

The following example is used to instrument the WSGI handler of the Django framework. It removes the coloring from the request, passed as the X-Mercury-Diag-HTTP-Color parameter using the `location.get_coloring_from_diag_envelope` method. The coloring is then returned to the python probe. In addition to the coloring, a `diag_arg` string and the IP address of the calling application and the called URI needs to get returned.

File `basehttprequesthandler_inbound.py`:

```
# Used by [BaseHTTPServerBaseHTTPRequestHandlerInbound]

import BaseHTTPServer, socket

def custom_fct_before(instance, location, args, kws):

    result = {}
    path = None

    if 'X-Mercury-Diag-HTTP-Color' in instance.headers:
        inbound_coloring =
location.get_coloring_from_diag_envelope(instance.headers['X-Mercury-Diag-HTTP-C
olor'])
        del (instance.headers['X-Mercury-Diag-HTTP-Color'])
        result['inbound_coloring'] = inbound_coloring

    if isinstance(instance, BaseHTTPServer.BaseHTTPRequestHandler):
        host, port = instance.client_address[:2]

        param_dict = {'name': instance.path, 'target': instance.headers['host']}
        diag_arg = location.create_diag_arg('http', param_dict)

        path = instance.path
        result['diag_arg'] = diag_arg
        result['remote_ip'] = host
        result['uri'] = path

    return (path, result)
```

## Argument Extraction

### **args:name**

**args:name** uses the string representation of the instance on which the instrumented method was called as call argument. For class or static methods or a module function, it returns the doc string of the instrumented function. If no doc string exists, it returns the module name.

### **args:n**

**args:n** uses the string representation of the argument on index 'n' as call argument in the GUI. 'n' can be in the range 0 - 254.

**args:name** and **args:n** can be used together with an **after:code** custom function, but not together with a **before:code** custom function. If a **before:code** function is referenced and **args** is used, it is undefined as to which one will be used.

## Available Out-of-the-box Configurations

The Python Agent comes with a number of out-of-the-box configurations as ready-to-use configuration or as starting point for own configurations. Currently available is instrumentation for:

- ▶ OpenStack Diablo and Essex releases
- ▶ Django and WSGI

### **OpenStack Instrumentation**

The Python Agent provides configuration for the instrumentation of the OpenStack cloud computing platform (Diablo and Essex Release).

For OpenStack, the following is provided in addition to the standard python agent:

- ▶ Points files for every component of OpenStack
- ▶ Setup scripts and configuration files for OpenStack



## Point Files

For every component of OpenStack one or more ready-made point files can be installed and used.

- common.point
- dashboard.point
- glance.point
- keystone.point
- nova-api.point
- nova-general-controller.point
- nova-queue-send.point
- nova-scheduler.point
- setup-openstack.conf
- setup-openstack.txt
- swift-common.point
- swift-account-server.point
- swift-container-server.point
- swift-object-server.point
- swift-proxy-server.point

## Setup of the OpenStack Instrumentation

The startup scripts of the OpenStack components that need to be instrumented must be changed in order to start the instrumentation together with a particular configuration. This can be done using a setup script **hpdiag\_setup\_openstack.py**.

```
hpdiag_setup_openstack.py -i|--install <os_version> | \
    -u|--uninstall <os_version> \
    [-m|--mediator <hostname_fqdn>] [-h|--help]
```

Install OpenStack instrumentation:

-i --install <os\_version> Install the OpenStack instrumentation

-u --uninstall <os\_version> Remove the OpenStack instrumentation

-m --mediator <mediator> Hostname of the Diagnostics Server

-h --help Display this message

OpenStack versions:

essex OpenStack 2012.1

diablo OpenStack 2011.3

The setup script uses the information about which component will be instrumented and where to find its start script that is provided in the **setup\_openstack.conf** file.

The **setup\_openstack.conf** file has the following syntax:

<probe id>:<absolute path to the start script>:<pyprobe.init call>

For example:

```
nova-compute:/usr/bin/nova-compute:pyprobe.init(config_dir="/opt/hpdiag/etc/
openstack", \
bin_dir="/opt/hpdiag/bin", config_file = "nova-compute.conf", \
point_file = "nova-general-controller.point")
```

In addition to setting up the instrumentation in the OpenStack start up scripts, the script **hpdiag\_setup\_openstack.py** creates a configuration file for each component of OpenStack. It uses the default configuration file **probe.conf** in **/opt/hpdiag/etc** as master and creates a copy for each component. Each configuration file contains the hostname of the Diagnostics server (mediator) and the probe ID which will be displayed in Diagnostics' user interface. The hostname and the probe ID are added to the component's configuration file automatically. The name of the configuration file is **<probe\_id>.conf**.

The instrumentation steps are:

**1** Stop all OpenStack processes

For

Diablo go into `/opt/hpdiag/etc/openstack-diablo`

Essex go into `/opt/hpdiag/etc/openstack-essex`

**2** Call `hpdiag_setup_openstack.py` to instrument the OpenStack components Swift, Nova, Glance, Keystone, and the dashboard. The script creates various `*.conf` files for the various Python probes that monitor OpenStack.

```
> hpdiag_setup_openstack.py -m <diagnostics_server_name> -i essex|diablo
```

When `-m` is omitted, then the hostname will be taken from the file `/opt/hpdiag/etc/probe.conf`. You may edit this file to set the HP Diagnostics server name for the OpenStack instrumentation.

**3** Restart the OpenStack services.

For all Swift servers only one Python source file is modified: `/usr/share/pyshared/swift/common/wsgi.py`. It is the central entry point for most Swift processes. The inserted `pyprobe.init` function call looks as follows:

```
pyprobe.init(config_dir = "/opt/hpdiag/etc/openstack-essex",
             config_file = "swift-" + log_name + ".conf",
             point_file = "swift-" + log_name + ".point")
```

As you can see, the name of the `*.conf` and `*.point` files is built based on the `log_name` variable. It must be "proxy-server", "account-server", "container-server", or "object-server" to match the generated files from `hpdiag_setup_openstack.py`. To ensure this, check the swift config files in `/etc/swift`. For example, the default `log_name` for the Swift proxy server in the Essex release is "swift-proxy". This does not match the generated `*.conf` files. Thus, edit the file `/etc/swift/proxy-server.conf` and change the value for `log_name` in the section `[app:proxy-server]` to "proxy-server". Alternatively, you can also rename the generated Swift `*.conf` and `*.point` files if you do not want to change the files in `/etc/swift`.

The original OpenStack scripts are preserved in `/opt/hpdiag/backup`. The instrumentation can be removed with the command `hpdiag_setup_openstack.py \-u essex\|diablo`.

## Django and WSGI Instrumentation

The Python Agent provides configuration for the instrumentation of the Django and WSGI. The provided point files can be used for that. The Django point file is:

django.point

This instruments a point in the WSGI handler, that provides the request information:

```
[DjangoWSGIHandlerInbound]
module = django.core.handlers.wsgi
class = WSGIHandler
method = __call__
layer = WSGIHandler
detail = inbound,before:code:django_wsgi_call_inbound
```

## Setup of the Django Instrumentation

The Django WSGI script needs to be changed to instrument a Django application. The Python Probe initialization needs to be done in that script.

Example Script:

```
import os, sys

# ---- Start of PyProbe section
# Calculate the path based on the location of the WSGI script.
sys.path.append(os.path.dirname(__file__))
sys.path.append('<path_to_the_app>')

# Instrument the application
from hpdiag import pyprobe
pyprobe.init(config_dir = '/opt/hpdiag/etc/mysite',
             bin_dir = '/opt/hpdiag/bin',
             config_file="probe.conf",
             point_file="mysite.point")
# ---- End of PyProbe section

os.environ['DJANGO_SETTINGS_MODULE'] = 'settings'
import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()
```

---

**Note:** It is recommended to put the original start-up code into a **try:** / **except:** / **finally:** block where the **finally:** executes **pyprobe.shutdown()**. This is not recommended for WSGI scripts because the WSGI scripts are executed and terminated for every request. Calling **pyprobe.shutdown()** would launch a new probe every time, which strongly reduces the correlation and presentation quality in the HP Diagnostics UI.

---

## Reconnect/Reinitialize Event Channel After Server Reboot

In case the Diagnostics server has been rebooted or shut down for some reason the python probe gets disconnected from the server. In this case everytime the probe wants to send data to the server it tries to reconnect first. In order to avoid that reconnection attempts occur too often, the probe only tries to reconnect to the server after a timeout. By default this timeout is set to 5 seconds. The value can be modified in the configuration file. See "Configuring the Python Agent" on page 309 for more information about this value. While the probe is disconnected from the server all collected data will be deleted. After the server is running the probe gets reconnected automatically and continues to send collected data. The maximum time needed for a reconnection after the server is up and running again, is the reconnection timeout mentioned above.

## Troubleshooting

Rotating log files are known to result in errors on Windows. The workaround is to set file size or the rotation interval in the **probe.conf** file to large values to ensure that rotation never happens.

## Removing the Python Agent

During the installation of the HP Diagnostics Python Agent, the installation script was copied into the **hpdiag** directory with the name **probe\_deinstall.py**. Executing this script will remove the Python Agent.

---

**Important:** Please make sure that no application is instrumented and that no probe is running when the probe will be removed. If a probe is still running on Windows, then the rename of the **hpdiag** directory will fail and an error is returned. On a Linux system we cannot detect if a probe is still running during uninstall. This may lead to unpredictable results.

---

On Linux:

```
/path/to/python /opt/hpdiag/probe_deinstall.py
```

On Windows:

```
cd %PROGRAMDATA%\Hewlett-Packard
```

```
\path\to\python hpdiag\probe_deinstall.py
```

Please note that on Windows it is necessary to call this script from outside of the **hpdiag** directory, because the hpdiag directory will be renamed during the de-installation. This rename fails when the console is opened in the hpdiag directory.

The deinstallation script will perform the following steps:

- 1** Remove the probe Python files from the Python **site-packages** directory.
- 2** Remove the **.egg-info** file.
- 3** Rename the hpdiag directory to **hpdiag.<date>-<time>**.

# Part IV

---

## **Custom Instrumentation for Monitoring Java and .NET Applications**

This section includes:

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





# 10

---

## 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 338
- Coding Points in the Capture Points File on page 340
- Defining Points With Code Snippets on page 348
- Controlling Class Map Capture on page 364
- Instrumentation Examples on page 365
- Understanding the Overhead of Custom Instrumentation on page 381
- Instrumentation Control on a Per Layer Basis on page 382
- Advanced Instrumentation Examples on page 383
- Configuring Cross VM Correlations for New or Custom Technologies on page 398
- Tutorial for Configuring Cross VM Correlation for Custom Technologies on page 403
- Maintaining Instrumentation from the Java Profiler UI on page 412
- Default Layers Defined for Typical Java Classes and Methods on page 423

## 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 926.

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 423.

---

**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 341
- “Optional Point Entries” on page 342

## 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
<b>Point-Name</b>	A unique name for the point.
<b>class</b>	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.
<b>method</b>	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.
<b>signature</b>	Specifies the signature (parameter and result types) of the method using javap symbolic encoding for method signatures (<jdk_install>/bin.javap -s).
<b>layer</b>	<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 926.

---

## Optional Point Entries

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

Argument	Description
<b>keyword</b>	The <b>keyword</b> indicates an instrumentation point handled by a special instrumentation class. The value of the keyword is looked up as a property in <b>inst.properties</b> , and the value of the found property is the instrumentation class name. The special instrumentation points can use implementation-specific arguments not documented here, refer to the comments in the <b>inst.properties</b> file.
<b>ignore_cl</b>	Specifies a comma-separated list of class names or regular expressions to ignore. Any class matching one of the classes specified with <b>ignore_cl</b> is not instrumented.
<b>ignore_method</b>	Specifies a comma-separated list of methods to ignore. Any method matching one of the methods specified with <b>ignore_method</b> is not instrumented.

Argument	Description
<b>ignore_tree</b>	A list of classes or regular expressions. Any subclass of a class matching one of the list items is excluded from the instrumentation.
<b>method_access_filter</b>	A list of method specifiers, separated by commas. The available specifiers are <b>static</b> , <b>private</b> , <b>protected</b> , <b>package</b> , and <b>public</b> . Any method matching this point is not instrumented if its access specifier matches any of the listed values.
<b>deep_mode</b>	<p>Specifies how subclasses are handled. This argument accepts three values:</p> <ul style="list-style-type: none"> <li>▶ <b>none</b> – A value of “none” is similar to not specifying a <b>deep_mode</b> argument. It has no effect on how subclasses are handled.</li> <li>▶ <b>soft</b> – 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.</li> <li>▶ <b>hard</b> – 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. <b>Caution:</b> Hard mode can lead to extensive instrumentation and very high probe overhead.</li> </ul>
<b>scope</b>	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.
<b>ignoreScope</b>	Lists method names or regular expressions and excludes certain packages, classes, and methods from those included in the scope specified in the scope argument.

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"> <li>▶ <b>caller</b> – causes caller side instrumentation to be performed. If this keyword is not specified, the default instrumentation, callee side instrumentation, is performed.</li> <li>▶ <b>args:n</b> – calls the <b>toString()</b> method of the <b>n-th</b> 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 <b>n</b> can be <b>1</b> through <b>256</b>.</li> <li>▶ <b>args:0</b> – calls the <b>toString()</b> on the current class instance or callee object. Static methods return the class name of the callee object.</li> <li>▶ <b>before:code:&lt;code-key&gt;</b> – 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 348 for information about code snippets and “Securing Code Snippets” on page 362 for information on code keys.</li> <li>▶ <b>after:code:&lt;code-key&gt;</b> – 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.</li> </ul>



Argument	Description
<p><b>detail</b> (continued)</p>	<ul style="list-style-type: none"> <li>▶ <b>disabled</b> – 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://&lt;probe_install_dir&gt;:&lt;probe_port&gt;/inst/layer">http://&lt;probe_install_dir&gt;:&lt;probe_port&gt;/inst/layer</a>.</li> <li>▶ <b>outbound</b> – 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.</li> <li>▶ <b>no-correlation</b> – used with those “outbound” points that do not use correlation supporting technologies.</li> <li>▶ <b>method-no-trim</b> – indicates that no latency-based trimming should take place when a method instrumented by this point is executed.</li> <li>▶ <b>method-trim</b> – 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.</li> <li>▶ <b>lifecycle</b> – identifies the instrumentation point as relevant for object lifecycle monitoring.</li> <li>▶ <b>no-layer-recurse</b> – prohibits recording of any methods called from the method instrumented by this point, unless the callee belongs to a different layer.</li> <li>▶ <b>is-statement</b> – marks calls into the <code>java.sql.Statement</code> class.</li> <li>▶ <b>is-prepare-statement</b> – marks calls returning <code>java.sql.Statement</code> objects to capture.</li> <li>▶ <b>method-cpu-time</b> – 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>).</li> </ul>

Argument	Description
<b>detail</b> (continued)	<ul style="list-style-type: none"> <li>▶ <b>condition</b> – 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 <b>inst.properties</b> (or on the command line).</li> <li>▶ <b>when-root-rename</b> – instructs the probe to rename the server request whenever the method instrumented by this point is the first one executed.</li> <li>▶ <b>diag</b> – marks the point as relevant for HP Diagnostics (default).</li> <li>▶ <b>tv:&lt;key&gt;</b> – marks the point as relevant for HP Transaction Vision.</li> <li>▶ <b>no-tv</b> – 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.</li> <li>▶ <b>add-field:&lt;access&gt;:&lt;type&gt;:&lt;name&gt;</b> – causes adding the specified field to the instrumented class.</li> <li>▶ <b>gen-instrument-trace</b> – causes printing of the thread stack trace onto stdout whenever this point is used for instrumentation.</li> <li>▶ <b>gen-runtime-trace</b> – causes printing of the thread stack trace onto stdout whenever the methods instrumented by this point are executed.</li> <li>▶ <b>trace</b> – causes printing of verbose instrumentation information into <code>probe.log</code> on each enter or exit from each method instrumented by this point.</li> <li>▶ <b>sub-point:&lt;key&gt;</b> – specifies additional processing during instrumentation; the key must be present in <b>inst.properties</b> and must identify a class name used for the processing.</li> <li>▶ <b>store-thread</b> – causes all special fields used in the corresponding code-snippet to be stored in a thread-local data structure.</li> <li>▶ <b>store-fragment</b> – causes all special fields used in the corresponding code-snippet to be stored as attributes of the current server request.</li> </ul>

Argument	Description
<b>detail</b> (continued)	<ul style="list-style-type: none"> <li>➤ <b>store-method</b> – 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.</li> <li>➤ <b>ws-operation</b> – 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.</li> </ul>
<b>rootRenameTo</b>	Identifies server requests whenever the <b>when-root-rename</b> detail is in effect.
<b>layerType</b>	<p>Specifies special handling for some instrumented methods and accepts the following values:</p> <ul style="list-style-type: none"> <li>➤ <b>method</b> – no special handling (default).</li> <li>➤ <b>trended_method</b> – identifies methods to be displayed in the Trended Methods view.</li> <li>➤ <b>Portlet</b> – identifies portlet lifecycle methods that are used for the Portal Components views. These are set by HP Diagnostics and should not be modified.</li> <li>➤ <b>sql</b> – identifies methods that are used to capture SQL for the SQL views. These are set by HP Diagnostics and should not be modified.</li> </ul>
<b>priority</b>	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 <b>priority</b> determines which point to use. Higher priority wins. The default is zero.
<b>active</b>	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 362.

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 349
- ▶ “Code Snippet Grammar” on page 350
- ▶ “Code Snippet Helper” on page 354
- ▶ “Securing Code Snippets” on page 362

## 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 &ws_op &ws_ns &ws_port (inbound only) &target (outbound only)	Web Service name Web Service Operation name Web Service namespace Web Service Port Name Outbound Web Service Target
jms	&name &target	Queue or Topic name 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
##SOAPHandler_wsname	Stores the web service name for later use by the SOAP Handler.
##<any_string>	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) {
        ...
    }
}

```

```

/*
 * Helper methods to allow our cross-vm coloring to piggyback ride across
 * the custom outbound calls in which the application passes [only] a String.
 * The actual transport technology is irrelevant.
 * Instead of sending the original message, a composite message ("envelope")
 * will be passed. The composite message includes both the original message
 * and Diagnostics Probe ENCODED cross-vm coloring.
 * On the receiving end, the composite message will be received, but only
 * the original message will be passed to the application, and the coloring
 * will be retained by the probe.
 */

/**
 * Create a composite message, given the coloring and the original message.
 * @param coloring - the correlation String obtained via the ENCODED coloring,
 * may be null
 * @param originalMessage - the original message sent by the application
 * @return - the composite message, null if and only if the originalMessage is null
 */
public static String createDiagEnvelope(String coloring, String originalMessage) {
    ...
}

/**
 * Extract the coloring from the composite message (envelope).
 * @param envelope - the composite message or the original message
 * @return the coloring as created on the sender side, or null if not present
 */
public static String extractColoringFromDiagEnvelope(String envelope) {
    ...
}

/**
 * Extract the original message from the composite message (envelope).
 * Works properly, even if the sender side has not been instrumented, and
 * there's no envelope.
 * @param envelope - the composite message or the original message
 * @return the original message (before coloring)
 */
public static String extractOriginalMessageFromDiagEnvelope(String envelope) {
    ...
}
}

```

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

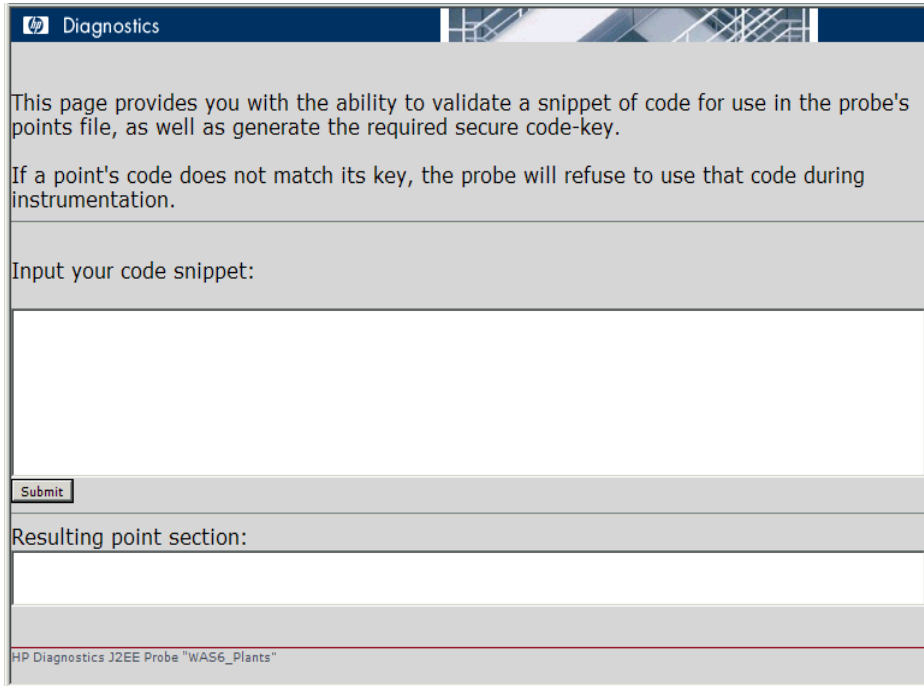
The Java Agent is installed with a tool that generates the code-key from the code snippet you input.

**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:



The screenshot shows a web browser window with the title "Diagnostics". The page content includes:

- A header bar with the HP logo and the word "Diagnostics".
- Text: "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."
- Text: "If a point's code does not match its key, the probe will refuse to use that code during instrumentation."
- A section titled "Input your code snippet:" followed by a large empty text input field.
- A "Submit" button.
- A section titled "Resulting point section:" followed by an empty text area.
- A footer: "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
- Changing SAP RFC Instrumentation Based on SAP JCO Library Version
- 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 383.

```
[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 383.

```
[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 383.

```
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 using the appropriate mode for your application server if you want to use the collection leak pinpointing (CLP) feature in the Java Agent. Chapter 6, “Preparing Application Servers for Monitoring with the Java Agent” for details on instrumenting the JRE.

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 546).

### `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 Layer page in the (<http://<probe-host>:<probe-port>/inst/layer> or from the Profiler select the Configuration tab and then select View instrumentation), 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
```

## Changing SAP RFC Instrumentation Based on SAP JCO Library Version

In the `<probe_install_dir>/etc/inst.properties` file there are two points defined depending on the version of SAP JCO used. Comment out the version you are not using. Starting with version 2.1.10 or later use `com.mercury.opal.capture.inst.SapRfcinstrumentationPoint2_1_10`. Otherwise the default setting will work for version 2.1.9 and earlier.

## 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 13, "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
<a href="#">(Other)</a>	0	2 / 2	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">(keyword) http</a>	129707	6 / 6	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">(keyword) lwmd</a>	31377106	4159 / 4217	<a href="#">[Enable]</a> <a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">(keyword) remote-http</a>	0	15 / 15	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">(keyword) rmi</a>	0	206 / 206	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">(keyword) soap fault</a>	0	2 / 2	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">Business Tier/EJB/Entity Bean</a>	989945	63 / 63	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">Business Tier/EJB/Session Bean</a>	110926	35 / 35	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">Database/JDBC/Connection</a>	107755	49 / 49	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">Database/JDBC/Execute</a>	105821	79 / 79	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">Directory Service/JNDI</a>	175	4 / 4	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">Legacy/JCA/Connection</a>	52877	1 / 1	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">Legacy/JCA/ECIConnectionFactory</a>	51936	2 / 2	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">Legacy/JCA/ManagedConnectionFactory</a>	2	2 / 2	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>
<a href="#">Web Tier/Servlet</a>	55198	11 / 11	<a href="#">[Disable]</a> <a href="#">[Clear # Hits]</a>

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 Examples

This section includes:

- “Using Caller Side Instrumentation” on page 383
- “URI Aggregation Instrumentation” on page 386
- “CORBA Cross VM Instrumentation” on page 387
- “Using RMI Instrumentation” on page 387
- “Using Thread Local Storage to Store the SOAP Payload” on page 388
- “Performing Correlation Across Multiple Threads” on page 389
- “Using Fragment Local Storage to Store Web Service Field” on page 392
- “Using Annotations for Custom Instrumentation” on page 396

### 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 (for example, <http://<myserver>/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://<myserver>/myApplication?page=home  
http://<myserver>/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/probeCorbaInterceptors.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 535.

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/bea/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 350.



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@.getConsumerIdFromDo
cument(##WS_inbound_service_name<java.lang.String>,#soap_payload<org.w3c.do
m.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 350.

**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 350.

**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 350.

**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 350.

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@.getConsumerIdFromDocument(##WS_inbound_service_name<java.lang.String>,#soap_payload<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 524.

**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 350.

## 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.

## Configuring Cross VM Correlations for New or Custom Technologies

Diagnostics can show call profiles that span multiple Java virtual machines (JVM). These Cross VM call profiles and topologies are very useful when a performance issue involves a client and a server. You want to know which application is the source of the problem but looking at the call profile for the client or server individually may not help with intermittent issues since they would not be correlated. The Cross VM call profile will show the client and the server correlated together in a single call tree.

Out-of-the-box the Java Agent provides support for this feature for many different technologies: for example, JMS, HTTP/S (Web Services only), RMI, SAP, TIBCO and Corba. With the latest version of Diagnostics, additional support was added to help you configure cross VM correlation for new or custom technologies.

The Cross VM correlation technique is exposed in code snippets, allowing you to prepare instrumentation points and code snippets to correlate almost any inter-process communication, including home-grown and legacy communication techniques. The only requirement for the communication technique is that its messages be able to carry an additional string, which is referred to as coloring.

The coloring string is created on the client side by the Java Agent, and attached to the outgoing message by a user-written code snippet. After the message is received, a user-written code snippet on the server side extracts the coloring from the message and passes it to the server side agent for parsing and processing.

Thus, your responsibility related to the cross-vm communication technique is primarily limited to embedding the coloring into the outgoing messages, and extracting the coloring from the received messages. This, of course, includes identifying the code locations (instrumentation points) for the client side (the outbound point), and for the server side (the inbound point). Refer to “Tutorial for Configuring Cross VM Correlation for Custom Technologies” on page 403 for a detailed example. And refer to “APIs Used to Facilitate Custom Transport Cross-VM Correlations” on page 401 for information on the three APIs provided to help you configure custom cross-vm correlation.

## Client Side

For the outbound calls (the client side), use the new **outbound:<coloring-type>** detail.

The available coloring types are:

- default
- sap
- none
- snippet

For all coloring types except **none**, there should be an associated code snippet, which will provide a special argument containing the technology type, the call target name and identification.

The argument has the following form:

**DIAG\_ARG:type=<type>&name=<name>&target=<target>**

where <type> is the technology type used for the remote call, and <name> and <target> are technology dependent values. The technology type should be the same as the one used for the inbound instrumentation point (see “Server Side” on page 401).

For all coloring types except **snippet**, the probe will generate the appropriate coloring and it will report the coloring to the Diagnostics Server for future correlation. However, the outgoing message remains unmarked at this time.

For all coloring types except **none**, a code snippet for another instrumentation point (which is hit after the outbound point, preferably during the outbound method execution) must attach the generated coloring to the outgoing message.

The most recently generated coloring can be obtained by calling **ICorrelationColor RemoteCaptureProxy.getCurrentColor(#location)**.

In developing support for your own cross-vm communication, you may use **snippet**, which means that the coloring will be explicitly created by a direct call from a code snippet. For the snippet coloring the above order is reversed, which means the coloring is generated (and, most often, immediately attached to the message) before the outbound point is hit. Please note that this includes a case where the **before** code snippet for the outbound point creates the coloring, because the code snippet will be executed before the agent is called.

**To create the coloring from code snippets:**

- 1 Make a call to **ICorrelationColor RemoteCaptureProxy.createColoring(#location, <type>, <diag-arg>)**

For type, use

RemoteCaptureProxy.ENCODDED\_COLORING for **default**

RemoteCaptureProxy.SAP\_R3\_COLORING for **sap**

If in doubt which type to use, use the default.

- 2 Make a call to **grabCorrelationString()** on the object returned in step 1, and insert the returned string into the outgoing message (using a technology-dependent technique). This is where you can use your creativity.

Tip: If using String messages, use the following helper API to accomplish this step:

ProbeCodeSnippetHelper.createDiagEnvelope(String coloring, String originalMessage)

- 3 Hit an instrumented point with the **outbound:snippet** detail. This will automatically use the most recently created coloring instead of creating a new one. Executing the outbound point informs the probe that the coloring was actually used, and identifies the method which will be considered the connection point for cross-vm call profiles. For synchronous cross-vm communication it is recommended to use **outbound** detail for a method that is used to both send the message and receive an acknowledgment, so the latency of the outbound call can be properly captured.



## Server Side

For the inbound calls (the server side), use the **inbound:<technology-type>** detail. Use your own technology type names when supporting new cross-vm technologies. Check to avoid conflicts with existing technology names (server request types). Examples of server request types include: ADO, CICS, Corba, HTTP, JDBC, JMS, MSMQ, RMI, Remoting (.NET), SAP ABAP types, Web Services. In addition, you may see server request types named Pseudo and RootRename.

**The before code snippet has to perform the following steps:**

- 1 Extract the correlation string from the incoming message, using the technology-dependent technique, corresponding to the one used for the outbound calls.

Tip: If the `ProbeCodeSnippetHelper.createDiagEnvelope()` was used previously, use `ProbeCodeSnippetHelper.extractColoringFromDiagEnvelope(String envelope)` to get the correlation string.

And use

`ProbeCodeSnippetHelper.extractOriginalMessageFromDiagEnvelope(String envelope)` to get the original message.

- 2 Leave TWO Strings on the stack: the capture argument (as any **before** code snippet should), and the extracted correlation string.

## APIs Used to Facilitate Custom Transport Cross-VM Correlations

Three helper APIs were added to facilitate custom transport cross-VM correlations (see the tips in the sections above and see “Code Snippet Helper” on page 354 for information on their use. There is also a “Tutorial for Configuring Cross VM Correlation for Custom Technologies” on page 403 to walk you through an example.

- `ProbeCodeSnippetHelper.createDiagEnvelope(String coloring, String originalMessage)`

- ▶ `ProbeCodeSnippetHelper.extractColoringFromDiagEnvelope(String envelope)`
- ▶ `ProbeCodeSnippetHelper.extractOriginalMessageFromDiagEnvelope(String envelope)`

## HTTP/S Support

The support for the server side HTTP/S is built in and is enabled by default. The Java Agent automatically recognizes standard J2EE implementation of `HttpServlet`, as well as Jetty and Apache Catalina implementations. No user action is required on the server side, if one of these technologies is used.

For the client side, the Java Agent automatically instruments the `openConnection` method from the `java.net.URL` class, to embed the most recently generated coloring (if it exists) into the outgoing HTTP request. One of the HTTP request headers is used to carry the coloring. The header will be recognized by the server side agent.

Therefore, HTTP support on the client side is an exception to the above rules. You still have to provide the outbound point and the corresponding `DIAG_ARG`, but you do not have to worry about embedding the coloring into the outgoing messages.

For example, Diagnostics mediators use the following point:

```
[RemoteHttpComponent-Outbound-1]
class    = com.mercury.diagnostics.common.net.registrar.RemoteHttpComponent
method   = getConnection
signature = (Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/
String;Ljava/lang/String;)Ljava/net/URLConnection;
priority = 1
detail   = method-no-trim,outbound:default,before:code:7b1125e2
layer    = Network.RemoteHttpComponent
```

The first argument for the `getConnection` method is a `String` representing the connection URL. The referred code snippet extracts from it the hostname and port and uses them for the target identification. A special utility method is provided by `RemoteCaptureProxy` to facilitate this conversion in a way consistent with the built-in part of the HTTP/S support.

```
7b1125e2 = #target=@RemoteCaptureProxy@.getTargetFromUri(#arg1); \
"DIAG_ARG:type=http&name="+#target+"&target="+#target;
```

## Tutorial for Configuring Cross VM Correlation for Custom Technologies

This tutorial takes a simplified client-server application that uses a shared blocking queue as its custom transport solution. The client sends a "String" message by adding it to the queue. The server receives a "String" message by removing it from the queue.

Although this example runs in a single JVM (to keep it simple), it uses two threads to simulate an application running in two JVMs. (If your intention is to correlate threads running in a single JVM, there is a simpler solution that will help you do this. See “Performing Correlation Across Multiple Threads” on page 389 for more details).

The sample code is shown below:

```
public class SimulatedCrossVM {

    private static int INTERVAL = 5 * 1000; // 5 seconds
    private static BlockingQueue<String> queue = new LinkedBlockingQueue<String>();

    private static class ReceiverSide extends Thread {

        public ReceiverSide() {
            super("Receiver");
        }

        public void execute(String receivedString) throws InterruptedException {
            System.out.println("Executing message: " + receivedString);
            sleep(2 * INTERVAL);
        }
    }
}
```

```
private void receiveAndHandleMessage() throws InterruptedException {
    String message = null;
    message = queue.take();
    // Handle it
    execute(message);
}

public void run() {
    try {
        while (true) {
            receiveAndHandleMessage();
        }
    }
    catch (Throwable t) {
        // oops
        t.printStackTrace();
    }
}

private static class SenderSide extends Thread {

    // For simulated TCP connection
    private String destHost;
    private int destPort;

    public SenderSide(String host, int port) {
        super(host + ":" + port);
        destHost = host;
        destPort = port;
    }

    public void sendMessage(String origMessage) throws InterruptedException {
        queue.put(origMessage);
    }

    private String generateMessage() {
        String message = "T" + System.currentTimeMillis();
        return message;
    }

    private void generateAndSendMessage() throws InterruptedException {
        sleep(2 * INTERVAL);
        // generate message
        String message = generateMessage();
    }
}
```

```
        System.out.println("Sender's original message: " + message);
        // And send it (outbound call)
        sendMessage(message);
        sleep(INTERVAL);
    }

    public void run() {
        try {
            while (true) {
                generateAndSendMessage();
            }
        }
        catch (Throwable t) {
            // oops
            t.printStackTrace();
        }
    }
}

public static void main(String[] args) {
    SenderSide sender = new SenderSide("fake-host", 12345);
    ReceiverSide receiver = new ReceiverSide();

    sender.start();
    receiver.start();
}
}
```

Executing this code will have the following output:

Sender's original message: T1313785958127

Executing message: T1313785958127

## Step 1: Instrument Your Methods

By instrumenting your methods, you let Diagnostics know which methods are important. Since these methods are custom, the out-of-the-box instrumentation points won't do anything. Edit the etc/autodetect.points file by adding the following instrumentation points. See “Maintaining Instrumentation from the Java Profiler UI” on page 412 for guidance on defining instrumentation points.

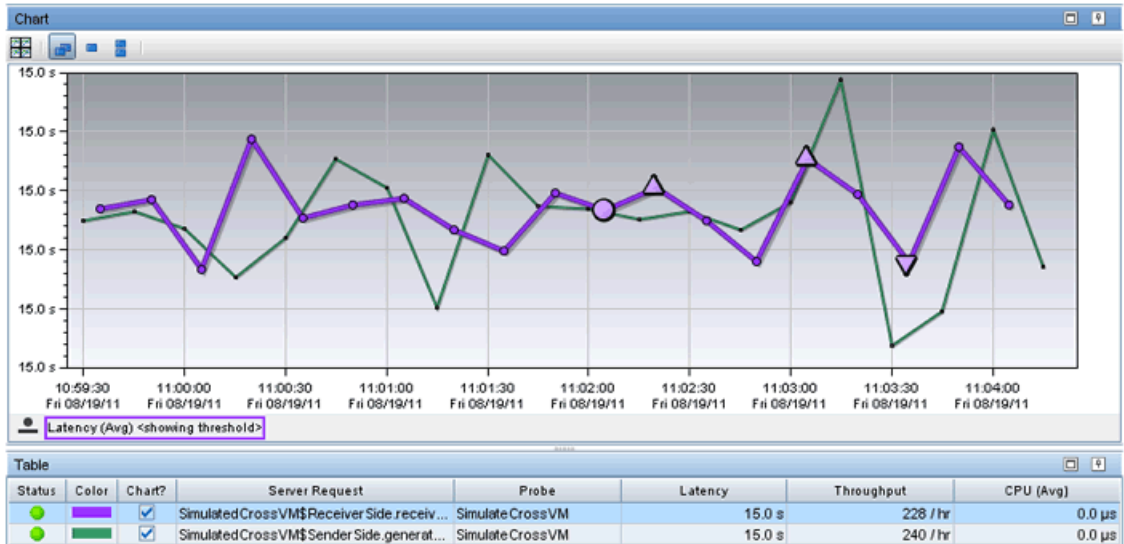
```
[SimCrossVM-Sender]
class    = SimulatedCrossVM$SenderSide
method  = generateAndSendMessage
signature = !.*
layer   = Sending
```

```
[SimCrossVM-Outbound]
class    = SimulatedCrossVM$SenderSide
method  = sendMessage
signature = !.*
layer   = Sending
```

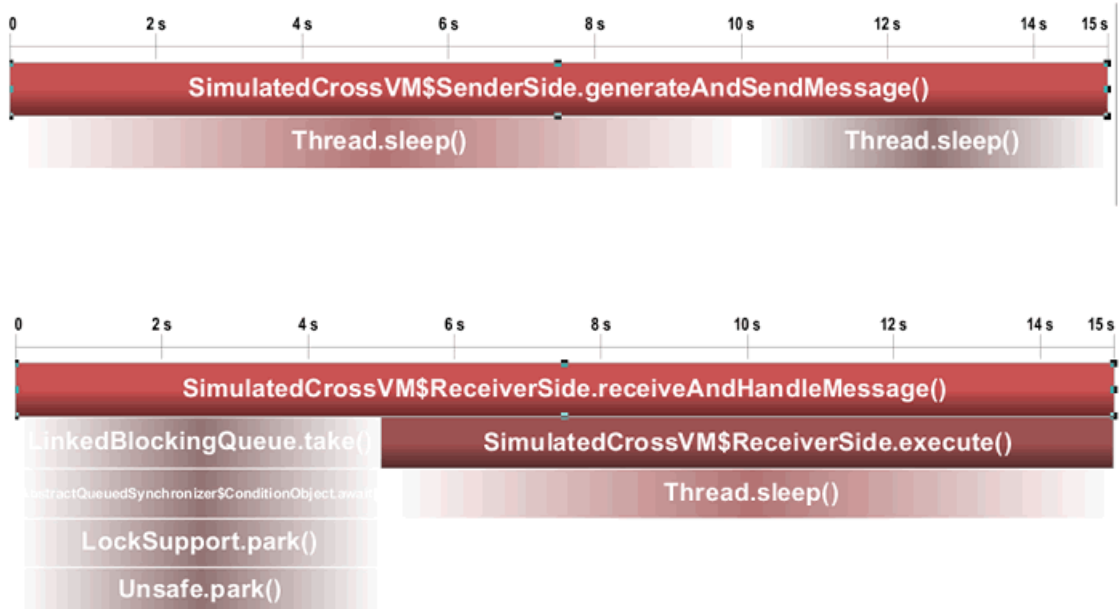
```
[SimCrossVM-Receiver]
class    = SimulatedCrossVM$ReceiverSide
method  = receiveAndHandleMessage
signature = !.*
layer   = Receiving
```

```
[SimCrossVM-Inbound]
class    = SimulatedCrossVM$ReceiverSide
method  = execute
signature = !.*
layer   = Receiving
```

Result: Running this instrumented test program, you see the following Server Requests:



Here are the call profiles shown for the sender and receiver.



## Step 2: Add “Coloring” to the Sender Logic

In this step, we add “coloring” to the messages sent by the client. When the instrumented server receives this colored message, HP Diagnostics will correlate them. This part is trickier, if you're not familiar with the code snippet syntax, it is described in “Defining Points With Code Snippets” on page 348.

First, we mark the method as an outbound point that uses a code snippet (outbound:snippet), and identify the code snippet to execute before invoking the method (before:code:5ea4753f). Since we're going to use the first argument, it's a good idea to provide a more specific signature (!\ (Ljava/lang/String;.\*).

```
[SimCrossVM-Outbound]
class    = SimulatedCrossVM$SenderSide
method  = sendMessage
signature = !\ (Ljava/lang/String;.*
layer   = Sending
detail   = outbound:snippet,before:code:5ea4753f
```

The corresponding code snippet is shown below. Line 1 creates a string (#target) that includes the hostname and destination port of the server. Line 2 defines a new string (#diagArg) that follows a special syntax (DIAG\_ARG:type=<type>&name=<name>&target=<target>). The “type” is the technology type and can be any name you choose; it will be used in the next step. The “name” and “target” are technology dependent values that will be shown in the UI; they can also be anything you choose. Line 3 defines a third string (#color) which will be used to identify this specific invocation of the method call from any other. Line 4 updates the method's 1st argument with the colored String, which will cause sendMessage to send a modified String. Finally, line 5, places the coloring on the stack for usage by HP Diagnostics.

```
1 5ea4753f = #target=#callee.destHost+"."+#callee.destPort; \
2 #diagArg =
    "DIAG_ARG:type=CB-TCP&name=Senders.sendMessage&target="+#target; \
```



- ```

3 #color = (null == #arg1 ? null :
    @RemoteCaptureProxy@.createAndGrabColor(#location,
    @RemoteCaptureProxy@ENCODED_COLORING, #diagArg.toString()); \
4 #arg1 = @ProbeCodeSnippetHelper@.createDiagEnvelope(#color, #arg1); \
5 #diagArg;

```

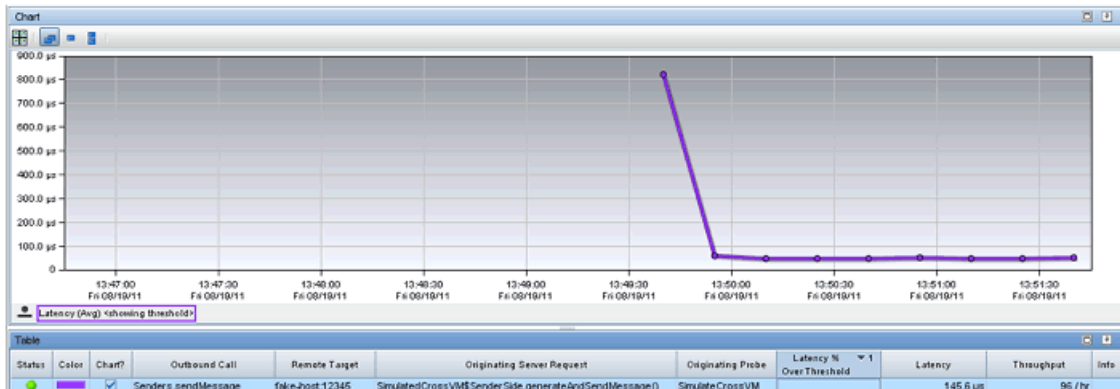
Running the example updates the output as follows. Notice the receiving side did not get the same string message that was sent. This is a result of the code snippet's Line 4. In many cases, the receiving side may not handle this well. It's a good idea to note the receiver's behavior as this can happen "accidentally" if the client and server are not both using the same instrumentation, and in particular, not both instrumented.

Sender's original message: T1313786970403

Executing message: HP\_DIAG1\_!Dhf/

ABAABKrh3Qf0cy7yaLsAAAAAAA9mYWtlLWhvc3Q6MTIzNDUAYTEzMTM3ODY5N  
 jAzODgmU2ltdWxhdGVDbm9zc1ZlNjlnpbXVsYXRlZENyb3NzVzV0kU2VuZGVyU2lk  
 ZS52b2lkIGdlbmVvYXRlQW5kU2VuZE1lc3NhZ2UoKSZcMCZcMCZcMCY=:T131378  
 6970403

At this point, the only change you'll see in the UI is some "Outbound Calls". Notice the values in the columns "Outbound Call" and "Remote Target", these are the values you provided in the code snippet "name" and "target".



### Step 3: Remove Coloring from the Receiver Side

The last step is to remove the coloring on the receive side so that the receiver can get the original "uncolored" message from the sender. First we mark the point as an inbound point with the technology type used in the code snippet defined in step 2, and assign a code snippet to run before this method is called. Again, we also specify a more specific signature since that argument will be used in the code snippet.

```
[SimCrossVM-Inbound]
class    = SimulatedCrossVM$ReceiverSide
method  = execute
signature = !(Ljava/lang/String;.*
detail   = before:code:d2c83d3c,inbound:CB-TCP
layer    = Receiving
```

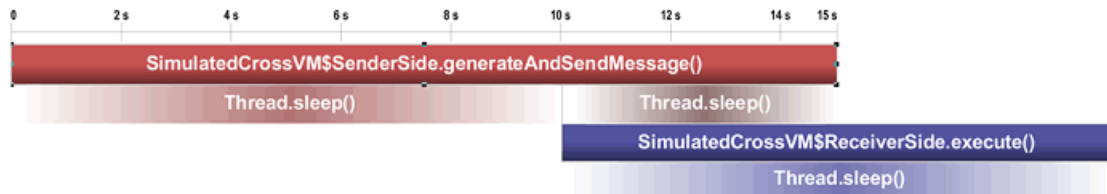
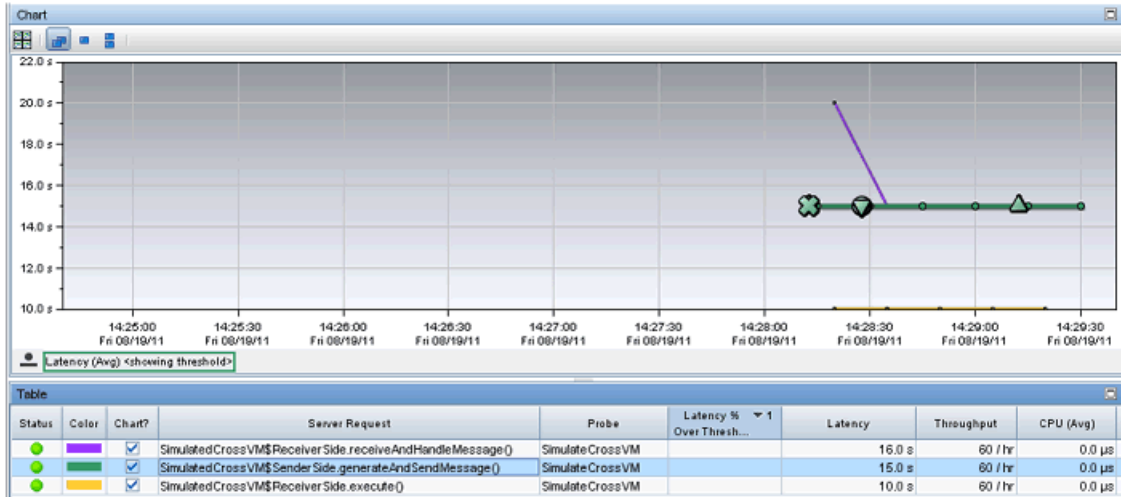
The corresponding code snippet is shown below. Line 1 extracts the coloring from the incoming message. Line 2 updates the method's 1st argument, restoring it to the original message sent by the client. Line 3 puts the coloring on the stack (and an empty String) for use by HP Diagnostics.

```
1 d2c83d3c =
   #coloring=@ProbeCodeSnippetHelper@.extractColoringFromDiagEnvelope(
   #arg1); \
2 #arg1=@ProbeCodeSnippetHelper@.extractOriginalMessageFromDiagEnvelope(
   #arg1); \
3 "";#coloring;
```

The program's output is now restored to the original:

```
Sender's original message: T1313789287234
Executing message: T1313789287234
```

The Server Request view now shows a Cross-VM call profile is available for the Sender's "generateAndSendMessage". Open this call profile and observe the client and server call profiles are now stitched together! They're not doing much in this sample application, but in a real application, you would be able to see if performance issues occur in the client, server, or both.



This call profile looks a bit strange but is typical for asynchronous applications. The client does not wait for a response, but does continue to do some processing (err sleeping for 5 seconds). During that time the server is processing the request and completes a few seconds afterwards. You will see the time durations for the methods in the tree as shown below. Notice also the diamonds with the number 2 inside, which represent the JVM depth. If your server made yet another outbound call, you could have 3 or more! In those cases, cross VM correlation because especially useful. Imagine trying to find the source of a performance issue across that many JVMs!

|       |                                                                                                 |        |
|-------|-------------------------------------------------------------------------------------------------|--------|
| 100%  | SimulatedCrossVM\$SenderSide.generateAndSendMessage()                                           | 15 s   |
| 66%   | Thread.sleep()                                                                                  | 9.9 s  |
| 0%    | Outbound Call to Senders.sendMessage on probe SimulateCrossVM on rasselin1.americas.hpqcorp.net | 0.8 ms |
| 66.6% | SimulatedCrossVM\$ReceiverSide.execute()                                                        | 10 s   |
| 65.9% | Thread.sleep()                                                                                  | 9.9 s  |
| 32%   | Thread.sleep()                                                                                  | 4.8 s  |

## 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.

**Instrumentation**

Currently Used Instrumentation:

Change Probe Instrumentation Plan

|                           |                                        |              |                               |
|---------------------------|----------------------------------------|--------------|-------------------------------|
| Shared Instrumentation:   | <input type="button" value="Edit..."/> | [157 points] | Used by all probe instances.  |
| Instance Instrumentation: | <input type="button" value="Edit..."/> | [0 points]   | Used by probes with Id: T-LC7 |

## 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                  |
| <a href="#">(Other)</a>                             | 97539   | 1 / 1         | [Disable] [Clear # Hits] |
| <a href="#">(keyword) http</a>                      | 78356   | 13 / 13       | [Disable] [Clear # Hits] |
| <a href="#">(keyword) lwmd</a>                      | 1004363 | 2861 / 2861   | [Disable] [Clear # Hits] |
| <a href="#">(keyword) remote-http</a>               | 0       | 12 / 12       | [Disable] [Clear # Hits] |
| <a href="#">(keyword) soap fault</a>                | 0       | 1 / 1         | [Disable] [Clear # Hits] |
| <a href="#">Business Tier/EJB/Entity Bean</a>       | 436449  | 596 / 596     | [Disable] [Clear # Hits] |
| <a href="#">Business Tier/EJB/Session Bean</a>      | 48922   | 110 / 110     | [Disable] [Clear # Hits] |
| <a href="#">Database/JDBC/Connection</a>            | 93203   | 57 / 57       | [Disable] [Clear # Hits] |
| <a href="#">Database/JDBC/Execute</a>               | 45968   | 64 / 64       | [Disable] [Clear # Hits] |
| <a href="#">Directory Service/JNDI</a>              | 479     | 5 / 5         | [Disable] [Clear # Hits] |
| <a href="#">HttpStatus</a>                          | 0       | 20 / 20       | [Disable] [Clear # Hits] |
| <a href="#">Legacy/JCA/Connection</a>               | 23075   | 1 / 1         | [Disable] [Clear # Hits] |
| <a href="#">Legacy/JCA/ECIConnectionFactory</a>     | 22918   | 2 / 2         | [Disable] [Clear # Hits] |
| <a href="#">Legacy/JCA/ManagedConnectionFactory</a> | 20      | 2 / 2         | [Disable] [Clear # Hits] |
| <a href="#">Messaging/JMS/Listener</a>              | 0       | 1 / 1         | [Disable] [Clear # Hits] |
| <a href="#">SOAPHandler</a>                         | 0       | 1 / 1         | [Disable] [Clear # Hits] |
| <a href="#">Web Services</a>                        | 0       | 1 / 1         | [Disable] [Clear # Hits] |
| <a href="#">Web Tier/Servlet</a>                    | 24073   | 23 / 23       | [Disable] [Clear # Hits] |
| <a href="#">Web Tier/Struts</a>                     | 0       | 2 / 2         | [Disable] [Clear # Hits] |

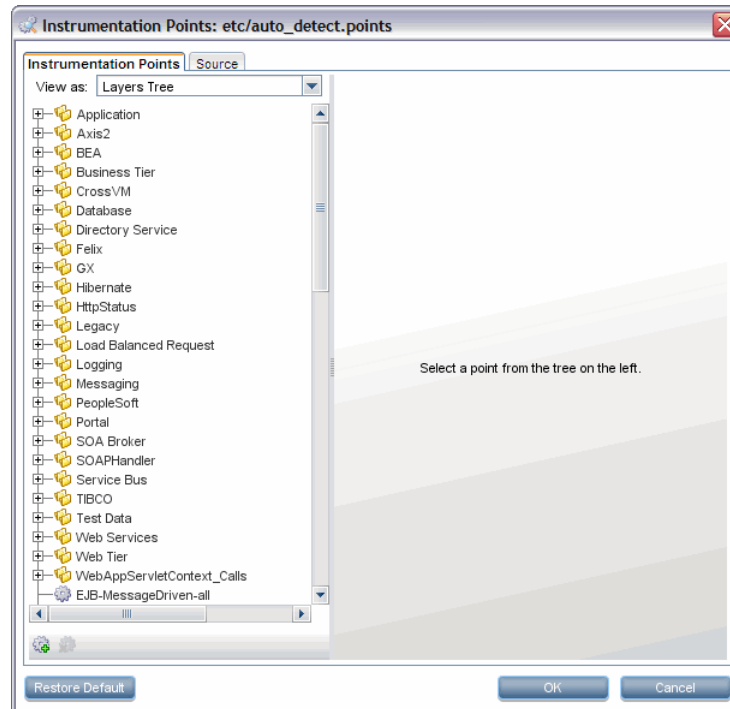
HP Diagnostics J2EE Probe "P81\_WA55\_Plants\_ovrmtt152\_W2k", version 9.00.70.1002

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                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Layer</b>         | 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. <b>Note:</b> Only the layers defined in points that were actually instrumented are listed.                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Hits</b>          | 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 <b>Clear # of Hits</b> link in the Actions column.                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Active Points</b> | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Actions</b>       | <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"> <li>▶ <b>Disable:</b> 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 340.</li> <li>▶ <b>Clear # Hits:</b> Resets the hit count displayed in the # Hits column for the selected layer.</li> </ul> |

## 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:

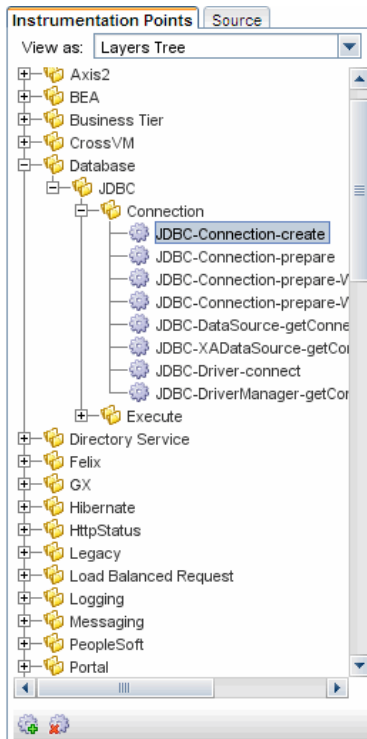


You can edit the instrumentation in two ways: visually, using a list or tree of points on the **Instrumentation Points** tab; or via the source of the capture points file on the **Source** tab.

## Selecting and Viewing an Existing Point

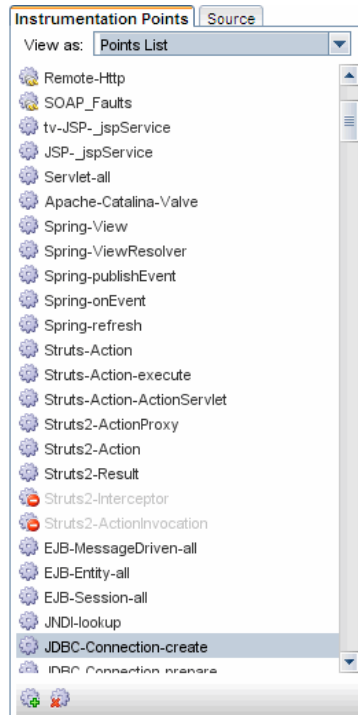
The navigation bar in the Instrumentation Points dialog helps locate the points in the capture points file that you would like to maintain. By making a selection from the **View as** dropdown, you can choose the format in which the points are listed.

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:

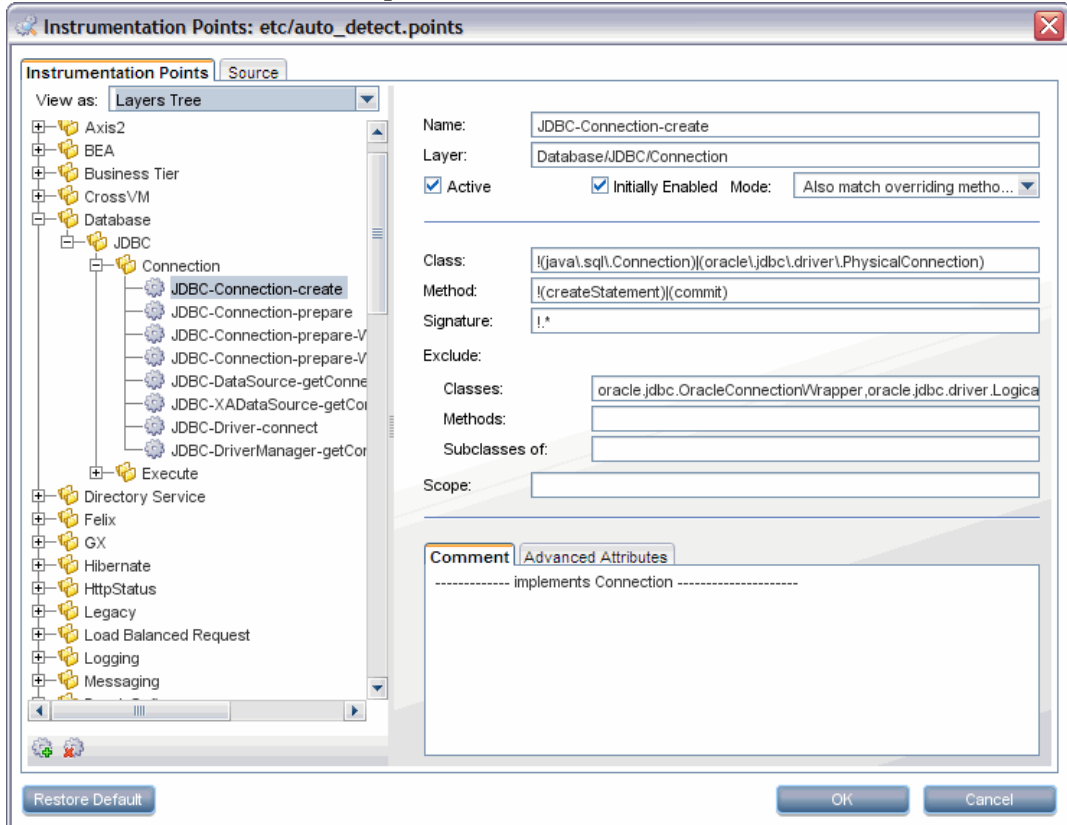


When you locate the point you want to view or maintain, select the point in the navigation bar. Then you see the details of the selected point in the view/edit panel where you can maintain the point.

## 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.

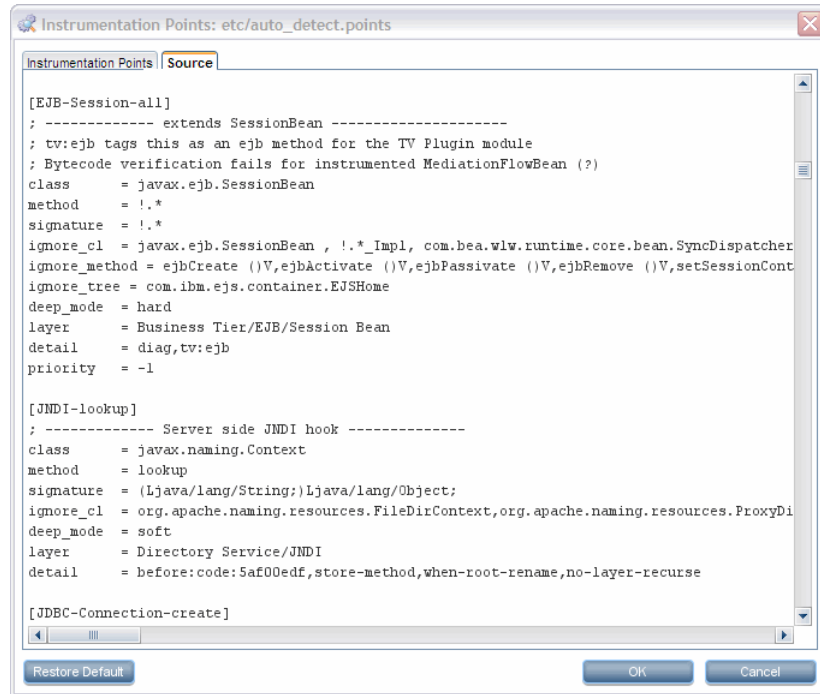
To update an existing point, select the point from the navigation bar so that the Profiler displays the details for the point in the Instrumentation Points tab of the view/edit panel:



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 340.

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**.

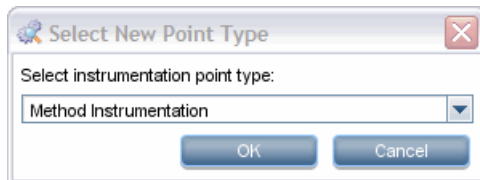
### Adding a New Point

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**.

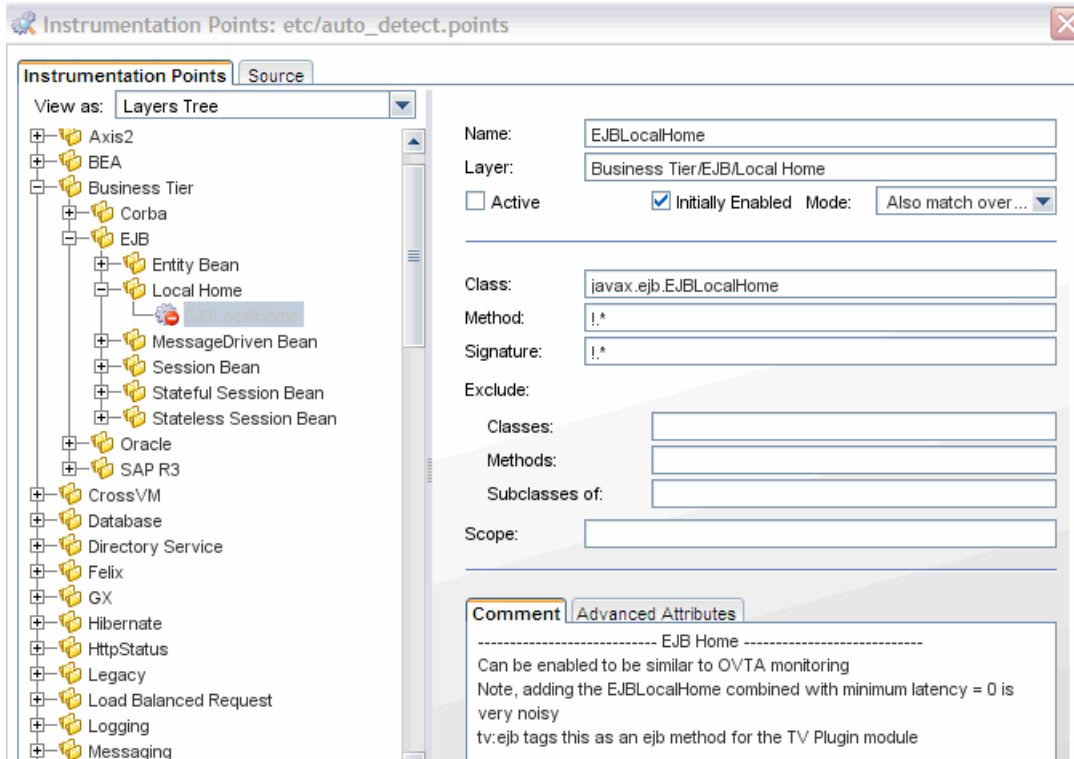
## Activating OVTA-like Points

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 416 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<br>Servlets<br>Struts<br>Session<br>Spring<br>Struts2                                                          |               |
| Business Tier     | EJB<br>Corba                                                                                                       |               |
| Web Services      |                                                                                                                    |               |
| EJB               | Entity Bean<br>Session Bean<br>Local Home<br>Stateless Session Bean<br>Stateful Session Bean<br>MessageDriven Bean | Business Tier |
| Directory Service | JNDI                                                                                                               |               |
| Database          | JDBC                                                                                                               |               |
| JDBC              | Execute<br>Connection                                                                                              | Database      |
| Messaging         | JMS<br>Spring                                                                                                      |               |
| JMS               | Producer<br>Listener<br>Consumer                                                                                   | Messaging     |
| Spring            | Producer<br>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*.



# 11

---

## 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 428
- ▶ Locating the .NET Capture Points Files on page 429
- ▶ Coding Points in the Capture Points File on page 430
- ▶ Instrumentation Examples on page 435
- ▶ Understanding the Overhead of Custom Instrumentation on page 461
- ▶ Default Layers for Typical .NET Applications on page 462

## 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 926.

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 462).

## 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 14, "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                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Point-Name</b> | A unique name for the point.                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>class</b>      | 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.                                                                                                                                                                                                                                                                    |
| <b>method</b>     | 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.                                                                                                                                                                                                                                                             |
| <b>layer</b>      | Specifies a layer, sublayer, or tier under which the data from this point is grouped. Layers are a part of the instrumentation collection control.<br><br>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 926.

---

## Optional Point Entries

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

| Argument           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>keyword</b>     | <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 is the <code>RemotingServer</code> keyword and the <code>Remoting.points</code> file.</p> <ul style="list-style-type: none"> <li>▶ <b>HttpCorrelation.</b> Turns on correlation of client/server method calls via HTTP.</li> <li>▶ <b>WsCorrelation.</b> Turns on web service correlation logic on the client side and turns on correlation of raw HTTP client request calls across both the .NET and Java technologies.</li> <li>▶ <b>WCF.</b> Turns on the WCF feature.</li> <li>▶ <b>REST.</b> Turns on the WCF REST service instrumentation.</li> <li>▶ <b>lwmd.</b> Turns on lwmd instrumentation.</li> <li>▶ <b>Remoting.</b> Turns on .NET Remoting framework instrumentation.</li> <li>▶ <b>RemotingServer.</b> Associates points in a .NET Remoting server to special .NET Remoting logic for these points. See "How to Configure Instrumentation for .NET Remoting" on page 451.</li> </ul> |
| <b>ignoreClass</b> | <p>Specifies a comma-separated list of classes to ignore. Any class matching one of the classes specified with <b>ignoreClass</b> is not instrumented.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |



| Argument            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ignoreMethod</b> | Specifies a comma-separated list of methods to ignore. Any method matching one of the methods specified with <b>ignoreMethod</b> is not instrumented.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>ignoreTree</b>   | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>deep_mode</b>    | <p>Specifies how subclasses are handled. This argument accepts three values:</p> <ul style="list-style-type: none"> <li>▶ <b>none</b> - A value of none is similar to not specifying a <b>deep_mode</b> argument. It has no effect on how subclasses are handled.</li> <li>▶ <b>soft</b> - 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.</li> <li>▶ <b>hard</b> - 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. <b>Caution:</b> Hard"mode can lead to extensive instrumentation and very high probe overhead.</li> </ul> |
| <b>scope</b>        | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>ignoreScope</b>  | 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                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>detail</b></p> | <p>Provides more specific capture instructions.</p> <p>For the following 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><b>args:n</b> – 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><b>args:0</b> – Calls the ToString() on the current class instance or callee object. This is invalid for static methods.</p> <p><b>*args:1</b> – Marks (*) the argument as a key argument for the server requests if the method is a top-level request.</p> <p>The detail argument also takes the following value:</p> <p><b>tv:user_event</b> - Generates a TransactionVision event for the methods that match. As part of the TransactionVision event the parameters to the method are collected as the Request Payload and the return value is collected as the Response Payload. The values displayed are the ToString() values returned by the parameters or the return value objects. Note that all parameters and return values may not be collected.</p> <p>Provides extensive support for transaction tracing by enabling TransactionVision event generation from practically any given method in any .NET application. You specify the method on which you want a TransactionVision event generated. It is highly recommended that event generation is specified for one method at a time to avoid too many events and performance degradation in TransactionVision. Avoid using wild card specifications (but they are supported for convenience).</p> |

| Argument         | Description                                                                                                                                                                                                                                                                                                                                                                  |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>layerType</b> | Specifies special handling for some instrumented methods and accepts three values: <ul style="list-style-type: none"> <li>▶ <b>trended_method</b> – Identifies methods to be displayed in the Trended Methods view.</li> <li>▶ <b>sql</b> – Identifies methods used to capture SQL for the SQL views. These are set by HP Diagnostics and should not be modified.</li> </ul> |
| <b>signature</b> | Specifies the signature (return and parameter types); for example, <code>System.String(System.int32, System.String)</code> . Any valid regular expression can be used.                                                                                                                                                                                                       |

## 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 436
- ▶ "Wildcard method" on page 436
- ▶ "Ignore Specified Methods" on page 436
- ▶ "Capture Methods for the Trended Methods View" on page 437
- ▶ "Capture Only a Specific Method In a Class" on page 437
- ▶ "Capture a Specific Method That Returns a String" on page 438
- ▶ "Caller Side Instrumentation" on page 438
- ▶ "Argument Capture" on page 440
- ▶ "Configure WCF REST Services for Monitoring" on page 444
- ▶ "Deep\_mode Examples" on page 446

- "How to Configure and Set Up Points for Non-ASP.NET or Windows Applications" on page 447
- "How to Configure Instrumentation for .NET Remoting" on page 451

### 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 = .ctor
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
```

## Argument Capture

The arguments to be captured are specified in the detail key of a points file section.

The following example 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: detail=args:1,...args:4, \*args:3

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 have not 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
```

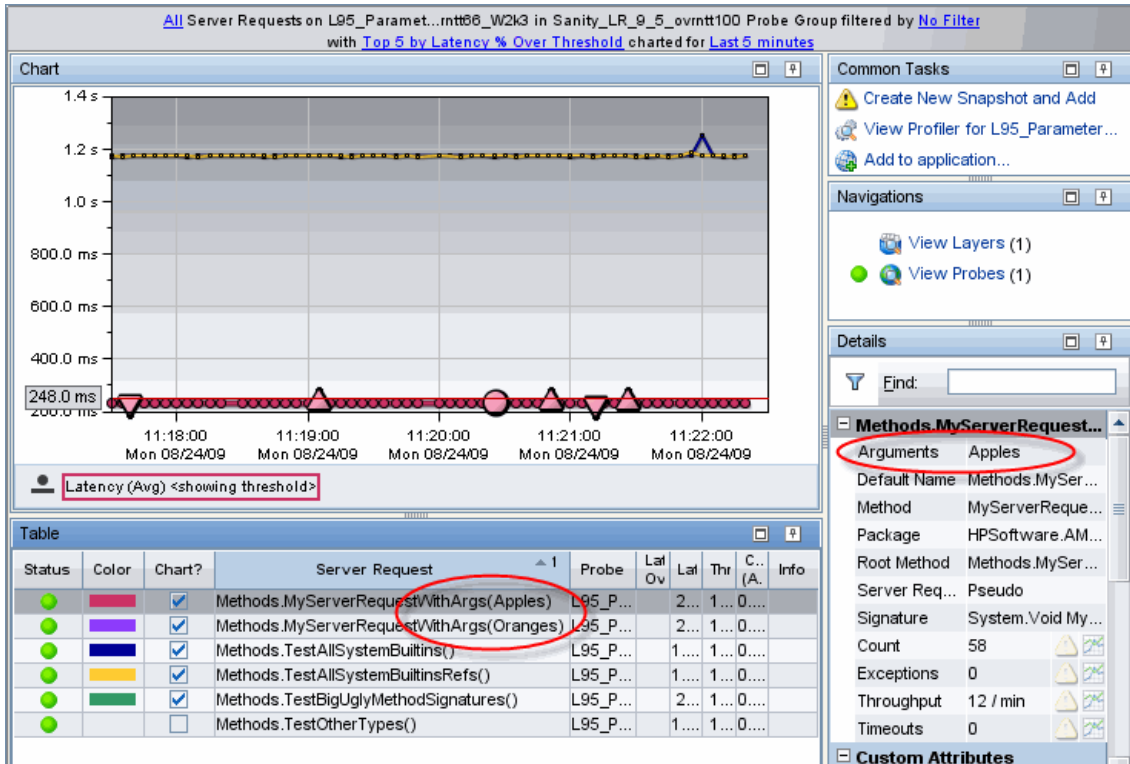
Here is the relevant code example:

```
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.
```

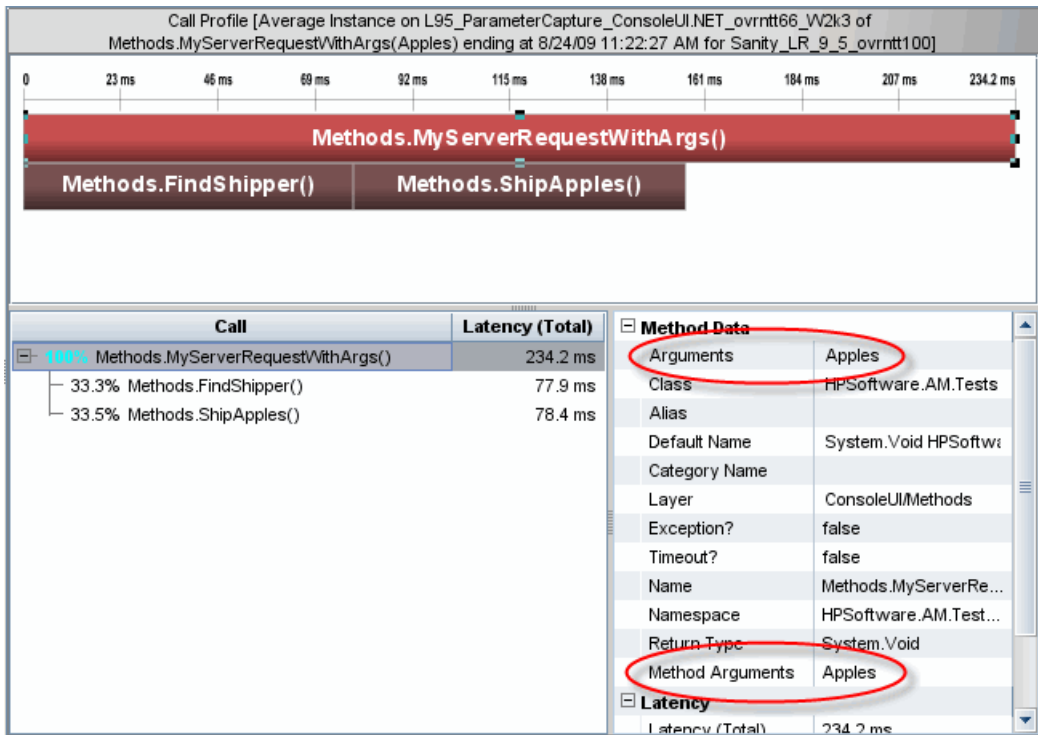
The value of QnameParam3 will be part of the identity of the server request if the top level method is methodWithParams.

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 no asterisk \*), they are displayed in the Call Profile view under Method Arguments only.

## Configure WCF REST Services for Monitoring

For a .NET Probe WCF REST services are monitored by default based on the **keyword=REST** value enabled out-of-the-box in the **WCF.points** file. These REST services will be monitored as web services and their performance data displayed in the Diagnostics UI SOA Services views.

You can further configure REST services as described in the sections below.

### REST Service Configuration

In WCF REST style services sometimes the operations are encoded as url parameters. For example:

HTTP Method: PUT Url: `http://localhost:81/RestNOSvc/AccountsRESTService/{ID}?op={OPERATION}` op can be "deposit" or "withdrawal"

To be able to distinguish operations in these types of services you can specify the operation parameters of the REST service method as a **key argument** to allow it to be displayed as a separate operation. See "Argument Capture" on page 440 for a general description of argument capture.

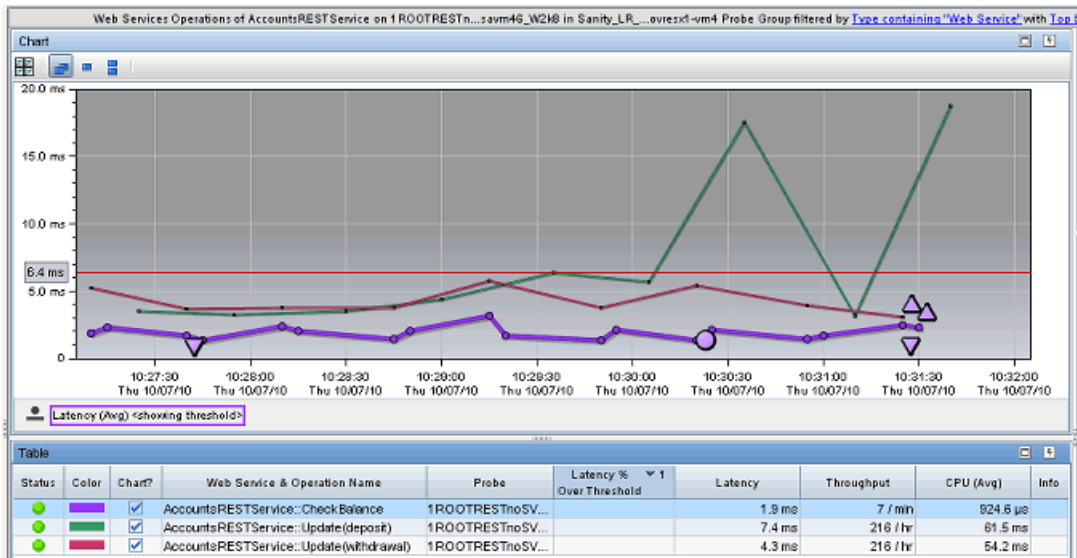
For example, for the method

```
[WebInvoke(UriTemplate = "{id}?op={operation}", Method = "PUT")]  
public TransactionResult Update(string id, string operation, long Amount)
```

The operation is the key argument and can be specified in the points file as:

```
[WebSite2-RestNOSvc]  
class = !HP.Test.WcfRestService.*  
method = Update  
detail = *args:2  
layer = WebSite2-RestNOSvc
```

The SOA Services Operations view example below shows the results of this configuration with separate operations shown in the table.



### REST Client Configuration

The REST service client is the same as an HTTP client call and cannot be distinguished. So for monitoring .NET applications that are REST service clients, the configuration option `<httpclient showurl="false"/>` should be set in the `probe_config.xml` file to avoid a large number of outbound calls and possible symbol table explosion. The number of calls is due to unique urls accessed by the client, often with ids encoded in the urls.

For example:

```
/RestNOSvc/AccountsRESTService/  
8FFD2F34-E334-4E1E-A940-50FCCCACE1D1
```

where the Guid represents different account ids.

## 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 the Diagnostics .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 cIread;

        public String readCl()
        {
            System.Console.WriteLine("Continue?");
            cIread = Console.ReadLine();
            return cIread;
        }
    }
}
```

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 634 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 `probe_config.xml` file. You can add it to the bottom of the 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>
```

## How to Configure Instrumentation for .NET 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 447, 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 447, 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>
```

You are 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(System...)
  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 447, 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 15, "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
- Understanding the .NET Agent Configuration File
- Advanced .NET Agent Configuration





# 12

---

## 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 466
- ▶ Configuring the Diagnostics Server for a Large Installation on page 470
- ▶ Overriding the Default Diagnostics Server Host Name on page 476
- ▶ Changing the Default Diagnostics Server Port on page 476
- ▶ Migrating Diagnostics Server from One Host to Another on page 477
- ▶ Configuring the Diagnostics Server for Multi-Homed Environments on page 479
- ▶ Reducing Diagnostics Server Memory Usage on page 483
- ▶ Configuring Server Request Name Based Trimming on page 484
- ▶ Automating Composite Application Discovery in HP Diagnostics on page 485
- ▶ Preparing a High Availability Diagnostics Server on page 488
- ▶ Configuring Diagnostics for HP ServiceGuard (HA solution) on page 489
- ▶ Diagnostics Server Assignments (LoadRunner/Performance Center Runs) on page 491
- ▶ Configuring the Diagnostics Server for LoadRunner Offline Analysis File Size on page 492

- ▶ Configuring Business Service Management Sample Queue Size and Web Services CI Frequency on page 495
- ▶ Configuring Diagnostics Using the Diagnostics Server Configuration Pages on page 496
- ▶ Optimizing the Diagnostics Server in Production to Handle More Probes on page 496
- ▶ Configuring a Custom Context Root on page 497

## 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 512.

### 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 gateway 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 commander server 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 22, “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 (especially if there aren’t enough CPU resources available such as multiple CPUs/cores or fast CPUs).

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	1400 MB
Up to 200 Java Probes	3000 MB (64bit)
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.

---

It is highly recommended to run Diagnostics on a system with more than two CPUs or cores (four cores are recommended). In such an environment, change the Garbage Collector to concurrent mark and sweep:

`-XX:+UseConcMarkSweepGC`

For 64bit JVMs, make sure to enable this option:

`-XX:+UseCompressedOops`

For VMware installations follow the best practices as described in VMware's "Enterprise Java Applications on VMware Best Practices Guide". In essence, use multiple vCPUs and fixed memory allocations (no ballooning or swapping to disk) and ensure installation of VMware Tools.

- 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). But note that by default trimming doesn't affect synthetic transactions (BPM/vugen/LoadRunner/Performance Center) so all these server requests are reported.

For more information on trimming, see “Configuring Server Request Name Based Trimming” on page 484 for the server, “Configuring Latency Trimming and Throttling” on page 641 and “Configuring Depth Trimming” on page 646 for .NET Agent, “Controlling Automatic Method Trimming on the Agent” on page 508 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 876.

## 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	<diagnostics_server_install_dir>/etc ► webservice.properties – jetty.port ► server.properties – commander.url ► probe/etc/dispatcher.properties – registrar.url
Diagnostics mediator server	<diagnostics_server_install_dir>/etc ► server.properties – commander.url <diagnostics_server_install_dir>/probe/etc ► dispatcher.properties – registrar.url
Probes	<probe_install_dir>/etc ► dispatcher.properties – registrar.url

## Migrating Diagnostics Server from One Host to Another

The following procedure shows how to migrate your Diagnostics Server from one host to another and assumes the new host name is different from the old host name.

**To migrate a Diagnostics server from one host to another:**

- 1** Ensure that the existing 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.
- 2** Unregister the Diagnostics Commander Server from Business Service Management.
- 3** Install the new Diagnostics Server on the new host.
- 4** On Windows, the Diagnostics Server is started automatically when the installer finishes so you must shut down the Diagnostics Server.

On UNIX the server is not automatically started so you do not need to shut it down.

Ensure that the Diagnostics Server has been shut down by verifying that there are no java/javaw processes in your process list. On Windows systems, you can use the Task Manager to do this and on UNIX systems, you can use ps.

Be sure you know the host name of the old Diagnostics Server (you can find the name in the /archive directory).

- 5** Delete the <diagnostics\_server\_install\_dir>/archive directory on the new Diagnostics Server.
- 6** Copy the <diagnostics\_server\_install\_dir>/archive folder and all subfolders from the old server into the new server <diagnostics\_server\_install\_dir>/.
- 7** If the host name for the new Diagnostics Server is different than the host name for the old Diagnostics Server, you must rename <diagnostics\_server\_install\_dir>/archive/mediator-<host-name> so that <host-name> reflects the new Diagnostics Server host name. For example, if your old host name 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>.
- 8** Delete the <diagnostics\_server\_install\_dir>/storage/ directory for the new Diagnostics Server.
- 9** Copy the <diagnostics\_server\_install\_dir>/storage/ folder and all subfolders from the old server into the new server <diagnostics\_server\_install\_dir>/.
- 10** On the new server rename <diagnostics\_server\_install\_dir>/storage/server-<hostname> so that <host-name> reflects the new Diagnostics Server host name. For example, if your old host name was oldhost and the new host name is newhost you would change  
  
<diagnostics\_server\_install\_dir>/storage/server-<oldhost> to  
<diagnostics\_server\_install\_dir>/storage/server-<newhost>
- 11** Copy the <diagnostics\_server\_install\_dir>/etc folder from the old server into the new server <diagnostics\_server\_install\_dir>/ and copy the new license to etc folder.

- 12 Start new Diagnostics server and register the new Diagnostics server in Business Service Management.
- 13 If the new server was the Commander then on all the mediators, you need to scan the etc folder and change the old server name to the new server name. Double check the dispatcher.properties file to make sure the commander server hostname changed. Then restart all the mediators.

There is no change required on the probe side unless the probe is directly reporting to the commander server or you are migrating the mediator server the probe is connected to. If that is the case, scan the etc folder on the probe system and change the old server name to the new server name (double check the dispatcher.properties file to make sure the mediator server hostname changed).

## 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 476.

---

## 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 Server Request Name Based Trimming

Server Request 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:** Server request 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 server request 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.

## Automating Composite Application Discovery in HP Diagnostics

Composite Application Discovery (CAM) provides a convenient way to group application servers (probes) and to continuously detect new components that are connected to these application servers by following the calls a probe is making to these other components.

In addition to configuring applications in the UI, Diagnostics provides scripting support for CAM. This allows the dynamic creation of new applications based on newly added probes outside the UI.

### Scripting Applications

Scripts that are used to create new applications are stored on each mediator in `etc/appDiscoveryRules.properties`. The script that ships with Diagnostics contains some examples.

Typically, applications are based on certain patterns that are available in entity properties such as probe name, probe group name or server request name. The `/groupby` path is used to query the Diagnostics data model and select instances. This query path is used in scripts such as the one below for application discovery and is also used to automate setting thresholds and alerts. See the *Diagnostics Data Model and Query API Guide* for details.

The following example illustrates an easy way to create new applications based on parts of the probe name.

```
# Example:
#
# Put all probes with a particular naming pattern into
# an application.
#
# If you have a very consistent naming convention for probes, you
# can auto-insert new probes into the appropriate application.
#
# In the below example, we put any probe that has a name starting
# with "cs_" into the "Sales" application.

/groupby[name\='Default\ Client']/probegroup/probe=\
  String probeName = probe.getName();\
  if( probeName.startsWith("cs_") ) {\
    uid=name="Sales";\
  }
```

The **/groupby** definition (in **blue bold text**) periodically queries all probes on this mediator and executes the script (JavaScript in *red italicized text*) against the returned probe names. In the example above, the code creates an application with the name **"Sales"** for all probes that start with **"cs\_"**.

In addition to the name, application permissions can be specified. The script includes more examples for specifying application permissions.

Further it is possible to automatically include all related probe entities such as Server Requests and SQL statements. To do this, set the variable **discoveryPolicies** with the value **"applyAppFilterToProbeContents"**:

```
/groupby[name\='Default\ Client']/probegroup/probe=\
  String probeName = probe.getName();\
  if( probeName.startsWith("cs_") ) {\
    uid=name="Sales";\
    discoveryPolicies="applyAppFilterToProbeContents";\
  }
```

## Moving Composite Applications Between Environments

The scripting approach provides an easy way to move applications between environments such as from QA to production. All application definitions, however, need to be created using the script. One master script can be used on all mediators even if the probes are not reporting to this mediator.

It is important to use a naming scheme that works between production and pre-production. This can be achieved by:

- ▶ Putting probes in specific probe groups that are constant between production and pre-production
- ▶ Using a probe naming convention that allows the script to create an application name as shown in the example above.

If the probes are in the same probe group and this name is constant between environments (but the probe name changes), use **probegroup.getName()** in the script to access the probe group name:

```
/groupby[name\='Default\ Client']/probegroup/probe=\
  String probegroupName = probegroup.getName(); \
  String probeName = probe.getName();\
  if( probegroupName.startsWith("cs") ) {\
    uid=name="Sales";\
    discoveryPolicies="applyAppFilterToProbeContents"; \
  } \
  else if (probegroupName.startsWith("is") ) { \
    uid=name="Information Systems";\
    discoveryPolicies="applyAppFilterToProbeContents"; \
  }
}
```

This script is generic and can be exchanged between environments.

## 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 488.

---

**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 Commander 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 reporting data to Business Service Management.

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 five 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 five minutes, note the size of the offline file.
- 3** Extrapolate the size of the offline file after an 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/server.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 793.

To cause the configuration changes to take effect, restart the Diagnostics Server in Mediator mode by clicking **Restart Mediator**.

## Configuring Business Service Management Sample Queue Size and Web Services CI Frequency

The following configurations are applicable to Business Service Management integrations.

### Configuring Business Service Management Sample Queue Size

Business Service Management Sample 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 Application Management for SOA. You can see the following message in the log: BAC sample being dropped since too many are waiting for delivery.

You can increase the samples queue size by setting the server property, **dispatcher.server.wdedelivery.max.queue.size** to configure the WDEDelivery queue size.

### Frequency of Web Service CIs

The Web Services CIs are created and added to the Run-time Service Model automatically by Diagnostics using a default frequency.

You could change the timing of the process that adds the Web Services CIs to the Run-time Service Model. 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=300s, 5m)
- **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
```

## Configuring a Custom Context Root

To configure a custom context root on Diagnostics commander server set the following in the `etc/webserver.properties` file:

```
# Reverse proxy prefix for Diag URLs (e.g. /diag/customername in ES
environment) # reverse_proxy.prefix=
```

If BSM is also configured with a custom context root (instead of '/topaz'), then in the BSM Admin Diagnostics Configuration page you will also need to specify the context root configured on the diagnostics side for the commander.

# 13

---

## 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 Overview on page 500
- ▶ Disabling the Java Diagnostics Profiler on page 501
- ▶ Controlling Probe Logging on page 502
- ▶ Setting the Probe's Host Machine Name on page 503
- ▶ Specifying a Different Probe IP Address on page 505
- ▶ Set the Active Products Mode on page 505
- ▶ Controlling Automatic Method Trimming on the Agent on page 508
- ▶ Configuring URI Truncation, Mapping and Trimming on page 510
- ▶ Configuring an Agent for a Proxy Server on page 511
- ▶ Time Synchronization for Probes Running on VMware on page 512
- ▶ Limiting Exception Tree Data on page 512
- ▶ Diagnostics Probe Administration Page on page 515
- ▶ Authentication and Authorization for Diagnostics Java Profilers on page 518
- ▶ Configuring Collection of CPU Time Metrics on page 521
- ▶ Configuring Consumer IDs on page 524

- ▶ Configuring SOAP Fault Payload Data on page 535
- ▶ Configuring REST Services on page 537
- ▶ Customizing Grouping JMS Temporary Queue/Topics on page 537
- ▶ Configuring SQL Query Parsing on page 537
- ▶ Configuring Display of Application Name for Server Requests on page 538
- ▶ Maintaining Probe Settings from the Java Profiler UI on page 539
- ▶ Generating Performance Reports for JUnit Tests on page 547

## Advanced Configuration Overview

The following bullet points list the probe configuration sources of information to consult to configure your environment.

- ▶ If you have a probe that you want to prevent others from using in Profiler mode, see “Disabling the Java Diagnostics Profiler” on page 501.
- ▶ 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 502.
- ▶ 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 503.
- ▶ 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 508.
- ▶ 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 511.
- ▶ 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 “Time Synchronization for Probes Running on VMware” on page 512.

- ▶ If you are running in a virtual environment, see “Time Synchronization for Probes Running on VMware” on page 512.
- ▶ If you need to limit the amount of exception data, see “Limiting Exception Tree Data” on page 512.
- ▶ 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 524.

## 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**.

## 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 511.

## Setting the Probe’s Host Machine Name

The probe’s host name registers the probe with the Diagnostics commander server. The Diagnostics commander server 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.

---



## Specifying a Different Probe IP Address

The `probe.host.ip.address.override` property (located in the property file `<probe_install_dir>/etc/dispatcher.properties`) enables you to override the Probe's IP address. You can use this property when you want the probe to provide the server with a different IP address, for example, a Virtual IP that would allow the server to communicate to the probe through a tunnel.

## Set the Active Products Mode

The Java Agent mode is typically set for you based on the options you enter in the setup program. But you can set the mode manually as described in this section.

The Java Agent can be set in different modes to do the following:

- ▶ Monitor applications from development through pre-production testing and into production
- ▶ Work with other HP Software products
- ▶ Be used as a standalone Diagnostics Java Profiler not reporting to a server or to other HP Software products

The mode the Java Agent works in is determined by the `modes` value of the `active.products` property located in the property file `<probe_install_dir>/etc/probe.properties`.

The `modes` value in the `active.products` property is also used in determining usage against the license capacity (see “License Information Based on Currently Connected Probes” on page 85). For Diagnostics there are two types of LTUs (License to use):

- ▶ AM - When using of the product in an enterprise mode, typically in a production environment.
- ▶ AD - When using the product in a pre-production load testing environment with probes in LoadRunner or Performance Center runs.

The value of the `active.products` property is initially set at the time you install the Java Agent.

To change the value of the **active.products** property you can edit the property file and restart the application server. Or you can re-run the Java Agent Setup Module and use the Change option to set the mode to Diagnostics Profiler Mode (PRO), Application Management/Enterprise Mode for Diagnostics (Enterprise) and/or TransactionVision (TV) or Diagnostics Mode for LoadRunner/Performance Center (AD).

---

**Note:** To use the standalone Diagnostics Profiler for Java trial copy in enterprise mode or integrated with other HP Software products, contact HP Software Customer Support to purchase HP Diagnostics.

---

To see Diagnostics data in the user interface of the interfacing HP Software products, you must perform additional configuration steps. See the sections in “Setting Up Integration with Other HP Software Products” on page 735 for details on integration with Business Service Management, LoadRunner or Performance Center. 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 performance metrics and presents them in the standalone Diagnostics Java Profiler user interface which is made available through a URL on the agent host.

If you are running the Java Agent as part of the Java Diagnostics Profiler trial copy, restrictions are placed on the agent to limit the load it can handle.

If you are running the Java Agent as part of the full Diagnostics enterprise product, or along with another HP Software product, the Profiler is enabled without the load restrictions.

PRO mode is not used in determine usage against license capacity.

## **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 the full Diagnostics enterprise product. Although you must also do additional configure to enable these integrations (see the sections in “Setting Up Integration with Other HP Software Products” on page 735 for details).

In Enterprise mode data will also be sent to the Diagnostics Java Profiler.

In Enterprise mode you must also register the agent with the Diagnostics Servers (see “Register the Agent with the Diagnostics Servers” on page 150).

Enterprise mode is the default for Java Agents (if you don’t specify AD or AM mode). In Enterprise mode the agents are counted against the AM license capacity.

## **AM Product Mode**

In AM mode the Java agent will capture all instrumentation data. You can set 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.

Agents in AM mode will always be counted against the AM license capacity.

AM mode supersedes all other modes except for AD.

## **AD Product Mode**

In AD mode the Java agent will only capture data during a LoadRunner or Performance Center run and the results will be stored in a specific Diagnostics database for that run, for example, Default Client:21.

When the agent is in AD mode it will not use resources or send any data to the server unless the probe is part of a LoadRunner/Performance Center run.

See Chapter 24, “Setting Up HP LoadRunner and HP Diagnostics Integration” for how to set up LoadRunner integration or see Chapter 25, “Setting Up Performance Center to Use Diagnostics” for how to setup Performance Center integration.

Use this mode to prevent an agent in a QA environment from using additional resources and continually report data to the Diagnostics server when a load test is not running.

Another advantage of running a probe in AD mode is that probes in AD mode are only counted against the AD license capacity when in a LoadRunner or Performance Center run. For example if you have 20 probes installed in LoadRunner/Performance Center AD mode but only 5 are in a run, then only 5 are counted against AD license capacity.

### **TV Product Mode**

This mode will send events to Transaction Vision. This mode can be combined with other modes. TV mode is not used to determine usage against HP Diagnostics license capacity.

## **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 and /login.do calls a() calls b() calls c() then only /login.do, a, and b are captured.

Note that setting a low **maximum.stack.depth** can significantly reduce the overhead of capture.

## Configuring URI Truncation, Mapping and Trimming

Any HTTP/S server request URI can be transformed before being reported by the probe. The possible transformations are based on regular expression matching and replacement controlled by the **uri.pattern.replace** property in **dynamic.properties**. The value of the property is a comma-separated list of pattern replacement operations to attempt on each URI.

This can be used when you are seeing too many server requests and you want to replace many server request URIs with one simplified server request URI that aggregates them.

Truncate or map URIs using `s/pattern/replace/` syntax. To perform multiple operations use a comma-separated list. The operations are performed in order.

For example, to truncate before a string, match the string and any characters that follow it and leave replace empty. In this example '\$' matches end-of-line.

```
s/string.*$//
```

Comments in the **dynamic.properties** file under URI Truncation and Mapping provide details and more examples.

---

**Important:** Overuse of this feature will impact performance.

---

If you have too many server requests you can also use the property **maximum.uri.pathsegments** in the **capture.properties** file to trim server requests down to *n* path segments.

The default is -1 which disables the property. For probes reporting in a SaaS environment (SaaS selected in the Java Agent setup) **maximum.uri.pathsegments** is set automatically to 2 to ensure the volume of server request data sent to HP hosted servers is not too large.

For example a setting of 2 would result in no more than two path segments. So `http://localhost:8080/path1/path2/path3` will trim down to `http://localhost:8080/path1/path2/`.

You could use `uri.pattern.replace` and then set `maximum.uri.pathsegments` to trim down to a certain number of path segments. Or use just one property or the other.

## 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.

## 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](http://www.vmware.com/pdf/vmware_timekeeping.pdf)) in a section on "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

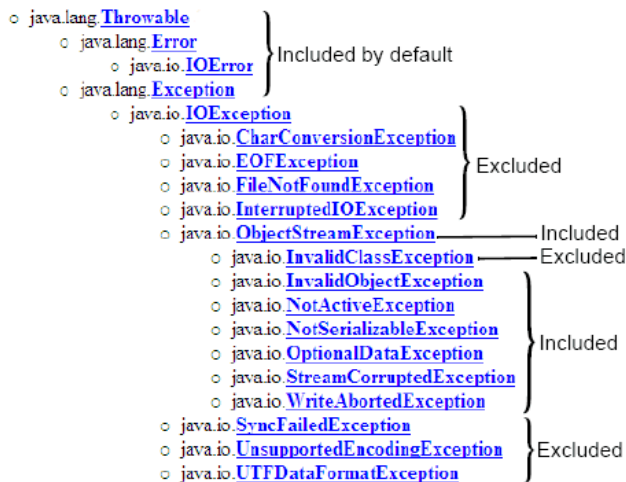
The following example causes exceptions of type `ClassNotFoundException` and all its subtypes to be ignored.

```
...
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

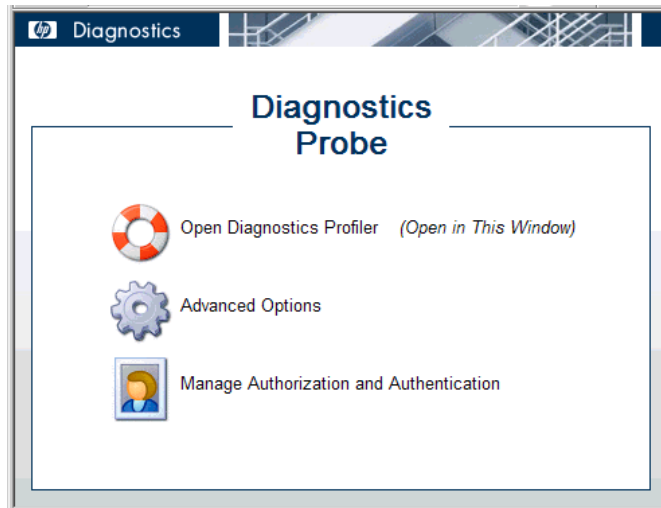
You can use the Diagnostics Probe Administration page to configure Java Agent and Profiler settings. Access the Diagnostics Probe Administration page directly from your browser.

### 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 516.
  - **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 Diagnostics Java Profilers” on page 518.

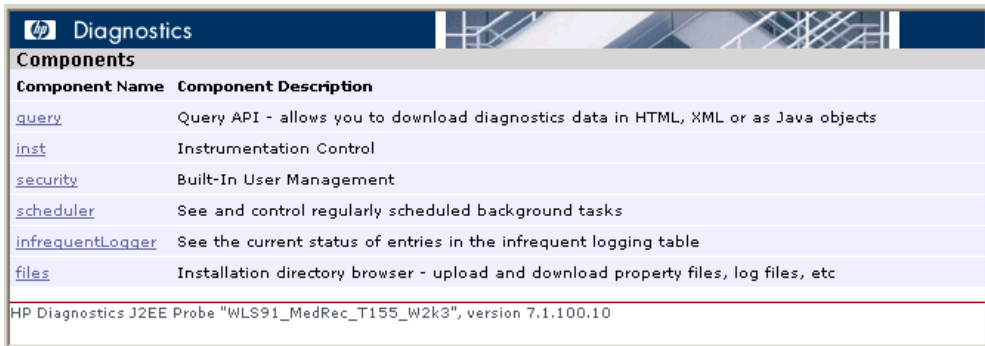
## Diagnostics Probe Components Page

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 515.
- 2 Click **Advanced Options**.
- 3 If prompted, enter your user name and password.

The Components page opens.



Diagnostics	
Components	
Component Name	Component Description
<a href="#">query</a>	Query API - allows you to download diagnostics data in HTML, XML or as Java objects
<a href="#">inst</a>	Instrumentation Control
<a href="#">security</a>	Built-In User Management
<a href="#">scheduler</a>	See and control regularly scheduled background tasks
<a href="#">infrequentLogger</a>	See the current status of entries in the infrequent logging table
<a href="#">files</a>	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 337.
- ▶ **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 797.
  - ▶ 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 Diagnostics Java Profilers” on page 518.
- ▶ **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 Diagnostics Java Profilers

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 797.

---

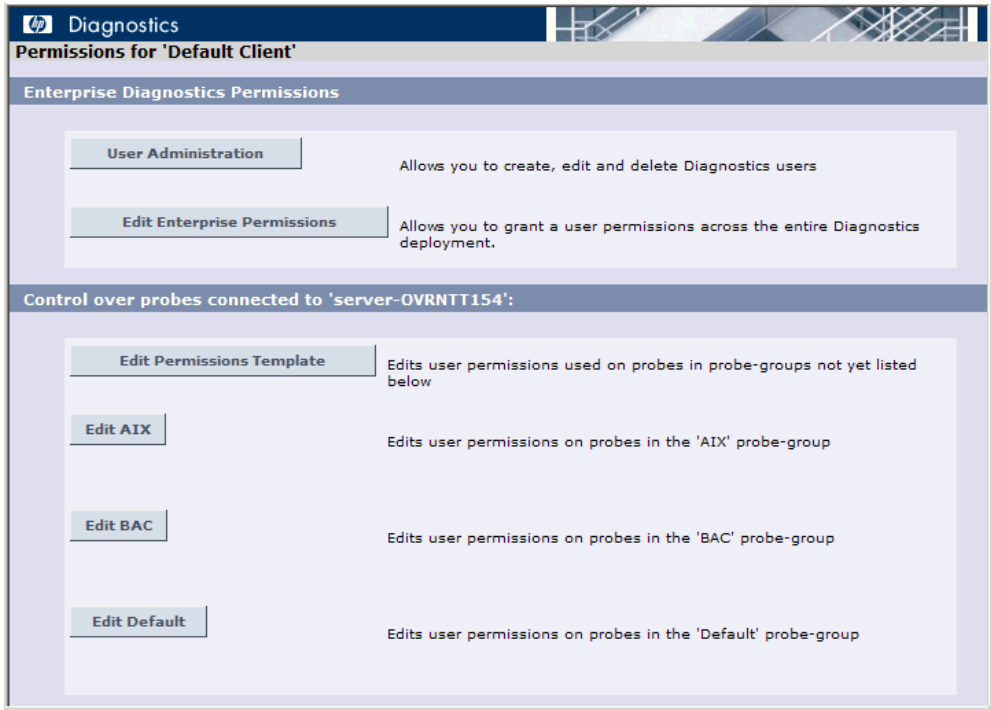
**To manage authentication and authorization for users of the standalone Java Diagnostics Profiler:**

### **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
<b>View</b>	The user can view Profiler data from the UI.
<b>Execute</b>	The user can perform garbage collection and clear the performance data held by the Profiler.
<b>Change</b>	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 Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](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](http://www.vmware.com/pdf/vmware_timekeeping.pdf) and “Time Synchronization for Probes Running on VMware” on page 512.

---

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 in a SOAP Header
- A Value in a SOAP Envelope
- A Value in the SOAP Body
- A Value in an HTTP Header
- A JMS Queue Name (or topic name) for SOAP over JMS web services
- A JMS Message Property for SOAP over JMS web services
- A JMS Message Header for SOAP over JMS web services
- A specific IP Address
- A Range of IP Addresses

**Important:** Defining consumer ID based on SOAP header, envelope, or body requires the Diagnostics SOAP message handler for Java probes. 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 239 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 Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](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.

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 526
- “About Consumer ID Rules” on page 527
- “Consumer ID Rules Syntax and Examples for Java Agents” on page 529

## 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 for SOAP payload capture.

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 529.
- 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 926 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 can be included with the SOAP fault data. SOAP payload is only captured when there is a SOAP fault.

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.

Because payloads can contain sensitive information such as credit card numbers, payload capture on SOAP faults is *disabled* by default.

To enable payload capture on SOAP fault set **max.soap.payload.bytes** to a value greater than zero, 5000 is recommended, in the **dispatcher.properties** file on the Java agent. This is the number of bytes captured, so if the payload you see in the UI indicates it is too small you can increase this number. By default the value is set to zero to disable payload capture.

Capturing SOAP payload requires the Diagnostics SOAP message handler for Java probes. 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 239 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 Support Matrix at

[http://support.openview.hp.com/sc/support\\_matrices.jsp](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 the comments in the following file for configuration details: `<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 SQL Query Parsing

If there are a large number of SQL queries using literals it can overwhelm the server symbol table. In these situations you can configure the `sql.parsing.mode` property in the `dispatcher.properties` file on the Java Agent. The possible mode settings are as follows:

- 1 - just methods, no SQL queries.
- 2 - main categories for SQL queries (select/update/insert/delete/...).
- 3 - (default) a measurement per whole SQL query aggregating similar statements into a single measurement (ignore literals, keyword case...).

4 - a measurement per whole SQL query aggregating only identical statements.

```
sql.parsing.mode = 3
```

Another property in the **dispatcher.properties** file can be used to limit the number of different SQL statements collected in case of temporary database tables, allowing you to fold down the table names using an SQL statement regular expression substitution. The property is **sql.pattern.replace** (see the comments in the **dispatcher.properties** file for more information).

## Configuring Display of Application Name for Server Requests

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 Diagnostics 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, by default 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 thread stack trace sampling, collection of CPU time metrics (using timestamping) and reporting collection leaks.

The screenshot shows the 'Probe Settings' configuration window. It includes the following controls:

- Thread Stack Trace Sampling:** A dropdown menu set to 'Auto'.
- Maximum Stack Trace Depth:** A text input field containing '60'.
- Sampling Interval:** A text input field containing '150' followed by 'ms'.
- Tardy Method Latency Threshold:** A text input field containing '100' followed by 'ms'.
- Collect CPU Timestamps:** A dropdown menu set to 'For Server Requests Only'.
- Report Collection Leaks:** A checked checkbox.
- Collection Leaks Flag Threshold:** A text input field containing '60' followed by 'minutes'.
- Collection Leaks Unflag Threshold:** A text input field containing '120' followed by 'minutes'.

At the bottom right, there are two buttons: 'Apply Changes' and 'Cancel Changes'.

When you click **Apply Changes** on the Java Diagnostics Profiler Configuration tab, all the updates you made in the Probe Settings sections of the Configuration tab are applied to the capture points file and the property files.

---

**Note:** Your changes take effect immediately. There is no need to restart the application (or probe).

---

The following sections describe each of the Probe Settings sections:

- “Configuring Thread Stack Trace Sampling” on page 540
- “Controlling CPU Timestamp Collection” on page 545
- “Enabling and Configuring Collection Leak Reporting” on page 546

## 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	Enabled ▾	Maximum Stack Trace Depth	60
Sampling Interval	150 ms	Tardy Method Latency Threshold	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](http://www.vmware.com/pdf/vmware_timekeeping.pdf) and “Time Synchronization for Probes Running on VMware” on page 512.

---

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 521) 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

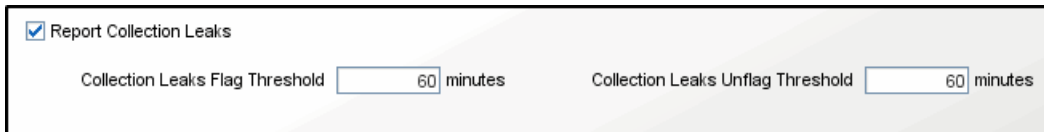
Data collection and reporting for collection leak pinpointing is enabled by default for the probe. The [**Collection Leak Pinpointing**] keyword = **clp** point in the probe's **etc/auto\_detect.points** file is set to true by default.

---

**Note:** You must run the JRE Instrumenter using the appropriate mode for your application server 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**.

## Generating Performance Reports for JUnit Tests

When you run JUnit tests, you can enable and configure the Java Agent so that it generates a performance report for all of your unit tests. This is useful for finding out if the performance (latency/CPU) of a particular test has changed over time.

When the unit test finishes, the Java Agent creates a CSV file for each test method (represented as a server request). This CSV file contains a complete listing of all test methods that were executed in each JVM instance, usually per test class. The CSV file can be opened in a spreadsheet program to analyze and visualize performance characteristics (the Filter function in Excel is very helpful for selecting specific methods).

Following is an example of a CSV file:

```
Date,Server Request,Avg Latency,Count,Min Latency,Max Latency,Cpu
Time,Exceptions
Fri Sep 23 12:55:22 PDT
2011,UT_SiSXmlDataReader.testDataSample(),1068.81,1,1068.81,1068.81,374.403,0
Fri Sep 23 12:55:40 PDT
2011,UT_SiSXmlDataReader.testDataSample(),1064.845,1,1064.845,1064.845,405.60
2,0
Fri Sep 23 12:55:57 PDT
2011,UT_SiSXmlDataReader.testDataSample(),1141.689,1,1141.689,1141.689,358.80
2,0
Fri Sep 23 12:56:27 PDT
2011,UT_SiSXmlDataReader.testDataSample(),1474.81,1,1474.81,1474.81,468.003,0
```

The latency times are in milliseconds (ms).

By default the data for each test execution is appended to the CSV files. This is especially useful when tests are run as part of a Continuous Integration cycle which allows you to capture results over time.

To use this functionality, enable the Java Agent in the JUnit test execution by specifying the following JVM parameters:

JVM Parameter	Description
-javaagent:<Java_Agent_Home>/DiagnosticsAgent/lib/probeagent.jar (UNIX) or -javaagent:<Java_Agent_Home>\DiagnosticsAgent\lib\probeagent.jar (Windows)	Enables the agent by specifying the path to the agent JAR file. You can use <b>-Xbootclasspath/p:&lt;JavaAgent_install_dir&gt;/DiagnosticsAgent/classes/boot</b> instead if you use JRE 1.4 .
-Ddispatcher.ac.autostart=true	Tells the agent to start profiling immediately.
-Dcapture.exit_report=dir=perftest:append	Instructs the agent to produce a performance report to the specified directory and to append the results. (To override the file, replace <b>append</b> with <b>override</b> .)
-Ddispatcher.minimum.fragment.latency=1ms	Collects only server requests (such as execution of JUnit test methods) that have latency above 1ms.

The following example shows an integraton into ANT:

```
<junit dir="${build}" fork="yes" forkmode="perTest" printsummary="yes"
jvm="${env.JAVA_HOME}/bin/java">

...

  <jvmarg value="-javaagent:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/
probeagent.jar"/>
  <jvmarg value="-Ddispatcher.ac.autostart=true"/>
  <jvmarg value="-Dcapture.exit_report=dir=<dir_name>:append"/>
  <jvmarg value="-Ddispatcher.minimum.fragment.latency=1ms"/>

...

</junit>
```

In addition to the above settings, the JUnit point needs to be activated (set **active=true**) in **<Java\_Agent\_Home>/DiagnosticsAgent/etc/auto\_detect.points**:

```
[JUnit]
class    = junit.framework.TestCase
method  = !test.*
signature = !.*
deep_mode = hard
layer    = JUnit
active   = true
```

---

**Note:** If you use JUnit 4.x and your unit test classes are not a subclass of `junit.framework.TestCase`, you need to change the class definition in the above JUnit point to match your unit test classes.

---



# 14

---

## 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 551
- .NET Agent Configuration Elements on page 552

### 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*.

## .NET Agent Configuration Elements

### <appdomain> element

#### Purpose

Builds an AppDomain inclusion list for processes that host multiple application domains. 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. (IIS path qualified, see the example below.)
website	string	none	The name of the Website for those appdomains that are Websites (information only)

#### Elements

Number of Occurrences	zero or more
Parent Elements	process
Child Elements	bufferpool, credentials, diagnosticserver, mediator, id, ipaddress, logging, lwmd, modes, points, profiler, sample, trim, webserver, symbols, filter, topology



**Example**

```
<appdomain enabled="true" name="1/ROOT/MSPetShop"/>
Where 1/ROOT is the Website ID and MsPetShop is the Virtual DirName
```

```
<appdomain enabled="false" name="1/ROOT" website="Default Web Site">
  <points file="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 demomode turned on, all outbound calls will be shown irrespective of any other instrumentation.

Once the calls leading to the outbound calls of interest are identified then demomode should be turned off and "custom" instrumentation added to ensure that call stacks leading to the outbound calls are apparent.

---

**Note:** It is recommended to TURN THIS OFF under production environments.

---

Demomode is used primarily to find outbound calls (webserver, http, remoteing, msmq) when the method making them is not instrumented. It is meant as a way to quickly find how applications may be connected without having to instrument application specific methods . This may be too noisy in production situations but is useful when you there is a lack of upstream instrumentation and you don't know where the outbound call is being made from. It can be used for all kinds of applications including ASP.NET.

**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 <diagnosticserver> element. The question marks (?) indicate that appropriate values need to be substituted.

```
<diagnosticserver 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 656.

## <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 user interface'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.  The value should be set to false for REST service client applications

**Elements**

Number of Occurrences	Zero to one.
Parent Elements	probeconfig, process, appdomain
Child Elements	none

**Example**

```
<httpclient showurl="true"/>
```

**<gentvhttpeventforwcf> element****Purpose**

Setting this option enables generation of a TransactionVision event for a WCF service with any binding that uses IIS (http based) hosting. Some WCF services may use a custom or private binding that is not supported as a true web service and in these types of cases TransactionVision web service events would not be generated unless you enable this option.

**Attributes**

Attributes	Valid Values	Default	Description
enabled	true, false	false	Enables/disables the generation of an http event for a WCF service with any binding that uses IIS (http based) hosting. If enabled, provides TransactionVision web service events.

**Elements**

Number of Occurrences	Zero to one.
Parent Elements	probeconfig, process, appdomain
Child Elements	none

**Example**

```
<gentvhttpeventforwcf enabled="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.

**Important:** The account under which the .NET process is running (for IIS the AppPool Account) has to be provided **delete** privileges on the log folder. This is not available by default on the NETWORK SRERVICE account or the App Pool Identity Account (which is the default Application Pool Account).

### 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 <hr/> 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="warning" />  
</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"	
max	number	10	The maximum size of a probe log file. After the log reaches this size no more logging will occur.



**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 max="10" level="INFO"/>
```

## <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 666 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 product mode(s) the .NET Agent should run in. See "Controlling Which HP Software Products the Agent can Work With" on page 637 for more information about using the different modes.

The **<modes>** element is also used in determining usage against the HP Diagnostics license capacity (see "License Information Based on Currently Connected Probes" on page 85 for more information on licenses).

The value of the **<modes>** element is initially set at the time you install the agent.

The .NET agent can set in different modes to do the following:

- ▶ Monitor applications from development through pre-production testing and into production.
- ▶ Used with other HP Software products.
- ▶ Used as a standalone Diagnostics Java Profiler not reporting to a server or to other HP Software products.



**Attributes**

Attributes	Valid Values	Default	Description
enterprise	true false	Depends on mode chosen in installation. <ul style="list-style-type: none"> <li>▶ true if pro is false</li> <li>▶ false if pro is true</li> </ul>	<p>Sets agent to run in enterprise mode (probe is working with Diagnostics Server).</p> <p>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.</p> <p>Enterprise mode is the default for .NET Agents (if you don't specify AD or AM mode). In Enterprise mode the agents are counted against the AM license capacity.</p>
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.

Attributes	Valid Values	Default	Description
ad	true false	false	<p><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>.</p> <p>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).</p> <p>Agents in AD mode will only be counted against AD license capacity when the probe is running in a LoadRunner or Performance Center test run. When not in a test run the agent does not count against license capacity.</p> <p>For example if 20 probes are installed in LoadRunner/ Performance Center AD mode but only 5 are in a run, then only 5 are counted against AD license capacity.</p>

Attributes	Valid Values	Default	Description
am	true false	► false	<p><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.</p> <p>Agents in AM mode will always be counted against the AM license capacity.</p>

Attributes	Valid Values	Default	Description
pro	true false	Depends on mode chosen in installation.  <ul style="list-style-type: none"> <li>▶ true if enterprise is false</li> <li>▶ false if enterprise is true</li> </ul>	Sets the agent to run in Profiler mode.  This mode sends data to the profiler. This mode can be combined with other modes. Agents in pro mode are not counted against license capacity.
tv	true false	false	Enables the capture of TransactionVision events. See "About Configuration of the .NET Agent for TransactionVision" on page 279 for details on setting transport and other TV options. This mode will send events to TransactionVision. This mode can be combined with other modes and in tv mode agents are not counted against Diagnostics's license capacity

**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, diagnosticsserver, 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"> <li>▶ for percent rate must be 0-100</li> <li>▶ for count rate must be &gt;1</li> <li>▶ for period rate must be one of standard Diagnostics time strings (3m for 3 minutes, 4s for 4 seconds, and so forth)</li> </ul>
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**

```
<xvm><ws><sample method="percent" rate="50"/></ ws ></xvm>
```

Sampling is a random percentage rate.

```
<xvm>< ws ><sample method="count" rate="50"/></ ws ></xvm>
```

Sampling is once every rate count.

```
<xvm>< ws ><sample method="period" rate="60000"/></ ws ></xvm>
```

## <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"> <li>▶ SOAP request capture for SOAP faults</li> <li>▶ SOAP requests and responses capture for TV mode</li> <li>▶ ConsumerID assigned via the SOAP rules.</li> </ul>
maxsize	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.
maxsize	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 SOAP request capture (including payloads) on SOAP Faults. Payloads can contain sensitive information such as credit card numbers so this element is disabled by default.

NOTE: If the <soapcapture> element is disabled it will override the <soaprequestforsoapfault> setting. Please refer to the documentation for the <soapcapture> element.

**Attributes**

Attributes	Valid Values	Default	Description
enabled	true false	false	Enables or disables the SOAP request capture on SOAP fault feature. Disabled by default.
maxsize	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 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>
```

**<sqlparsing> element****Purpose**

This element is used to indicate in what mode SQL queries should be parsed. If there are a large number of SQL queries using literals it can overwhelm the server symbol table so the default is set to mode 3 to avoid this problem.

**Attributes**

Attributes	Valid Values	Default	Description
mode	1, 2, 3, 4	3	<p>Mode indicates how to parse SQL queries.</p> <p>1 - just methods, no SQL queries</p> <p>2 - main categories for SQL queries (select/update/insert/delete/...)</p> <p>3 - (default) a measurement per whole SQL query aggregating similar statements into a single measurement (ignore literals, keyword case...)</p> <p>4 - a measurement per whole SQL query aggregating only identical statements</p>
keywordsfile	string	None	<p>Optionally allows you to specify a file containing keywords you want the agent to find in the SQL statement and highlight in uppercase when stored or displayed by Diagnostics. This helps ensure similar queries are recognized as the same query irrespective of case.</p>

## Elements

Number of Occurrences	1
Parent Elements	probeconfig
Child Elements	

## Example

```
<sqlparsing mode="4" keywordsfile="C:\myfolder\mykeyword.txt"/>
```

**<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	urireplacepattern

**Example**

```
<symbols maxuri="1000" maxuriname="Maximum number of unique URIs exceeded" maxsql="1000" maxsqlname="Maximum number of unique SQLs exceeded"/>
```

**<timeskew> element****Purpose**

Used in configuring HP TransactionVision. Calculates the time difference between the time server and the host on which the .NET Agent is running. The frequency of checking with the time server can be configured.

**Attributes**

Attributes	Valid Values	Default	Description
historysize	number	24	(Read on startup) number of time skew samples to store and compare for best sample.
checkinterval	number	300,000 ms.	(Dynamic) The time in milliseconds to wait before checking the time server for the skew time calculation.
latencythreshold	number	100 ms.	(Dynamic) The maximum time in milliseconds a reply from a time server can take for a valid time skew value.
retrythreshold	number	8	(Dynamic) Number of times to try when request to time server fails.

**Elements**

Number of Occurrences	1 (one)
Parent Elements	tv
Child Elements	none

**Example**

```
<timeskew historysize="24" checkinterval="300000" latencythreshold="100"
retrythreshold="8"/>
```

**<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">
```

## <transport> element

### Purpose

Configure the events channel used by TransactionVision.

### Attributes

Attributes	Valid Values	Default	Description
type	mqseries sonicmq	sonicmq	The event transport provider being used by the Agent.
connectionString	See below.		The connection information for the event transport provider.

### connectionString Syntax when type=sonicmq

```
broker = <broker>; port = <port>; user = <user>; password = <password>;
configurationQueue = <configurationQueue>
```

Where:	Is:
broker	Host name on which the Sonic broker is running. Typically this will be the Analyzer hostname.
port	The port on which the broker communicates. By default, 21111.
user	User id if required by SonicMQ installation for connection. By default, no username is required.
password	Password if required by SonicMQ installation for connection. This is in the obfuscated form created by using the PassGen utility. By default, no password is required. For more information about <b>PassGen</b> , see "Administration Utilities" in the <i>BSM Application Administration User Guide</i> .
configurationQueue	Name of the queue which has the configuration messages for the .NET TransactionVision Agent.



**connectionString Syntax when type=mqseries**

```
host= <host>; queuemanager=<queuemanager>; port= <port>; channel=,channel>
configurationQueue = <configurationQueue>
```

Where:	Is:
host	Host on which the TransactionVision configuration queue is hosted.
queuemanager	Name of the queuemanager.
port	MQSeries port on which the QueueManager communicates.
channel	MQSeries channel which will be used to communicate.
configurationQueue	Name of the queue which has the configuration messages for the .NET TransactionVision Agent.

**Elements**

Number of Occurrences	1 (one)
Parent Elements	tv
Child Elements	None

**Example**

For SonicMQ:

```
<transport type="sonicmq" connectionString="broker=brokerHost;
port=21111; user=; password=;
configurationqueue=TVISION.CONFIGURATION.QUEUE"/>
```

For MQ Series:

```
<transport type="mqseries" connectionString="host=mqHost;
queuemanager=; port=1414; channel=TRADING.CHL;
configurationqueue=TVISION.CONFIGURATION.QUEUE"/>
```

## <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>
```

**<tv> element****Purpose**

Configure the .NET Agent for use with TransactionVision.

**Attributes**

Attributes	Valid Values	Default	Description
eventthreads	number	3	(Read on startup) The number of threads spawned by the Agent to send events to the Analyzer.
eventthreadsleep	number	100	(Dynamic) The time in milliseconds the event thread sleeps after sending a message(event package).
eventmemorythreshold	number	25,000,000	(Dynamic) The memory consumed by the internal buffer (Q) after which the Agent will try and send the message on the application thread.
configthreadsleep	number	10,000	(Dynamic) The time in milliseconds the event thread sleeps after browsing the configuration queue.

**Elements**

Number of Occurrences	1 (one)
Parent Elements	ProbeConfig
Child Elements	transport, timeskew

### Example

```
<tv eventthreads="3" eventthreadsleep="80"  
eventmemorythreshold="25000000" configthreadsleep="10000" >  
  <timeskew historysize="24" checkinterval="300000" latencythreshold="100"  
    retrythreshold="8"/>  
  <transport type="sonicmq"  
    connectionstring="broker=myhost.mydomain.com;  
    port=21111; user=; password=;  
    configurationqueue=TVISION.CONFIGURATION.QUEUE"/>  
</tv>
```

**<urireplacepattern> element****Purpose**

Used to reduce the number of server requests by replacing many server requests with one simplified server request URI that aggregates them. Uses regular expression pattern matching. See "Configuring URI Truncation and Mapping" on page 647.

**Attributes**

Attributes	Valid Values	Default	Description
enabled	true false	false	Enables uri pattern replacement.
pattern value	s/string/string/	If enabled there are two default patterns defined for you.	The syntax for the pattern value is s/search_pattern/replace_pattern/. If / is used in the pattern then the character # should be used instead of / as the separator. Patterns are applied to all server requests and are applied in the order they are specified in probe_config.xml.

**Elements**

Number of Occurrences	1
Parent Elements	probeconfig, symbols
Child Elements	none

**Example**

```
<symbols maxuri="" maxsql="">
  <urireplacepattern enabled="true">
    <pattern value="s/TestService1/CommonService/">
      <pattern value="s/TestService2/CommonService/">
    </urireplacepattern>
  </symbols>
```

## <vmware> element

### Purpose

Controls the ability to adjust timestamps to be more accurate when running in a VMware environment.

### Attributes

Attributes	Valid Values	Default	Description
attempttime stampadjustments	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.
disableperfcounters	true false	false	Set this option to true if the .NET Agent causes IIS worker process to crash in a VMWare environment. This is a workaround for a Microsoft-VMWare environment problem related to accessing perfmon counters in certain VMWare environments.

**Elements**

Number of Occurrences	1
Parent Elements	probeconfig
Child Elements	none

**Example**

```
<vmware attempttimestampadjustments="false"/>
```

## <webserver> element

### Purpose

Specifies the local Web server properties for communication with the probe.

### Attributes

Attributes	Valid Values	Default	Description
start	number	35000	Starting port for webserver.
end	number	35100	Ending port for webserver.
ipaddress	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><sample method="percent" rate="50"/></ 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>
```

# 15

---

## 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 628
- ▶ Customizing the Instrumentation for ASP.NET Applications on page 628
- ▶ Discovering the Classes and Methods in an Application on page 634
- ▶ Controlling Which HP Software Products the Agent can Work With on page 637
- ▶ Configuring Support for MSMQ Based Communication on page 641
- ▶ Configuring Latency Trimming and Throttling on page 641
- ▶ Configuring Depth Trimming on page 646
- ▶ Configuring URI Truncation and Mapping on page 647
- ▶ Configuring the .NET Agent for Lightweight Memory Diagnostics on page 649
- ▶ Limiting Exception Stack Trace Data on page 652
- ▶ Disabling Logging on page 655
- ▶ Overriding the Default Probe Host Machine Name on page 656
- ▶ Listing the Probes Running on a Host on page 657
- ▶ Authentication and Authorization for .NET Profilers on page 658
- ▶ Configuring Consumer IDs on page 660

- ▶ Configuring SOAP Fault Data on page 665
- ▶ Collecting Additional Probe Metrics or Modifying Probe Metrics on page 666

## 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](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 634 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 11, "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. See "Virtual Directories (AppDomains) Under Different IIS Paths with the Same Names" on page 632 for more examples.

```
<appdomain name="1/ROOT/your_app_name" website="Default Web Site"
enabled="true">
  <points file="DefaultWebsite-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. Note that the IIS path is included in the appdomain tag.

```
<?xml version="1.0" encoding="utf-8"?>

<probeconfig>
  <id probeid="" probegroup="Umatilla"/>

  <credentials username="" password=""/>
  <profiler authenticate=""><authentication username="" password=""/></profiler>

  <diagnosticserver url="http://issaquah:2006"/>
  <mediator host="issaquah" port="2612"/>
  <webservice start="35000" end="35100"/>
  <modes am="true"/>

  <instrumentation><logging level="" threadids="no"/></instrumentation>

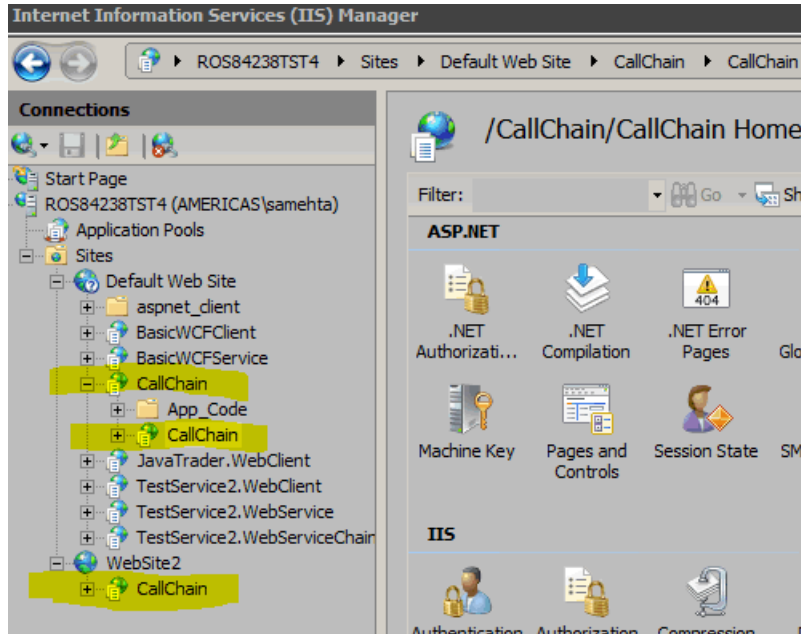
  <lwmd enabled="true" sample="1m" autobaseline="1h" growth="10" size="10"/>

  <process name="ASP.NET", enablealldomains="false">
    <logging level=""/>
    <points file="ASP.NET.points"/>
    <appdomain name="1/ROOT/MSPetShop" website="Default Web Site"
  enabled="true">
      <points file="DefaultWebsite-MSPetShop.points"/>
    </appdomain>
  </process>
</probeconfig>
```

- 4 Restart IIS as instructed in "Discovery and Standard Instrumentation" on page 282.

## Virtual Directories (AppDomains) Under Different IIS Paths with the Same Names

You can distinguish two or more appdomains on the same IIS server which have the same name. Consider the configuration below where there are 3 virtual directories (AppDomains) with the name CallChain.



In the `probe_config.xml` file you can distinguish the AppDomains by including the IIS configuration path.



The configuration for the 3 CallChain applications in the example above would be as follows:

```
<appdomain enabled="false" name="1/ROOT/CallChain/CallChain" website="Default
Web Site">
  <points file="Default Web Site-CallChain-CallChain.points" />
</appdomain>
<appdomain enabled="false" name="1/ROOT/CallChain" website="Default Web Site">
  <points file="Default Web Site-CallChain.points" />
</appdomain>
<appdomain enabled="false" name="2/ROOT/CallChain" website="WebSite2">
  <points file="WebSite2-CallChain.points" />
</appdomain>
```

The resultant probes are distinguished using the IIS path and are displayed in the Enterprise UI as: 1ROOTCallChain.NET, 1ROOTCallChainCallChain.NET, 2ROOTCallChain.NET

## Backward Compatibility with Pre-9.01 Releases

For the sake of backward compatibility, the 9.01 or later version of the agent will be able to read and process versions of the probe configuration earlier than 9.01 for ASP.NET AppDomains. The 'earlier' format is shown in the example below:

```
<appdomain name="CallChain">
  <points file="CallChain.points" />
</appdomain>
```

If you use the earlier format, then the behavior of the agent will revert to the previous version's behavior.

- All AppDomains with name "CallChain" (in this example) will be enabled or disabled simultaneously.
- All CallChain probe instances will be consolidated on the server into one probe.
- Trend lines for probes and server requests should continue from previous versions.

It is recommended that you do NOT use the earlier format of configuration where backward compatibility (such as trend lines) is not required.

For an appdomain configured using the earlier format, if the new behavior is desired, the "old" format entry should be deleted from the probe\_config.xml file. Then run **Rescan ASP.NET Applications** from the start menu on the probe system. This will result in the addition of AppDomain entries with the new format, allowing you to distinguish different probes on the same IIS server with the same name.

The upgrade install will retain the earlier version of the appdomain configuration and modify probe\_config.xml to add the new format configuration for any unlisted AppDomains.

## 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 628.

## Controlling Which HP Software Products the Agent can Work With

The .NET Agent can be set in different modes for the following:

- ▶ Monitoring applications from development through pre-production testing and into production.
- ▶ Use with other HP Software products.
- ▶ Use as a standalone Diagnostics Java Profiler not reporting to a server or to other HP Software products.

The mode the .NET Agent works in is determined by the **<modes>** element set in the **<probe\_install\_dir>/etc/probe\_config.xml** file.

The **<modes>** element is also used in determining usage against the license capacity (see "License Information Based on Currently Connected Probes" on page 85). For Diagnostics there are two types of LTUs (License to use):

- ▶ AM - When using of the product in an enterprise mode, typically in a production environment.
- ▶ AD - When using the product in a pre-production load testing environment with probes in LoadRunner or Performance Center runs.

The value of the **<modes>** element is initially set at the time you install the .NET agent. See Chapter 8, "Installing .NET Agents."

To change the value of the <modes> element you can edit the probe\_config.xml file. Or you can re-run the .NET Agent installer and use the Change option to set the mode to Diagnostics Profiler Mode (PRO), Application Management/Enterprise Mode for Diagnostics (Enterprise) and/or TransactionVision (TV) or Diagnostics Mode for LoadRunner/Performance Center (AD).

---

**Note:** To use the standalone Diagnostics Profiler for .NET in enterprise mode or integrated with other HP Software products, contact HP Software Customer Support to purchase HP Diagnostics.

---

To see Diagnostics data in the user interface of the interfacing HP Software products, you must perform additional configuration steps. See the sections in "Setting Up Integration with Other HP Software Products" on page 735 for details on integration with Business Service Management, LoadRunner or Performance Center.

The sections that follow provide instructions for configuring each product mode of the <modes> element (see also "<modes> element" on page 592).

### **PRO Mode - Diagnostics Profiler for .NET**

When PRO mode is set, the agent gathers performance metrics and presents them in the standalone 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.

PRO mode is not used in determining usage against license capacity.

### **Enterprise Mode**

When configured in Enterprise mode, the agent works with HP Software products such as Business Service Management, LoadRunner, Performance Center, and as the full Diagnostics enterprise product. It will capture data for LoadRunner/Performance Center runs in a separate database as well as capture data outside of LoadRunner/Performance Center runs.

Both AD and AM modes will override this mode.

In Enterprise mode data will also be sent to the Diagnostics .NET Profiler. 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 data until the profiler UI is started.

Enterprise mode is the default for .NET Agents (if you don't specify AD or AM mode). In Enterprise mode the agents are counted against the AM license capacity.

### **AM Mode**

In AM mode the .NET agent will capture all instrumentation data. You can set 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.

Agents in AM mode will always be counted against the AM license capacity.

AM mode supersedes all other modes except for AD.

### **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 24, "Setting Up HP LoadRunner and HP Diagnostics Integration" for how to set up LoadRunner integration or see Chapter 25, "Setting Up Performance Center to Use Diagnostics" for how to setup Performance Center integration.

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.

Another advantage of running a probe in AD mode is that probes in AD mode will only be counted against AD license capacity when the probe is running in a LoadRunner or Performance Center test run. For example if you have 20 agents installed in LoadRunner/Performance Center AD mode but only 5 are in a run, then only 5 are counted against AD license capacity.

### **TV mode**

This mode will send events to Transaction Vision. This mode can be combined with other modes. TV mode is not used in determining usage against HP Diagnostics license capacity.

### **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 14, "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 14, "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 14, "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 14, "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 14, "Understanding the .NET Agent Configuration File".

## Configuring URI Truncation and Mapping

Any HTTP/S server request URI can be transformed before being reported by the probe. This transformation is based on regular expression matching and replacement controlled by the **urireplacepattern** element in the **probe\_config.xml** configuration file. It is turned off by default.

This can be useful when you are seeing too many server requests and you want to replace many server request URIs with one simplified server request URI that aggregates them.

---

**Important:** Overuse of this feature will impact performance.

---

An example is shown below:

```
<symbols maxuri="" maxsql="">
  <urireplacepattern enabled="true">
    <pattern value="s/TestService1/CommonService/" />

    <pattern value="s/TestService2/CommonService/" />
  </urireplacepattern>
</symbols>
```

The syntax used for the pattern value is s/search\_pattern/replace\_pattern/.

The search\_pattern and replace\_pattern should be enclosed in /. If / is used in the pattern then the character # should be used instead of / as the separator.

The patterns are applied to all server requests and are applied to the uri in the order they are specified in the probe\_config.xml file.

If urireplacepattern is enabled, then two default patterns are configured by default.

The first of these default patterns is used to trim server requests that contain a ; or !. All content after these tokens is removed from the server request.

The pattern used is : s#(;|/?\ \!).\*\$##"

The second of these default patterns replaces loading of images, pdfs and docs with a fixed token ("/Static Content").

The pattern used is:

```
s#(?<word1>^.*)(/.*\ \.js|css|jpg|gif|png|pdf|html|doc|docx)#${word1}/Static Content#
```

Both of these patterns can be customized.



## Configuring the .NET Agent for Lightweight Memory Diagnostics

The Lightweight 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 586).
- ▶ 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="1/ROOT/your_app_name" website="Default Web Site"
  enabled="true">
    <points file="DefaultWebsite-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="1/ROOT/your_app_name" website="Default Web Site"
  enabled="true">
    <points file="DefaultWebsite-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 586).
- ▶ 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 438 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

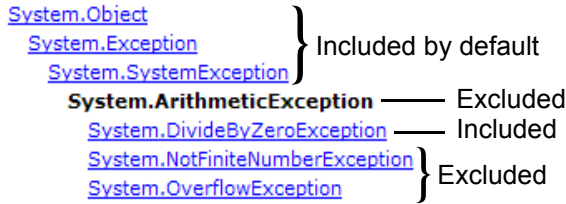
### Limit Specific Exception Types

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

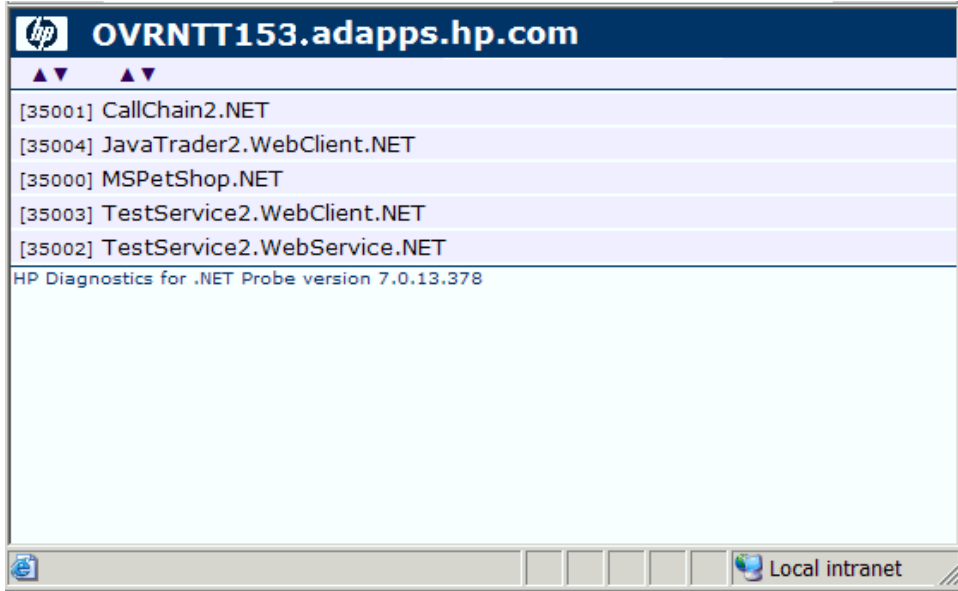
When more than one probe is running on a single host, you cannot know which port each probe is using since the port that is assigned is based on the one that is available at the time the application (and probe) is started. As the applications are started and stopped, the port that is assigned to the 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

You can 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 797.

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 the password required 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 .NET Diagnostics Profiler:**

- 1** Generate a new username and password using the **PassGen.exe** utility located in the `<probe_install_dir>/bin` directory. Enter the user name and password for encryption. The encrypted password generated for the user is FIPS-2 compliant.
- 2** In the `probe_install_dir>/etc/probe_config.xml` file, after the `<profiler authenticate="true">` line, enter the 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 when running the PassGen utility.
- 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 Business Service Management. BSM users can look at the performance of the same application based on consumers to compare their performance characteristics.

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 661.
- 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 926 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 665.

## 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" consumerid="Client3"/>
  </iprules>
</consumeridrules>
```

## Configuring SOAP Fault Data

If a SOAP fault is detected, the SOAP payload can be included with the SOAP fault data. SOAP payload is only captured when there is a SOAP fault.

In the Diagnostics UI, you can view the payload information as part of the SOAP fault instance tree (call profile).

Because payloads can contain sensitive information such as credit card numbers, payload capture on SOAP faults is disabled by default. To enable payload capture on SOAP faults set **<soaprequestforsoapfault enabled="true"/>** in the **probe\_config.xml** file on the .NET probe system.

You can also define the limit for the payload size using the **maxsize** attribute in the **<soaprequestforsoapfault>** element. For example, the following entry increases the SOAP payload length to 10000 from its default of 5000:

```
<soaprequestforsoapfault enabled="true" maxsize="10000"/>
```

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 590 and "<metric> element" on page 590 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



# 16

---

## 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 672
- ▶ Enabling HTTP Proxy Communications for the Java Agent on page 673
- ▶ Enabling HTTP Proxy Communications for a .NET Agent on page 674

## Enabling HTTP Proxy Communications for the Diagnostics Servers

The following section describes how to configure the Diagnostics commander server and Diagnostics mediator server 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 commander 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 70.



## Enabling HTTP Proxy Communications for the Java Agent

The following section describes how to configure the Java Agent to communicate with Diagnostics commander server through an HTTP proxy. You can also have proxy communications configured for you by selecting the Use Proxy Server checkbox in the Java Agent installation and setup program.

**To configure the Java Agent for HTTP proxy communications:**

- 1** Set the following properties in `<probe_install_dir>/etc/dispatcher.properties`:
  - Set **proxy.enabled** to true to enable proxy communications for the Java Agent and then enter the following options.
  - 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 commander 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/metricis.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.

# 17

---

## 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 676
- ▶ Collating Offline Analysis Files over a Firewall on page 679
- ▶ Installing and Configuring the MI Listener on page 680
- ▶ Configuring the Diagnostics mediator server to Work with a Firewall on page 681
- ▶ Configuring LoadRunner and Performance Center to Work with Diagnostics Firewalls on page 687

---

**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.

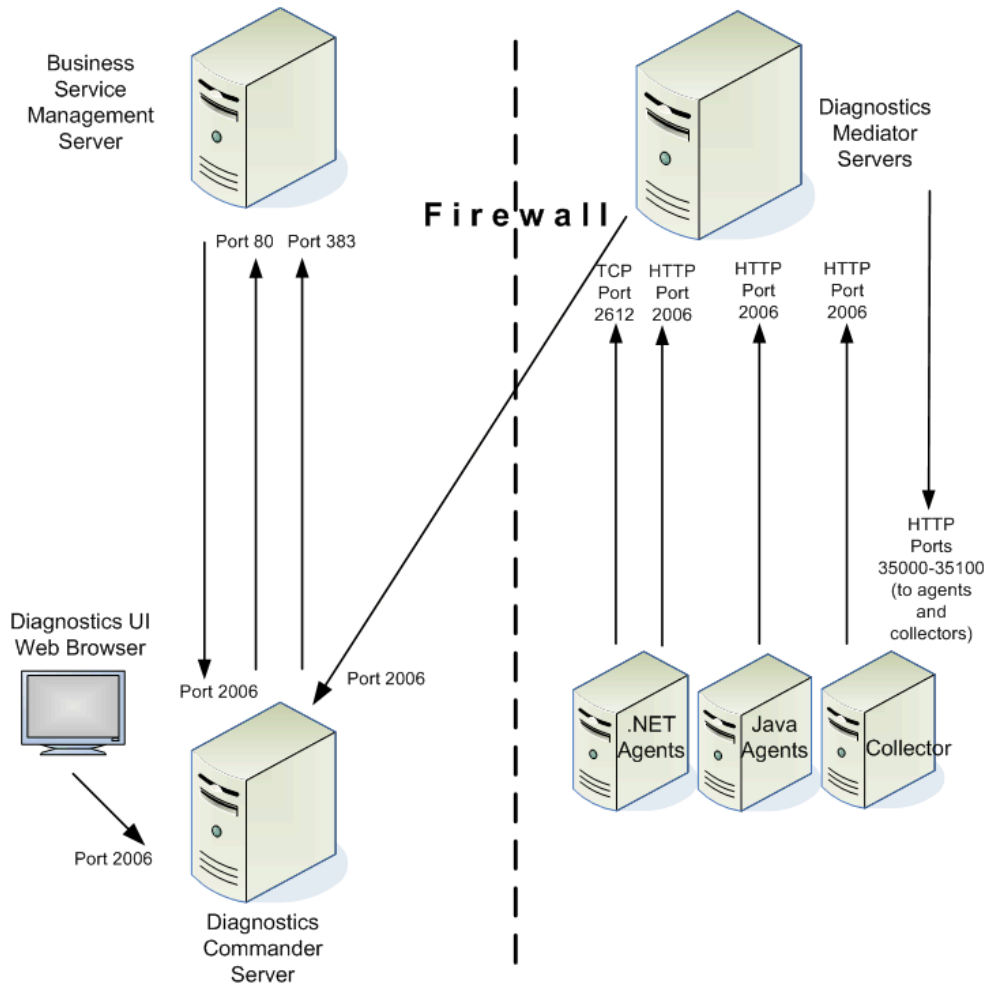
---

## 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.

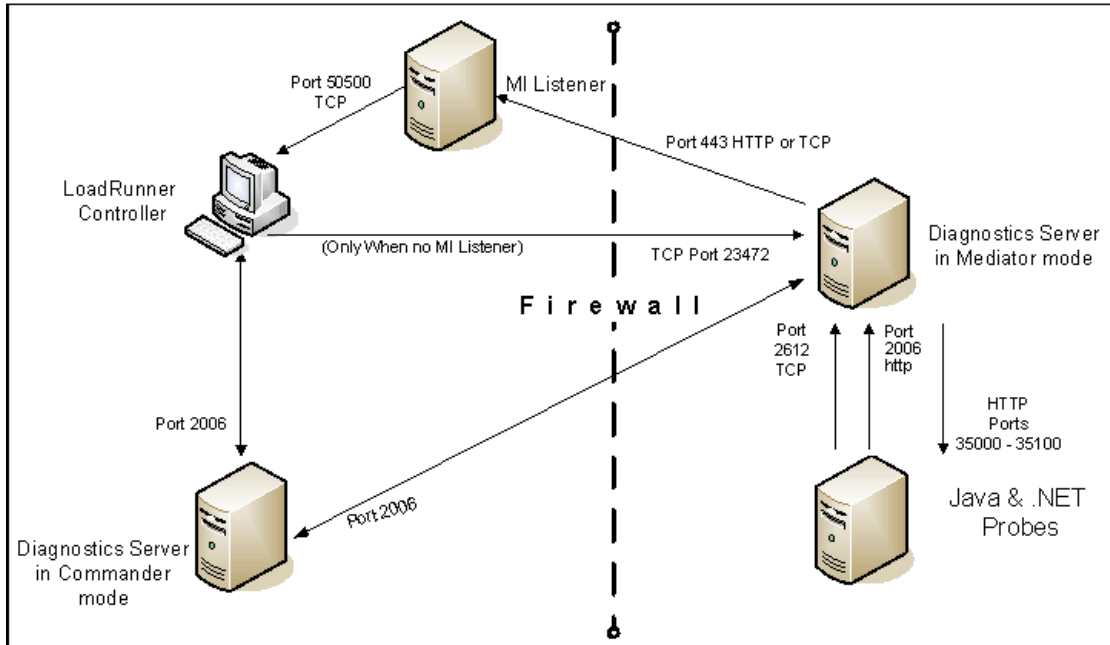


To configure the firewall to enable the communications between Diagnostics components and Business Service Management, open the ports that will allow the following:

- ▶ HTTP requests from the Business Service Management server(s) to the Diagnostics Commander Server, on port 2006.
- ▶ HTTP requests from the Diagnostics Commander Server to Business Service Management Server on port 80. HI updates from the Diagnostics Commander Server to Business Service Management on port 383.
- ▶ HTTP requests from the Diagnostics UI web browser client machine to the Diagnostics Commander Server on port 2006. Note that you can also access the Java or .NET profilers from the Diagnostics UI.
- ▶ HTTP requests from the Diagnostics Mediator Servers to the Diagnostics Commander Server on port 2006.
- ▶ HTTP requests from the Diagnostics Mediator Servers to ports 35000-35100 of the probe. 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 “Configure Monitoring of Multiple Java Processes on an Application Server” on page 233
- ▶ TCP requests from the .NET Agent to the Diagnostics Mediator Server on port 2612.
- ▶ HTTP requests from the Java Agent, Collector, .NET Metrics Collector to the Diagnostics Mediator Server on port 2006.

## 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 mediator server to the Diagnostics commander server 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.

---

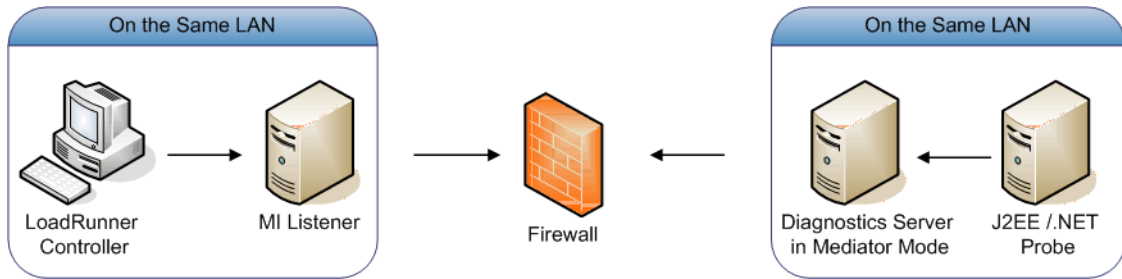
**Note:** In addition to the above topology, if you are using the LoadRunner Analysis Tool to view offline J2EE results, see “Collating Offline Analysis Files over a Firewall” to properly configure the Controller and the Diagnostics Servers in Mediator mode for offline file retrieval.

---

## **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*.



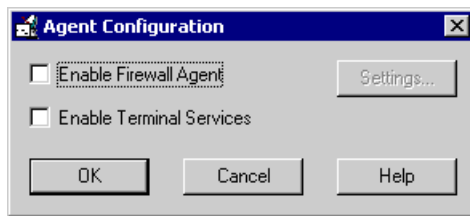
## Configuring the Diagnostics mediator server to Work with a Firewall

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:**

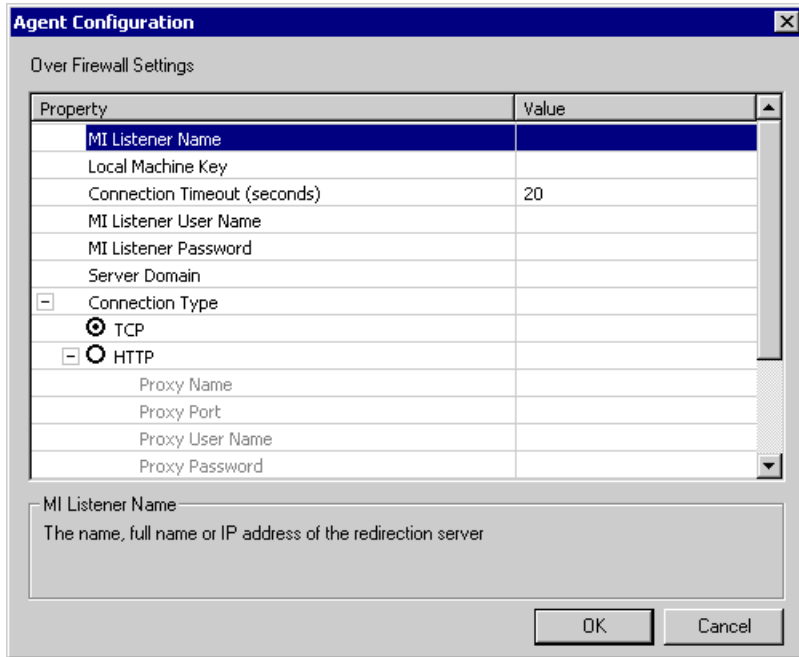
- 1 Launch the Agent Configuration by running `<diagnostics_server_install_dir>/nanny/windows/bin/AgentsConfig.exe`.

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.



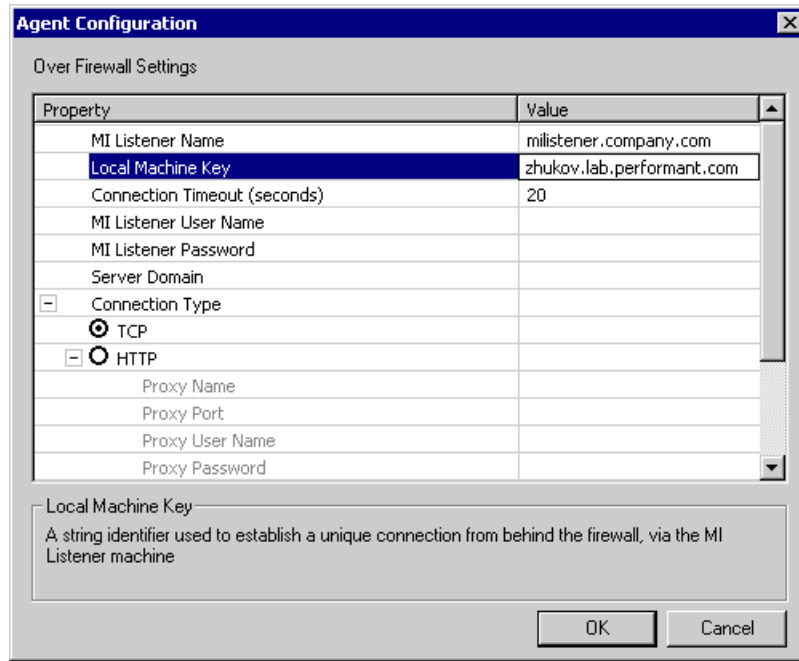
- 5 For the **Local Machine Key** property, enter the machine name of the host of the Diagnostics mediator server.

---

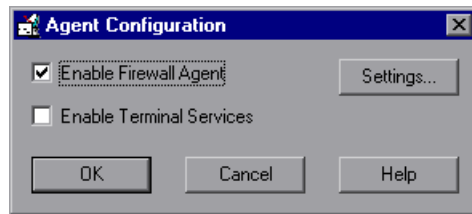
**Important:**

- ▶ 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 view in the Diagnostics UI to determine the machine name for the host of the Diagnostics mediator server. For more information on the System views, see Appendix D, “Using System Views for Administrators.”
-

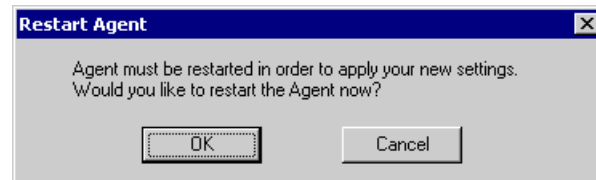
6 Click **OK** to close the dialog box.



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.

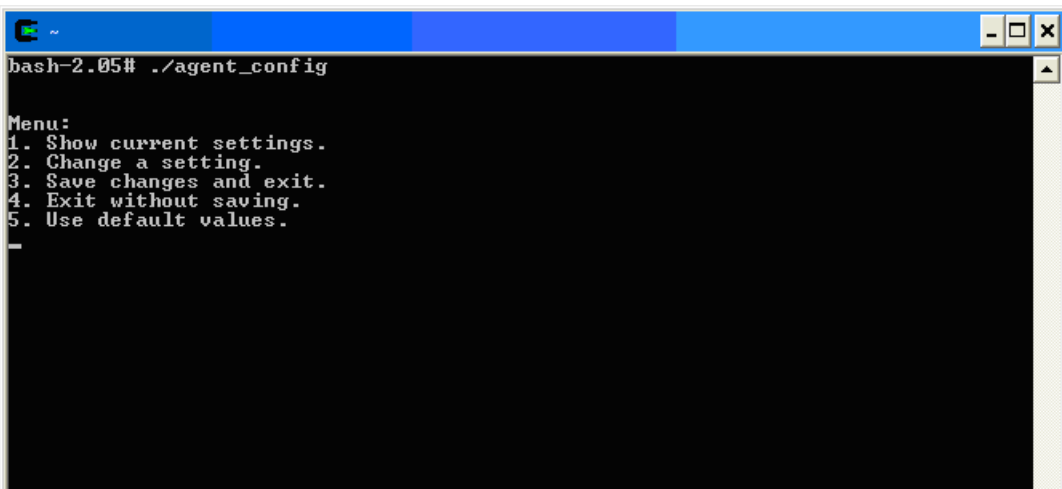
For Solaris and Linux, where `<platform>` is either `solaris` or `linux`:

```
export LD_LIBRARY_PATH=.
export M_LROOT=<diagnostics_server_install_dir>/nanny/<platform>
cd $M_LROOT/bin
./agent_config
```

For HP-UX:

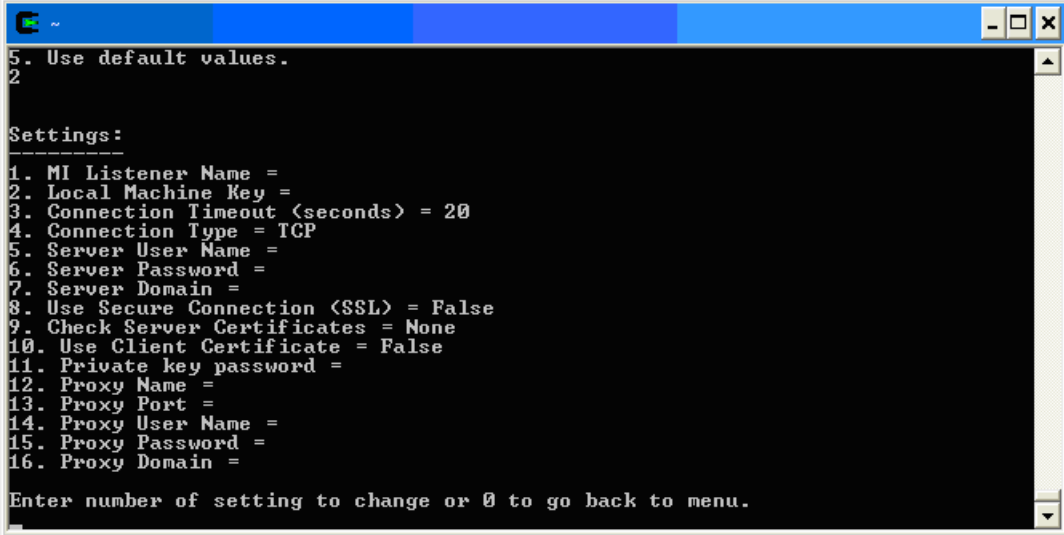
```
export SHLIB_PATH=.
export M_LROOT=<diagnostics_server_install_dir>/nanny/hpux
cd $M_LROOT/bin
./agent_config
```

- 3 In the Agent Configuration Utility window, press **2**, **Change a Setting**.



**4** A list of settings appears.

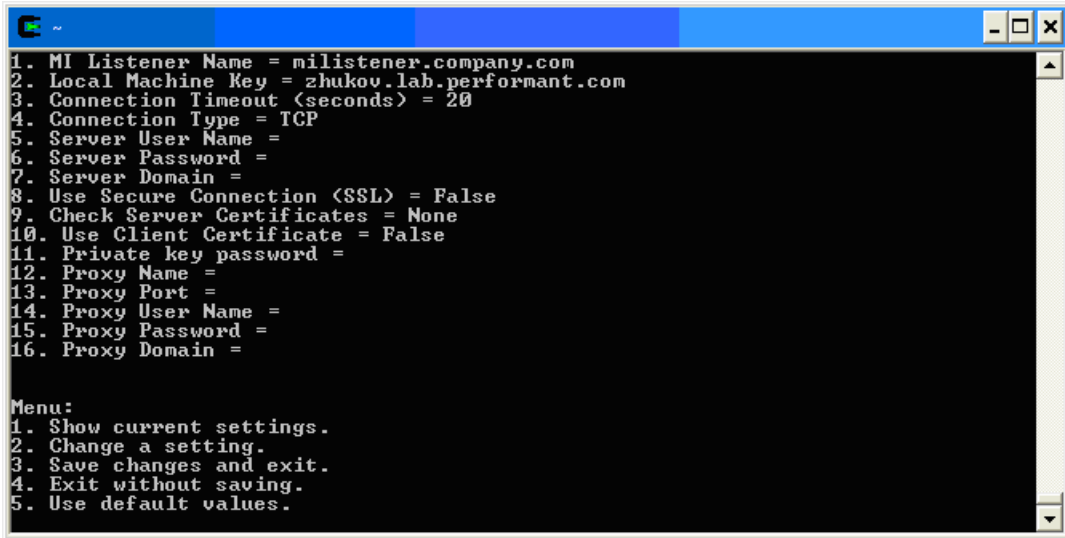
Press **1** to select **MI Listener Name**, and enter the machine name or IP address of the MI Listener host.

A screenshot of a terminal window with a blue title bar. The terminal displays a menu of settings. At the top, it says "5. Use default values." followed by "2". Below that is a section titled "Settings:" with a dashed underline. The settings are listed as follows: 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 =. At the bottom, it says "Enter number of setting to change or 0 to go back to menu." The terminal has a scrollbar on the right side.

```
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.

Use the System Health view in the Diagnostics UI to determine the machine name for the host of the Diagnostics mediator server. For more information on the System views, see Appendix D, “Using System Views for Administrators.”



```

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 some Linux systems, if you encounter an error saying that the `libstdc++.so.5` shared library is missing, you may need to install it. For example, on CentOS, enter the following command to install the library:

**`yum install compat-libstdc++-33`**

## 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



# 18

---

## **.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 691
- System Metrics Captured by Default on page 692
- Configuring .NET System Metrics Capture on page 693
- Adding System Metrics Using the Windows Performance Monitor on page 696
- Default Entries in the .NET Agent metrics.config File on page 698
- Keywords in the metrics.config File on page 699

### **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 666.

---

## 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

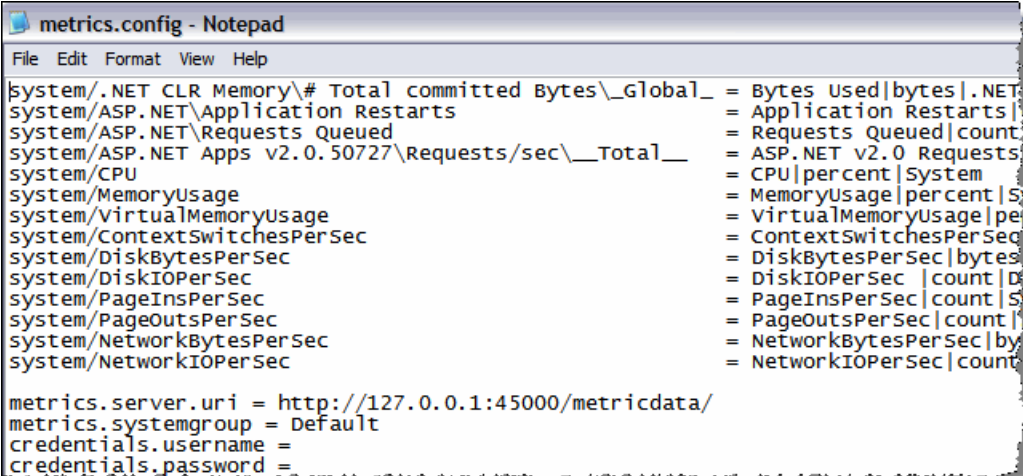
In addition to the default system metrics listed above, the following system metrics are also captured by default on .NET Agent systems. (The layout of these entries is described in “Understanding the system/ Metrics Collector Entries” on page 694).

```
.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|count|S
system/ASP.NET\Requests Queued = Requests Queued|count|S
system/ASP.NET Apps v2.0.50727\Requests/sec\__Total__ = ASP.NET v2.0 Requests/sec|count|S
system/CPU = CPU|percent|System
system/MemoryUsage = MemoryUsage|percent|S
system/VirtualMemoryUsage = VirtualMemoryUsage|percent|S
system/ContextSwitchesPerSec = ContextSwitchesPerSec|count|S
system/DiskBytesPerSec = DiskBytesPerSec|bytes|S
system/DiskIOPerSec = DiskIOPerSec|count|S
system/PageInsPerSec = PageInsPerSec|count|S
system/PageOutsPerSec = PageOutsPerSec|count|S
system/NetworkBytesPerSec = NetworkBytesPerSec|bytes|S
system/NetworkIOPerSec = NetworkIOPerSec|count|S

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 19, “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 696 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
```

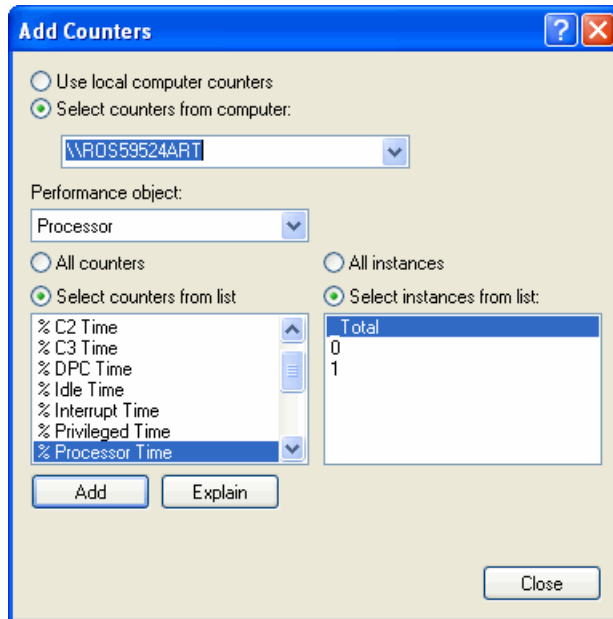
## Adding System Metrics Using the Windows Performance Monitor

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 694.

**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 694.

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 693.

The use of each of the other keywords is described in the following section.

<b>credentials.password</b>	This setting must match the setting for the password attribute of the <code>&lt;credentials&gt;</code> element in the <code>probe_config.xml</code> file. See “ <code>&lt;credentials&gt;</code> element” on page 558 for more details.
<b>credentials.username</b>	This setting must match the setting for the username attribute of the <code>&lt;credentials&gt;</code> element in the <code>probe_config.xml</code> file. See “ <code>&lt;credentials&gt;</code> element” on page 558 for more details.
<b>default.sampling.rate</b>	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:  <code>default.sampling.rate = 10s</code>

<p><b>metrics.server.uri</b></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.</p> <p>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://&lt;my_diag_server&gt;: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 &lt;mediator&gt; 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><b>metrics.systemgroup</b></p>	<p>This setting is automatically generated at install time. Do not change this setting.</p>
<p><b>metrics.agent.publish.interval</b></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><b>metrics.agent.registered_host name</b></p>	<p>Refer to the “Overriding the Default Probe Host Machine Name” on page 656 for a description of when and how to use this setting.</p>
<p><b>proxy.password</b></p>	<p>This setting must match the setting for the proxypassword attribute of the &lt;diagnosticsserver&gt; element in the probe_config.xml file. See “&lt;diagnosticsserver&gt; element” on page 562 for more details. Also refer to Chapter 16, “Configuring Diagnostics Servers and Agents for HTTP Proxy.”</p>

<b>proxy.user</b>	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 562 for more details. Also refer to Chapter 16, “Configuring Diagnostics Servers and Agents for HTTP Proxy.”
<b>proxy.uri</b>	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 562 for more details. Also refer to Chapter 16, “Configuring Diagnostics Servers and Agents for HTTP Proxy.”



# 19

---

## 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 703
- What Metrics are Being Collected by the Java Agent on page 705
- Understanding Metric Collector Entries on page 706
- About Collecting Additional Probe Metrics on page 708
- Modifying Probe Metrics Already Being Captured on page 708
- Stopping Capture of a Metric on page 708
- Using Customized metrics.config Files for Multiple JVM Applications on a System on page 709

### **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 691).

---

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.

---

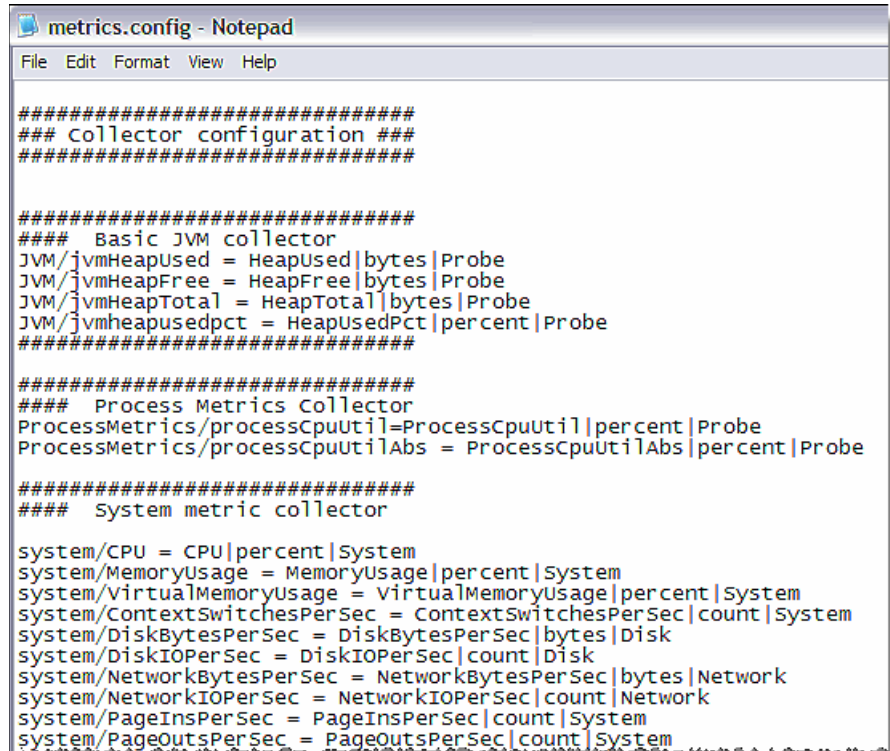
**Note:** If you update the metrics configuration file, the metric collectors automatically restarts so that your changes can take effect.

---



## 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.



```

#####
### 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 “Getting a List of Available JMX or WebSphere PMI Metrics” on page 725 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**.

The collector-name along with metric names can also be found on the Advanced Query page in the Diagnostics UI ([http://<diagnostics\\_sever>:2006/query](http://<diagnostics_sever>:2006/query)).

- ▶ **<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 715. For information on formatting the **metric\_config** property for JMX metrics, see “Creating New JMX or WebSphere PMI Metrics Entries” on page 728.
- ▶ **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 706.

See “Capturing Additional Custom System Metrics” on page 715 for details on capturing additional system metrics.

See “Additional Custom JMX Metrics” on page 725 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.

## Using Customized `metrics.config` Files for Multiple JVM Applications on a System

There may be times when you only need to collect certain metrics, or customize the metric collector properties for select JVM applications running on a system with multiple JVMs, and such changes would negatively impact the other instrumented JVMs running on the system. In these cases, you can create and customize different `metrics.config` configuration files and configure those JVM applications to use the customized settings by following these steps:

---

**Note:** You only need to configure the JVM applications that need customized `metrics.config` files. The other JVM applications can use the out-of-the-box `metrics.config` configuration.

---

- 1** Copy the `etc/metrics.config` file for each JVM application requiring special customization and name the file, such as `metrics_<app_name>.config`. This file must be in the same `<probe_install_dir>/etc` folder as the original `metrics.config` file. Customize this file as needed.
- 2** Create a copy of the `lib/modules.properties` file for each `metrics_<app_name>.config` file created, and name the file, such as `modules_<app_name>.properties`. This file must be in the same `<probe_install_dir>/lib` folder as the original `modules.properties` file.

Change the `metrics.properties` property of this new file to point to the new `metrics_<app_name>.config` file as shown in the following example:

```
#####
## Metrics capture module
#####
metrics.class.name=com.mercury.diagnostics.capture.metrics.MetricsModule
metrics.class.loader=probeLoader
metrics.properties=metrics_<app_name>.config
```

- 3 Update each JVM start script that needs customized metrics collection to use the new corresponding **lib/modules\_<app\_name>.properties** file by adding the following to the JVM property definition:

```
-Dmodules.properties.file=module_<app_name>.properties
```

# 20

---

## 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 711
- ▶ System Metrics Captured by Default on page 712
- ▶ Configuring the System Metrics Collector on page 713
- ▶ Capturing Additional Custom System Metrics on page 715
- ▶ Enabling z/OS System Metrics Capture on page 721

### **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 713 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 715.

For information on z/OS system metrics see “Enabling z/OS System Metrics Capture” on page 721.



## 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 19, “Java Agent Metrics Collectors” for general information on the metrics collector and see “Understanding Metric Collector Entries” on page 706 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 691).

---

---

**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:

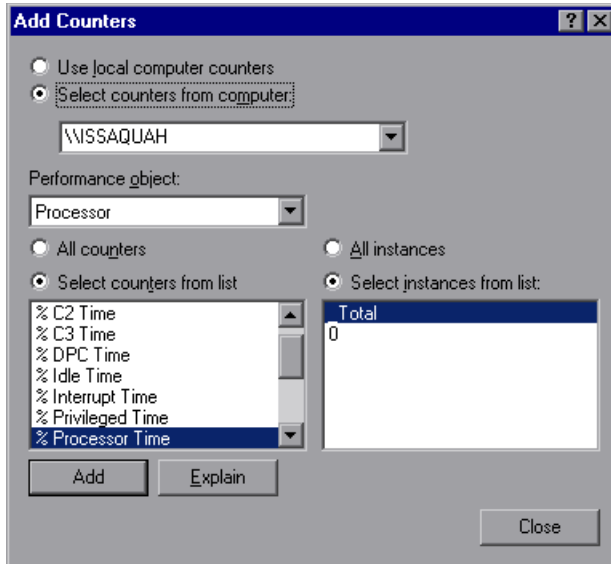
- a** Select **Start > Run** from the Start menu.
- b** In the **Open** box on the **Run** dialog box type `perfmon`.

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 706.

**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 706.

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 706. 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.

# 21

---

## 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 723
- About Configuring JMX Metric Collectors on page 724
- Additional Custom JMX Metrics on page 725
- Getting a List of Available JMX or WebSphere PMI Metrics on page 725
- Creating New JMX or WebSphere PMI Metrics Entries on page 728

### **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
- Apache Tomcat
- JBoss J2EE Server
- TIBCO Business Works

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 Infrastructure (PMI) service on the WebSphere server to start receiving JMX metrics.

See “Configuring WebSphere for JMX Metric Collection” on page 211 for information on how to configure WebSphere 5.x, 6.x and 7.0 servers for JMX metrics collection.

### **Configuring TIBCO for JMX Metric Collection**

For TIBCO JMX metric collection you need to enable JMX metric collection see “To configure TIBCO BusinessWorks for JMX Metrics Collection:” on page 184 for instructions.

## **About 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 19, “Java Agent Metrics Collectors” for general information on the metrics collector and see “Understanding Metric Collector Entries” on page 706 for an explanation of the metrics collector entries and syntax.

## 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 723. You configure these collectors by defining entries in the `metrics.config` file, see “Understanding Metric Collector Entries” on page 706. 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.

In order to create new entries in the JMX metric collectors you can get a list of the available JMX metrics and WebSphere Performance Monitoring Infrastructure (PMI) metrics. Then you can create new metrics entries in the `metrics.config` file. The following sections provide instructions for creating new entries in the JMX metric collectors so that additional JMX metrics and PMI metrics can be monitored.

## Getting a List of Available JMX or WebSphere 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 or WebSphere PMI 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.

Some examples of the metrics list file are shown below. You can use this type of information to configure additional JMX or PMI metrics in the probes' **etc/metrics.config** file.

The following example shows the available MBean ObjectNames and their collectable attributes:

```
===== 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=JDBCDataSource,mbeanIdentifier=cells/yli87Node01Cell/nodes/yli87Node01/servers/server1/resources.xml#DataSource_1244231364323,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
.....
```

The following example shows the available MBean ObjectNames and their collectable attributes and fields:

```
===== MBean ObjectNames and Available Attributes and Fields =====
MBean ObjectName:
java.lang:name=PS Old Gen,type=MemoryPool
Available Metrics:
Attribute: CollectionUsage type: javax.management.openmbean.CompositeData
Field: committed, type: java.lang.Long
Field: init, type: java.lang.Long
Field: max, type: java.lang.Long
Field: used, type: java.lang.Long
```

The following example shows the available MBean ObjectNames and their collectable operations and fields:

```

===== MBean ObjectNames and Available Operations and Fields =====
MBean ObjectName:
com.tibco.bw:key=engine,name="MortgageBroker-BrokerService"
Available Metrics:
Operation: java.lang.Integer GetActiveProcessCount()
Operation: javax.management.openmbean.CompositeData GetExecInfo()
    Field: Threads, type: java.lang.Integer
    Field: Uptime, type: java.lang.Long

Operation: javax.management.openmbean.CompositeData GetMemoryUsage()
    Field: FreeBytes, type: java.lang.Long
    Field: PercentUsed, type: java.lang.Long
    Field: TotalBytes, type: java.lang.Long
    Field: UsedBytes, type: java.lang.Long

```

For WebSphere JMX collectors, besides the 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 configuration information as shown in the example below:

```

===== 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

```

## Creating New JMX or WebSphere PMI Metrics Entries

The following instructions guide you through the process of creating the JMX or PMI 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 706.

**To capture JMX or WebSphere PMI 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. See syntax below.

MBean object and attributes:

```
<MBean object name pattern>.<attribute name>
```

MBean Object, attribute and fields:

```
<MBean object name pattern>.<attribute name>#<field name>
```

MBean object and operations:

```
<MBean object name pattern>.<operationname>()
```

MBean object, operations and fields:

```
<MBean object name pattern>.<operationname>()#<field name>
```



- **<MBean object name pattern>** is the string representation of the object name of an MBean. For an explanation of metric patterns see “Understanding Metric Patterns” on page 731. For an explanation of how to group JMX metrics see “JMX GROUPBY and EXPAND\_PMI Modifiers” on page 732.
- **<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
```

See “Getting a List of Available JMX or WebSphere PMI Metrics” on page 725 for an example of a metrics dump showing available attributes.

- **<attribute name>#<field name>** JMX Attributes that return Composite Data can have their numeric fields used as metrics. Simply append the symbol # followed by the name of the field after the MBean name.

For example:

```
Java\ Platform\java.lang:type=MemoryPool,name=\Perm\
Gen.Usage#used
```

will track the <used> field of the <Perm Gen> MBean's <Usage> composite data attribute.

- **(<operationname>())** where the operation name is followed by open and close parentheses. And the entire operation name is enclosed in parentheses. If the operation returns a composite attribute, suffix the composite attribute field after the () as for attributes.

For example:

```
Tibco/com.tibco.bw:key=engine,name=*. (GetActiveProcessCount()) =
Active Process Count|count|Tibco
```

Note that only operations that don't take arguments are supported.

- ▶ **<operation name>()#<field name>** JMX Operations that return Composite Data can have their numeric fields used as metrics. Simply append the symbol # followed by the name of the field after the MBean name.

For example:

```
Tibco/com.tibco.bw:key\=engine,name\=*.getStatus()#Total\ Errors) =
Total Errors|count|Tibco
```

will track the "Total Errors" field of the Composite data object returned by the getStatus() operation.

- 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 "Getting a List of Available JMX or WebSphere PMI Metrics" on page 725 for an example of the PMI module and PMI module branches and their available statistics names.

See "JMX GROUPBY and EXPAND\_PMI Modifiers" on page 732 for an example of how to group PMI metrics.

- 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
```

## JMX GROUPBY and EXPAND\_PMI Modifiers

You can 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 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.



# 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 HP LoadRunner and HP Diagnostics Integration
- ▶ Setting Up Performance Center to Use Diagnostics





# 22

---

## 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 documentation is relevant for Diagnostics' integration with Business Service Management 9.x unless otherwise stated. For the most recent information on supported integrations, see the Diagnostics Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](http://support.openview.hp.com/sc/support_matrices.jsp).

---

**This chapter includes:**

- ▶ About Setting Up the Integration Between Business Service Management and Diagnostics on page 739
- ▶ Registering the Diagnostics Server in Business Service Management on page 740
- ▶ Removing the Diagnostics Registration on page 747
- ▶ Understanding the Diagnostics Admin Page on page 747
- ▶ Assigning Permissions for Diagnostics Users in Business Service Management on page 748
- ▶ Password for Data Collectors to Access RTSM on page 750
- ▶ Accessing the Diagnostics Pages in Windows 2003 on page 751

- ▶ Accessing the Diagnostics Application from Business Service Management on page 751
- ▶ Data Samples Sent to Business Service Management on page 752
- ▶ Diagnostics Populates CIs and Models in Business Service Management on page 753
- ▶ Synchronize CIs Between Diagnostics and Business Service Management on page 753
- ▶ Diagnostics Provides KPI/HI Coloring to Business Service Management on page 754
- ▶ Enabling Diagnostics Integration with BSM's Service Health Analyzer on page 755
- ▶ Integration with BSM's Performance Graphing on page 756
- ▶ Diagnostics and OM Server Co-existence on page 756
- ▶ Configuration of Separate BSM Servers for DPS and Gateway on page 761
- ▶ Additional Information on Integration on page 763

## 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” on page 740.

### **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 748.

### **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 751.

### **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 Server as a trusted site in the browser configuration.

## Registering the Diagnostics Server in Business Service Management

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. Differences for BAC 8.x are noted.

---

**Important:** After a Business Service Management upgrade, you must re-register the Business Service Management-Diagnostics integration.

---

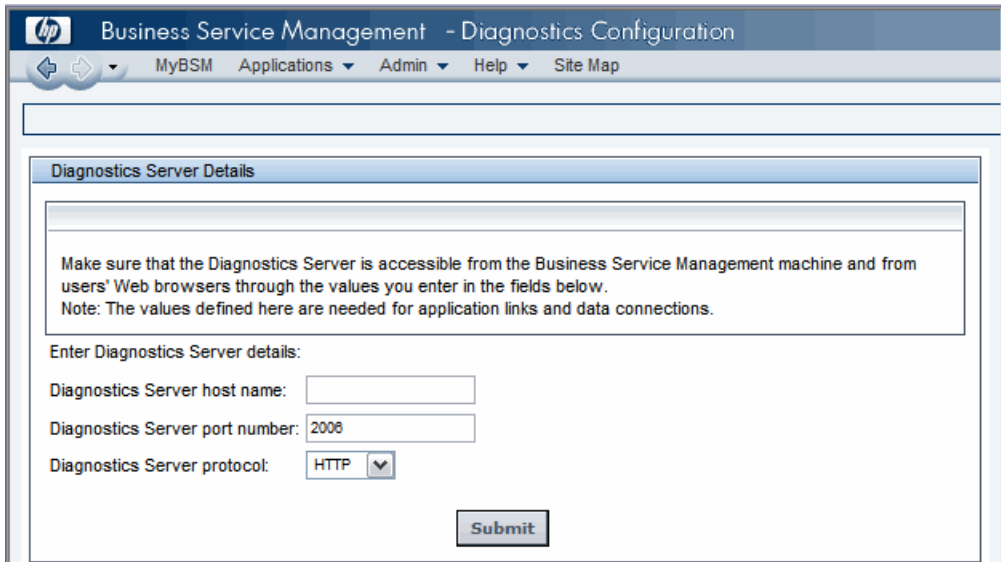
**Note:** If you are using Windows 2003, you must configure your Internet browser settings to access the Diagnostics Admin page. For more details, see “Accessing the Diagnostics Pages in Windows 2003” on page 751.

---

If Diagnostics is integrated with Business Service Management then after an upgrade of the Diagnostics Commander Server, you must copy over the **RegistrarPersistence.xml** file from the **etc.old** folder to the new **etc** folder. Then check the Diagnostics Integration in the **BSM > Admin > Diagnostics** page and re-do the registration of Diagnostics server in BSM if it is not working properly.

**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**.
- ▶ **Diagnostics root context.** If BSM is configured to use a custom context root and you have configured Diagnostics commander server to use a custom context root, enter the Diagnostics commander context root. See “Configuring a Custom Context Root” on page 497.

---

**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.”

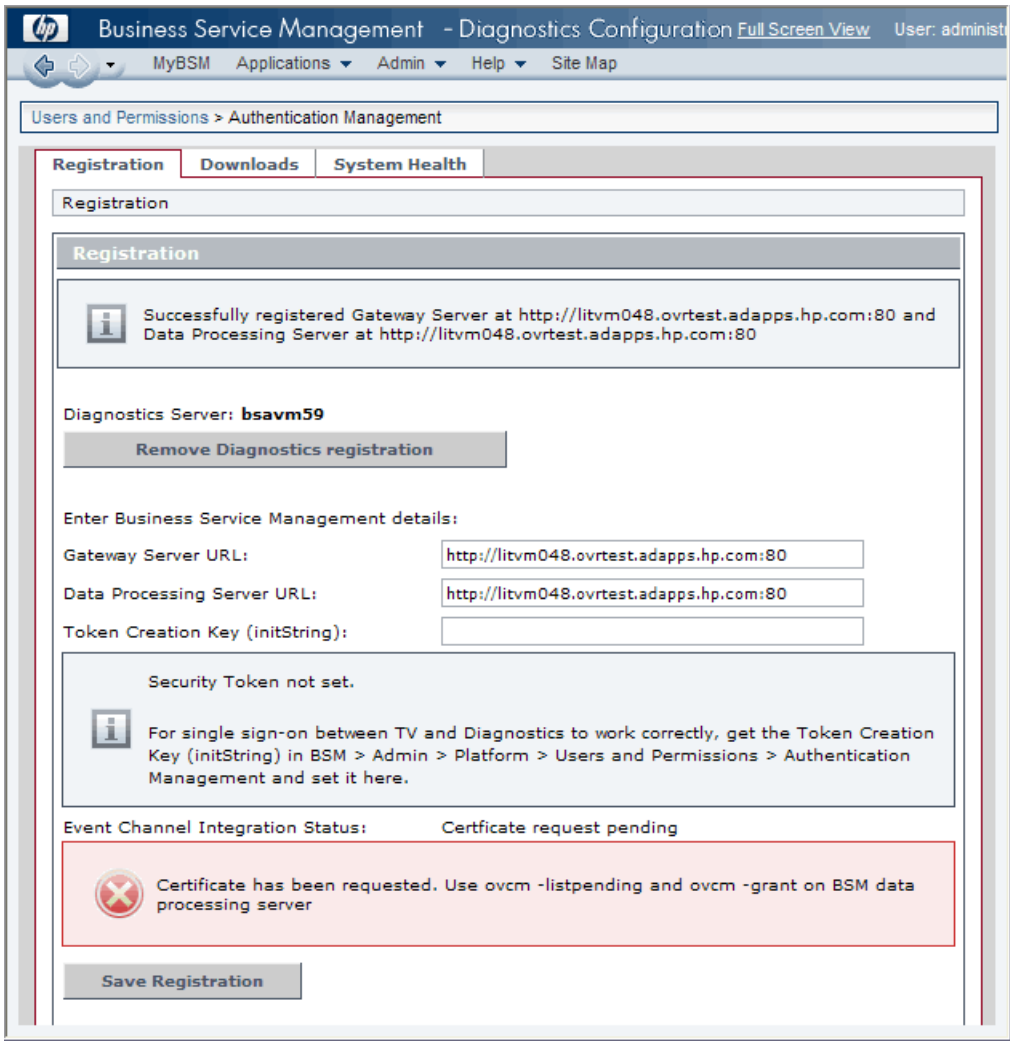
---

- 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.



## Business Service Management Details

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.

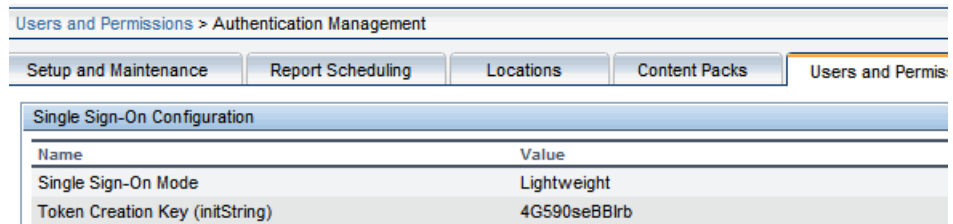
---

**Note:** In the cases where the BSM Processing Server cannot be accessed from the Diagnostics Server and the BSM Gateway Server has been configured to tunnel certificate requests to the Processing Server (such as when the Gateway and Processing Server are on the other side of a Load Balancer or SSL Accelerator), then use the BSM Gateway as both the BSM Gateway Server and Processing Server in this Diagnostics Server registration page. You may still need to manually grant certificates on the Processing Server as described on the following page.

---

**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.



Users and Permissions > Authentication Management	
Single Sign-On Configuration	
Name	Value
Single Sign-On Mode	Lightweight
Token Creation Key (initString)	4G590seBB1rb



**Event Channel Integration Status.** When integrating Diagnostics 9.x with Business Service Management 9.x, the event 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.

The registration executes a script to do the event channel integration (<**Diagnostics\_install\_dir**>/server/bin/switch\_ovo\_agent.sh or .vbs. Root access on UNIX is required to run the script. If the Diagnostics commander server is not run as root on UNIX the registration of Diagnostics in BSM will fail with a permission denied error. After you get this failure you must execute the script manually as root on the Diagnostics commander server. It is important that the BSM registration step is done first prior to running the script manually.

---

**Important:** For communications between BSM gateway server and BSM processing server with an event channel integration there must be a trust relationship between the machines if the servers are on different systems. See “Configuration of Separate BSM Servers for DPS and Gateway” on page 761 if you need to set this up.

---

When the Event Channel Integration Status is red, see if the message displayed indicates a certificate request is pending. The registration executes a script to do the event channel integration (<**Diagnostics\_install\_dir**>/server/bin/switch\_ovo\_agent.vbs or .sh). But you must do some manual steps to grant the certificates, so the status indicates the certificate request is pending.

In this case you need to do the following to complete the event channel integration.

**To manually grant the certificate:**

- 1 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 919 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.

---

- 2 Click **Save Registration** and verify the Event Channel Integration Status is OK and the other values are correct.

The screenshot shows the HP Business Service Management - Diagnostics Configuration web interface. The browser address bar shows the URL. The navigation menu includes MyBSM, Applications, Admin, Help, and Site Map. The main content area has three tabs: Registration, Downloads, and System Health. The Registration tab is active, displaying a registration form. The form includes a 'Registration' header, a 'Diagnostics Server: OVRNTT150' label, a 'Remove Diagnostics registration' button, and a section for 'Enter Business Service Management details:'. This section contains three input fields: 'Gateway Server URL' with the value 'http://ovrntt123.ovrtest.adapps.hp.com:80', 'Data Processing Server URL' with the value 'http://ovrntt123.ovrtest.adapps.hp.com:80', and 'Token Creation Key (initString):' with the value '4G590seBBlrb'. Below these fields, the 'Event Channel Integration Status: OK' is displayed, and a 'Save Registration' button is at the bottom.

## Removing the Diagnostics Registration

You can 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” on page 740.

## 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 “Removing the Diagnostics Registration” on page 747.
- 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” on page 740.

### **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.”

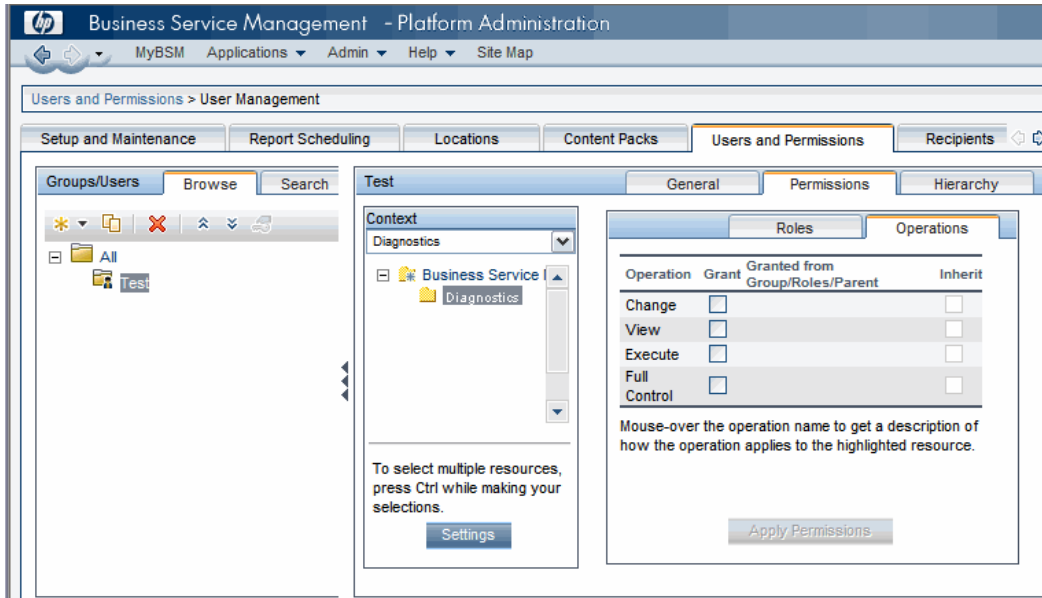
### **System Health Tab**

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.

## **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*.

## Password for Data Collectors to Access RTSM

During the Business Service Management configuration server launch you can choose to override the default password for data collections, such as Diagnostics, to access RTSM. When you override this password in the Business Service Management Setup and Database Configuration Utility's Login Settings page the same password will be used for all data collectors (TV, BPI, RUM and Diagnostics).

If you override the default password in BSM, you must make a corresponding change in the password in Diagnostics to match the new password. You make the change on the Diagnostics server in the **etc/cmdbProperties.xml** file.

```
<customer>
  <!-- customerId is an Integer -->
  <customerId>1</customerId>
  <customerName>Default Client</customerName>
  <userName>diagnostics</userName>
  <!-- userPassword may be obfuscated -->
  <userPassword>integration</userPassword>
</customer>
<customer>
```

## 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 Run-time Service Model (RTSM) for application infrastructure elements and business transactions. See *HP Diagnostics User's Guide* section on Integrations for more information.

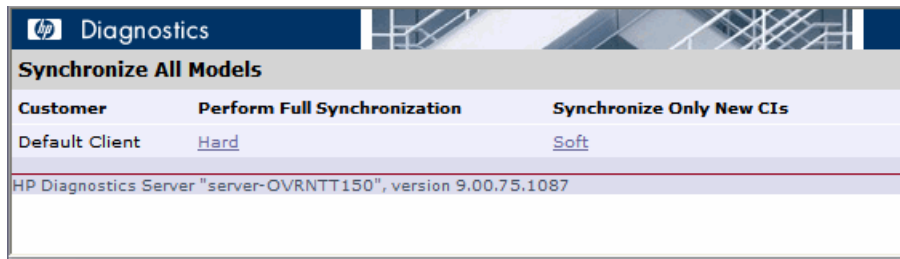
For ASP.NET applications see “Discovery of IIS Metadata for CI Population of IIS Deployed ASP.NET Applications” on page 283 for information on discovery of IIS metadata required for Run-time Service Model population and running a Rescan if you modify an existing ASP.NET application deployment.

In rare cases where you want to change the timing of the process that adds these CIs to the Run-time Service Model, 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 12 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" on page 740 for details on checking the Event Channel status. See "Event Based Health Indicator Status Troubleshooting Flow" on page 915 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.

## Enabling Diagnostics Integration with BSM's Service Health Analyzer

You can enable an integration between Diagnostics and BSM's Service Health Analyzer (SHA). With this integration samples containing host metrics and probe metrics are sent from the Diagnostics Mediator Servers and relayed through the Diagnostics Commander Server to BSM where the metrics are put into the BSM SHA database.

The SHA application uses these Diagnostics probe and host metrics as well as metrics from other samples to create baselines. The SHA application compares metrics to the baseline and reports anomalies as performance issues are detected. For an anomaly you can drill down to Diagnostics Probes view or Hosts view for detailed Diagnostic data (see BSM's Service Health Analyzer documentation for details on using SHA).

The integration of Diagnostics with SHA is not enabled by default.

### To enable the integration:

- 1** On each Diagnostics Mediator locate the `/etc/server.properties` file.
- 2** Make the following changes.
 

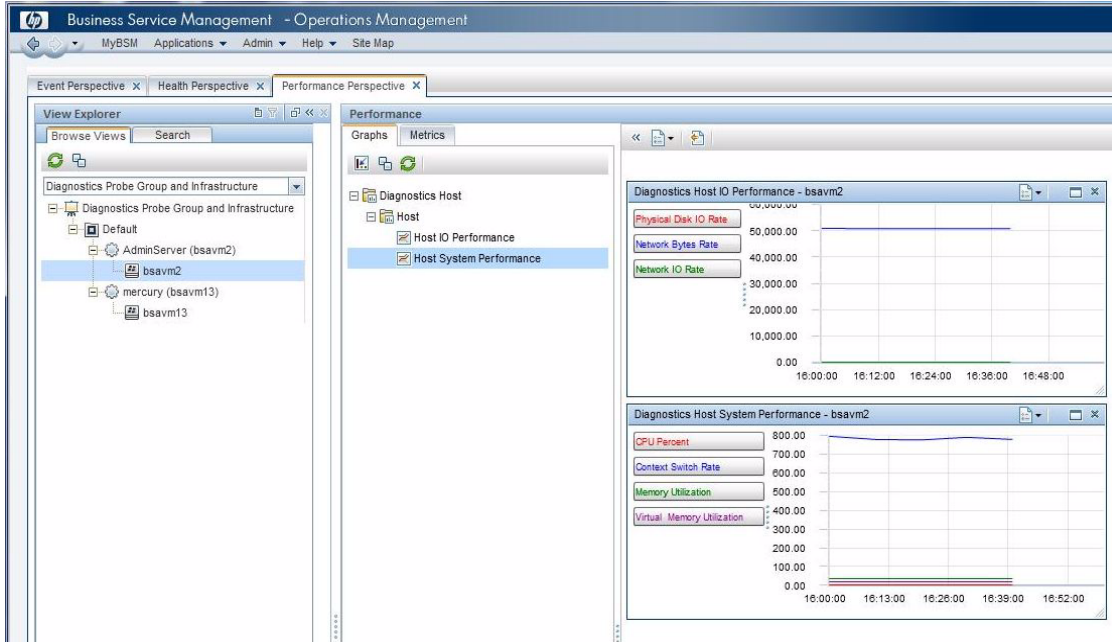
```
# Send host metrics for Service Health Analyzer (SHA)
bac.diag.sha.host.metric.create.samples=true
# Send probe metrics for Service Health Analyzer (SHA)
bac.diag.sha.probe.metric.create.samples=true
```
- 3** Restart each Diagnostics Mediator.
- 4** Once the integration is enabled and the host metrics and probe metrics from Diagnostics are available in Business Service Management's SHA database you then select these CIs in the SHA Admin application for use in anomaly detection.

You can also define filters in Diagnostics to determine which host and probe metrics are sent to SHA's database. Use the following XML files in the Diagnostics server's `/etc` directory to filter these metrics. Filters are based on regular expression matching similar to data exporting.

- **shaHostMetrics.xml.** Include/exclude filters for host metrics
- **shaProbeMetrics.xml.** Include/exclude filters for probe metrics

## Integration with BSM's Performance Graphing

Once you register the Diagnostics commander server with BSM, then in the BSM UI you can graph Diagnostics data. In the BSM UI (Applications > Operations Management > Performance Perspective) when you select a CI in the View Explorer tree you can see applicable Diagnostics graph templates in the Graphs tab. You can also graph individual Diagnostics metrics from the Metrics tab. An example from BSM is shown below.



## Diagnostics and OM Server Co-existence

Diagnostics 9.x bundles and uses the OM agent to send Health Indicator (HI) update events to BSM 9.x. The following procedure describes the necessary changes if the system that is used for the Diagnostics commander server already contains an OM agent and you want to configure reporting to an OM Server as well as to BSM servers.

## Configure Trusted Certificates

In an environment with multiple BSM/OM servers, you must configure each server to trust certificates that the other servers issued. This task involves exporting every server's trusted certificate, and then importing this trusted certificate to every other server. You must also update the agent's trusted certificates, so that the agent also trusts the BSM/OM servers.

### To configure trusted certificates for every BSM/OM:

- 1 On every BSM/OM server, export the trusted certificate to a file using the following command:

```
ovcert -exporttrusted -file <file>
```

The command generates a file with the name that you specify.

- 2 Copy each file to every other server, and then import the trusted certificate using the following commands:

```
ovcert -importtrusted -file <file>
```

```
ovcert -importtrusted -ovrg server -file <file>
```

- 3 On the Diagnostics system (in case an agent was already installed), update the trusted certificates using the following

```
ovcert -updatetrusted
```

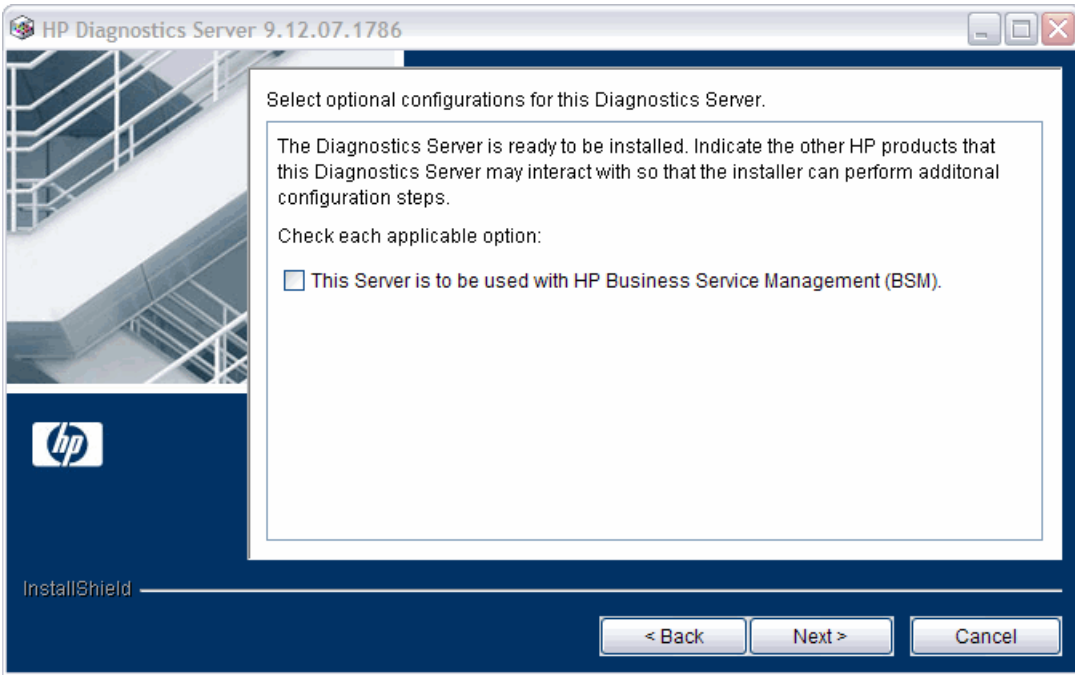
## OM Agent Installed Before Diagnostics is Installed

This scenario assumes that the OM agent is already present and reporting to an OM server on the system where Diagnostics commander server is to be installed.

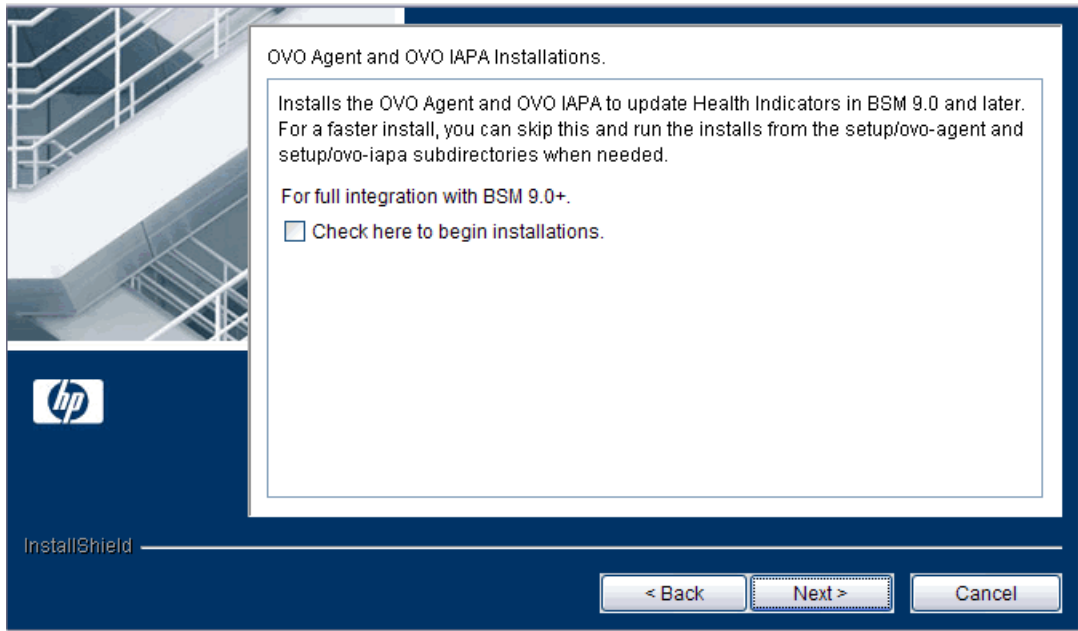
### To setup coexistence when OM Agent is installed first:

- 1 Install the Diagnostics commander server (see “Installing the Diagnostics Server” on page 53).

At this step, check the BSM integration box:



When the install comes to this step, don't check the box below:



- 2** On the Diagnostic commander server, install the IAPA component (see “Manual Installation of OM Agent and IAPA Components” on page 74 for instructions).
- 3** Register Diagnostics in BSM via the Diagnostics Admin page in BSM (see “Setting Up the Integration Between Business Service Management and Diagnostics” on page 737). Note if the Diagnostics commander server is on Linux and you aren’t running the Diagnostics Server as root, you will get an error during BSM registration of Diagnostics. Ignore this error and continue.

- 4 On the Diagnostics commander server, go to `<Diagnostics_server_install_dir>\bin` and execute `switch_ovo_agent.vbs` or on UNIX `switch_ovo_agent.sh`, specifying the OM server as the target for `-server` and `-cert_srv`. Note, on Linux, you have to run this command as root.

For example:

```
cscript switch_ovo_agent.vbs -server ovruxt65.rose.hp.com -cert_srv  
ovruxt65.rose.hp.com
```

- 5 Determine the core IDs for the BSM Gateway Server and OM Server. On the Diagnostics commander execute:

```
bbcutil -ping <OM Server>
```

```
bbcutil -ping <BSM Gateway Server>
```

- 6 Copy directory:

```
<Diagnostics_server_install_Dir>\newconfig\ovo-agent\policies\mgrconf  
to <Diagnostics_server_install_dir>\newconfig\ovo-agent\policies\tmp
```

If the `mgrconf` directory doesn't exist, contact support to get the content of this directory. Also if you have a more complex setup (for example with multiple OM managers) you may need to make additional changes to the file below.

- 7 Edit the file:

```
<Diagnostics_server_install_dir>\newconfig\ovo-agent\policies\tmp\mgrc  
onf\FF9A8F04-B5E3-43C3-999B-7A9492C35014_data.
```

- ▶ Locate the string `${OM_MGR_SRV}` and replace all occurrences with the FQDN of the HPOM management server.
- ▶ Locate the string `${OM_MGR_SRV_ID}` and replace all occurrences with the core ID of the HPOM management server.
- ▶ Locate the string `${OMi_MGR_SRV}` and replace all occurrences with the FQDN of the BSM gateway server.
- ▶ Locate the string `${OMi_MGR_SRV_ID}` and replace all occurrences with the core ID of the BSM gateway server.

Note in case of a more complex OM setup you may need to add additional entries in this file.



- 8 Go to the directory:  
<Diagnostics\_server\_install\_dir>\newconfig\ovo-agent\policies\tmp and install the policy:  
  
ovpolicy -install -dir mgrconf
- 9 The Diagnostics specific logfile encapsulator template that comes with the agent will now report to the BSM server and all of the other policies will report to the OM server.

## Diagnostics Already Installed

It is recommended to un-install the OM agent first and then use the above procedure.

Diagnostics 9.x bundles and uses the OM agent to send Health Indicator (HI) update events to BSM 9.x. The following procedure describes the necessary changes if you already have Diagnostics installed and you have installed the OM agent and IAPA components that come with Diagnostics you now want to configure events to flow to an OM Server as well as to BSM servers.

### To setup coexistence when Diagnostics is already installed:

- 1 Uninstall the OM agent component you installed with Diagnostics (see “Manual Uninstall of OM Agent and IAPA Components” on page 76 for instructions).
- 2 Go to your OM Server and deploy an OM agent to the Diagnostics commander server.
- 3 Then follow steps 2 through 9 on the previous pages.

## Configuration of Separate BSM Servers for DPS and Gateway

When integrating Diagnostics 9.0 or later with Business Service Management 9.0 or later, the OMi event channel is used by Diagnostics (actually the OM agent and IAPA components are used) to send Health Indicator status events from Diagnostics to the Business Service Management gateway server.

Diagnostics requires the OMi communication channel to be setup, see “Business Service Management Details” on page 743 for information on checking event channel integration status in BSM.

Your BSM Gateway server and BSM Data Processing Server can be set up on separate systems (see the BSM documentation for how to set this up). When the servers are running on different systems and you have event channel integration, there must be a trust relationship between the machines for communications between the BSM gateway server and BSM processing server.

**To set up certificates on a separate Gateway Server do the following:**

- 1** On the BSM Gateway Server, execute the following commands:  
**ovconfchg -ns sec.cm.client -set CERTIFICATE\_SERVER <processing\_server>**  
and:  
**ovcert -certreq**
  - 2** On the BSM Processing Server, execute the following commands:  
**ovcm -listpending -l**  
and:  
**ovcm -grant <reqid>**
  - 3** On the BSM Gateway Server, execute the following commands:  
**ovcert -list**  
and:  
**bbcutil -ping <processing\_server>**
- See the *Business Service Management Online Help* for more information.

## Additional Information on Integration

Some additional information on setting up integration between Diagnostics and Business Service Management is provided below.

### Authentication Dialog Displayed

When the Diagnostics server is installed on a different domain than the Business Service Management server, the MyBSM Diagnostics Dashboard may show an authentication dialog before the Diagnostics dashboard applet is displayed if Lightweight Single Sign-On (LWSSO) is not set up to add the Diagnostics server domain as a trusted domain.

To fix this issue, ensure that the domain that the Diagnostics server is running on is listed in Business Service Management's Single Sign-On page.

- 1** In Business Service Management select **Admin > Platform > Users and Permissions > Authentication Management > Single Sign-On Configuration**.
- 2** Click the **Configure** button.
- 3** Click **Next** in the wizard to get to the Single Sign-On page.
- 4** Click the **Add a Trusted host/domain** icon and enter the Diagnostics Server's domain.
- 5** Click **Next**.
- 6** Click **Next**.
- 7** Click **Finish**. This logs you out of Business Service Management. Log back in to Business Service Management and open the MyBSM Diagnostics Dashboard.

### Missing Link in the Diagnostics UI

If the Diagnostics UI is launched from Business Service Management and in addition the Diagnostics UI is launched in standalone mode on the same system, the Maintenance link in the Diagnostics UI in standalone mode will not be available.

To resolve this issue close both instances of the Diagnostics UI and re-launch the Diagnostics standalone UI.

## **HI Events not Flowing to Business Service Management**

When integrated with Business Service Management 9.x, Diagnostics sends Health Indicator status events to the Business Service Management gateway server. If there are problems with HI events flowing to Business Service Management see Appendix H, “Troubleshooting HP Diagnostics” sections on the OM Agent, BSM Gateway Trust Relationship and Event Based HI Status Troubleshooting Flow.

# 23

---

## 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 and connect to the Diagnostics UI.

**This chapter includes:**

- ▶ Before Installing the LoadRunner Diagnostics Add-in on page 766
- ▶ Installing the LoadRunner Diagnostics Add-in on page 766

## Before Installing the LoadRunner Diagnostics Add-in

Before installing the LoadRunner Diagnostics Add-in, Diagnostics commander server and LoadRunner must be installed. To install LoadRunner, see the *HP LoadRunner Installation Guide*.

You need to use the latest published Diagnostics 9.10 LoadRunner add-in for Diagnostics versions 8.0x, 9.0x, 9.10 and later.

## 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.

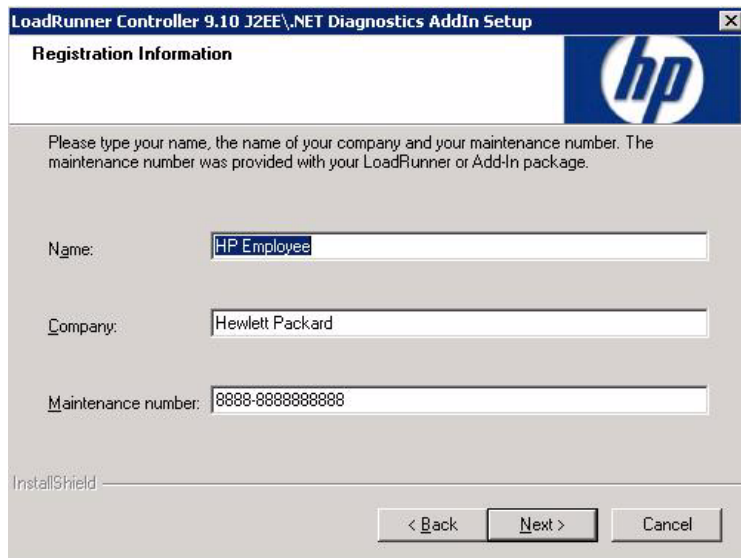
---

### 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.

The Registration Information dialog box opens.



The screenshot shows a Windows-style dialog box titled "LoadRunner Controller 9.10 J2EE\,NET Diagnostics AddIn Setup". The dialog has a blue header bar with the HP logo on the right. Below the header, the text "Registration Information" is displayed. A message reads: "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." There are three text input fields: "Name:" with the value "HP Employee", "Company:" with the value "Hewlett Packard", and "Maintenance number:" with the value "8888-8888888888". At the bottom left, there is a small "InstallShield" logo. At the bottom right, there are three buttons: "< Back", "Next >", and "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.

- 4 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 24, “Setting Up HP LoadRunner and HP Diagnostics Integration” for information on configuring LoadRunner for integration with Diagnostics.



# 24

---

## Setting Up HP LoadRunner and HP Diagnostics Integration

General information is provided on setting up HP LoadRunner and HP Diagnostics integration in load testing runs and offline analysis.

**This chapter includes:**

- ▶ How You Can Use HP Diagnostics with LoadRunner on page 770
- ▶ About Setting Up LoadRunner to Integrate with HP Diagnostics on page 773
- ▶ Configuring LoadRunner Scenarios to use HP Diagnostics on page 774
- ▶ Selecting Probe Metrics to Include in the Offline Analysis File on page 774
- ▶ Improving Transfer of Large Offline Analysis Files on page 777
- ▶ Out of Memory Issue in LoadRunner Controller's Diagnostics UI on page 777

## How You Can Use HP Diagnostics with LoadRunner

LoadRunner diagnostics modules and the integration with HP Diagnostics provide detailed performance information in LoadRunner to help you rapidly identify and pinpoint performance problems in Siebel, Oracle, SAP, J2EE and .NET environments.

In LoadRunner, the **J2EE/.NET diagnostics** functionality is provided by HP Diagnostics allowing you to monitor, analyze and solve complex performance problems in your J2EE and .NET application test environments.

Once you set up LoadRunner integration with HP Diagnostics you can view HP Diagnostics data in LoadRunner in a number of ways:

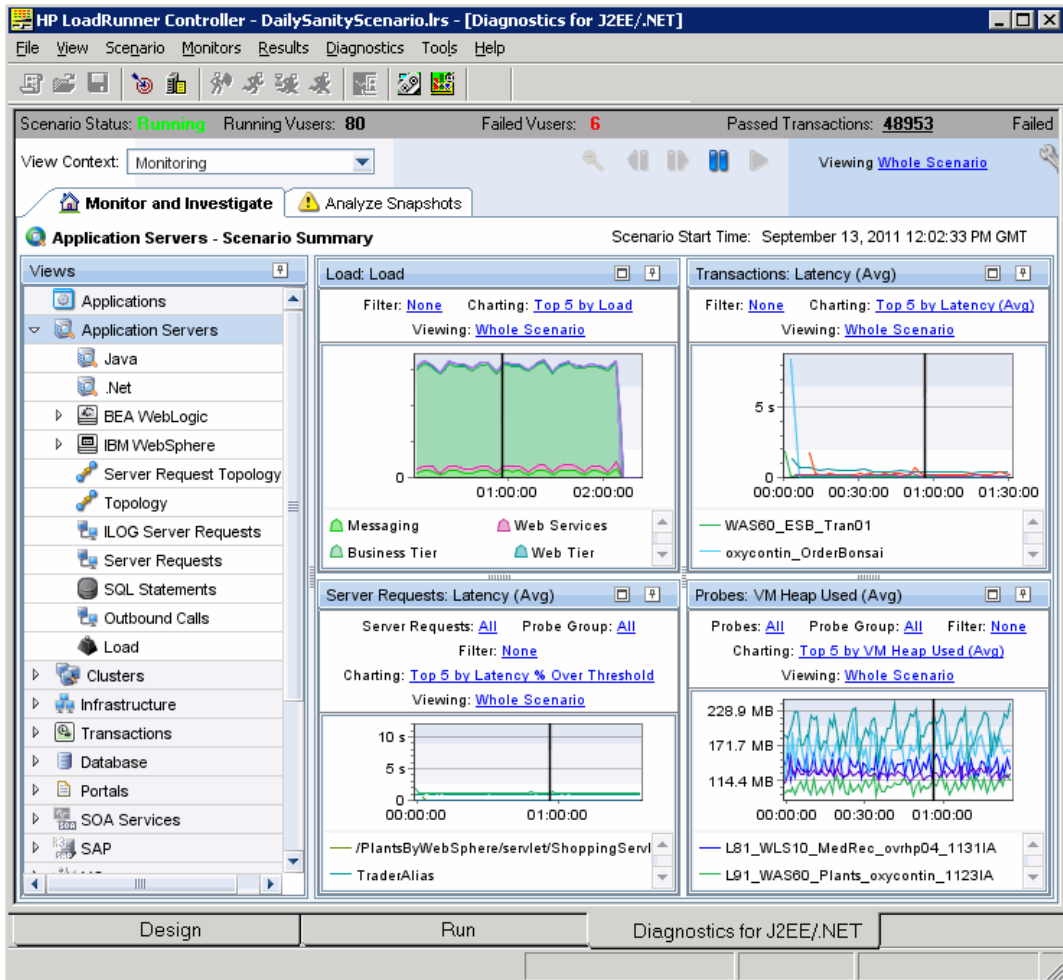
- ▶ Diagnostics UI Views for a LoadRunner Scenario
- ▶ Drill Down to the Diagnostics UI for Details on a Transaction
- ▶ LoadRunner Analysis J2EE and .NET Diagnostics Graphs

### **Diagnostics UI Views for a LoadRunner Scenario**

For a LoadRunner load test scenario you can open the HP Diagnostics UI and get detailed performance data for the whole scenario.

In LoadRunner you select the Diagnostics for J2EE/.NET tab at the bottom of the scenario window and the HP Diagnostics UI opens displaying detailed diagnostic data for the scenario. In the HP Diagnostics UI you can navigate to other views to identify, isolate, analyze and solve performance problems detected during the run.

Following is an example of the HP Diagnostics UI:



## Drill Down to the Diagnostics UI for Details on a Transaction

In LoadRunner you can drill down to the HP Diagnostics UI for a particular transaction to get diagnostics data for that transaction.

In LoadRunner select one of the Transactions graphs (such as Transaction Response Time) and from the graph you drill down to the HP Diagnostics UI which opens displaying the Transactions view. From here you can navigate to other views in the HP Diagnostics UI to troubleshoot problems related to the transaction.

The screenshot shows the HP LoadRunner Controller interface for a scenario named 'DailySanityScenario.lrs'. The status bar indicates the scenario is 'Running' with 80 running users, 6 failed users, 49340 passed transactions, and 0 failed transactions. The main content area displays the 'Transactions - Synthetic Transactions' view, showing a chart of 'Latency (Avg)' over time and a table of transaction details.

Status	Chart	Transaction	BPM Profile...	Latency	Throu...	CPU (Avg)
<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	ovrntt121_IIS_MSPetSho...		42.8 ms	9 / min	26.8 ms
<input type="radio"/>	<input type="checkbox"/>	WLS9_T209_MedRec_St... WLS9_...		30.5 ms	16 / hr	10.9 ms
<input type="radio"/>	<input type="checkbox"/>	WLS9_T209_MedRec_St... WLS9_...		349.4 ms	18 / hr	115.4 ms
<input type="radio"/>	<input type="checkbox"/>	WLS9_T209_MedRec_St... WLS9_...		26.6 ms	15 / hr	14.0 ms
<input type="radio"/>	<input type="checkbox"/>	WAS60_ESB_Tran01		91.3 ms	13 / hr	74.7 ms
<input type="radio"/>	<input type="checkbox"/>	T155_CallChain_CrossVM		2.6 s	7 / min	4.4 ms
<input type="radio"/>	<input type="checkbox"/>	SOA_wl91_ovrntt155_s...		1.5 s	14 / hr	653.1 ms

## LoadRunner Analysis J2EE and .NET Diagnostics Graphs

In LoadRunner Analysis the **J2EE and .NET Diagnostics graphs** are based on data provided by HP Diagnostics. These graphs enable you to trace, time and troubleshoot individual transactions and server requests through J2EE and .NET web, application and database servers. you can also quickly pinpoint problem servlets and JDBC calls to maximize business process performance, scalability and efficiency.

The J2EE & .NET Diagnostics graphs in LoadRunner are comprised of two groups: J2EE & .NET Diagnostics Graphs, J2EE & .NET Server Diagnostics Graphs.

Refer to the *HP LoadRunner Controller User Guide* and the *HP LoadRunner Analysis User Guide* for information on viewing HP Diagnostics data in LoadRunner.

## About Setting Up LoadRunner to Integrate with HP Diagnostics

Before you can access the Diagnostics UI from within LoadRunner, you must provide LoadRunner with the information it needs to communicate with the HP 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.

## Out of Memory Issue in LoadRunner Controller's Diagnostics UI

If you find `OutOfMemory` errors in the LoadRunner UI log (`Mercury_Diagnostics_UI.log`) this may be caused by a memory limit imposed on the Diagnostics applet displaying data in the Diagnostics tab of the LoadRunner Controller.

To fix this you can increase the heap memory of the Diagnostics applet by defining the system environment variables in the OS of the LoadRunner system: `APPCRITIC_MAX_MEM=256m` and `JAVA_TOOL_OPTIONS=-Xmx256m` (in case 256MB Max VM Heap is also needed).

# 25

---

## 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:**

- ▶ How You Can Use HP Diagnostics with Performance Center on page 780
- ▶ About Setting Up Performance Center to Use Diagnostics on page 782
- ▶ Configuring Performance Center Load Tests to Use Diagnostics on page 783
- ▶ Managing Performance Center Offline Files on page 784

## How You Can Use HP Diagnostics with Performance Center

Performance Center diagnostics modules and the integration with HP Diagnostics provide detailed performance information in Performance Center to help you rapidly identify and pinpoint performance problems in Siebel, Oracle, SAP, J2EE and .NET environments.

In Performance Center, the **J2EE/.NET diagnostics** functionality is provided by HP Diagnostics allowing you to monitor, analyze and solve complex performance problems in your J2EE and .NET application test environments.

Once you set up Performance Center integration with HP Diagnostics you can view HP Diagnostics data from Performance Center.

For a Performance Center load test run you can drill down into HP Diagnostics UI and get detailed performance data for the whole load test or for a particular transaction. After the load test run, you can use HP LoadRunner Analysis to analyze offline diagnostics data generated during the load test.

# Chapter 25 • Setting Up Performance Center to Use Diagnostics

The screenshot displays the HP Performance Center interface for a test named 'Sanity Sta...'. The top status bar shows 'Scheduler Action: Duration' and 'Next Step: Stopping 5 Vusers in 11:23:30'. The test status is 'Scheduler Running...'. Key performance metrics are: Running Vusers: 96, Time: 05:48:20, Hits/sec: 49 (last 60 sec), Passed trans: 21552, Failed trans: 106540, Errors: 111575. A 'Trans details' table is visible, showing columns for Group, Down, Init, Ready, Run, Rendez, Exiting, Passed, Failed, Stopped, and Error. A red box highlights the 'Diagnostics' button in the right-hand panel.

Below the test summary, a 'Transaction Response Time' graph is shown, plotting response time (0-50) against time. A red box highlights the graph's toolbar, which includes a magnifying glass icon.

The bottom portion of the image shows the 'Monitor and Investigate' section, titled 'Application Servers - Scenario Summary'. It features a 'Views' tree on the left and several monitoring graphs on the right:

- Load: Load**: A line graph showing load over time (02:00:00 to 04:00:00).
- Transactions: Latency (Avg)**: A line graph showing transaction latency over time.
- Server Requests: Latency (Avg)**: A line graph showing server request latency over time.
- Probes: VM Heap Used (Avg)**: A line graph showing VM heap usage over time.

The interface also includes a sidebar with navigation options like 'Project', 'Load Tests', 'Now Running', 'User Management', and 'Miscellaneous'. The bottom right corner shows a vertical list of values: 844, 188, 969, 047, 024, 109.

## 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 799.

- ▶ **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 675.

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 System Views for Administrators
- 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 787
- Using the Diagnostics Administration UI on page 790

## Accessing the Diagnostics Administration UI

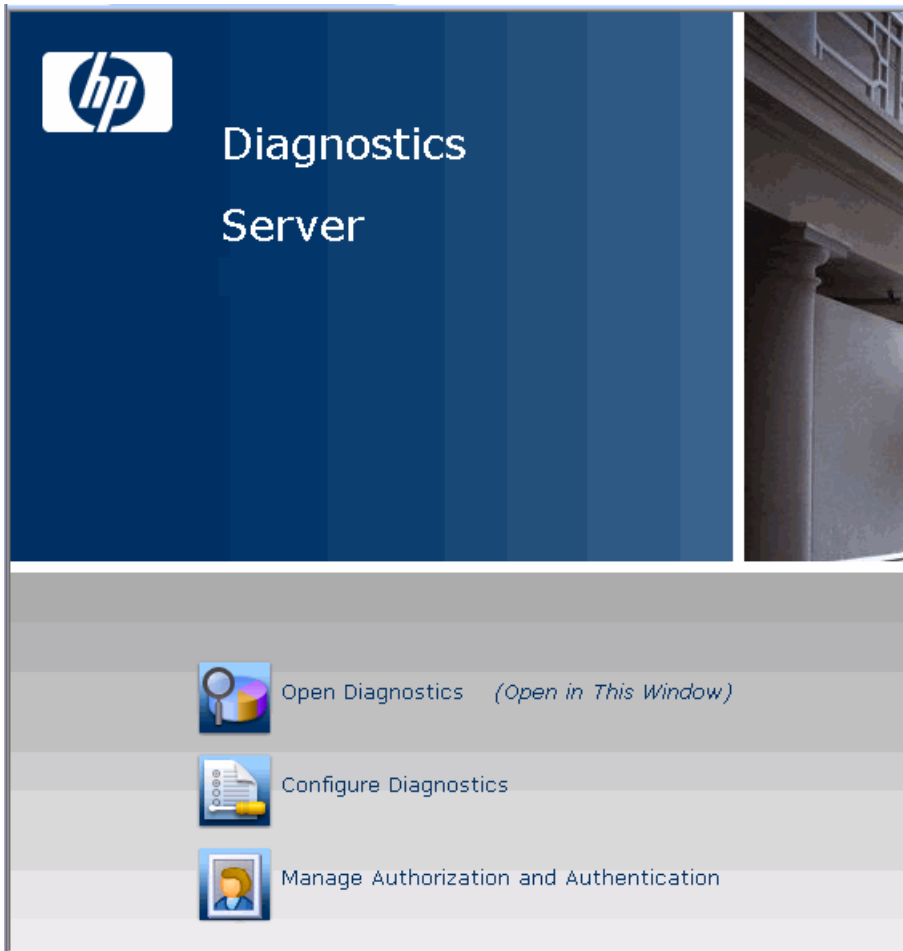
You can view information about the Diagnostics configuration, set the user privileges, configure Diagnostics settings and manage your Diagnostics software directly from the main UI of Diagnostics.

### **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 > All 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 793.

- **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 797.

- 2 From the main Diagnostics UI select **Configure Diagnostics**. See “Using the Diagnostics Administration UI” on page 790 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
<a href="#">registrar</a>	Central list of all Diagnostics component deployments
<a href="#">query</a>	Query API - allows you to download diagnostics data in HTML, XML or as Java objects
<a href="#">security</a>	Built-In User Management
<a href="#">logging</a>	Configure log files and logging details.
<a href="#">configuration</a>	Configuration
<a href="#">files</a>	Installation directory browser - upload and download property files, log files, etc
<a href="#">license</a>	License Management
<a href="#">synchronize</a>	Synchronize uCMDB Models in BSM
<a href="#">thresholding</a>	Script thresholds and alert rules

[\(Show Advanced Options\)](#)

HP Diagnostics Server "server-ovrntt150", version 9.20.51.1091

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. Also see the *HP Diagnostics Data Model and Query API Guide* (pdf) available from the online help Home page and in the Documentation directory.

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 802.
- **Logging.** Configure log files and logging details.
- **Configuration.** Configure the Diagnostics Server. See “Making Server Configuration Changes” on page 793 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 79 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).
- **Thresholding.** Script statements for setting thresholds and alerts.

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.

Configuration	
Name	Description
<a href="#">Customer Information</a>	Properties that are used to configure the customer information for the Server.
<a href="#">Alert Properties</a>	Properties that are used to configure the alert settings for the Server. All changes take effect dynamically, without server restart.
<a href="#">Component Communications</a>	Properties that are used to configure how this Diagnostics Component communicates with the other diagnostic components.
<a href="#">Memory Diagnostics</a>	Properties that configure the way that the Server will handle memory diagnostics
<a href="#">Online Cache</a>	Properties that are used to configure the Server's Online Cache.
<a href="#">logging</a>	Configure log files and logging details.

[Show Advanced Options](#)

HP Diagnostics Server "server-OVRNTT150", version 7.0.9.214

- 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

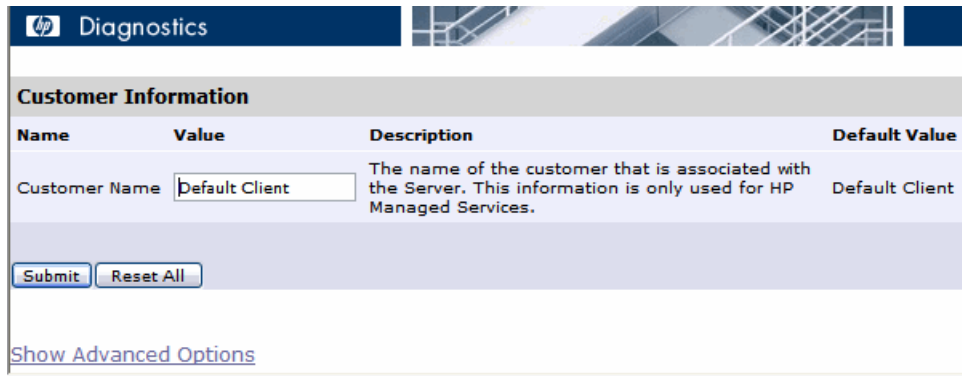
The configuration options displayed on this page are the commonly configured options. The more advanced configuration options are hidden by default. Select **Show Advanced Options** to see more configuration options.

---

**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.



The screenshot shows the HP Diagnostics interface. At the top, there is a blue header with the HP logo and the word "Diagnostics". Below the header, the page title "Customer Information" is displayed. The main content area contains a table with the following structure:

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

Below the table, there are two buttons: "Submit" and "Reset All". At the bottom of the form, there is a link labeled "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 798
- Understanding User Privileges on page 799
- Understanding Roles on page 800
- Accessing Diagnostics Using Default User Names on page 801
- Understanding the Diagnostics Server Permissions Page on page 802
- Creating, Editing and Deleting Users on page 810
- Assigning Privileges Across the Diagnostics Deployment on page 812
- Assigning Privileges for Probe Groups on page 813
- Authentication and Authorization for Users of Integrated HP Software Products on page 816
- Tracking User Administration Activity on page 818
- List of Active Users on page 819
- Configuring Diagnostics to use JAAS on page 820

## 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 Diagnostics Java Profilers” on page 518.
  - ▶ For information about managing authentication and authorization for the .NET Agent installed as a profiler only, see “Authentication and Authorization for .NET Profilers” on page 658.
-

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
<b>View</b>	The user can view Diagnostics data from the UI.
<b>Execute</b>	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.
<b>Change</b>	The user can access the <b>Configure Diagnostics</b> 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 812.

## 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
<b>user</b>	View	Can only view the data from the UI.
<b>superuser</b>	View, Execute	Can view data, change thresholds, and create alerts and comments from the UI.
<b>admin</b>	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

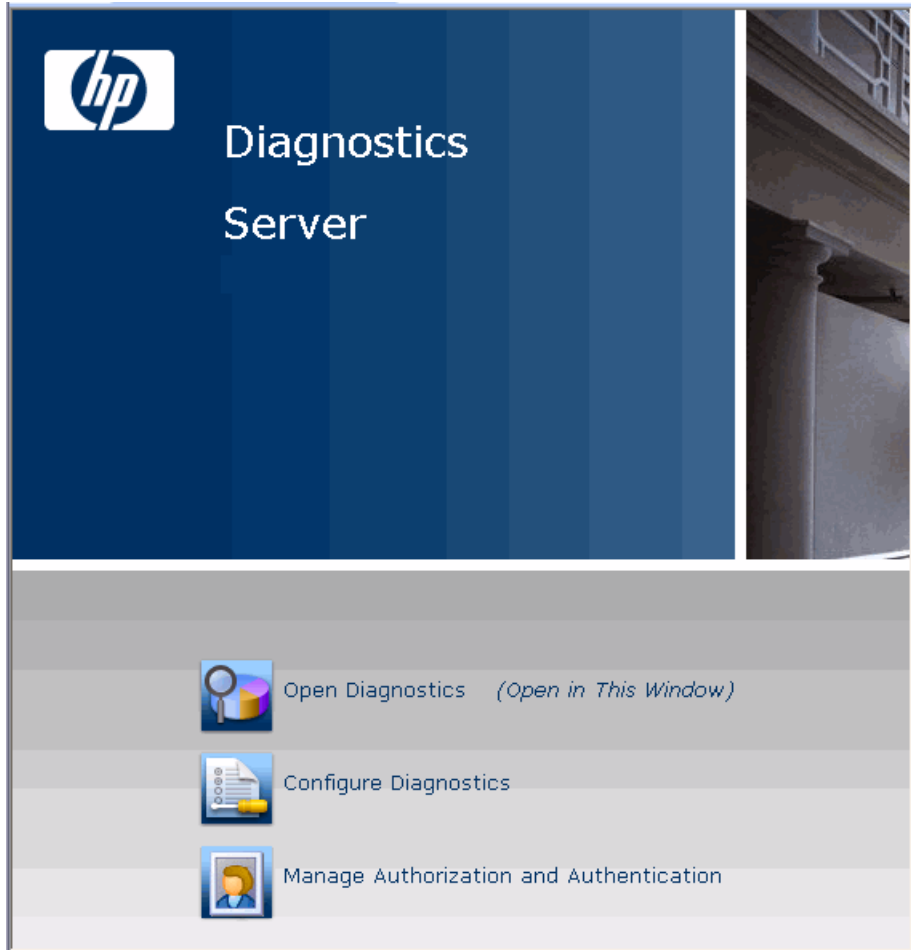
You manage users and assign user privileges in the Permissions page.

This section includes:

- “Accessing the Permissions Page” that follows.
- “The Permissions Page at a Glance” on page 805
- “Enterprise and Application Permissions” on page 806

## 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 <Diagnostics\_commander\_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 Commander 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
<b>Diagnostics UI</b>						
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			
Configure Business Transactions		X				

**Appendix B • User Authentication and Authorization**

Area and Action	Enterprise Permissions			Application Permissions		
	view	execute	change	view	modify	edit screens
View system health <b>Note:</b> To view system health, you also require <b>system</b> enterprise permission.			X			
Manage authorization and authentication for other users			X			
Access maintenance page			X			
Working with incidents	X					
Working with custom views	X					
<b>Profiler UI</b>						
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			
<b>User defined applications</b>						
Create application		X				
Delete application		X			X	
Rename application		X			X	
Change application path		X			X	



Area and Action	Enterprise Permissions			Application Permissions		
	view	execute	change	view	modify	edit screens
Modify application permissions		X			X	
Edit custom application screen	X					X
Add entity to application	X				X	
Remove entity from application	X				X	
Edit entity properties (thresholds, comments, etc)		X		X		
View application	X			X		
<b>Auto discovered applications, transaction applications or Entire Enterprise</b>						
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 803.
- 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 812.

**To assign roles:**


- 1 Access the Diagnostics Server Permissions page as described in “Accessing the Permissions Page” on page 803.
- 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 812 and “Assigning Privileges for Probe Groups” on page 813).

---

**To delete a user:**

- 1 Access the Diagnostics Server Permissions page as described in “Accessing the Permissions Page” on page 803.
- 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 803.
- 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 803.
- 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 799.

---

**To assign user privileges across the entire Enterprise:**

- 1 Access the Diagnostics Server Permissions page as described in “Accessing the Permissions Page” on page 803.
- 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 810.

---

- 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 801. 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 799.

---

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 803.
- 2 In the **Control over probes connected to <Diagnostics\_commander\_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 803.
- 2 In the **Control over probes connected to <Diagnostics\_commander\_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 816
- ▶ “Authentication and Authorization for Performance Center and LoadRunner Users” on page 817

### **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 748.

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 810.

---

**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 > All 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 799.
- 

**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 commander 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 Oracle's documentation on JAAS and Oracle's javadoc on `javax.security.auth.login.Configuration`.

---

**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 Diagnostics commander 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**

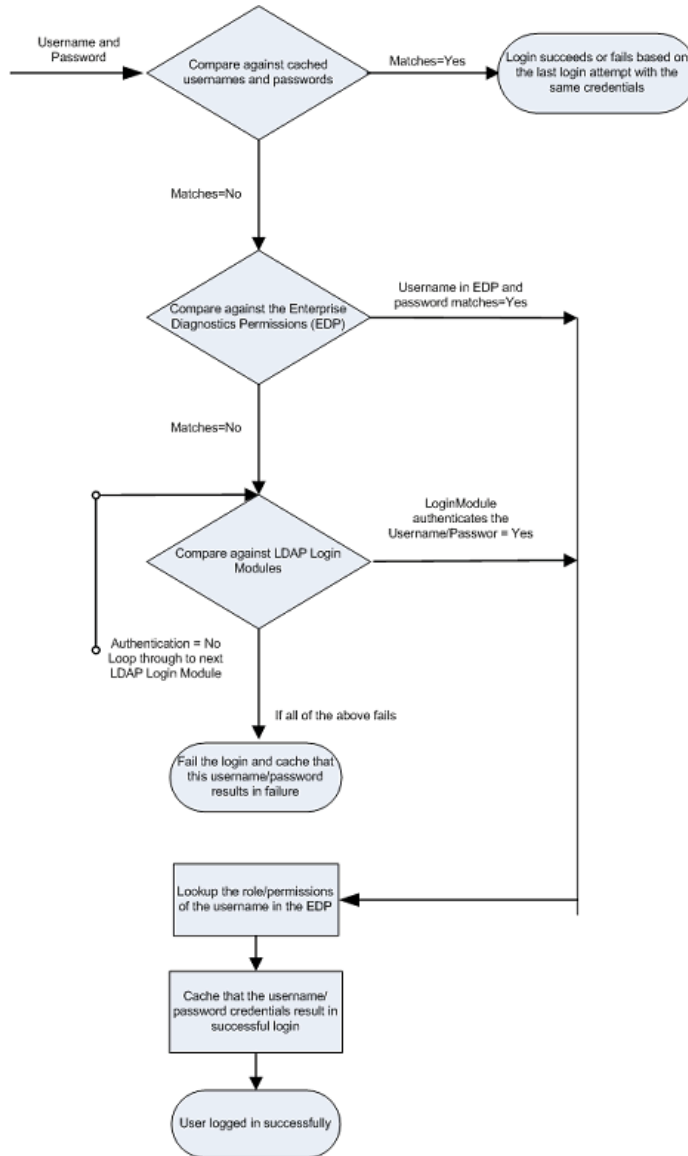
To configure LDAP authentication in Diagnostics you must first configure Diagnostics to use JAAS (see “Configuring Diagnostics to use JAAS” on page 820) and then configure the LDAPLoginModule on the Diagnostics commander server.

**In general here is how Diagnostics handles permissions and how LDAP authentication is handled (see the flow diagram on the following page):**

- 1** Accept a username and password from the user.
- 2** Compare the username and password against cached usernames and passwords.
  - a** If the username is in the cache and the password matches, then the login succeeds or fails based on the last login attempt with the same credentials.
  - b** Otherwise, proceed to the next step.

- 3** Compare the username and password against the Enterprise Diagnostics Permissions (EDP).
  - a** If the username is in the EDP and the password matches, then go to step 6.
  - b** Otherwise, proceed to the next step.
- 4** Loop through all the LDAP Login Modules configured in jaas.configuration.
  - a** If the LoginModule authenticates the username/password, then go to step 6.
  - b** Otherwise, proceed to the next Login Module.
- 5** If all the above fail, then fail the login and cache that this username/password results in failure.
- 6** Lookup the role/permissions of the username in the EDP.
- 7** Cache that the username/password credentials result in a successful login.
- 8** Return that the user logged in successfully.

### Diagnostics LDAP Authentication Flow





Edit the Diagnostics JAAS realm (application) block in `<INSTALL_DIR>/etc/jaas.configuration` with option values specific to your LDAP server.

The LDAPLoginModule may be used in a simplified or an advanced mode.

- In both modes:
  - SSL and a server certificate may be configured.
  - Roles may be configured.
  - Debug information may be requested.
- In simplified mode:
  - Anonymous directory searches may be done.
  - A predefined search filter is used.
  - Only a single base DN (distinguished name) may be configured.
  - Referrals are not available.
- In advanced mode:
  - Credentials must be provided for directory searches.
  - RFC 2254 compliant search filters may be used.
  - Multiple base DN's may be configured.
  - Referrals are available.

You can launch the **ldp.exe** utility on your Active Directory system to test settings.

The following table lists **LDAPLoginModule common attributes** (for all modes):

Attribute	Description and Examples	Values
authType	Specifies the security level to use when authenticating the user.	"simple" (default) "none" "strong"
debug	Specifies whether to write debug information to server.log.	"false" (default) "true"
defaultRoles	Comma-delimited list of roles to assign each authenticated user. Example: "SuperUsers"	
roleAttributes	Comma-delimited list of the user's DN attributes whose values will be used as the user's roles. If <b>defaultRoles</b> is also set, the resulting roles will be the union of the <b>defaultRoles</b> and <b>roleAttributes</b> . Example: "employeeType,hpJobFunction"	"roles" (default)
serverCertificate	Path to the trust store file containing the LDAP server's certificate. Path can be absolute or relative to the server's installation directory. See Appendix C, "Enabling HTTPS Between Components" for information on generating a keystore. Example: "etc/jssecacerts"	
useSSL	If set to <b>true</b> , use SSL to connect to the LDAP server.	"false" (default) "true"

The following table lists **LDAPLoginModule simple mode attributes**:

Attribute	Description and Examples	Values
allowAnonymous	If set to <b>true</b> , then allow anonymous searches of the LDAP server to retrieve the user's principal DN. To be effective the <b>searchFirst</b> attribute must also be set to <b>true</b> .	"false" (default) "true"
baseDN	Used to construct the principal's DN. If anonymous searches are allowed, then it is also used to specify which base DN to search for the user in. (required) Example: "OU=Users,DC=your,DC=ldap,DC=domain,DC=com"	
providerURL	URL to the LDAP server. Used for authentication. If anonymous searches are allowed then it is also used to search for the user. (required) Example: "ldap://your.ldap.domain.com:389" SSL example: "ldaps://yourldap.domain.com:636"	
searchFirst	If set to <b>true</b> and <b>allowAnonymous</b> is also <b>true</b> then do an anonymous search for the users; otherwise, construct the user's principal DN from the <b>uidAttribute</b> , the user's login name and the <b>baseDN</b> attribute.	"false" (default) "true"
uidAttribute	Used in the construction of the user's principal DN. If anonymous searches are allowed, it is also used to construct the search filter.	"uid" (default) common values: "uid", "CN"

Example of constructing the user's principal DN:

If uidAttribute="UID", and user login name is jsmith, and baseDN="OU=Users,DC=your,DC=ldap,DC=domain,DC=com", then the user's principal DN will be:

"UID=jsmith,OU=Users,DC=your,DC=ldap,DC=domain,DC=com"

The following table lists **LDAPLoginModule advanced mode attributes**:

Attribute	Description and Examples	Values
providerURL	<p>URL to the LDAP server used for authentication. Used to authenticate the user.</p> <p>Example: "ldap://yourldap.domain.com:389"</p> <p>SSL example: "ldaps://your.ldap.domain.com:636"</p>	Default is the value of the searchProviderURL attribute
searchBaseDNs	<p>Semicolon-separated list of base DN's to which to apply the search filter. (required)</p> <p>Example: "DN=America,DN=ns,DN=root,DN=com; DN=asia,DN=ns,DN=root,DN=com; DN=europe,DN=ns,DN=root,DN=com"</p> <p>Referral Example: "DN=ns,DN=root,DN=com"</p>	

Attribute	Description and Examples	Values
searchDN	<p>The principal's DN used to search for the user principal to authenticate. (required) Assumes that searchFirst is set to true even if you don't specify this.</p> <p>Example:  "CN=SearchAdmin,OU=Administrators,DC=americas,DC=ns,DC=root,DC=com"</p>	
searchFilter	<p>An RFC 2254 compliant search filter (see <a href="http://www.ietf.org/rfc/rfc2254.txt">http://www.ietf.org/rfc/rfc2254.txt</a>). The "{USERNAME}" string in the filter will be replaced with the user's login name before the directory is searched. When connecting to Active Directory, it is useful to test search filters using <b>ldp.exe</b> before putting them in the <b>jaas.configuration</b> file. (required)</p> <p>Example1: "(uid={USERNAME})"</p> <p>Example2:  "(&amp;(CN={USERNAME})(objectClass=user))"</p> <p>Example 3:  "(sAMAccountName={USERNAME})"</p>	
searchFirst	Is set to <b>true</b> .	"true"

Appendix B • User Authentication and Authorization

Attribute	Description and Examples	Values
searchPassword	<p>The password of the <b>searchDN</b> attribute. It may be plain text or obfuscated. (required) See Appendix C, “Enabling HTTPS Between Components” for information on password obfuscation.</p> <p>Example: "Secret123"</p> <p>Obfuscated example: "OBF:1fof1j1u1igh1ym51t331ym91idp1iz01fmn"</p>	
searchProvider URL	<p>URL to the LDAP server used to search for the user’s principal DN. This is used to find the user.</p> <p>Example: "ldap:// america.ns.root.com:389"</p> <p>SSL example: "ldaps:// america.ns.root.com:636"</p> <p>Referral example: "ldaps:// ns-root.com:636"</p>	Default is the value of the <b>providerURL</b> attribute.
searchReferral	<p>If set to <b>follow</b>, then the LDAP sever will refer search requests to other LDAP servers. if it cannot resolve the search or authentication request. If set to <b>follow</b> then only the forest’s principal DN needs to be listed in the <b>searchBaseDNs</b>.</p>	<p>"ignore" (default)</p> <p>"follow"</p> <p>"throw"</p> <p>(see <a href="http://download.oracle.com/javase/1.5.0/docs/guide/jndi/jndi-ldap-gl.html#referral">http://download.oracle.com/javase/1.5.0/docs/guide/jndi/jndi-ldap-gl.html#referral</a>).</p>

---

**Note:** After making any changes to **server.properties** or the **jaas.configuration** file, you must restart the Diagnostics commander server.

---

The following example is a configuration where all users have the same base DN that starts with "CN", so their principal DBs can be determined without searching for them.

```
Diagnosics {  
  baseDN="OU=Users,DC=simple,DC=domain,DC=com"  
  providerURL="ldap://simple.domain.com:389"  
  uidAttribute="CN"  
  ;  
};
```

If "larry" logs in, then his principal DN will be "CN=larry,OU=Users,DC=simple,DC=domain,DC=com".

The following example is a configuration where all users have the same base DN that starts with "CN", so the principle DNs can be determined without searching for them, but you want to search for them anyway.

```
Diagnosics {  
  allowAnonymous="true"  
  baseDN="OU=Users,DC=simple,DC=domain,DC=com"  
  providerURL="ldap://simple.domain.com:389"  
  searchFirst="true"  
  uidAttribute="CN"  
  ;  
};
```

If "sally" logs in, then her principal DN will be "CN=sally,OU=Users,DC=simple,DC=domain,DC=com".

The following example is a configuration where users may be from anyplace in the world, but we are only interested in IT employees from three regions.

```

Diagnostics {
  searchFirst="true"
  searchReferral="follow"
  useSSL="true"
  serverCertificate="etc/key.store"
  searchProviderURL="ldaps://america.ns.root.com:636"
  searchDN="CN=Searcher,OU=Admins,DC=america,DC=ns,DC=root,DC=com"
  searchPassword="OBF:1fof1j1u1igh1ym51t331ym91idp1iz01fmn"
  searchFilter="(&(CN={USERNAME})(objectClass=IT))"
  searchBaseDNs="DC=america,DC=ns,DC=root,DC=com; \
                DC=africa,DC=ns,DC=root,DC=com; \
                DC=russia,DC=ns,DC=root,DC=com"
;
};

```

If "ororro" in Africa logs in, then her principal DN will be "CN=ororro,OU=IT,DC=africa,DC=ns,DC=root,DC=com".

The following example is a configuration where users may be from anyplace in the world and hosted on different LDAP servers. In addition, users CNs may be localized but their sAMAccountNames are guaranteed to be ISO 8859-1.

```

Diagnostics {
  searchFirst="true"
  searchReferral="follow"
  useSSL="true"
  serverCertificate="etc/key.store"
  searchProviderURL="ldaps://ns.root.com:636"
  searchDN="CN=Searcher,OU=Admins,DC=america,DC=ns,DC=root,DC=com"
  searchPassword="OBF:1fof1j1u1igh1ym51t331ym91idp1iz01fmn"
  searchFilter="(sAMAccountName={USERNAME})"
  searchBaseDNs="DC=ns,DC=root,DC=com"
;
};

```

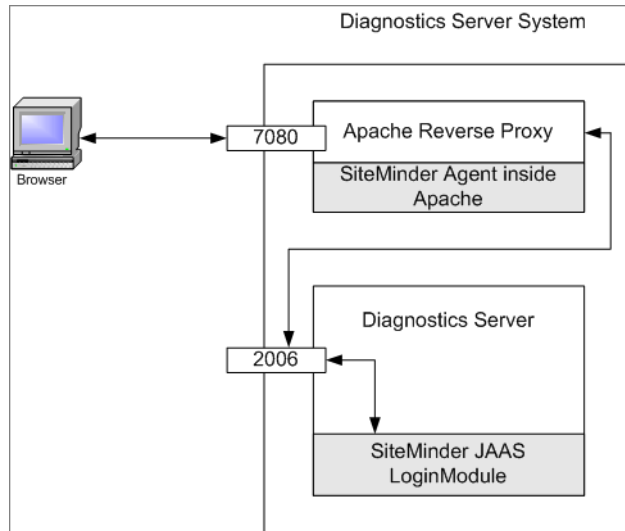
If Σαλοθ in Greece logs in as his sAMAccountName "Saloth", then his principal DN used for authentication will be "CN=Σαλοθ,OU=Υσερς,DC=greece,DC=ns,DC=root,DC=com".



## 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 /siteminderagent !
- ▶ 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)

---

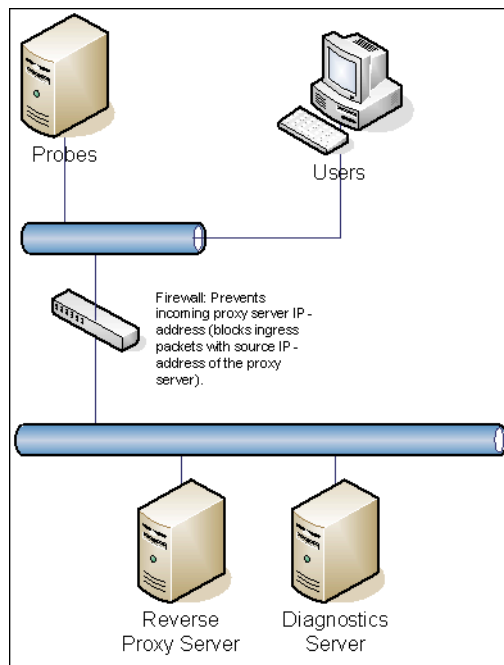
**Note:** After making any changes to the `httpd.conf` file, you must restart the Apache server (`apachectl stop` and `apachectl start`).

---

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 Diagnostics commander server:

- 1 Configure Diagnostics to use JAAS (see “Configuring Diagnostics to use JAAS” on page 820).
- 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 commander 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 840
- ▶ Filtering Encryption Cipher Suites on page 840
- ▶ HTTPS Checklist per Diagnostics Component on page 841
- ▶ Enabling Incoming HTTPS Communication for Diagnostics Components on page 843
- ▶ Generate Client Certificate on page 843
- ▶ Enabling Outgoing HTTPS Communication from Diagnostics Components on page 853
- ▶ Enabling HTTPS Communications for the Business Service Management Server on page 860

---

**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."

---

## Filtering Encryption Cipher Suites

A cipher suite defines the security algorithms and key sizes used for HTTPS/SSL encryption. The supported cipher suites are based on the version of Java used. Your organization may have a policy of filtering out certain ciphers either because they are not secure enough or are not allowed. There are two properties related to cipher suites that you can set in the **webserver.properties** file:

- ▶ **log.cipher.suites:** Prints messages related to cipher suites to the **server.log** file. These messages include the supported cipher suites and the cipher suites enabled after applying the **cipher.suites.filters**.
- ▶ **cipher.suites.filters:** A common-separated-value list of regular expression excludes and includes used to filter the cipher suites.



For example, suppose the following cipher suites are listed for your installation:

```
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA
SSL_DH_anon_WITH_RC4_128_MD5
```

But you want to filter out the 40-bit, anonymous and DES-based cipher suites and you only want to include RC4-based encryptions. Then you can specify a filter that looks like this:

```
cipher.suite.filters=\
exclude:.*[0-9]?40[0-9]?.*, \
exclude:.*_anon_.*, \
exclude:.*_DES_.*, \
exclude:.*_3DES_.*, \
INCLUDE:.*_WITH_RC4_.*, \
exclude:.*
```

## 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	Commander 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 commander server certificate	No	Yes	No	No	No
Copy mediator server certificate	Yes	No	Yes	Yes	Yes

Appendix C • Enabling HTTPS Between Components

Configuration step	Commander Server	Mediator Server	Java Probe	Collector	.NET Probe
Copy Java Probe certificates	Yes	Yes	No	No	No
Copy Collector certificates	No	Yes	No	No	No
Edit security.properties: enablessl=true, keystorepassword, keypassword	Yes	Yes	Yes	Yes	No
Edit security.properties: add commander 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 diagnosticserver 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 844
- “Configuring the Java Agent for Incoming HTTPS Connections” on page 847
- “Configuring the Collector for Incoming HTTPS Connections” on page 850

### Generate Client Certificate

Generate client certificate using advanced settings in Certificate Services and specify FQDN. Mark keys exportable and use the Friendly Name=CLIENT.

## Configuring the Diagnostics Server for Incoming HTTPS Connections

---

### Notes:

- ▶ To avoid issues with DNS and host name resolution, the Commander URL for the Diagnostics commander 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 Diagnostics commander server, you must use port 8443 to run the Enterprise UI, for example:  
`https://<commander_server>:8443`.
- 

### To configure the Diagnostics Server for incoming HTTPS connections:

- 1 Generate a keystore in the `<diagnostics_server_install_dir>/etc` directory. An example command is shown below:

```
<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:1ytc1vu91v2p1y831y7v1v1p1vv11yta
MD5:179ad45c6ce2cb97cf1029e212046e81
```

---

**Note:** If you did not use the same password for **keypass** and **storepass**, you must run this command twice to create an obfuscated version for each password.

---

- 4** Change the following properties in the file `<diagnostics_server_install_dir>/etc/security.properties` for the Diagnostics commander 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, for example:  
https://<my\_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: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 `<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:1ytc1vu91v2p1y831y7v1v1p1v11yta
```

---

## 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:1ytc1vu91v2p1y831y7v1v1p1v11yta
```

---

## 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 commander 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 commander server.
- 2** Change the value of the `trusted.certificate` property in the file `<diagnostics_server_install_dir>/etc/security.properties` for the Diagnostics commander 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 commander server.

Set **commander.url** to **https://<diagserver\_commander\_hostname>:8443**

**To enable the Diagnostics Mediator Server for outgoing communication to the Diagnostics commander server via HTTPS:**

- 1 Copy the certificate file from **<diagnostics\_server\_install\_dir>/etc/diag\_server\_commander.cer** on the Diagnostics commander 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_mediatorservice_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 commander 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 843 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 commander 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 “Discovery and Standard Instrumentation” on page 282.

**To verify that you successfully configured the .NET probe for HTTPS communication with the Diagnostics commander server:**

- 1** Browse to your .NET application to activate the .NET Agent.
- 2** Verify that the .NET Agent is available by checking the System Health view in the Diagnostics UI.

## 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 commander server and Business Service Management:**

- 1** Copy the Diagnostics certificate file, **diag\_server\_commander.cer**, from the Diagnostics commander 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 commander 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 commander 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 commander server exactly as it was specified in the **CN** parameter when you created the keystore for the Diagnostics commander 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 843.
    - Enter **HTTPS** for the protocol.
    - Enter **8443** for the Web port of the Diagnostics Server.
  - e** Click **Submit Configuration**.



# D

---

## Using System Views for Administrators

You can use the Diagnostics System views to monitor the health of the Diagnostics components and verify that they are working properly.

**This chapter includes:**

- System Views for Diagnostics' Administrators on page 863
- System Health View Description on page 865
- System Capacity View Description on page 866

### System Views for Diagnostics' Administrators

In large scale Diagnostics deployments you can use the System Views instead of the System Health Monitor. The specialized System Views allow you to quickly locate systems or groups of system based on various system attributes. Allowing you to more easily monitor system health and identify when systems are nearing capacity.

In many cases, the System views 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. At a glance, you can determine which components are experiencing problems.

**To access the System Views:**

- 1** Open the Diagnostics UI as the Mercury System customer from [http://<Diagnostics\\_Commanding\\_Server\\_Name>:2006/query/](http://<Diagnostics_Commanding_Server_Name>:2006/query/).
- 2** In the query page locate the Mercury System customer in the list and select the link to Open Diagnostics.

- 3 Log in to Diagnostics and on the Applications window select Entire Enterprise and select any link to open the Diagnostics Views.
- 4 In the Views pane you'll see the System Views view group. Open the view group and select either the System Health view or System Capacity view.

---

**Note:** The System Health Monitor is still available for compatibility and can be accessed from Performance Center, LoadRunner Controller or Business Service Management. Access to the System Health Monitor requires system permissions. From Diagnostics you can access it from `http://<Diagnostics command server host>:<Diagnostics command server port>/registrar/health`.

---



## System Health View Description

The System Health view in the Diagnostics UI provides overall health information for the components you have installed in your Diagnostics environment. The information is similar to that provided in the System Health Monitor but allows you to filter, sort and select metrics and attributes of interest.

Heartb...	Name	Type	Host	Mediator	Port	Run	Events per s...	Vers...
❌	L91_MSMQCon...	probe	ovresx2-vm3.ovrt...	Comman...	35007		0	9.20....
✅	CommandingSe...	Comma...	ovrntt150.ovrtest....		2006		0	9.20....
✅	server-hpsw-v...	mediator	hpsw-vm118.ovrt...	Comman...	2612		0	9.20....
✅	server-HPSWR...	mediator	HPSWROS018.o...	Comman...	2612		0	9.20....
✅	server-ovresx1...	mediator	ovresx1-vm4.ovrt...	Comman...	2612		85	9.20....
✅	server-ovresx1...	mediator	ovresx1-vm5.ovrt...	Comman...	2612		0	9.20....
✅	server-ovrlxd14...	mediator	ovrlxd14.ovrtest....	Comman...	2612		12	9.20....
✅	server-ovrntt100	mediator	ovrntt100.ovrtest....	Comman...	2612		0	9.20....
✅	server-OVRNT...	mediator	OVRNTT154.ovrt...	Comman...	2612		0	9.20....
✅	server-ovrsun2...	mediator	ovrsun28.rose.hp...	Comman...	2612		0	9.20....
✅	L95_1RootSimpl...	probe	OVRNTT66.ovrte...	Comman...	35001		0	9.20....
✅	L95_1RootSimpl...	probe	OVRNTT66.ovrte...	Comman...	35007		0	9.20....
✅	CruiseMasterA...	probe	amkibld03.rose.h...	server-h...	35000		0	9.00....
✅	CruiseNativeAm...	probe	amkibld05.rose.h...	server-h...	35000		0	9.00....
✅	CruiseNativeAm...	probe	amkibld06.rose.h...	server-h...	35000		0	9.00....
✅	CruiseNativeAm...	probe	amkibld07.rose.h...	server-h...	35000		0	9.00....
✅	CruiseNativeGe...	probe	gerstner.rose.hp...	server-h...	35000		0	9.00....
✅	CruiseNativeHp...	probe	hpswbld02.rose.h...	server-h...	35000		0	9.00....

Metric	
BPM transactions	0.0
BPM transaction...	0.0
Events per sec	0.0
Events since lau...	3816.0
Load Runner tra...	0.0
Load Runner tra...	0.0
Number of instru...	4.0
Throttling	ON
Time skew	-2 sec (beh
Trim latency	2.0
Trim depth	25.0
Last update (co...	Thu Mar 29
Up time	11 hour(s) +
Configuration	
Host	ovresx2-vm...
IP Address	16.77.35.14
Install dir	C:\Mercury
Instrumentation	MSMQCon...
Lightweight Mem...	OFF
Log file 1	C:\Mercury

## System Capacity View Description

The System Capacity view in the Diagnostics UI provides information for managing capacity in your Diagnostics environment. This view shows the number of probe groups and probes that are assigned to each Diagnostics mediator.

Name	Host	Port	Probe Group Count	Probe Count								
[-] server-ovresx1-vm5	ovresx1-vm5.ovrtest.adapps.hp.com	2612	3	8								
<table border="1"> <thead> <tr> <th>Probe Group Name</th> <th>Probe Count</th> </tr> </thead> <tbody> <tr> <td>Sanity_LR_9_1_ovresx1-vm5</td> <td>4</td> </tr> <tr> <td>Sanity_CallChain_WebService_ovresx1-vm5</td> <td>3</td> </tr> <tr> <td>Sanity_Collectors_ovresx1-vm5</td> <td>1</td> </tr> </tbody> </table>		Probe Group Name	Probe Count	Sanity_LR_9_1_ovresx1-vm5	4	Sanity_CallChain_WebService_ovresx1-vm5	3	Sanity_Collectors_ovresx1-vm5	1			
Probe Group Name	Probe Count											
Sanity_LR_9_1_ovresx1-vm5	4											
Sanity_CallChain_WebService_ovresx1-vm5	3											
Sanity_Collectors_ovresx1-vm5	1											
[+] server-HPSWROS018	HPSWROS018.ovrtest.adapps.hp.com	2612	4	8								
[+] server-ovrntt100	ovrntt100.ovrtest.adapps.hp.com	2612	4	8								
[+] server-ovrsun28.rose.hp.com	ovrsun28.rose.hp.com	2612	2	11								
[+] CommandingServer	ovrntt150.ovrtest.adapps.hp.com	2006	3	5								
[+] server-ovrlxd14.ovrtest.adapps....	ovrlxd14.ovrtest.adapps.hp.com	2612	1	9								
[+] server-OVRNTT154	OVRNTT154.ovrtest.adapps.hp.com	2612	3	5								
[+] server-hpsw-vm118.ovrtest.adap...	hpsw-vm118.ovrtest.adapps.hp.com	2612	1	10								
[+] server-ovresx1-vm4.ovrtest.ada...	ovresx1-vm4.ovrtest.adapps.hp.com	2612	6	17								

# E

---

## Diagnostics Data Management

Detailed information is provided on how Diagnostics data is managed and stored.

**This chapter includes:**

- About Diagnostics Data on page 868
- Custom View Data on page 868
- Performance History Data on page 870
- Data Retention on page 876
- Disk Space Issues on the Server on page 882
- Pre-Installation Data Management Considerations on page 882
- Backing Up Diagnostics Data on page 883
- Handling Diagnostics Data when Upgrading Diagnostics on page 888

## About Diagnostics Data

There are two main types of Diagnostics data:

- ▶ The custom views that each user has created.
- ▶ 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 LoadRunner/Performance Center runs and Business Service Management as well as the application metrics. The organization and maintenance of the data files that make up the data base are described in this chapter.

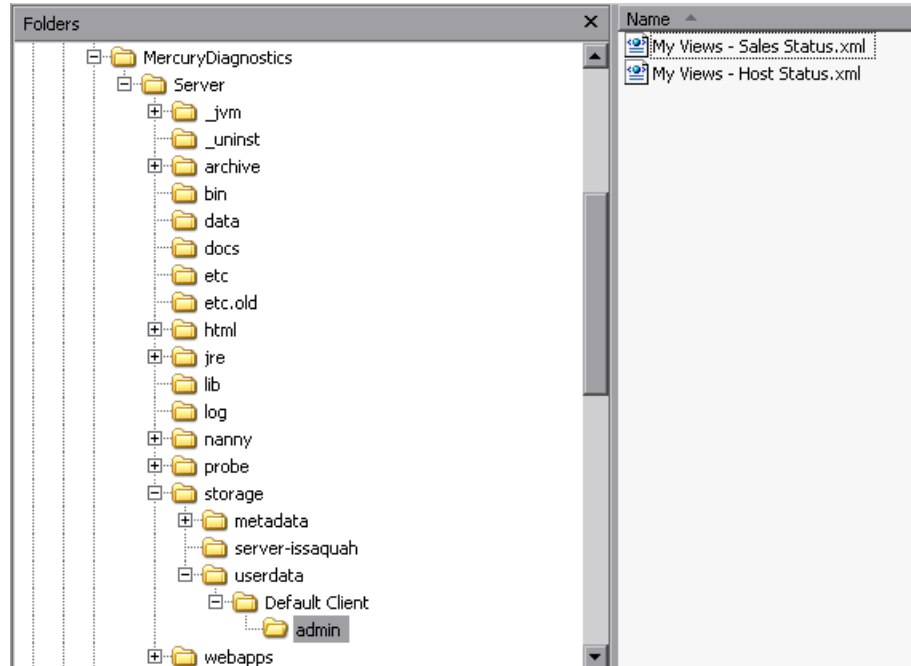
## Custom View Data

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 876.

This section includes:

- ▶ “Performance History Data Organization” on page 870
- ▶ “Performance History Data File Types” on page 874

### 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.

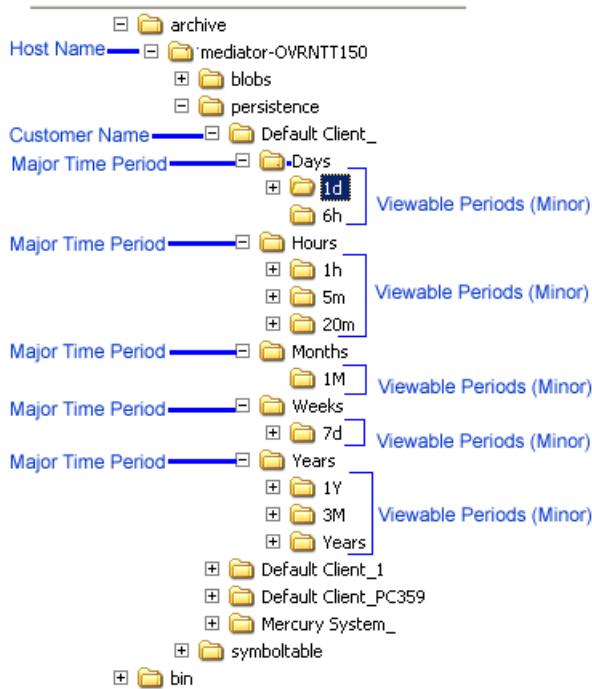
The archive **archive.dirname** property in **etc/server.properties** identifies where the archive should be stored (either absolute or relative to the `<server_install_dir>`). If you would like to move or store the archive on a NAS drive (for example `//<hostname>/<sharename>`) the share you configure in **archive.dirname** needs to have read/write permissions for the user(s) Diagnostics will run as.

---

**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:

Name	Size	Type
2009_4_27_0.trend	21,40...	TREND File
2009_4_27_0.summary	361 KB	SUMMARY File
2009_4_26_0.summary.zip	94 KB	WinZip File
2009_4_26_0.trend.zip	3,777 KB	WinZip File
2009_4_26_0.tree.zip	1 KB	WinZip File
2009_4_27_0.tree	1 KB	TREE File
2009_4_25_0.summary.zip	101 KB	WinZip File
2009_4_25_0.trend.zip	3,382 KB	WinZip File
2009_4_25_0.tree.zip	1 KB	WinZip File
2009_4_24_0.summary.zip	101 KB	WinZip File
2009_4_24_0.trend.zip	3,414 KB	WinZip File
2009_4_24_0.tree.zip	1 KB	WinZip File
2009_4_23_0.summary.zip	105 KB	WinZip File
2009_4_23_0.trend.zip	3,328 KB	WinZip File
2009_4_23_0.tree.zip	1 KB	WinZip File
2009_4_22_0.summary.zip	106 KB	WinZip File
2009_4_22_0.trend.zip	3,743 KB	WinZip File
2009_4_22_0.tree.zip	1 KB	WinZip File
2009_4_21_0.summary.zip	107 KB	WinZip File
2009_4_21_0.trend.zip	3,595 KB	WinZip File
2009_4_21_0.tree.zip	1 KB	WinZip File
2009_4_20_0.trend.zip	2,968 KB	WinZip File
2009_4_20_0.summary.zip	97 KB	WinZip File

### 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. Default settings for data retention are set out of the box. You should monitor your systems to check available disk space and change the data retention settings accordingly.

Data retention settings are taken into consideration in determining when purging occurs so if you see unexpectedly high disk space usage you should check your data retention settings to see if they need to be modified. See “Symbol Table Purging” on page 879.

This section includes:

- ▶ “Data Retention on the Mediators” on page 876
- ▶ “Data Retention Configuration” on page 877
- ▶ “Symbol Table Purging” on page 879

### 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 870) 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

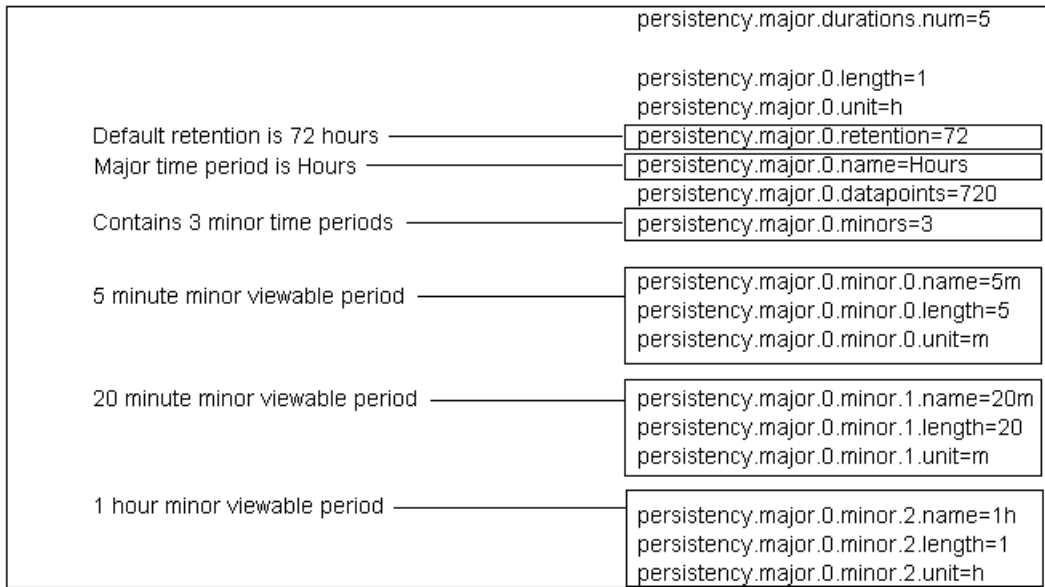
The above table only applies when the entities don't change and are constantly available for the specified periods of time. See “Symbol Table Purging” that follows.

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. Default settings for data retention are set out of the box. You should monitor your systems to check available disk space and change the data retention settings accordingly.

An example showing the persistence section of the server.properties file is shown below:



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].

```
persistency.major.0.minor.2.name=1h
persistency.major.0.minor.2.length=1
persistency.major.0.minor.2.unit=h
persistency.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 (months) x 31 (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

The purging mechanism must take a number of settings and other factors into account when determining when and how to purge data.

By default purging is set to run every 6 months (4320 hours). 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).

---

**Important:** Increasing the purging interval requires more server memory.

---

Data that is part of a snapshot is not purged since doing so renders the snapshot useless.

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.

A property can be specified for how much space you want the TSDB to use and data will typically be deleted to maintain a size less than this specified threshold. The **persistence.purging.threshold** property is set in the `<diagnostics_server_install_dir>/etc/server.properties` file. But note that a number of other settings can take priority over this threshold value and can result in too much data being retained.

If you find that disk space is being exhausted this does not mean that purging isn't working it may mean that one of the following factors has affected the purging mechanism. For example if you have allocated 10GB of disk space on the server for Diagnostics but you see the archive at 20GB in danger of exhausting disk space on the system, this could be possible for any of the following reasons:

- ▶ Purging interval has not been reached yet. You can adjust to a shorter interval.
- ▶ There are a large number of snapshots on the system that by design do not get purged.
- ▶ The data retention settings may be requiring too much data to be retained. You may need to adjust data retention in order to save disk space (see “Data Retention” on page 876).

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 (provided other factors do not affect the purging mechanism). 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.

## Disk Space Issues on the Server

You can set up E-mail alerts to be sent to the Diagnostics administrator for disk space issues on the server. The Administrator E-mail address can be entered during the server installation or setup later using the Alert Properties page.

Alerts are issued if the server has less than 100MB of free disk space for the archive directory. If the server has less than 50MB of space the server stops collecting data and exits. The server will also not startup if there is less than 50MB of free disk space. These precautions help ensure the server stops collecting data before running out of disk space to maintain server stability.

The thresholds that determine these types of alerts for the Diagnostics administrator are factory configured in the server's **server.properties** file. For more details see the comments in this file for the various **watchdog** properties.

## 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.

If you use your own backup approach, you have to shutdown the server (because of Locked PathSymbolTable.pst). But use of the backup script provided with Diagnostics is recommended.

---

**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 488

---

This section includes:

- ▶ “Backing up Data Remotely” on page 884
- ▶ “Configuring Symbol Table Backup” on page 886
- ▶ “Restoring Data After a Failure” on page 887

## 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 also 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` be located in `<diagnostics_install_dir>/server/bin/wget` directory (the `.sh` script just requires `wget` in the path).

The backup script can backup data remotely and from that directory you can do your traditional backup if you want. The script can also backup data to a local directory (ideally another drive on the same host).

---

**Important:** 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
<b>-h</b>	The host (or IP address) to download from
<b>-o</b>	The directory to store the backup in
<b>-u</b>	The HTTP username to use Default: admin
<b>-p</b>	The HTTP password to use Default: admin
<b>-P</b>	The HTTP port number to use (optional) Default: 2006
<b>-r</b>	The ID of the Diagnostics Server in Mediator mode being read from (for rhttp backups) (optional). For example, if you have 2 servers "commander" and "mediator", you could backup mediator over rhttp with: <code>-h commander -r mediatorld</code> .
<b>-c</b>	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).
<b>-v</b>	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	<b>etc/</b>
User custom views	<b>storage/userdata</b>
Raw performance history data	<b>archive/.../persistence/</b>
Symbol table	<b>archive/.../symboltable/</b>

### 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 **persistence.major.<n>.name** property (see “Data Retention Configuration” on page 877). For example, to backup the symbol table weekly, use the **persistence.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:**

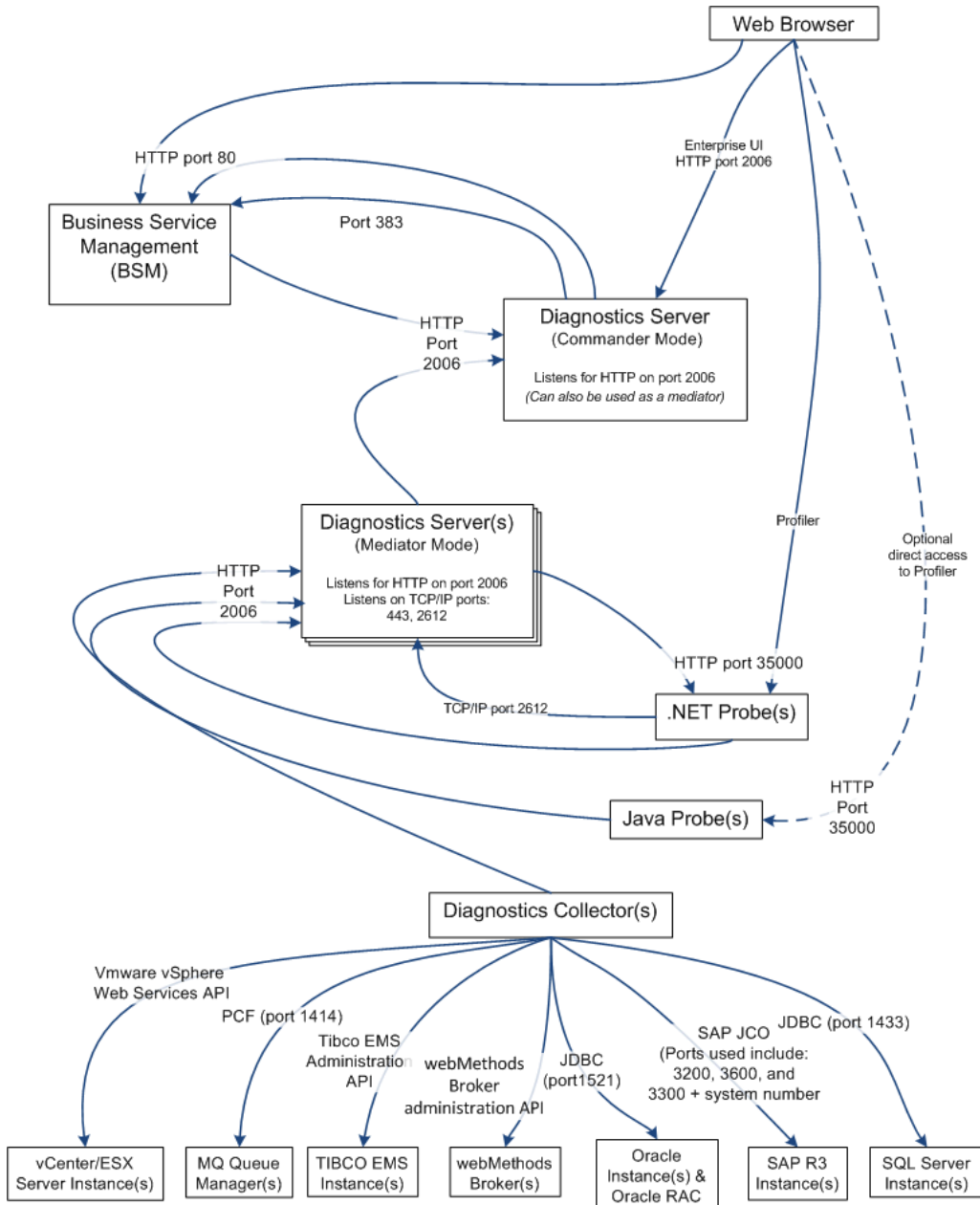
- Communications with Business Service Management on page 890
- Communications with LoadRunner and Performance Center on page 891
- .NET Probe Aggregator Data Flow on page 892

---

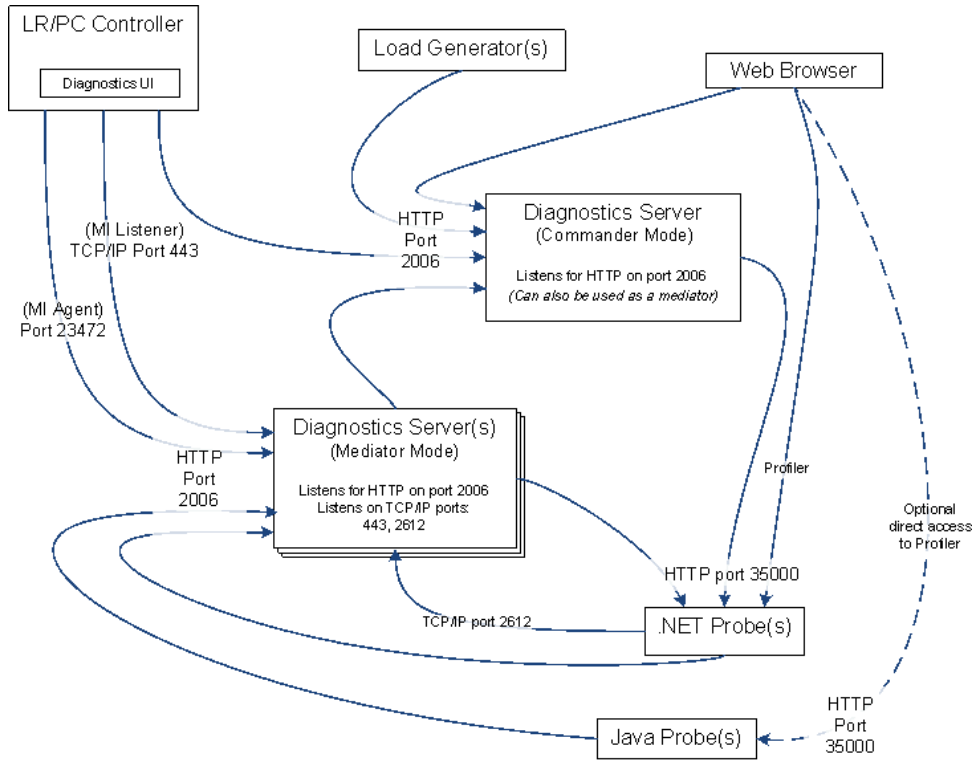
**Note:** The diagrams are intended to provide a high-level view, not provide an in-depth knowledge of the working of the components.

---

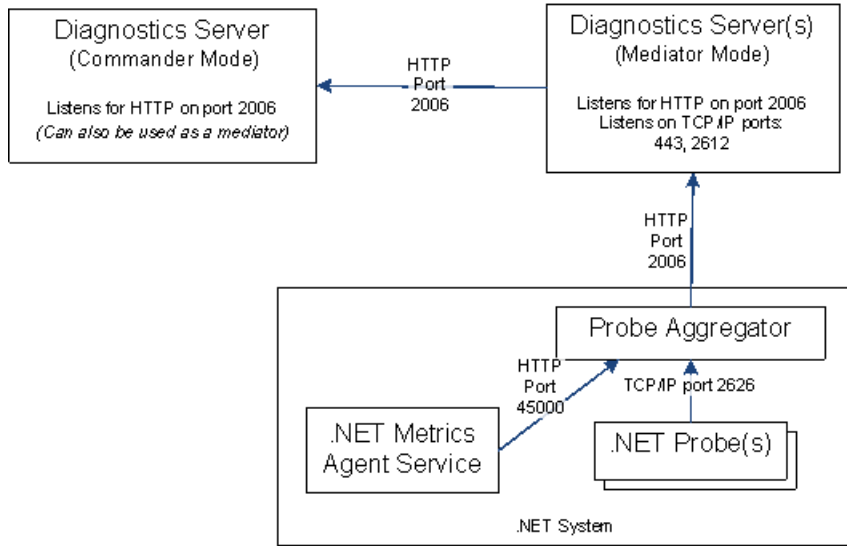
# Communications with Business Service Management



## 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 and .NET Agent instructions apply when upgrading or installing a patch release of a Diagnostics Agent AND a TransactionVision Agent.

---

**This chapter includes:**

- ▶ Before You Begin on page 894
- ▶ Diagnostics Compatibility with Earlier Diagnostics Versions on page 894
- ▶ Upgrade or Patch Install Instructions for Diagnostics Components on page 894
- ▶ Diagnostics Compatibility with Other HP Software Products on page 907

## Before You Begin

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 Diagnostics Release Notes 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 and refer to the upgrade documentation for these products for important instructions relevant to the Diagnostics integration.

## Diagnostics Compatibility with Earlier Diagnostics Versions

The Diagnostics Server is supported to work with the following earlier agent and collector versions:

- ▶ Java Agent 8.x, 9.0x, 9.1x
- ▶ .NET Agent 8.x, 9.0x, 9.1x
- ▶ Collectors 8.x, 9.0x, 9.1x

## 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 895
- ▶ “Java Agent” on page 899

- “.NET Agent” on page 903
- “Diagnostics Collector” on page 904

## Diagnostics Server

This section contains instructions for upgrading your Diagnostics Server from an earlier version. 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.

---

During the installation the keystore is *overwritten* along with the JRE. As a result your trusted certificates will be unavailable after the upgrade.

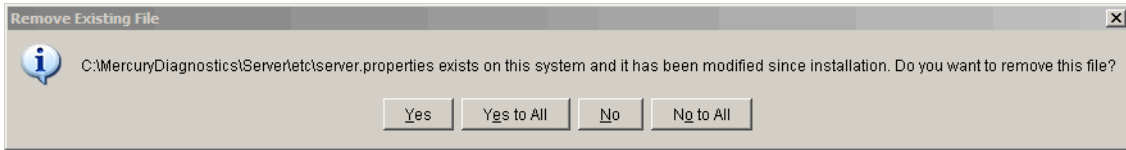
With each new release of Diagnostics you should *re-record* the Diagnostics Server silent install response files prior to performing silent installation on multiple machines.

### To upgrade a Diagnostics Server:

- 1** Shut down the current Diagnostics Server.
- 2** Make a **backup copy** of the current Diagnostics Server directory. By default this is C:\MercuryDiagnostics\Server on Windows and /opt/MercuryDiagnostics/Server on UNIX although a different directory 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:

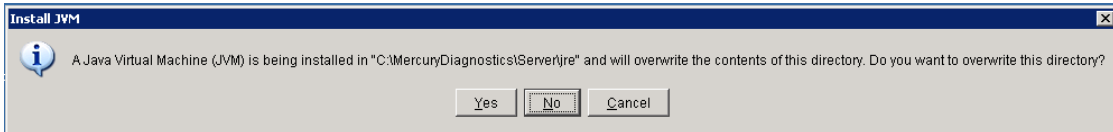


For information about uninstalling and removing the Diagnostics Server, completely see the *HP Diagnostics Installation and Configuration Guide*.

During the upgrade the existing **etc** directory is renamed to **etc.old\_<timestamp>**.

- 4 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:



- 5 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.

- 6 Compare the **etc** directory and the **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.



---

**Important:** When upgrading from a version prior to 9.20, you may have to modify the following two properties whether or not you previously customized them:

- If a copy of *server.properties* exists in `etc.old_<timestamp>`, copy the **thresholding.evaluation.status.red.for.availability** property and its value from the old file to the new file.
- If a copy of *thresholds.configuration* exists in `etc.old_<timestamp>`, copy the **com.mercury.diagnostics.common.data.graph.node.ProbeData.Availability** property and its value from the old file to the new file; otherwise, set the value in the new file to "-95" instead of ",-95".

These changes preserve the behavior of previously set Availability thresholds. If you missed these changes during the initial upgrade, you may make them later.

---

Apply any differences that were caused by the customizations that were made (found in the `etc.old_<timestamp>` 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
<b>alerting.properties</b>	SNMP and SMTP servers, mail addresses.
<b>security.properties</b>	If the system is set up for SSL mode, all parameters should be updated and certificates manually copied to the new <code>/etc</code> folder.
<b>server.properties</b>	Timeout/Trimming settings, Commander's URL. See the Important note above regarding the <code>thresholding.evaluation.status.red.for.availability</code> property.
<b>thresholds.configuration</b>	Copy any customizations that were made. See the Important note above regarding the <code>com.mercury.diagnostics.common.data.graph.node.ProbeData.Availability</code> property.

Property File	Configuration Properties To Be Copied to the New Diagnostics Server
<b>webservice.properties</b>	Default port information.
<b>.htaccess</b>	The .htaccess file is for security and it needs to be copied to the new /etc folder to retain your original settings for user roles.

- 7** If the system is integrated with LoadRunner or Performance Center, copy **run\_id.xml** from the **etc.old\_<timestamp>** directory to the new **etc** directory to ensure that the Run ID is properly incremented for future runs.
- 8** If you are upgrading the Diagnostics command server, copy the **DiagnosticsLicFile.txt** or **DiagnosticsServer.lic** file from the **etc.old<timestamp>** directory to the new **etc** directory.
- 9** Start the Diagnostics Server.
- 10** Clear your browser's cache and restart the browser before you attempt to access the Diagnostics UI.
- 11** You can verify that the upgraded Diagnostics Server is running by checking the version in the System Health view in the Diagnostics UI. The version should be the latest version if the upgrade was successful and the Diagnostics Server was restarted. To access the System Health view you must open the Diagnostics UI as the Mercury System customer from [http://<Diagnostics\\_Commanding\\_Server\\_Name>:2006/query/](http://<Diagnostics_Commanding_Server_Name>:2006/query/). Then in the Views pane you can select the System Views view group.

- 12** If Diagnostics is integrated with **Business Service Management** you need to do the following:
  - a** Check the Diagnostics Readme for any installation notes when integrating Diagnostics with BSM.
  - b** After a Diagnostics upgrade of the Commander Server, you must copy over the **RegistrarPersistence.xml** file from the **etc.old\_<timestamp>** folder to the new **etc** folder. Then check the Diagnostics Integration in the BSM -> Admin -> Diagnostics page and re-do the registration of Diagnostics server in BSM if it is not working properly. See the *HP Diagnostics Installation and Configuration Guide* chapter on "Setting Up the Integration Between Business Service Management and Diagnostics".
- 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 a newer version, 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. 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.

---

---

**Caution:** With each new release of Diagnostics you should re-record the Java agent silent install response files prior to performing silent installation on multiple machines.

---

**To upgrade a Java Agent:**

---

**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 do the following. This ensures that the persisted data for your application will match up with the metrics captured by the new agent.

- ▶ configure the Java Agent to work with a Diagnostics Server or as a standalone Diagnostics Profiler. The Java Agent can also be configured to work with a TransactionVision Server 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
- 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/JASMUutilities/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\JASMUutilities\Sna
pins>runPropertyScanner -console -diffOnly yes -Source1
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\etc -Source2
C:\MercuryDiagnostics\JavaAgent8\DiagnosticsAgent\etc
```

Sample Output:

```
***** Property dispatcher.properties:stack.trace.method.calls.max
PropertyFile=dispatcher.properties
Property=stack.trace.method.calls.max
Source1=
Source2=1000
```

Apply any differences that were caused by the customizations that you made to the previous 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
<code>auto_detect.points</code>	Copy custom points that you have created and points that you have modified from the <code>auto_detect.points</code> file in the old <code>etc</code> directory to the new <code>etc</code> directory. Be sure to check the points for RMI, LWMD, <code>args_by_class</code> when looking for points you may have modified.
<code>capture.properties</code>	Depth and latency trimming.

Property File	Configuration Properties To Be Copied to the New Diagnostics Agent
<b>inst.properties</b>	define.pre.process
<b>dispatcher.properties</b>	minimum.sql.latency sql.parsing.mode
<b>dynamic.properties</b>	cpu.timestamp.collection.method
<b>metrics.config</b>	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.
<b>security.properties</b>	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** Prepare your application servers to be monitored using the JRE instrumentation methods described in the *HP Diagnostics Installation and Configuration Guide* chapter on "Preparing Application Servers for Monitoring with the Java Agent". In particular you need to update the application's startup script or JVM parameters to point to the upgraded agent installation. The parameters include 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 Diagnostics Agent is running by checking the version in the System Health view in the Diagnostics UI. The version should be the latest version if the upgrade was successful. To access the System Health view you must open the Diagnostics UI as the Mercury System customer from [http://<Diagnostics\\_Commanding\\_Server\\_Name>:2006/query/](http://<Diagnostics_Commanding_Server_Name>:2006/query/). Then in the Views pane you can select the System Views view group.
- 9** When all your applications have been migrated over to be the latest version 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. 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.

---

### **To upgrade a .NET Agent:**

- 1** Install the new Diagnostics Agent for .NET (select Upgrade).

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 Diagnostics Agent is running by checking the version in the System Health view in the Diagnostics UI. The version should be the latest version if the upgrade was successful. To access the System Health view you must open the Diagnostics UI as the Mercury System customer from [http://<Diagnostics\\_Commanding\\_Server\\_Name>:2006/query/](http://<Diagnostics_Commanding_Server_Name>:2006/query/). Then in the Views pane you can select the System Views view group.

## Diagnostics Collector

This section contains instructions for upgrading your Diagnostics collector from an earlier version. 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.

---

---

**Important:** With each new release of Diagnostics you should re-record the Diagnostics Collector silent install response files prior to performing silent installation on multiple machines.

---

### To upgrade a Diagnostics Collector:

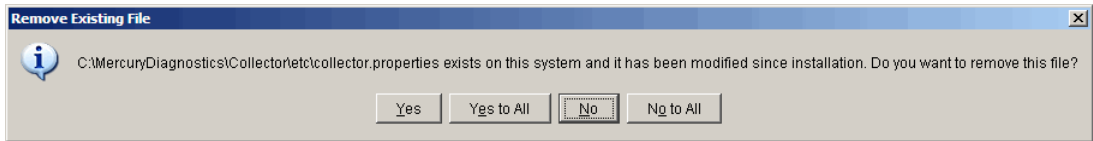
- 1 Stop or shut down the Diagnostics Collector that you want to upgrade.
- 2 Back-up the directory for the current collector installation.

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 the *HP Diagnostics Installation and Configuration Guide*.

During the upgrade the existing **etc** directory is renamed to **etc.old\_<timestamp>**.

- 4** 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.

- 5** If installing on Windows, stop the Collector.

The Collector is started automatically when the installer finishes.

On UNIX the Collector is not automatically started so you do not need to stop it.

- 6** Compare the new **etc** directory and the **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 (found in the **etc.old\_<timestamp>**) 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
<b>mq-config.xml</b>	If the collector is monitoring an MQ environment.
<b>oracle-config.xml</b>	If the collector is monitoring an Oracle environment.
<b>sqlserver-config.xml</b>	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.
<b>r3config.xml</b>	If the collector is monitoring an SAP ABAP environment.
<b>vmware-config.xml</b>	If the collector is monitoring a VMware environment.
<b>tibco-ems-config.xml</b>	If the collector is monitoring a TIBCO EMS environment.
<b>wm-broker-config.xml</b>	If the collector is monitoring a webMethods Broker environment.
<b>security.properties</b>	If the system is set up for SSL mode, set all properties and copy the certificates from the old property file to the property file.

- 7** Start the Diagnostics Collector.
- 8** You can verify the upgraded Collector by checking the version.
- 9** Once you are satisfied that the Diagnostic Collector has been upgraded successfully, remove the backup copy you created in Step 2.

## **Diagnostics Compatibility with Other HP Software Products**

For the most recent information on version compatibility, see the Diagnostics Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](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 910
- ▶ Diagnostics Installers Do Not Work on Some 64-bit Linux Systems on page 910
- ▶ Error During Linux Install - Missing libstdc++.so.5 Shared Library on page 911
- ▶ Java Agent Fails to Operate Properly on page 911
- ▶ Error During WAS Startup with Diagnostics Profiler for Java on page 912
- ▶ Missing Server-Side Transactions on page 913
- ▶ Event Capture Buffer Full Warning on page 913
- ▶ WebSphere Application Server Startup Issue on page 914
- ▶ Java Agent Support Collector on page 915
- ▶ Event Based Health Indicator Status Troubleshooting Flow on page 915
- ▶ OM Agent Troubleshooting on page 919
- ▶ Troubleshooting Registration of OMi Between the BSM Gateway Server and Data Processing Server on page 922

## 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`.

## Diagnostics Installers Do Not Work on Some 64-bit Linux Systems

On some 64-bit Linux systems (for example, RedHat Enterprise Linux 6, CentOS 6, Ubuntu 11), the Diagnostics Installers (.bin files) may not work because the 32-bit glibc library is not installed. To work around this issue, you need to install the 32-bit glibc library.

To install the 32-bit glibc package in RHEL 6/CentOS 6, run this command:  
**yum install glibc.i686.**

To install the 32-bit glibc package in Ubuntu 11, run this command:  
**sudo apt-get install libc6:i386.**

Then try to run the install again.

## Error During Linux Install - Missing libstdc++.so.5 Shared Library

On some Linux systems, when you install the Diagnostics Server or Collector, if you encounter an error saying that the libstdc++.so.5 shared library is missing, you may need to install it. For example, on CentOS, enter the following command to install the library:

```
yum install compat-libstdc++-33
```

## 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, make sure you have instrumented the JRE as this is what creates it. See Chapter 6, “Preparing Application Servers for Monitoring with the Java Agent.”

## 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 483.

**"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 BSM 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.

## WebSphere Application Server Startup Issue

**Symptoms:**

With the Java probe enabled, the WebSphere application server throws exceptions such as "java.lang.NoClassDefFoundError: javax.xml.rpc.handler.Handler" during application startup.

**Reason:**

The SOAP Handler cannot be loaded in this configuration.

**Solution:**

Turn off the SOAP Handler (impacts SOAP consumer ID and payload capture) by changing the property below to false in the probe's `etc\inst.properties` file. Then restart the application server.

```
details.conditional.properties=\  
mercury.enable.SOAPHandler=false  
mercury.enable.autoLoadSOAPHandler= false, \  

```

OR

Add the missing classes to the application server's class path so they can be accessed by the probe's SOAP Handler.

For example from `<WebSphere>/lib/j2ee.jar` there are several jars that contain the missing classes.

## 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\_agent\_install\_dir>\config directory
- ▶ Files from the <TransactionVision\_agent\_install\_dir>\logs directory
- ▶ Property Scanner report, which compares two agent directories and reports differences between property files, points files, and XML files (TransactionVision Agents 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.

Note that the utility is also available in the <probe\_install\_dir>\bin directory

- 2** Execute `.\runSupportSnapshot.cmd -console` on Windows, or `./runSupportSnapshot.sh -console` on UNIX or Linux.
- 3** A .zip file is created. The default location of the saved zip file is the .../DiagnosticsAgent/ folder.

## Event Based Health Indicator Status Troubleshooting Flow

When integrated with Business Service Management 9.0 or later, Diagnostics sends Health Indicator status events to the Business Service Management gateway server.

---

**Important:** For communications between BSM gateway server and BSM processing server with an event channel integration there must be a trust relationship between the machines. See “Configuration of Separate BSM Servers for DPS and Gateway” on page 761 if you need to set this up.

---

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|17b87476ac6de3938ff9898cd19c8bd8|mercury|Default Client|1|J2EE|PROBE|2010-05-17 13:40:51|latency [BpmTxJpalmpl.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
```

For additional troubleshooting steps go to OM Agent Troubleshooting in the following section. And you can also check “Configuration of Separate BSM Servers for DPS and Gateway” on page 761.

## OM Agent Troubleshooting

When integrating Diagnostics with Business Service Management 9.0 or later, 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.

- ▶ **OM Agent Installation Verified.** Make sure the OM Agent components were installed and if there are problems with the installation, errors are reported in the <Diagnostics\_install\_dir>/server/log.txt file.
- ▶ **Root Access Requirement.** Root access is required for the installation of the OM Agent and IAPA component. If you need to install the Diagnostics Server without root access you can chose to not install these two components and install them later manually. When you see the dialog box: OM Agent and IAPA component installations leave the box unchecked and install later (see “Manual Installation of OM Agent and IAPA Components” on page 74).
- ▶ **Grant Certificates on Gateway Server.** 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 or later. The required certificates are requested during the registration and then you perform an additional step to grant the certificate on the gateway server. (See “Registering the Diagnostics Server in Business Service Management” on page 740 in the section "To manually grant the certificate".)

**To check that the Diagnostics Commander Server can ping the Business Service Management Gateway Server:**

- 1 Execute the following on the Diagnostics Commander Server:

```
bbcutil -ping <gateway server hostname>
```

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 Server is used by another OM server):

```
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.

## Troubleshooting Registration of OMi Between the BSM Gateway Server and Data Processing Server

Complete the following steps to register OMi between the BSM Gateway Server and Data Processing Server on separate systems in order to get Diagnostics 9.x events to show up in BSM.

### To troubleshoot:

- 1 First check the certificate on the BSM Gateway Server. From a command prompt on the Gateway Server, run the command: **ovcert --check**.

If there are problems, you'll see an error like the following:

```
C:\Documents and Settings\Admin>ovcert -check
OvCoreId set           : OK
Private key installed  : FAILED
Certificate installed  : FAILED
Certificate valid      : FAILED
Trusted certificates installed : FAILED
Trusted certificates valid : FAILED
```

Otherwise, you'll see a message like the following:

```
C:\Documents and Settings\Admin>ovcert -check
OvCoreId set           : OK
Private key installed  : OK
Certificate installed  : OK
Certificate valid      : OK
Trusted certificates installed : OK
Trusted certificates valid : OK
Check succeeded.
```

- 2 Check if the OMi BSM Gateway Server is registered properly with the BSM Data Processing Server. From a command prompt on the BSM Gateway Server run the command: **bbcutil -ping <Data Processing Server>**.

If there are problems, you will get an error like the following:

```
(bbc-289) status=eServiceError coreID= bbcV= appN= appV=conn=0 time=109
ms
```

Otherwise, you will see a message like the following:

```
status=eServiceOK coreID=09139942-991a-7549-1eae-ee2cbe62289a
bbcV=11.00.044 appN=ovbbccb appV=11.00.044 conn=2 time=156 ms
```

- 3 Check if OMi BSM Data Processing Server is registered properly with the BSM Gateway Server. From a command prompt on the BSM Data Processing Server run the command: **bbcutil -ping <Gateway Server>**.

If there are problems, you'll see an error like the following:

```
(bbc-288) status=eServiceError coreID= bbcV= appN= appV=
conn=0 time=109 ms
```

Otherwise, you'll see a message like the following:

```
status=eServiceOK
coreID=c3475f92-a584-7546-1cfb-b2612a96538f
bbcV=11.00.044 appN=ovbbccb appV=11.00.044
conn=2 time=94 ms
```

- 4 If you have any of the problem conditions from above, use the following steps to set up certificates on separate BSM Gateway Server and Data Processing Server (As described in the BSM Deployment Guide).

- a From a command prompt on the BSM Gateway Server run the following commands:

```
ovconfchg -ns sec.cm.client -set CERTIFICATE_SERVER <Data Processing
Server>
```

```
ovcert -certreq
```

```
INFO: Certificate request has been successfully triggered
```

- b From a command prompt on the BSM Data Processing Server run the following command:

```
ovcm -listpending -l
```

```
RequestID: e0609222-7ea6-754d-0a03-e1a09dc5dd43
Context:
CN: c3475f92-a584-7546-1cfb-b2612a96538f
Nodename: bsavm9.ovrtest.adapps.hp.com
IPAddress: 16.93.25.199
PeerAddress: 16.93.25.199
```

Platform: Windows 5.2, CPU: x64  
InstallType: Manual  
TimeReceived: 1/19/2011 1:49:18 PM Pacific Standard Time

- c Grant the RequestID.

**ovcm -grant e0609222-7ea6-754d-0a03-e1a09dc5dd43**

- d From a command prompt on the BSM Gateway Server run the command:

**ovcert --list**

You should see Certificates and Trusted Certificates as in the example below:

```
+-----+
| Keystore Content |
+-----+
| Certificates: |
| c3475f92-a584-7546-1cfb-b2612a96538f (*) |
+-----+
| Trusted Certificates: |
| CA_09139942-991a-7549-1eae-ee2cbe62289a |
+-----+
```

- e From a command prompt on the BSM Gateway Server run the command:

**bbcutil -ping <Data Processing Server>**

Ping should now work.

status=eServiceOK coreID=09139942-991a-7549-1eae-ee2cbe62289a

bbcV=11.00.044 appN=ovbbccb appV=11.00.044 conn=2 time=156 ms

- 5 Should be working now.

# General Reference Information

---

This section includes general reference topics.

**This chapter includes:**

- Using UNIX Commands on page 925
- Using Regular Expressions on page 926
- Multi-Lingual User Interface Support on page 934

## 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 929).

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 931. For more information on the | regular expression character, see "Matching One of Several Regular Expressions" on page 932.

## 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 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).

---

**Caution:** Diagnostics does not support localization of agent names.

---

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.



# J

---

## Data Exporting

The metric data collected by Diagnostics can be archived directly to a third-party database where it can be retained 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 867.

### **This chapter includes:**

- ▶ Task 1: Prepare the target database on page 938
- ▶ Task 2: Determine which metrics you want to export on page 939
- ▶ Task 3: Determine the frequency and the recovery period on page 942
- ▶ Task 4: Modify the data export configuration file on page 943
- ▶ Task 5: Monitor the data export operation on page 947
- ▶ Task 6: Verify the results on page 949
- ▶ Task 7: Select the data from the target database on page 950
- ▶ Sample Queries on page 950

## Task 1: Prepare the target database

The target database for the exported data can be an SQL Server or Oracle database to which the Diagnostics commanding server has access. For the most recent information on supported environments, see the Diagnostics Support Matrix at [http://support.openview.hp.com/sc/support\\_matrices.jsp](http://support.openview.hp.com/sc/support_matrices.jsp).

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.

---

**Note.** If you are upgrading to Diagnostics 9.10 or later but you want to keep the old pre-9.10 database content you can alter the database to gain the following new functionality: In 9.10 or later the min and max values use doubles instead of integers allowing exported data to show decimal places. Alter the database after upgrading as follows:

Oracle:

```
ALTER TABLE RECORD MODIFY (
  REC_COUNT NUMBER(38),
  TOTAL     FLOAT,
  MINIMUM   FLOAT,
  MAXIMUM   FLOAT)
```

SQL Server:

```
ALTER TABLE RECORD ALTER COLUMN REC_COUNT  DECIMAL(19)
ALTER TABLE RECORD ALTER COLUMN TOTAL     FLOAT
ALTER TABLE RECORD ALTER COLUMN MINIMUM   FLOAT
ALTER TABLE RECORD ALTER COLUMN MAXIMUM   FLOAT
```

## Task 2: Determine which metrics you want to export

There are different ways to control which metrics are exported. You can specify to get all metrics for a particular entity type. You can exclude metrics from that grouping or you can specify to include only specific metrics. For more information on the Diagnostics data model see the *Diagnostics Data Model and Query API* document available on the DVD and in the help.

---

**Important:** The data export operation exports metric data only, that is counts, latencies, and averages. No instance data or status data is exported. Also you cannot export call profile data.

---

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 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.

The following tables give examples of types of metrics and the categories they belong to:

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

Category	Metric
Latency	latency total_cpu exception_count timeout_count throughput

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 configure the export processing of the mediators in batches to reduce the load on the commanding server (**servers-per-query** attribute set in the **etc/data-export-config.xml** file 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 Commander Server.

**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-uri>jdbc:oracle:thin:@testapps.hp.com:1521:ORCL</connection-uri>
  <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 125.

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. Spaces are not allowed in the id value.

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, you can use the servers-per-query attribute on a query as a way to reduce the load on the server.

servers-per-query= specifies processing the export query in batches.

```
<query id="Probes" frequency="1h" recovery-periods="2" servers-per-query="10">
  <entity-path>/probegroup/probe</entity-path>
</query>
```

For example, if there are 30 mediator servers and you set `servers-per-query=10`, then the export gets results for 10 mediators, exports these results and then processes the next 10 mediators and so on.

This attribute should only be set when using more than 10 mediators.

- 8 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 926.

- 9 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"/>
```

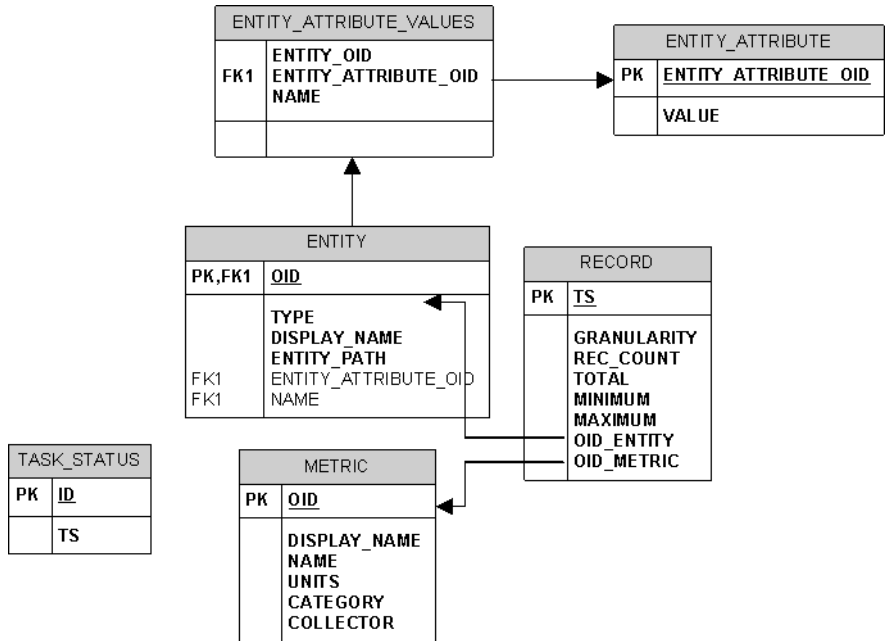
- 10 Save the changes to the `data-export-config.xml` file.

## 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 [MedRecE...	JDBC [MedRecEAR@MedRecAp...	COUNT	JDBC [MedRec
3aa2764fb92b5...	EJB Response Time	EJB Response Time	MILLISECONDS	EJB
3aa4caf58987c7...	IOBusyTime	IOBusyTime	MILLISECONDS	SqlServer
3ad15a083c484...	User Commits Pe...	User Commits Percentage	PERCENT	Oracle
3ae792e1509c6...	Servlets Respon...	Servlets Response Time	MILLISECONDS	Web Applicati
3b49b2d59252d...	JDBC [MedRecGl...	JDBC [MedRecGlobalDataSourc...	MILLISECONDS	JDBC [MedRec
3b769a0780452...	Physical Writes ...	Physical Writes Direct Per Txn	COUNT	Oracle
3be4dbe933822...	JDBC [MedRecP...	JDBC [MedRecPool-PointBase] ...	COUNT	JDBC [MedRec
3c0afe8166fcbc...	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 282

.NET agent  
  .NET framework requirements 252  
  add an HP software product to  
    configuration 637  
  advanced configuration 627  
  collect additional metrics 666  
  configure for outgoing HTTPS 857  
  enabling and disabling 288  
  HTTP proxy 674  
  installing 251  
  launching the installers 254  
  performance counters 666  
  probe\_config file elements 551  
  system metrics collector 691  
  uninstall 294  
  uninstalling 294  
  upgrade 903

.NET agent installer  
  how it works 252

.NET agent version information 288

.NET configuration file 551

.NET instrumentation 427

.NET layers 462

.NET points files 429

## A

active 347

active users  
  list of 791, 819

active.products property 505

AD license 81, 86

AD mode  
  java agent 143

add-in  
  LoadRunner Diagnostics 765

admin alert message configuration 63

adonet.points 429

advanced options  
  show or hide 792

after  
  code 344

agent  
  .NET system requirements 38, 39  
  Java 133, 295

agent administration UI 515

agent name not localizable 47

agent, .NET, *See* .NET agent

agent, Java, *See* Java agent

agent.Java properties file 338

altert properties 793

AM license 81, 85

AM product mode 507

AM/Enterprise mode  
  java agent 142

appdomain element 552

application level permissions 806

application name  
  configure show in server request 538

application server  
  multiple JVM instances 233

application server configuration  
  generic 219  
  Oracle 224  
  SAP NetWeaver 178

application server startup script  
  generic 231

application servers  
  supported 30

application servers, supported 30

ApplicationPoolIdentity 666

## Index

- args 344
- argument capture 434, 440
- arguments
  - mandatory for .NET 431
  - mandatory for Java 341
  - optional for .NET 432
  - optional for Java 342
- ASP.NET applications
  - automatic configuration 283
  - discovering 282
- aspnet.points 429
- asynchronous thread sampling 540
- authentication element 553
- authorization and authentication 798
- auto\_detect.points file 338
- automatic explicit mode 221
- automatic implicit mode 224
- automatic jsp instrumentation 247
- Automatic Method Trimming
  - controlling 508
- Azure Cloud 287
- B**
- backslash (\) 928
- backup 883
- backup directories 886
- backup symbol table
  - configuring 886
- before
  - code 344
- BizTalk 263
- BSM performance graphing 756
- bufferpool element 554
- Business Availability Center
  - assigning permissions for Diagnostics users 748
  - Diagnostics configuration 747
- Business Service Management
  - Admin>Diagnostics 747
  - changing the Diagnostics Server details 747
  - configure HTTPS communications 860
  - Diagnostics downloads 748
  - Diagnostics registration 747
  - samples queue size 495
  - setup to use Diagnostics 737
  - specifying Diagnostics Server details 740
- Business Service Management server HTTPS communication 860
- C**
- caller 344
- capture points
  - .NET 428
- capture points files
  - mandatory point entries 341, 431
  - optional point entries 342, 432
  - using for instrumentation 338, 428
- captureexceptions element 555
- change
  - privilege level 799
- CI population
  - discovery of IIS metadata 284
- cipher suites 840
- CIs
  - Diagnostics populates in BSM 283, 753
  - discover new J2EE servers 241
  - synchronize with BSM 791
- class 341, 431
- class map capture 364
- ClassLoader class, recreating 911
- client monitoring
  - disable 249
  - instrumentation 247
- cloud
  - monitoring applications in the 287
- CLP 377
- code snippet helper 354
- code snippets 348
  - secure code-key 362
- code-key
  - generating 362
- collect CPU timestamps 545
- collector
  - configure for incoming HTTPS 850
  - configure for outgoing HTTPS 857
  - configuring active system property

- files 106
  - installing on Windows 94
  - starting and stopping 128
  - supported platforms 92
  - uninstalling 130
  - upgrade 904
  - verifying installation 103, 127
  - version information 130
  - communication diagrams 889
  - compatibility with other HP Software 907
  - component and communication diagrams 889
  - component communications 793
  - Components page 790
  - compressed files 876
  - configure Diagnostics 787
  - consumer ID
    - look deeper in xml to find 530
  - consumer IDs
    - configuring 524
  - consumeridrules element 556
  - CORBA instrumentation 387
  - CPU 521
  - CPU time metrics 521
    - configuring 521
  - CPU timestamp 545
  - cpu.timestamp.collection.method 522
  - cputime element 557
  - credentials element 558
  - cross VM
    - RMI instrumentation 387
  - custom context root 497
  - custom dashboard 791
  - custom data 868
  - custom sub-layer instrumentation 366
  - custom\_code.properties 350
  - customer information 793
- D**
- data compression 876
  - Data exporting
    - about 937
    - configuration file 943
    - frequency 942
    - recovery period 942
  - sample scripts 950
  - supported metrics 939
    - target database 938, 943
  - data management 867
  - data management, *See* Diagnostics data management
  - data port 265
  - data retention 876
  - database name
    - automatically discovered 114
  - data-export-config.xml file 943
  - days data 873
  - deep\_mode 343, 369, 433
  - deployed into
    - show for server request 538
  - depth element 561
  - depth trimming 509, 646
  - detail 344
  - Diagnostics
    - integration with LoadRunner 769
    - integration with Performance Center 779
    - integration with Business Service Management 737
  - Diagnostics components
    - description 28
    - host requirements 31
    - synchronizing time between 466
  - Diagnostics data management
    - backing up data 883, 888
    - custom screen data 868
    - data sizes & data retention 876
    - performance considerations 882
    - performance history data 870, 874
  - Diagnostics mediator
    - firewall configuration 681
  - Diagnostics Server
    - adjusting heap size 471
    - administration 787
    - advanced configuration 465
    - changing default port 476
    - configuration pages 492
    - configure for incoming HTTPS 844
    - configure for outgoing HTTPS 853
    - configuring 72
    - configuring for large installation 470

- configuring for multi-homed environments 479
- configuring LoadRunner offline file 492
- configuring time synchronization 468
- configuring, advanced 465
- high availability 488
- HTTP proxy 672
- information required for installation 40
- installing 54
- jetty.xml file, modifying 480
- jetty.xml file, sample 482
- launching the installers 55
- optimized to handle more probes 496
- override assignment for a probe 491
- overriding default host name 476
- reducing memory usage 483
- reducing memory use 483
- running the installation 59
- setting event host name 479
- setup in BSM 740
- starting and stopping 70
- synchronizing time between
  - Diagnostics components 466
- system requirements 47
- UI 787
- uninstalling 73
- upgrade 895
- verifying installation 67
- Diagnostics Server host name
  - overriding 476
- Diagnostics Server installation 54
- Diagnostics Server port
  - changing default 476
- Diagnostics Server version information 72
- Diagnostics UI 788
- diagnosticsserver element 562
- discovery
  - new J2EE servers 241
- disk space exhausted 876
- documentation updates 23

## E

- embedded java probe and HTTPS 856

- enable.stack.trace.sampling 540
- encrypted password 125
- encryption cipher suites
  - filtering 840
- EncryptPassword.jsp 125
- enterprise level permissions 806
- eve files 777
- event host, setting name for 479
- exception data
  - limiting 652
- exception tree data
  - limiting 513
- exceptiontype element 564
- exclude element
  - parent is captureexceptions 565
  - parent is lwmd 566
- excludeassembly element 567
- execute
  - privilege level 799
- explicit mode 221
- exporting data 937

## F

- files 791
- files page
  - accessing 517
- filter element 568
- firewall
  - enabling communications through 675
- fragment name based trimming 484

## G

- gentvhttpeventforwcf element 570
- granularity 872
- GROUPBY for JMX metrics 732

## H

- heap size 471
  - adjusting for large installation 471
  - adjusting in startup script 237
- high availability 488
- host requirements, Diagnostics components

- hours data 873
  - HP Software Support Web site 22
  - HP Software Web site 22
  - HTTP proxy communication
    - .NET Probe 674
    - Diagnostics Server in Mediator Mode 672
    - enabling 671
    - Java Probe 673
  - httpclient element 569
  - httpd.conf 834
  - httpheaderrule element 571
  - httpheaderrules element 572
  - HTTPS
    - configuration steps 841
    - configure for incoming communication 843
    - configure for outgoing communication 853
  - HTTPS communication
    - enabling for Business Service Management server 860
  - HTTPS communications
    - enabling between components 840
- I**
- IAPA 61
  - id element 573
  - ignore\_cl 342, 432
  - ignore\_method 342, 433
  - ignore\_tree 343
  - ignoreScope 343, 433
  - IIS Host Header 657
  - IIS worker process crash in VMWare 622
  - iis\_discovery\_data.xml 284
  - implicit mode 224
  - include element
    - parent is captureexceptions 575
    - parent is lwmd 576
  - installation
    - .NET agent 251
    - collector 92
    - Diagnostics Server 53
    - Diagnostics Server (Windows) 54
    - gathering information for 40
    - interruption during 910
    - Java agent 133
    - order of 48
    - planning 40
    - recommended order 48
  - installation requirements
    - .NET agent 38, 39
    - .NET Diagnostics Profiler 39
    - Diagnostics Server 32
    - Java agent 36
    - Java Diagnostics Profiler 37
  - instance tree files 875
  - instant on license 80
  - instrumentation
    - deep\_mode examples 446
  - instrumentation
    - .NET applications 427
    - .NET remoting 451
    - access filter 374
    - advanced 383
    - allocation analysis 375
    - always trim 379
    - argument capture 371, 440
    - attributes in instance trees 374
    - caller side 375, 438
    - capture for trended methods view 367, 437
    - capture with controlled scope 369
    - collection leak pinpointing 377
    - correlation across multiple threads 389
    - CPU time collection 380
    - custom layers 366, 436
    - deallocation 375
    - deep\_mode hard and soft 369
    - direct recursion 375
    - edit points 415
    - edit points from the profiler 412
    - enable at runtime 379
    - enabling for .NET 290
    - fragment local storage 392
    - ignore specific methods 366, 436
    - Java applications 337
    - LWMD 376
    - never trim 379
    - non-ASP.NET applications 447

- object lifecycle 377
- printing 380
- RMI 387
- RootRename 373
- thread local storage 388
- TransactionVision related 346
- URI aggregation 386
- using annotations 396
- using wildcards 366, 436
- view current 413
- web services 372
- instrumentation element 577
- instrumentation examples
  - .NET 435
  - Java 365
- instrumentation overhead 381, 461
- instrumentation page
  - accessing 517
- iprule element 578
- iprules element 579

**J**

- JAAS authentication 820
- Java agent
  - add an HP software product to
    - configuration 505
  - advanced configuration 499
  - automatic method trimming 508
  - configure for incoming HTTPS 847
  - configure for outgoing HTTPS 855
  - configuring and installing, about 134
  - configuring for multiple JVMs 233
  - configuring for proxy server 511
  - controlling log messages 502
  - edit probe settings from the profiler
    - 539
  - failed operation 911
  - HTTP proxy 673
  - installing 133
  - installing using generic installer 156
  - JMX metric collector 723
  - JMX metrics collector 723
  - launching windows installer 135
  - log messages 502
  - metrics collectors 703

- proxy server 511
- reverse HTTP for agent in SaaS 512
- silent installation 157
- system metrics 711
- system requirements 36
- uninstalling 160
- Unix install 134
- upgrade 899
- upgrade steps 900
- Java agent installer
  - how it works 134
- Java agent windows install 137
- Java Diagnostics Profiler
  - disabling 501
  - WAS startup error 912, 913
- Java instrumentation 337
  - code snippets 348
- Java layers 424
- Java Profiler
  - PRO product mode 506, 507
- Java profiler
  - configuration tab 412, 539
- Java system properties 153
- JDK/JRE executable 149
- jetty.xml 480
- jetty.xml file
  - modifying 480
  - sample 482
- JMS temporary queues
  - grouping 537
- JMX metrics
  - accessing 723
  - add custom 725
  - collecting 725
  - custom 728
  - GROUPBY 732
  - understanding patterns 731
- JMX metrics collector 723
  - configuring 724
- JRE instrumenter
  - how it works 226
- JRE instrumenter options to invoke 219

**K**

- keyword 342, 432

**L**

- large deployments
  - data management 882
- latency element 580
- latency trimming 508, 641
- layer 341, 431
- layer\_type 435
- layers
  - .NET 462
  - .NET layers 462
  - about instrumentation 423, 462
  - Java 423
  - Portals 425
  - view current 414
- layers page
  - instrumentation control and edit 382
- layers tree 416
- layerType 347
- LDAP authentication 822
- license 791
- light-weight memory Diagnostics (LWMD) 649
- Linux custom metrics 719
- licensing 80
- LoadRunner
  - Diagnostics probe metrics 774
  - Diagnostics, configuring scenarios to use 774
  - Diagnostics, setting up 773
  - firewall configuration 687
  - installing add-in for Diagnostics 765
- LoadRunner offline analysis
  - reducing file size 493
- LoadRunner Offline File
  - advanced Diagnostics Server configuration 493
  - estimating size of 493
  - reducing size of 493
- LoadRunner scenarios
  - configure Diagnostics parameters 774
- log messages 502
- logdirmgr element 581
- logging 791, 793
  - controlling 502
  - disable .NET agent 289
  - disabling 655

- logging element
  - parent is appdomain probeconfig process 584
  - parent is instrumentation 582
- LR/PC runs
  - assign Diagnostics server for a probe 491
- lwmd element 586
- LWMD *See* lightweight memory Diagnostics (LWMD)

**M**

- major time periods 872
- manual mode 226
- max.search.level.depth 530
- mediator element 587
- memory diagnostics 793
- memory usage, reducing 483
- message handler 237
- method 341, 431
- method\_access\_filter 343
- metric collector default port 714
- metric element 590
- metric patterns 731
- metric port 266
- metrics
  - add custom JMX 725
  - add custom system 696, 715
  - list available 705
  - list available JMX 725
- metrics agent 691
- metrics collector
  - modifying default port 714, 715
  - system metrics 713
- metrics element 589
- metrics entries 694, 706, 728
- metrics.config
  - .Net 693
  - Java 703, 713
  - keywords 699
- MI listener 680
- migrate servers 477
- minor time periods 872
- modes
  - .NET agent 638

## Index

- Java agent 505, 637
- modes element 592
- months data 873
- MQ probe
  - configuring 117
  - permissions required 117
- multi-homed environments 479
- multiple JVM instances 233
- MyBSM authentication issue 763

## N

- nanny
  - start and stop using 70
- negative latency with VMWare guest 622
- NET Diagnostics Profiler
  - enabling 641
- non ASP.NET applications 285
- non-ASP.NET applications
  - instrumenting 447

## O

- offline analysis files
  - improve transfer 777
- offline.xml 775
- OM agent and IAPA component installation 64
- online cache 793
- Oracle 10g JAX-RPC
  - SOAP message handler 240
- Oracle application server startup scripts
  - modifying 172
- Oracle probe, configuring 110
- Oracle, application server configuration 224
- order of installation, recommended 48
- OSGi 242
- OVTA like points 421

## P

- parked fragments 390
- password obfuscation 125
- patch installation instructions 893
- path segment trimming 510
- perfmon 666
  - system metric counters 696

- Performance Center
  - firewall configuration 687
  - integration with Diagnostics 779
  - load tests, configuring to use Diagnostics 783
  - offline analysis files 784
  - setting up to use Diagnostics 782
- performance counters
  - access rights 666
- performance history data 870
- performance monitor users group 666
- permission denied
  - server install 64, 919
- Permissions Page 805
- permissions, *See* user permissions
- persistence
  - configure 877
  - data file types 873, 874
  - data files 876
- persistence directory
  - history data 872
- persistence.purging.threshold 880
- PMI
  - EXPAND\_PMI 733
- PMI integration 756
- points
  - .NET 428
  - arguments defined for .NET 430
  - arguments defined for Java 340
  - edit 418
  - Java 338
  - specifying for Java applications 349
- points element 597
- points list 417
- portal layers 425
- preinstallation considerations 47
- priority 347
- privileges
  - user 799
- PRO product mode 506, 507
- probe administration UI 515
- probe aggregator 286
- probe host machine name
  - getting 503
- probe level permissions 806
- probe metrics



- additional .NET 590
  - collect additional 666
- probe metrics in offline analysis 774
- probe name
  - unique 234
- probe settings
  - edit from the profiler 539
- Probe, .NET
  - enabling 628
- Probe, .NET advanced configuration
  - classes/methods 628
  - customizing instrumentation for
    - ASP.NET applications 628
  - depth trimming 646
  - disabling logging 655
  - elements and attributes 551
  - latency trimming and throttling 641
  - light-weight memory Diagnostics (LWMD) 649
  - overriding default Probe host name 656
- probe.id 235
- probe\_config.xml
  - elements and attributes defined 551
- probeconfig element 598
- process element 599
- product security 798
- profiler
  - disabling 501
- profiler element 601
- profilers
  - authentication for standalone 518
- proxy
  - enabling communication through 671
- proxy server
  - configuring for agent 511
- purging
  - symbol table 879
- Python agent
  - installing 295

**Q**

- query 791
- query page

- accessing 517
- queue size 495

**R**

- reflector 634
- registrar 791
- regular expressions 926
- regular expressions, backslash (\) 928
- remote-backup.sh 884
- remoting
  - instrumentation for 451
- requirements, Diagnostics Server 47
- REST client
  - .NET configuration 445
- REST services
  - .NET WCF 444
  - configuring as web services 537
- restore 887
- resumeFragment 390
- retention 876
- reverse HTTP
  - configuring for SaaS 512
- RMI instrumentation 387
- roles 800
- rootRenameTo 347
- rum element 603
- Run-time Service Model
  - timing for adding web service CIs 753

**S**

- SaaS
  - .NET agents 253
  - java agent and mediator connection 147
  - java agents 135
  - port 147
  - servers 55
- sample element 605
- samples queue size 495
- samples sent to BSM 752
- sampling
  - thread stack trace 540
- SAN drive 33
- SAP collector

## Index

- out of memory 109
- SAP NetWeaver, application server
  - configuration 178
- SAP probe, configuring 106
- scalability information 34
- scheduler page
  - accessing 517
- scope 343, 433
- secure communications 839
- security 791
- security page
  - accessing 517
- security permissions 799
- server configuration pages 496
- server requests
  - too many URIs 510
- Server, Diagnostics *See* Diagnostics Server
- servers
  - migrating 477
- Service Health Analyzer (SHA) 755
- ServiceGuard
  - high availability server 489
- setup
  - installation 55
- signature 341, 435
- silent install
  - log file 69, 104, 159
  - server 68
  - specify temp dir 69, 104, 159
- SiteMinder
  - reverse proxy setup in Apache 834
- SiteMinder JAAS authentication 836
- SiteMinder JAAS LoginModule
  - reverse proxy 833
- SMTP settings 63
- snapshot persistence 880
- SOAP Fault Data, configuration for .NET probes 665
- SOAP fault data, configuring for Java probes 535
- SOAP faults
  - configuring capture 535, 665
- SOAP message handler 152, 237
  - disable 238
  - loading 239
- soapcapture element 606
- soaprequestforsoapfault element 608
- soaprule element 609
- soaprules element 610
- Solaris custom metrics 718
- SQL Server collector
  - Windows NT security 115
- SQL Server probe
  - configuring 113
- sql statement parsing 537
- sql statements
  - limit number of 538
- sqlparsing element 611
- stack trace data
  - limiting 654
- stack trace sampling 540
  - examples 542
  - troubleshooting 543
- startup script
  - multiple probes 235
- summary files 874
- support collector 915
- switch\_ovo\_agent.sh 745
- symbol table backup
  - configuring 886
- symbol table files 874
- symbols element 613
- synchronize web service CIs 753
- synchronize with BAC 791
- system metrics
  - .NET 691
  - about 711
  - add custom 696, 715
  - capture 691
  - customizing in Windows 715
  - customizing on Linux host 719
  - customizing on Solaris host 718
  - default metrics collected by .NET metrics agent 692
  - default metrics collected by Java agent 712
  - Java 711
  - metrics collector 713
  - metrics collector entries 705
  - modifying captured metrics 708
  - stopping capture 708
- system metrics collector 711

- system requirements 31
    - .NET agent 38
    - Diagnostics Server host 32
    - Java agent host 36
    - with .NET probes 34
    - with Java probes 33
  - system/ 694
- T**
- temporary queues
    - grouping into a single node 537
  - thread stack trace sampling 540
  - throttling 644
  - TIBCO ActiveMatrix 3.x configuration 185
  - TIBCO BusinessWorks configuration 183
  - TIBCO EMS probe
    - configuring 120
  - TIBCO JMX metrics collection 184
  - time synchronization 466
  - time synchronizing between components 466
  - timestamping 521
  - timestamps
    - CPU 545
  - Tomcat application server configuration 187
  - topology element 615
  - TransactionVision related instrumentation 346
  - trend files 875
  - trial license 80
  - trim element 618
  - trimming
    - controlling depth 509
    - controlling latency 508, 641
    - server request name based 484
    - server request URIs 510
  - trimming parameters 474
  - troubleshooting Diagnostics 909
  - TV event generation
    - points driven 434
- U**
- UI requirements
    - JRE version 32
  - unique probe name 234
  - UNIX installer
    - how to select an option 925
  - updates, documentation 23
  - upgrade paths
    - general recommendations 894
  - upgrade procedures 893
  - upgrade recommendations 894
  - upgrading earlier versions of Diagnostics 50
  - URI truncation and trimming 510
  - use.cpu.timestamps 522
  - user permissions
    - about 798
    - accessing Diagnostics, default users 801
    - assigning for Diagnostics users in Business Availability Center 748
    - managing user details 802, 810
  - user roles 800
  - users
    - list of 791, 819
- V**
- view
    - privilege level 799
  - VMWare
    - time synchronization 628
  - VMware
    - time synchronization 512
  - VMware and CPU time metrics 522, 545
  - vmware element 622
  - VMWare issues on .NET 622
  - VMware probe
    - configuring 123
- W**
- WCF supported transports 252
  - WCF.points 429
  - WDEDelivery queue size 495
  - weblogic over T3
    - instrumentation for cross VM 388
  - webservice element 624
  - WebSphere application server startup script
    - modifying 196

## Index

- WebSphere application startup issue 914
- WebSphere IDE configuration 211
- WebSphere JAX-RPC
  - SOAP message handler 239
- WebSphere JMX metric collection 211
- weeks data 873
- windows credentials
  - SQL server collector 115
- Windows custom metrics 715
- ws element 625

## **X**

- xml
  - look deeper in xml for consumer ID 530
- xvm element 626

## **Y**

- years data 873

## **Z**

- z/OS
  - installing the Java agent 154
  - Java agent install 154
- zip files 876