Quest™

Foglight® for Java EE Technologies 5.9.13

Installation Guide

owners.

Legend

WARNING: **A WARNING icon indicates a potential for property damage, personal injury, or death.**

CAUTION: **A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.**

IMPORTANT NOTE, **NOTE**, **TIP**, **MOBILE**, or **VIDEO:** An information icon indicates supporting information.

Foglight for Java EE Technologies Installation Guide
Updated - March 2018
Software Version - 5.9.13

Contents

1

Installing and Configuring Foglight for Java EE Technologies

This guide provides installation and configuration instructions for Foglight for Java EE Technologies and the instrumented agents for Java EE. It is intended for any user who wants to install and configure Foglight for Java EE Technologies.

Foglight for Java EE Technologies contains a Java EE Integration Agent and a Java EE Agent.

The Java EE Integration Agent is responsible for the following activities:

- Transferring the files that the Java EE Agent needs to the remote application server hosts in your monitored environment.
- Integrating the Java EE Agent with application servers
- Helping to configure the Java EE Agent.

The Java EE Agent is responsible for collecting performance and monitoring data from the application servers.

Before installing Foglight for Java EE Technologies, ensure that you have the correct environment as outlined in the *Foglight for Java EE Technologies Release Notes* and the *Supported Platforms and Servers Guide*.

IMPORTANT: To install or upgrade Foglight for Java EE Technologies, you must belong to a user group that has Foglight Security role permissions. For more information about assigning user permissions, see the *Foglight Administration and Configuration Guide*.

Adjusting the Agent Manager settings for the Nexus

If you intend to use the default Nexus running in the embedded Foglight for Java EE Technologies Agent Manager, you must adjust the Agent Manager disk cache settings before installing and configuring Foglight for Java EE Technologies. The default Nexus is created as part of the installation process.

The Agent Manager accumulates messages that are destined to be sent either upstream or downstream in queues between connections. This prevents messages from getting lost in the event of a disconnection.

Increase the Agent Manager disk cache size to ensure that data submissions from the Java agent are not discarded. See the documentation in the *fglam.config.xml* file for detailed descriptions of the options.

NOTE: This procedure is recommended for all installations.

To adjust the Agent Manager settings for use with the Nexus:

1. Navigate to your Agent Manager `config` folder. For example: `C:\Quest_Software\Foglight_Agent_Manager\state\default\config`.
2. Open the *vm.config* file in an editor.

   IMPORTANT: In Agent Managers version 5.6.10 and later, this file is named baseline.jvmargs.config.

3 Uncomment and set the following setting to:

```
vmparameter.0 = "-Xmx1024m";
```

4 Save and close the *.config* file.

5 In the same config directory, open the *fglam.config.xml* file in an editor.

6 Edit the `config:queue-sizes` and `max-disk-space` settings as follows:

```
<config:queue-sizes>
    <config:upstream max-queue-size="500" max-disk-space="100000" max-batch-size="500" allow-runtime-change="false"/>
    <config:upstream-verified max-queue-size="250" max-disk-space="50000" max-batch-size="250" allow-runtime-change="false"/>
    <config:downstream max-queue-size="500" max-disk-space="1024" max-batch-size="500" allow-runtime-change="false"/>
</config:queue-sizes>
```

This sets the `max-disk-cache` size to 100MB for the upstream and upstream-verified queues (normal and verified submissions).

7 Save and close the *fglam.config.xml* file.

8 Restart the Agent Manager.

If the Nexus is running in a non-embedded Agent Manager, the Agent Manager service must be restarted.

If the Nexus is running in the embedded Agent Manager (for example, if you are using the default Nexus), do one of the following:

- Restart the Management Server.

Or

- Go to `<server>/jmx-console` (for example, *http://yourhost:8080/jmx-console/*) and click: `service=EmbeddedFglAm`.

  Invoke the `Stop()` operation, and then invoke the `start()` operation.

If you still find warnings in the Agent Manager log about the disk cache being too small, increase the `max-disk-space` values for the `upstream` and `upstream-verified` queues.

Next step:

- After you have adjusted the settings and restarted the Agent Manager, continue with Installing and Configuring Foglight for Java EE Technologies.

Installing Foglight for Java EE Technologies

Foglight for Java EE Technologies is distributed in a *.car* file, *ApplicationServers-Java-<version>.car*. Installing the cartridge file on the Management Server adds the capabilities for monitoring Java EE systems to your Foglight for Java EE Technologies environment.

Prerequsites:

- Ensure that you have the correct environment as outlined in the *Foglight for Java EE Technologies Release Notes* and the *Supported Platforms and Servers Guide*.
- Adjusting the Agent Manager settings for the Nexus.
- You must belong to a user group that has Foglight Security role permissions. For more information about assigning user permissions, see the *Foglight Administration and Configuration Guide*.

The installation process is common to all Foglight cartridges. For instructions for installing and enabling the cartridge, see Install Foglight cartridges in the *Foglight Administration and Configuration Help.*

NOTE: The cartridge file installation requires approximately 50MB of additional hard drive disk space on the Foglight Management Server.

To view the Cartridge for Java EE Technologies:

- On the navigation panel, under Dashboards, click **Administration > Cartridges > Cartridge Inventory**. Search for “Java”.

Nexus services

Foglight for Java EE Technologies includes technology for distributed Java EE performance and monitoring data collection called the Nexus. The Nexus runs as a service hosted on a Foglight for Java EE Technologies Agent Manager. This service is part of the cartridge; disabling the cartridge stops Nexus services and enabling the cartridge starts them.

For information about configuring the Nexus, see Nexus and Agent-Nexus connections on page 43.

Installing the agents

Foglight for Java EE Technologies includes the Application Servers Administration dashboard. Use this dashboard to install agents and integrate them with application servers.

NOTE: Agents should be installed using an account that has administrator privileges. For more information, see Java EE Integration Agent user permissions on page 10.

Installation and integration process

1. Set up the integration host (the application server with which you want to integrate):
   - Deploy a Foglight Agent Manager of the correct version to each host that you want to monitor with a Java EE Agent. Use the **Deploy Agent Package** option on the **Administration > Agents > Agent Hosts** dashboard.

     TIP: If you are uncertain of the required Agent Manager version, see the *Foglight for Java EE Technologies Supported Platforms and Servers Guide*, or the *Foglight for Java EE Technologies Release Notes*.
2. Make sure that you have a Nexus available, and check that the Nexus agent is up and running.

   The Nexus receives and correlates data from the Java agent. For more information about the Nexus, and for tasks relating to Nexus creation and management, see the *Foglight for Application Servers Administration and Configuration Guide* or online help.
3. Use the **Java — Setup New Application Server Integration** wizard to configure an integration. This wizard is available through **Application Servers > Administration > MonitoringAgents > Setup agents... > Java.**

   Use the wizard for the following activities:
   - Create the Integration Agent that manages the Java (monitoring) agent and your application server integration.
   - Set the required information for the connection with the Nexus.
   - Set logging, agent configuration (data recording settings), and installation directory details.

For detailed instructions on using the wizard, see Installing a Java EE agent and the appropriate topics for the application server type you are attempting to integrate. You may also want to review the information in Java EE Integration Agent user permissions before using the wizard.

Installing a Java EE agent

Prerequisites:

Before installing the Java EE agent, make sure that the following components are installed and running:

1 An Agent Manager installed and running on the target host.

2 The Nexus is installed and the Nexus agent is running.

To install an agent:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

The Application Servers Administration dashboard opens.

2 On the Monitoring Agents view, click **Setup agents** and select **Java**.

The Java — Setup New Application Server Integration wizard opens.

3 Select whether you want to set up a new application server integration configuration, or reuse an existing one, and click **Next**.

For information about reusing an integration configuration, see Reusing an integration configuration on page 50.

For new integrations, continue with the next step.

4 Select the type of integration you want to perform and click **Next**.

Follow the on-screen instructions to configure the integration and create the agent.

Integrating with application servers

For detailed instructions, see the section applicable to the type of integration you are creating:

- Integrating with JBoss on page 13
- Integrating with Apache Tomcat on page 18
- Integrating with WebLogic Domain Startup Scripts on page 21
- Integrating with WebSphere on page 24
- Integrating with Oracle AS on page 33
- Creating a Generic Installation for Manual Java EE Agent Integration on page 39
- Reusing an integration configuration on page 50

NOTE: Some older versions of Java EE agents can still connect to this version of Foglight for Java EE Technologies. See the *Foglight for Java EE Technologies Release Notes* for details.

Java EE Integration Agent user permissions

Foglight for Java EE Technologies consists of two separate agents, each of which require access to the file system and other resources.

- **Java EE Integration Agent** - run by the same user that is running the Agent Manager process during agent activation.
- **Java EE Agent -** run by the user running the integrated application server.

The required permissions can be supplied at any level appropriate to your organization or IT structure:

- **Owner/User Permissions** — The Agent Manager and the Application Server must be run by the same user. Usually in this case there is no need to set permissions explicitly.
- **Group Permissions** — The Agent Manager user and the Application Server user belong to a common group.
- **Others Permission** — The Agent Manager user is different from the Application Server user and these two users cannot be added to a common group. Because Others permissions include all other users, it is recommended that Owner or Group permissions be used whenever possible.

For more information, see Appendix: Managing Permissions for the Java EE Integration Agent on page 109.

Java EE Integration Agent installation directory

The installation directory (`DEPLOYMENT_DIRECTORY`) is created during the Java EE Integration Agent activation. After the Java EE Integration Agent is activated, all future version updates of Foglight for Java EE Technologies use this directory to store sub-directories containing the latest Java agent libraries, configuration files, and log files. In addition, the installation directory contains the integrate scripts, *integrate.cmd,* and *integrate.sh*, that are used to configure your application servers.

The default value for this installation directory is *JavaEE*. This path is relative to the *<FglAM_HOME>\agents* directory created during the Foglight Agent Manager (FglAM) installation.

NOTE: You can specify a custom location during installation. If you do not specify a fully qualified path, it is relative to the <FglAM_HOME>\agents directory.

IMPORTANT: Do not use the JavaEE Installation directory anywhere in your application server startup scripts. This directory is used by the Java EE Integration Agent to maintain a copy of the original agent files/libraries.

For more information about managing installations, see Managing Java EE Agent Installations, Integrations, and Configurations on page 42.

Upgrading Java EE Agents and Integration Agents

The *Foglight Upgrade Guide* provides full instructions for upgrading to the latest version of the cartridge and agents. Read the topic Upgrade Foglight for Java EE Technologies in the Upgrading the Application Monitoring Cartridges section for details.

Depending on the version you are upgrading from, you may also need to upgrade the Nexus. See the *Foglight Upgrade Guide* for details. .

Configuration migration during upgrades

If you have modified the configuration files and you perform a minor version upgrade (for example from x.y.z to x.y.z+1), your modified files are preserved. The default contents of the configuration files from the new version are overwritten with your changes.

For example, you have version 5.9.3.2 currently installed and you have modified the Nexus *recording.config* and the Java EE agent *instrumentation.config*. Upgrading to version 5.9.4 would overwrite the default contents of the 5.9.4 versions of those files with your file contents from 5.9.3.2. All other 5.9.4 configuration files are unchanged, and contain the default contents installed by the 5.9.4 cartridge.

NOTE: Configuration files managed through the Advanced dashboards (for example, *nexus/aggregation.config* or *agent/java/collector/jboss.config*) are not automatically migrated. If you have made any modifications to these files, they must be migrated manually, with the assistance of Customer Support where necessary.

In addition, the *nexus/compatible-builds.config* file is never automatically migrated. If you have any patch build IDs, these must be manually added to the new version's *compatible-builds.config* with the assistance of Customer Support.

IMPORTANT: No automatic migration occurs during an upgrade to the next major version (for example, from 5.8.x to 5.9.x), however the migration from 5.9.x to 5.10.0 is supported.

2

Integrating with JBoss

You can use the application server agent setup wizard to configure and integrate the Java EE agent and the Java EE Integration Agent with JBoss® server startup scripts. You must supply the JBoss Home directory for one or more application servers with which you want to integrate. Optionally, you can also specify pre-instrumentation and agent options.

IMPORTANT: Foglight for Java EE Technologies version 5.9.11 is the last major release that supports JBoss AS (all versions), JBoss EAP 4.x, and JBoss EAP 5.x.

For an overview of the installation and integration process, see Installing the agents on page 9.

To integrate with JBoss on UNIX® or Windows®:

IMPORTANT: Ensure that the target application server has an active Foglight Agent Manager of a compatible version installed and running.

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, in the Monitoring Agents section, click **Setup agents..**.
3. In the menu that opens, select **Java**.

   The Java — Setup New Application Server Integration wizard opens.
4. On the Setup Options page, select the type of integration. This example describes how to **Setup a new application server integration** (the default selection).

   TIP: For other integration methods, Reusing an integration configuration on page 50.
5. On the New Integration — Type page, select the type of Java application server you want to integrate with. In this case, select either **JBoss on Solaris, Linux, HP-UX, or AIX**, or **JBoss on Windows**, as appropriate.
   a. The wizard refreshes, indicating that it will use the JBoss Startup Scripts. This option integrates the JBoss *run.bat* or *run.sh* startup scripts.
   b. Click **Next**.
6. On the New Integration — Details page, specify any additional information required to integrate with the application server.
   a. Type or select the absolute path to the **JBoss Home** directory. This is the JBoss installation location on the remote file system.
   b. Optional — If your application server uses startup scripts with customized names, click the arrow beside **Other Options** to open a list of script paths and names. Type the custom names in the appropriate boxes.

      NOTE: Always use forward slashes to separate directory names, regardless of whether the application server is running on Windows or UNIX.
   c. Optional — Click the arrow beside **Command Line Options** to show the agent and pre-instrumentor option boxes. Use these boxes to provide command-line parameters for the agent or pre-instrumentor.

- To add agent options, type them in the **Agent Options** box.

  For example, if the JBoss *.jar* files are located in a `lib/endorsed` directory, add the following option in the Agent Options box:
  `-Dquest.agent.instrument.bootstrap=true`

  > **NOTE:** If you need to add more than one option to the Agent Options, separate them with spaces. Do not use quotes.

- To add pre-instrumentation options, type them in the **pre-instrumentor Options** box.

  To set pre-instrumentor options specific to the JVM you are running, type the desired VM options in the pre-instrumentor Options box.

  For example, to set the memory limit for the pre-instrumentor to 256MB, type the following:
  `-Xmx256m`

  > **NOTE:** If you need to add more than one option to the pre-instrumentor Options, separate them with spaces. Do not use quotes.

  > **TIP:** For information about pre-instrumentation, see Configuring Java Homes pre-instrumentation on page 39.

d Click **Next**.

7 On the Nexus Connection page, select or create the connection that the agent should use to locate and communicate with a Nexus.

> **TIP:** For more information about the Nexus, see Nexus and Agent-Nexus connections on page 43.

If you are using the default Nexus on the embedded Agent Manager, or if you have created a single remote Nexus, a default Nexus Connection is created and pre-selected for you.

Otherwise, select or define at least one Nexus connection. For more information, see Creating or editing Agent-Nexus connection settings on page 44.

Click **Next**.

8 On the Logging page, select or create a logging configuration to control how logs are written, including retention policies and rollovers.

- Select an existing log configuration from the list. The default provides a basic log configuration that you can use to get started.

Or

- Click **New log configuration**. The New Log Configuration dialog box opens.

> **TIP:** If you are uncertain about how to adjust these settings, click **Help** for an overview of log targets and their purposes. For details, see Creating or editing logging properties on page 48.

Click **Next**.

9 On the Integration Hosts page, select the monitored host where the JBoss application server is installed. By default, only eligible hosts (that is, hosts that have an active Foglight Agent Manager of a compatible version installed and running) are listed.

Select the host and click **Next**.

10 On the Agent Configuration page, select the configuration for the agent to use. This agent configuration defines the requests to monitor and collect details.

> **TIP:** You can create agent configurations or edit existing ones from the Java Administration dashboard. For more information, see Managing Java EE Agent Configurations on page 52.

Select a configuration and click **Next**.

11 On the Installation Directory page, define the location for the directory that contains all installation components and log files. To accept the default directory, leave the **Directory name** box empty.

> **IMPORTANT:** If you have previously configured the host for agent installation, the existing installation directory is used.

To specify a custom directory, type a path in the **Directory name** box.

Click **Next**.

12 On the Review page, verify the details of the integration.

a By default, the instrumentation script is assigned a name based on the integration type. For example: *JBoss on Windows: ServerRunScript*. If you want to customize this name, type the new value in the **Name** box.

b Click **Finish**.

The Application Servers Administration dashboard refreshes, and the integration task appears in the Task History list, where you can monitor its progress.

13 Restart your application server.

The agent starts collecting data after the application server has restarted and the agent is successfully integrated.

Next steps:

- If you are running JBoss with Jetty, see Running JBoss with Jetty on page 15.
- If you are integrating with the Java Service Wrapper, see Integrating a Java Service Wrapper Windows Service with a Java EE agent on page 16.
- Use the Application Servers Monitor dashboard to view the application servers status. For more information about the Application Servers Monitor dashboard, see the *Foglight for Application Servers User Guide*.
- If for any reason you need to manully integrate a JBoss agent, see JBoss remote agent integration from the command-line on page 88.

Running JBoss with Jetty

Jetty is a pure Java-based HTTP server and servlet container (application server). If you are running JBoss® with Jetty, make the following changes to the property file *agent_type_jboss-<version>.config* before you start recording.

> **IMPORTANT:** Foglight for Java EE Technologies version 5.9.11 is the last major release that supports JBoss AS (all versions), JBoss EAP 4.x, and JBoss EAP 5.x.

The following table lists the <*version*> names:

Table 1. Application Server version names.

Application Server	Version
EAP 6	*agent/java/plugins/jboss-eap-6.config*

To edit the agent .config file:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, click the **Java** tab.

3 On the Java view, click the **Agent Configurations** tab.

4 In the Agent Configurations view, click **Advanced Configurations**.

The Advanced Configuration view opens.

5 Select the version of the Java EE agent that you want to edit. Click **Version x.x.x** to open a list of compatible versions.

6 Click the name of the `agent/java/plugins/jboss-<version>.config` for the JBoss version you want to edit, and select **Edit**.

> **TIP:** Click **Show fully qualified names** to see the directory location for the files. Type all or part of the filename in the **Search** box (for example, `jboss-`) to locate the file.

7 In the text editor, scroll down to locate and comment out the following line:

```
"com.quest.pas.agent.plugin.instrumentor.method.ServletInstrumentor",
```

8 Uncomment the following lines:

```
#"com.quest.pas.agent.plugin.instrumentor.method.HttpServletInstrumentor",
#"com.quest.pas.agent.plugin.instrumentor.ServletRequestInstrumentor",
```

9 Click **Save**.

10 Restart your application server.

Integrating a Java Service Wrapper Windows Service with a Java EE agent

Foglight for Java EE Technologies supports the use of Windows Services configured using the Java Service Wrapper (JSW) application by Tanuki Software.

The JSW uses configuration files to control how your Java application is started and stopped as a Windows Service. To integrate the Java EE agent with your JSW, the JDK must be pre-instrumented (using the Java EE Integration Agent) and then the JSW configuration file needs to be manually modified to include Java EE agent configuration settings.

To integrate the Java Server Wrapper with a Java EE agent:

1 Create an installation, as described in Creating a Generic Installation for Manual Java EE Agent Integration on page 39.

2 After the generic installation is complete, a new file appears in the following directory on the machine where you ran the pre-instrumentation.

Windows®: `<VERSIONED_SUBDIRECTORY>\bootstrap\`*<INTEGRATIONID>*

UNIX®: `<VERSIONED_SUBDIRECTORY>/bootstrap/`*<INTEGRATIONID>*

You can find the *<INTEGRATIONID>* in the file that the Java EE Integration Agent creates in step 1.

3 Run the integration `bat/sh` file that was created by the manual installation.

4 Open the JSW configuration file in a text editor, and add the following code before the first `wrapper.java.additional` code:

```
#include <INSTALLATION DIRECTORY>\jee_builds.properties
#set.JAVA_HOME=<JVMPath>
set.QUEST_INTEGRATIONID=<INTEGRATIONID>
set.PERFORMASURE_BOOTSTRAP=-
Xbootclasspath/p:%QUEST_AGENT_HOME%\bootstrap\%QUEST_INTEGRATIONID%\<BOOTSTRAP.JAR
FILENAME>
set.PERFORMASURE_JAVAAGENT=-javaagent:%QUEST_AGENT_HOME%\lib\performasure-agent.jar
```

a Replace *<INSTALLATION DIRECTORY>* with the absolute path of the Installation Directory. The path can be found on the Installations tab of the Java Administration page (accessible from the

Applications Servers Administration dashboard). If the Installation Directory path is specified as a relative path, as it is by default, you may need to examine the agent machine to determine the absolute path.

b If `set.JAVA_HOME` is not specified earlier in the JSW configuration file, uncomment the second line in the code block above and set *<JVMPath>* to the absolute path for the JVM that the service uses.

For example: `set.JAVA_HOME=C:\work\jvms\jdk1.5.0_12`

Or, if the path is set in a different variable, such as `JRE_HOME`:
`set.JAVA_HOME=%JRE_HOME%`

c Replace `<INTEGRATIONID>` with the value determined in step 2.

d Replace `<BOOTSTRAP .JAR FILENAME>` with the filename of the bootstrap JAR file created in the `<VERSIONED_SUBDIRECTORY>/bootstrap/`*<INTEGRATIONID>* directory by the pre-instrumentation task in step 1.

5 In the JSW configuration file, add the following code after the last existing `wrapper.java.additional` entry:

```
wrapper.java.additional.<N>=%QUEST_INTEGRATIONID%
wrapper.java.additional.<N>=%PERFORMASURE_JAVAAGENT%
wrapper.java.additional.<N>=%PERFORMASURE_BOOTSTRAP%
```

and replace **<N>** with incremented numbers starting after your highest `wrapper.java.additional` number.

6 Save your JSW configuration file and then restart your JSW service.

3

Integrating with Apache Tomcat

Use the application server agent setup wizard to configure and integrate the Java EE agent and the Java EE Integration Agent with Apache Tomcat Catalina startup scripts or with Tomcat running as a Windows® Service.

You must supply the Catalina Home directory or the Windows Service name for one or more application servers with which you want to integrate. Optionally, you can also specify pre-instrumentation options and agent options.

Before you begin

For most configurations of a Tomcat server running as a Windows Service, the Java EE Integration Agent automatically integrates the Java EE agent. Read the **Important** note below to determine if you are required to complete additional steps to integrate the Java EE agent with a Tomcat server running as a Windows Service.

If you are not required to perform the additional steps, complete the integration procedure (Integrating with Tomcat on page 18), then restart your Tomcat application server. No other steps are required.

IMPORTANT: If you are using one of the following configurations, you are required to integrate manually. Extra steps are required.

- On 32-bit or 64-bit Windows: Startup mode is set to Java and you have not specified a Java home using the `JavaHome` command line option.
- On 64-bit Windows: Startup mode is set to `jvm` and the Use Default check box is selected. If you are using one of these configurations, complete this procedure: Configuring a Tomcat Windows Service that requires manual integration on page 93.
- Best Practices: We recommend that you explicitly set the Java home or the Java Virtual Machine when setting up a Tomcat application server that is running as a Windows Service. This prevents the Tomcat server from running with an unexpected JRE and makes the integration easier.

Integrating with Tomcat

To integrate with Tomcat on Solaris®, Linux®, HP-UX®, or AIX®, or on Windows®:

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, in the Monitoring Agents section, click **Setup agents...**.
3. In the menu that opens, select **Java**. The Java — Setup New Application Server Integration wizard opens.
4. On the Setup Options page, select the type of integration. This example describes how to **Set up a new application server integration** (the default selection).

   TIP: For other integration methods, Reusing an integration configuration on page 50.

5. On the New Integration — Type page, select the type of Java application server you want to integrate with. In this case, select **Tomcat on Solaris, Linux, HP-UX, or AIX**, or **Tomcat on Windows**.
   a. The wizard refreshes, providing the option to select the Tomcat application server startup method.

      Select **Catalina Scripts** to integrate with Tomcat *catalina.sh* or *catalina.bat* startup scripts.

      Or

      Select **Windows Service** to instrument Tomcat (*procrun*) Windows Services.

> **NOTE:** If for any reason you need to integrate manually see Manual integration with Tomcat on page 91.

b Click **Next**.

6 On the New Integration — Details page, provide the details required for integrating with your Tomcat server, including command-line options.

a Type or select the absolute path to the Catalina startup script in the **Catalina Home** box.

Or

Type the Windows Service name in the **Service Name** box.

> **IMPORTANT:** You must have a Windows Service defined to proceed with this type of integration. If you do not yet have a Windows Service defined, first install your Tomcat server to run as a Windows Service.

b Optional — If your application server uses startup scripts with customized names, click the arrow beside **Other Options** to open a list of script paths and names. Type the custom names in the appropriate boxes.

> **NOTE:** Always use forward slashes to separate directory names, regardless of whether the application server is running on Windows or UNIX.

c Optional — Click the arrow beside **Command Line Options** to show the agent and pre-instrumentor option boxes. Use these boxes to provide command-line parameters for the agent or pre-instrumentor.

- To add agent options, type them in the **Agent Options** box.

> **TIP:** If you need to add more than one option to the Agent Options, separate them with spaces. Do not use quotes.

- To add pre-instrumentation options, type them in the **pre-instrumentor Options** box.

  To set pre-instrumentor options specific to the JVM that you are running, type the desired VM options in the pre-instrumentor Options box.

  For example, to set the memory limit for the pre-instrumentor to 256MB, type the following:
  `-Xmx256m`

> **NOTE:** If you need to add more than one option to the pre-instrumentor Options, separate them with spaces. Do not use quotes.

> **TIP:** For information about pre-instrumentation, see Configuring Java Homes pre-instrumentation on page 39.

d Click **Next**.

7 On the Nexus Connection page, select or create the connection that the agent should use to locate and communicate with a Nexus.

> **TIP:** For more information about the Nexus, see Nexus and Agent-Nexus connections on page 43.

If you are using the default Nexus on the embedded Agent Manager, or if you have created a single remote Nexus, a default Nexus Connection is created and pre-selected for you.

Otherwise, select or define at least one Nexus connection. For more information, see Creating or editing Agent-Nexus connection settings on page 44.

Click **Next**.

8 On the Logging page, select or create a logging configuration to control how logs are written, including retention policies and rollovers.

- Select an existing log configuration from the list. The default log provides a basic log configuration that you can use to get started.

Or

- Click **New log configuration**. The New Log Configuration dialog box opens.

> **TIP:** If you are uncertain about how to adjust these settings, click **Help** for an overview of log targets and their purposes. For details, see Creating or editing logging properties on page 48.

Click **Next**.

9 On the Integration Hosts page, select the host where Tomcat is installed. By default, only eligible hosts (that is, hosts that have an active Foglight Agent Manager of a compatible version installed and running) are listed.

Select the host and click **Next**.

10 On the Agent Configuration page, select the configuration that the agent will use. This agent configuration defines the requests to monitor and collect details.

> **TIP:** You can create agent configurations or edit existing ones from the Java Administration dashboard. For more information, see Managing Java EE Agent Configurations on page 52.

Select a configuration and click **Next**.

11 On the Installation Directory page, define the location for the directory that contains all installation components and log files. To accept the default directory, leave the **Directory name** box blank.

> **IMPORTANT:** If you have previously configured the host for agent installation, the existing installation directory is used.

To specify a custom directory, type a path in the **Directory name** box.

Click **Next**.

12 On the Review page, verify the details of the integration.

a By default, the instrumentation script is assigned a name based on the integration type. For example: *Tomcat on Windows: CatalinaScripts*.

If you want to customize this name, type the new value in the **Name** box.

b Click **Finish**.

The Application Servers Administration dashboard refreshes, and the integration task appears in the Task History list, where you can monitor its progress.

13 Restart your application server.

The agent starts collecting data after the application server has restarted and the agent is successfully integrated. Use the Application Servers Monitor dashboard to view the application servers status. For more information about the Application Servers Monitor dashboard, see the *Foglight for Application Servers User Guide*.

4

Integrating with WebLogic Domain Startup Scripts

You can use the Java EE Integration Agent to integrate the Java EE agent with WebLogic® domain startup scripts. Supply a description and the domain home path for the application server with which you want to integrate. If you are integrating with WebLogic running as a Windows® Service, specify the name of the Windows Service.

Prerequisite:

- You must have a Windows Service defined before attempting an integration with WebLogic running as a Windows Service.

IMPORTANT: If you are integrating with WebLogic 8.1.x, you cannot use the Integration Wizard described in this section. Instead, you must manually integrate. For detailed information, see Creating a Generic Installation for Manual Java EE Agent Integration on page 39.

To integrate with WebLogic on Solaris®, Linux®, HP-UX®, or AIX®, or on Windows®

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, in the Monitoring Agents section, click **Setup agents...**.
3. In the menu that opens, select **Java**. The Java — Setup New Application Server Integration wizard opens.
4. On the Setup Options page, select the type of integration. This example describes how to **Setup a new application server integration** (the default selection).

   TIP: For other integration methods, see Reusing an integration configuration on page 50.

5. On the New Integration — Type page, select the type of Java application server you want to integrate with. In this case, select **WebLogic on Solaris, Linux, HP-UX, or AIX**, or **WebLogic on Windows**.
   a. The wizard refreshes, providing the option to select the WebLogic application server startup method:

Table 2. Integration types

Integration Type	Description
Windows Service	Integrates a WebLogic Server or NodeManager Windows Service.
Domain Startup Script	Integrates the *startWebLogic.sh/.cmd* scripts within a WebLogic Domain (`DOMAIN_HOME/bin/startWebLogic`).
NodeManager Startup Script	Integrates the *startNodeManager.sh/.cmd* scripts within a WebLogic installation (`WL_HOME/server/bin/startNodeManager`).
Managed Server Startup Script	Integrates the *startManagedWebLogic.sh/.cmd* scripts within a WebLogic 12.1.1 or earlier installation (`WL_HOME/common/bin/startManagedWebLogic`).

Table 2. Integration types

Integration Type	Description
Non-Domain startWLS Startup Script	Integrates the startWLS script within a WebLogic 10.0 or earlier installation (`WL_HOME/server/bin/startWLS`).
All Non-Domain Scripts	Combines the NodeManager, Managed Server, and startWLS Startup Script tasks into a single integration.

b Select a startup method.

c Click **Next**.

6 On the New Integration — Details page, provide the details required for integrating with your WebLogic server, including command-line options.

a For most WebLogic integrations, type or select the absolute path to the **WebLogic Home** directory. WebLogic Home is the installation directory of a particular WebLogic version.

If you are integrating using *startWebLogic* as the startup method, type or select the absolute path to the **Domain Home**. Domain Home is the directory containing a WebLogic Domain.

If you are integrating with WebLogic as a Windows Service, type the name of the Windows service in the **Service Name** box.

b Optional — If your application server uses startup scripts with customized names, click the arrow beside **Other Options** to open a list of script paths and names. Type the custom names in the appropriate boxes.

NOTE: Always use forward slashes to separate directory names, regardless of whether the application server is running on Windows or UNIX.

c Optional — Click the arrow beside Command Line Options to show the agent and pre-instrumentor option boxes. Use these boxes to provide command-line parameters for the agent or pre-instrumentor.

- To add agent options, type them in the **Agent Options** box.

NOTE: If you need to add more than one option to the Agent Options, separate them with spaces. Do not use quotes.

- To add pre-instrumentation options, type them in the **pre-instrumentor Options** box.

  To set pre-instrumentor options specific to the JVM you are running, type the desired VM options in the pre-instrumentor Options box.

  For example, to set the memory limit for the pre-instrumentor to 256MB, type the following: `-Xmx256m`

NOTE: If you need to add more than one option to the pre-instrumentor Options, separate them with spaces. Do not use quotes.

TIP: For information about pre-instrumentation, see Creating a Generic Installation for Manual Java EE Agent Integration on page 39.

d Click **Next**.

7 On the Nexus Connection page, select or create the connection that the agent should use to locate and communicate with a Nexus.

TIP: For more information about the Nexus, see Nexus and Agent-Nexus connections on page 43.

If you are using the default Nexus on the embedded Agent Manager, or if you have created a single remote Nexus, a default Nexus Connection is created and pre-selected for you.

Otherwise, select or define at least one Nexus connection. For more information, see Creating or editing Agent-Nexus connection settings on page 44.

Click **Next**.

8 On the Logging page, select or create a logging configuration to control how logs are written, including retention policies and rollovers.

- Select an existing log configuration from the list. The default log provides a basic log configuration that you can use to get started.
- Click **New log configuration**. The New Log Configuration dialog box opens.

TIP: If you are uncertain about how to adjust these settings, click **Help** for an overview of log targets and their purposes. For details, see Creating or editing logging properties on page 48.

Click **Next**.

9 On the Integration Hosts page, select the host where WebLogic is installed. By default, only eligible hosts (that is, hosts that have an active Foglight Agent Manager of a compatible version installed and running) are listed.

Select one or more hosts and click **Next**.

10 On the Agent Configuration page, select the configuration that the agent will use. This agent configuration defines the requests to monitor and collect details.

TIP: You can create agent configurations or edit existing ones from the Java Administration dashboard. For more information, see Managing Java EE Agent Configurations on page 52.

Select a configuration and click **Next**.

11 On the Installation Directory page, define the location for the directory that contains all installation components and log files.

a To accept the default directory, leave the **Directory name** box blank.

NOTE: If you have previously configured the host for agent installation, the existing installation directory is used.

To specify a custom directory, type a path in the **Directory name** box.

b Click **Next**.

12 On the Review page, verify the details of the integration.

a By default, the instrumentation script is assigned a name based on the integration type. For example: *WebLogic on Windows: startWebLogic*.

If you want to customize this name, type the new value in the **Name** box.

b Click **Finish**.

The Application Servers Administration dashboard refreshes, and the integration task appears in the Task History list, where you can monitor its progress.

13 Restart your application server.

The agent starts collecting data after the application server has restarted and the agent is successfully integrated. Use the Application Servers Monitor dashboard to view the application servers status. For more information about the Application Servers Monitor dashboard, see the *Foglight for Application Servers User Guide*.

5

Integrating with WebSphere

Before you begin remote agent integration with an IBM® WebSphere® application server, first review the following sections:

- Understanding what should be instrumented on page 24
- Setting the PMI levels on WebSphere on page 25
- Running WebSphere with security enabled on page 25

Next, follow the procedure in Integrating with WebSphere startup scripts on page 26, as appropriate for your system.

Understanding what should be instrumented

The following guidelines identify the components in a WebSphere® domain that require instrumentation to ensure that the Nexus can do a full data collection.

For standalone servers:

- Each standalone server process must be integrated and running.
- To collect WebSphere metrics on the average size of an HTTP session, enable **Average Size (b) Current** in the Show columns list and then restart your WebSphere application server using the `-Dquest.configfilter.httpsessionsize=true` command. For more information about show and hide columns, see the *Foglight for Application Servers User Guide*.
- WebSphere standalone servers must be integrated by their server startup scripts, or by individual Windows® server processes.

NOTE: Standalone servers started solely by an instrumented Admin agent do not need to be integrated individually. However, if a standalone server is started by other means (startup script, or Windows Service), then integrating using these methods is required.

- WebSphere version 8.x/9.0.0.x standalone servers registered with a WebSphere 8.x/9.0.0.x Admin agent possess an extra integration option: integrating the Admin agent process instead of individual registered servers. Any standalone servers launched by an instrumented Admin agent are also instrumented. However, the Admin agent is not required for monitoring of any standalone servers it may have started, and it may be shut down once all startup activities have been completed. To manually integrate a WebSphere version 8.x/9.0.0.x Admin agent, see Manual integration with IBM WebSphere Application Server on page 96.

For WebSphere Servers federated in a WebSphere Cell:

- The Domain Manager of each WebSphere Cell must be integrated for complete cell monitoring, and must remain running during the monitored period. Any interruption in the service of an instrumented domain manager results in incomplete Nexus data collection, and possible false alarms.
- Each Node agent must be integrated with the agent, and must remain running during the monitored period. Any interruption in the service of an instrumented `nodeagent` results in incomplete data collection, and

possible false alarms. Individual application servers started by an instrumented Node agent are instrumented automatically.

- No integration is required for individual servers. Attempts to start an individual server using a startup script fail, as WebSphere requires you to send the start command by using the server's node agent.

CAUTION: For any server with a corresponding Windows service: Attempts to start a server by using a startup script may launch a Windows service instead. If the startup script is instrumented with the Java EE Integration Agent, then the resulting WebSphere process is instrumented as well. Otherwise, the integration properties of the Service are used. This release changes the behavior of services launched as a result of a start script invocation.

Running WebSphere with security enabled

Set security levels and define PMI levels before configuring your environment.

If you want to run WebSphere® while J2EE security or Global security is enabled, edit the WebSphere's *server.policy* file to prevent problems between WebSphere and the Java EE agent.

To edit the *server.policy* file:

1. Open the *server.policy* file located in *<websphere_home>/properties* or in *<websphere_profile_home>/properties*.
2. Add the following block to the file:

```
grant codeBase "file:* QUEST_DEPLOYMENT_DIRECTORY */-" {
     permission java.security.AllPermission;
};
```

3. Restart your application server for the changes to take effect.

TIP: To understand what metrics Foglight for Java EE Technologies collects, review the *agent_collector_<SERVER>.config* files for your server type. For example, *agent_collector_websphere-6.config*. You can access these files from the Application Servers **Administration > Java Administration > Advanced Configuration** view.

Setting the PMI levels on WebSphere

If you want the Java EE agent to collect detailed performance statistics on runtime and application components, enable Performance Monitoring Infrastructure (PMI). After you enable PMI, the Java EE agent automatically enables the required counters at the start of data collection. You can also set the level of data collection if necessary.

To set the PMI levels on WebSphere® :

1. In the WebSphere Administrative Console navigation tree, do the following:
   - For WebSphere 8.x/9.0.0.x — click **Servers > Server Types > WebSphere Application servers**.

   In the right pane, the Application Servers page appears with the Configuration tab open.
2. In the right pane, select the server.
3. Under Performance, click **Performance Monitoring Infrastructure (PMI)**.
4. Select the **Enable Performance Monitoring Infrastructure (PMI)** check box.
5. Optional — Select or clear the **Use sequential counter update** check box to enable or disable precise statistic update.

6 Optional — Set the currently monitored statistic set to any level.

> **IMPORTANT:** The higher the setting, the more significant the performance impact of the metric collection process is on your application server. For more information on PMI settings, see your WebSphere documentation.

7 Click **Apply**, then **OK** to return to the server settings.

8 In the Messages pane, click **Save** to apply your changes.

9 For the PMI settings to take effect, restart all WebSphere Application Servers in your environment.

Integrating with WebSphere startup scripts

You can use the Java EE Integration Agent to integrate the Java EE agent with WebSphere® Startup Scripts. Specify the WAS Home and WAS Profile type for the application server that you want to integrate with.

To integrate with WebSphere on Solaris®, Linux®, HP-UX®, or AIX®, or on Windows®:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, in the Monitoring Agents section, click **Setup agents**...

3 In the menu that opens, select **Java**.

The Java — Setup New Application Server Integration wizard opens.

4 On the Setup Options page, select the type of integration. This example describes how to **Setup a new application server integration** (the default selection).

> **TIP:** For other integration methods, Reusing an integration configuration on page 50.

5 On the New Integration — Type page, select the type of Java application server you want to integrate with. In this case, select **WebSphere on Solaris, Linux, HP-UX, or AIX**, or **WebSphere on Windows**.

a The wizard refreshes, providing the option to select the WebSphere application server startup method:

Table 3. Startup Method descriptions.

Startup Method	Description
Deployment Manager Startup Scripts	Startup Scripts for a Deployment Manager Node
Node Startup Scripts	Startup Scripts for a Federated Node
Deployment Manager Windows Service	Windows Service for a Deployment Manager
Server Startup Scripts	Startup Scripts for a non-federated WebSphere Application Server
Admin Agent Startup Scripts	Startup Scripts for an Admin Agent Node
Node/Admin Agent Windows Service	Windows Service for a Nodeagent or an Adminagent
WAS Server Windows Service	Windows Service for a WebSphere Application Server

b Select a startup method.

> **IMPORTANT:** If your WebSphere Windows Service uses encoded parameters (for example, if you provided the `-encodeParams` option when the service was created, or if you ran the `-encodeParams <service_name>` command sometime after service creation) you cannot use the integration method described here. Instead, use the instructions in Manual integration with IBM WebSphere Application Server on page 96.

c Click **Next**.

6 On the Integration Details page, provide the details required for integrating with your WebSphere server, including command-line options.

a Type or select the information that your integration type requires. Use the following table to determine what information is needed.

Table 4. Startup Method integration details.

Startup Method	Integration Details Required
Deployment Manager Scripts	Profile Path WebSphere Home
Node Scripts	Profile Path WebSphere Home
Deployment Manager Windows Service	Profile Path Service Name WebSphere Home
Standalone Server Scripts	Profile Path WebSphere Home
Admin Agent Scripts	Admin Agent Profile Path WebSphere Home
Node/Admin Agent Windows Service	Profile Path Service Name WebSphere Home
WAS Server Windows Service	Profile Path Service Name WebSphere Home

The **Profile Path** is the absolute file system path to a WebSphere Profile.

The **Admin Agent Profile Path** is the absolute file system path to a WebSphere Admin Agent Profile.

WebSphere Home is the absolute file system path to the root of a WebSphere Installation.

b Optional — If your application server uses startup scripts with customized names, click the arrow beside **Other Options** to open a list of script paths and names. Type the custom names in the appropriate boxes.

> **NOTE:** Always use forward slashes to separate directory names, regardless of whether the application server is running on Windows or UNIX.

c Optional — Click the arrow beside **Command Line Options** to show the agent and pre-instrumentor option boxes. Use these boxes to provide command-line parameters for the agent or pre-instrumentor.

- To add agent options, type them in the **Agent Options** box.

> **NOTE:** If you need to add more than one option to the Agent Options, separate them with spaces. Do not use quotes.

- To add pre-instrumentation options, type them in the **pre-instrumentor Options** box.

  To set pre-instrumentor options specific to the JVM you are running, type the desired VM options in the pre-instrumentor Options box.

  For example, to set the memory limit for the pre-instrumentor to 256MB, type the following:
  `-Xmx256m`

NOTE: If you need to add more than one option to the pre-instrumentor Options, separate them with spaces. Do not use quotes.

For more information, see Configuring Java Homes pre-instrumentation on page 39.

d Click **Next**.

7 On the Nexus Connection page, select or create the connection that the agent should use to locate and communicate with a Nexus.

TIP: For more information about the Nexus, see Nexus and Agent-Nexus connections on page 43.

If you are using the default Nexus on the embedded Agent Manager, or if you have created a single remote Nexus, a default Nexus Connection is created and pre-selected for you.

Otherwise, select or define at least one Nexus connection. For more information, see Creating or editing Agent-Nexus connection settings on page 44.

Click **Next**.

8 On the Logging page, select or create a logging configuration to control how logs are written, including retention policies and rollovers.

- Select an existing log configuration from the list. The default log provides a basic log configuration that you can use to get started.
- Click **New log configuration**. The New Log Configuration dialog box opens.

  TIP: If you are uncertain about how to adjust these settings, click **Help** for an overview of log targets and their purposes. For details, see Creating or editing logging properties on page 48.

Click **Next**.

9 On the Integration Hosts page, select the monitored host where WebSphere is installed. By default, only eligible hosts (that is, hosts that have an active Foglight Agent Manager of a compatible version installed and running) are listed.

Select the host and click **Next**.

10 On the Agent Configuration page, select the configuration that the agent will use. This agent configuration defines the requests to monitor and collect details.

TIP: You can create agent configurations or edit existing ones from the Java Administration dashboard. For more information, see Managing Java EE Agent Configurations on page 52.

Select a configuration and click **Next**.

11 On the Installation Directory page, define the location for the directory that contains all installation components and log files.

- To accept the default directory, leave the **Directory name** box blank.

NOTE: If you have previously configured the host for agent installation, the existing installation directory is used.

To specify a custom directory, type a path in the **Directory name** box.

Click **Next**.

12 On the Review page, verify the details of the integration.

a By default, the instrumentation script is assigned a name based on the integration type. For example: *WebSphere on Windows: StandaloneServerScripts*.

If you want to customize this name, type the new value in the **Name** box.

b Click **Finish**.

The Application Servers Administration dashboard refreshes, and the integration task appears in the Task History list, where you can monitor its progress.

13 Restart your application server.

The agent starts collecting data after the application server has restarted and the agent is successfully integrated. Use the Application Servers Monitor dashboard to view the application servers status. For more information about the Application Servers Monitor dashboard, see the *Foglight for Application Servers User Guide*.

6

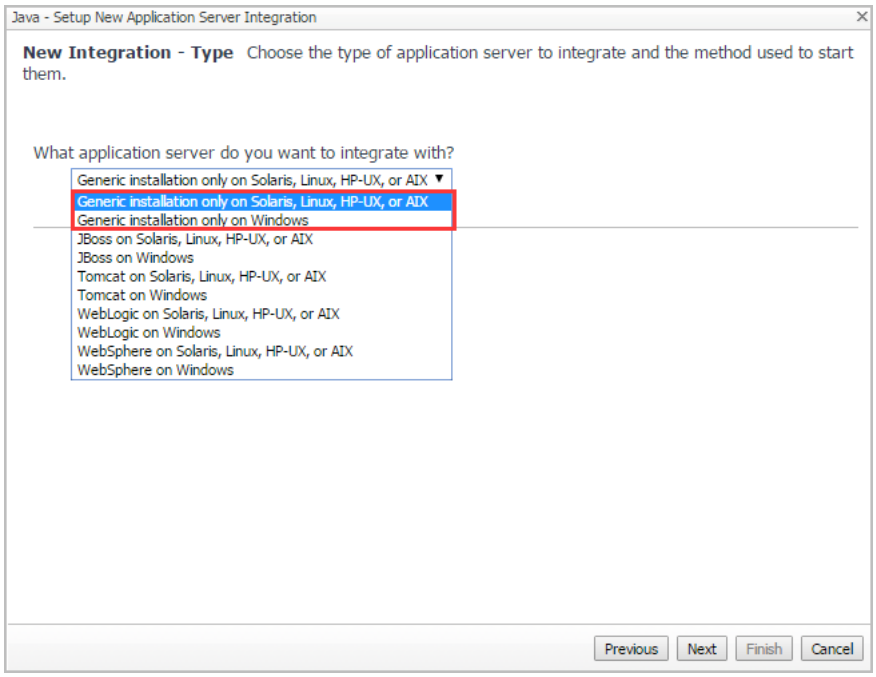
Integrating with WebSphere Liberty

You can use the Java EE Integration Agent to integrate the Java EE agent with an IBM® WebSphere® Liberty on Windows and Linux.

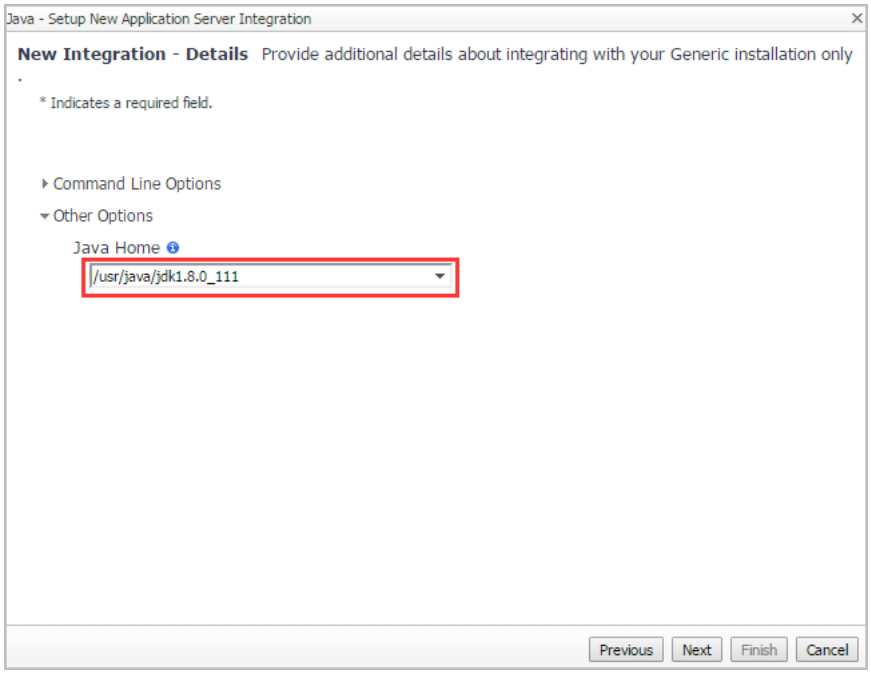
To integrate with WebSphere Liberty on Linux® or on Windows®:

1 Create a *Generic installation only* integration on Linux or Windows. For more information, see Creating a Generic Installation for Manual Java EE Agent Integration on page 39.

- **New Integration - Type**: Select *Generic installation only on Solaris, Linux, HP-UX, or AIX* for Linux, or select *Generic installation only on Windows* for Windows.



- **New Integration - Details**: Type your Java Installation directory in the **Java Home** field.



2 After the generic installation is complete, create a new file named *jvm.options* under the *<Liberty Home>/usr/servers/<Server Name>* directory.

3 Open the new *jvm.options* in a text editor, and then add the following contents.

```
-Dquest.agent.appserverinfo=<Server Name>:Generic:<Liberty Version>:<System Name>
-Xbootclasspath/p:<Bootstrap jar path>/<Bootstrap jar name>.jar
-javaagent:<Performasure agent jar path>/performasure-agent.jar
-Dquest.integrationid=<Integration ID>
```

IMPORTANT: During the integration, change the variables inside *"<>"* to the actual values used in your environment.

IMPORTANT: If the JDK version or any instrument configuration has been changed, make sure to redo the integration to generate a new bootstrap JAR.

NOTE: The value of *quest.integrationid* is generated during the agent setup, and this value is a part of *bootclasspath*. For example, if the *bootclasspath* is "*/opt/quest/foglightagentmanager/agents/JavaEE/5.9.10-20161219-1320/bootstrap/rVbXsTY9XOGObZR/-usr-java-jdk1.8.0_111.jar*", then the value of *quest.integrationid* will be "*rVbXsTY9XOGObZR*".

See the following samples for reference.

- **Linux**: Assume Liberty is installed under "*/opt/IBM/wlp-webProfile7*", Java EE agent directory is "*/opt/quest/foglightagentmanager/agents/JavaEE/5.9.10-20161219-1320*", and the Liberty server to be monitored is "*JDBCSample*". Then add the following contents into *jvm.options* under "*/opt/IBM/wlp-webProfile7/usr/servers/JDBCSample/*".

```
-Dquest.agent.appserverinfo=JDBCSample:Generic:16.0.0.2:LibertyDomain1
-Xbootclasspath/p:/opt/quest/foglightagentmanager/agents/JavaEE/5.9.10-20161219-
1320/bootstrap/rVbXsTY9XOGObZR/-usr-java-jdk1.8.0_111.jar
-javaagent:/opt/quest/foglightagentmanager/agents/JavaEE/5.9.10-20161219-
1320/lib/performasure-agent.jar
-Dquest.integrationid=rVbXsTY9XOGObZR
```

- **Windows**: Assume Liberty is installed under "*C:\IBM\wlp-webProfile7-java8-win-x86_64-16.0.0.2*", Java EE agent directory is "*C:\Quest\foglightagentmanager\agents\JavaEE\5.9.10-20161219-*

1320", and the Liberty server to be monitored is "*JDBCSample*". Then add the following contents into *jvm.options* under "*C:\IBM\wlp-webProfile7-java8-win-x86_64-16.0.0.2\usr\servers\JDBCSample*".

```
-Dquest.agent.appserverinfo=JDBCSample:Generic:16.0.0.2:LibertyDomain2
-Xbootclasspath/p:c:\Quest\foglightagentmanager\agents\JavaEE\5.9.10-20161219-
1320\bootstrap\rVbXsTY9XOGObZR\c--IBM-wlp-webProfile7-java8-win-x86_64-16.0.0.2-
java-java.jar
-javaagent:c:\Quest\foglightagentmanager\agents\JavaEE\5.9.10-20161219-
1320\lib\performasure-agent.jar
-Dquest.integrationid=rVbXsTY9XOGObZR
```

4 Save *jvm.options*.

5 Execute the following command to start WebSphere Liberty using the <Server Name>.

```
<Liberty Home>\bin>server run <Server Name>
```

6 You will get the following output in the Liberty console, which means the Java Agent integration with Liberty is complete.

```
2016-12-26 17:19:40.287 INFO Cartridge For Application Servers 5.9.10 (Build
5.9.10-20161219-1320) on windows x86_64 6.1, Java 1.8.0 (64 bit).; Agent
Type: Generic
2016-12-26 17:19:41.046 INFO Connected to Nexus at APMWin7Server0:41705
2016-12-26 17:19:41.468 INFO Agent started
2016-12-26 17:19:41.609 INFO Application Server started.
2016-12-26 17:19:42.690 INFO Session recording started
```

7

Integrating with Oracle AS

In OracleAS 10g R3, the cluster name is implicitly set to `default` whether a server is running in a cluster or in a standalone environment. However, the Java EE agent can only detect a cluster if the cluster name is explicitly set to something other than `default`.

In Oracle 10g R3, you can set the cluster name in the file *<oracle_home>/opmn/conf/opmn.xml* by adding the `cluster-name` attribute to the `ias-instance` tag for each server in the cluster and setting this attribute to the desired value. For example: `<ias-instance id="ID" name="name" cluster-name="mycluster1">`. You must add the `cluster-name` attribute and set it to the same value for each application server in your cluster.

Use the following procedure to integrate the Java EE agent with an Oracle Application Server version 10g R3 (v10.1.3.[0-5]).

IMPORTANT: The instructions in this procedure assume that no more than two JDKs (1.4, 1.5, or 1.6) are in use simultaneously.

To configure Oracle Application Server 10g R2 or 10g R3:

1. Create a *Generic installation only* integration on the host, with the JAVA_HOME specified for pre-instrumentation. For more information, see Creating a Generic Installation for Manual Java EE Agent Integration on page 39.
2. After the generic installation is complete, a new file appears in the following directory on the machine where you ran the pre-instrumentation.

   Windows: `<VERSIONED_SUBDIRECTORY>\bootstrap\`*<INTEGRATIONID>*

   Unix: `<VERSIONED_SUBDIRECTORY>/bootstrap/`*<INTEGRATIONID>*

   You can find the *<INTEGRATIONID>* in the file created by the Java EE Integration Agent in step 1.

   TIP: Make a backup copy of your original *<oracle_home>/opmn/conf/opmn.xml* file before completing the next step.
3. Open *<oracle_home>/opmn/conf/opmn.xml* in a text editor.
4. Add the following environment variable definitions at the `<ias-instance>` level.

   Code Block:

```
<opmn xmlns="http://www.oracle.com/ias-instance">
<process-manager>
<ias-instance id="xyz" name="xys" cluster-name="XYZ">
   <environment>
      <variable id="JDK14_INTEGRATIONID" value="<jdk14_integrationid>"/>
      <variable id="JDK15_INTEGRATIONID" value="<jdk15_integrationid>"/>
      <variable id="PERFORMASURE_HOME"
            value="&amp;quot;<VERSIONED_SUBDIRECTORY>&amp;quot;"/>
      <variable id="PERFORMASURE_JDK14_OPTIONS" value="-Dquest.debug=0 -
Dquest.integrationid=$JDK14_INTEGRATIONID -
Xbootclasspath^/p:$PERFORMASURE_HOME/bootstrap/$JDK14_INTEGRATIONID/<oracle_jd
k14_bootstrap_jar>" />
```

```
      <variable id="PERFORMASURE_JDK15_OPTIONS" value="-Dquest.debug=0 -
Dquest.integrationid=$JDK15_INTEGRATIONID -
Xbootclasspath^/p:$PERFORMASURE_HOME/bootstrap/$JDK15_INTEGRATIONID/<oracle_jd
k15_bootstrap_jar> -javaagent:$PERFORMASURE_HOME/lib/performasure-
agent.jar=oracle"/>
   </environment>
</ias-instance>
</process-manager>
</opmn>
```

TIP: Forward and backward slashes (“/”, “\”) in opmn.xml-environment-variable-values are automatically converted to the platform-specific path-separator character, unless preceded by a caret (^). The new environment variables can therefore work on both Windows and Unix platforms without requiring any modifications to the slash direction.

c If you are using JDK 1.4:

- Omit the definition of `JDK15_INTEGRATIONID` and `PERFORMASURE_JDK15_OPTIONS`.
- Replace `<oracle_jdk14_bootstrap_jar>` with the file name of the bootstrap JAR file created in the `<VERSIONED_SUBDIRECTORY>/bootstrap/`*<INTEGRATIONID>* directory by the pre-instrumentation task in step 1, for your JDK 1.4 installation (for example, *C--AppServers-oracle-10gR2_10.1.2-jdk.jar*).

d If you are using JDK 1.5 or 1.6:

- Omit the definition of `JDK14_INTEGRATIONID` and `PERFORMASURE_JDK14_OPTIONS`.
- Replace `<JDK15_INTEGRATIONID>` with the name of the directory created in step 1 by the pre-instrumentation task for your 1.5 JDK.
- Replace `<oracle_jdk15_bootstrap_jar>` with the file name of the bootstrap JAR file created in the `<VERSIONED_SUBDIRECTORY>/bootstrap/`*<INTEGRATIONID>* directory by the pre-instrumentation task in step 1, for your JDK 1.5 or 1.6 installation (for example, *C--AppServers-oracle-10gR3_10.1.3-jdk.jar*).

e Replace `<VERSIONED_SUBDIRECTORY>` with the path to the Java EE agent installation directory created in step 1.

NOTE: The **"** characters surrounding `<VERSIONED_SUBDIRECTORY>` are required to enable the use of spaces in the value. Do not remove them.

IMPORTANT: If you are integrating the Java EE agent with OracleAS 10g R2 or 10g R3 in a clustered environment and you have two or more OC4J server instances with the same name, ensure that the id attribute of the process-set tag for each OC4J instance is unique within the cluster.

f Append the `$PERFORMASURE_JDK14_OPTIONS` or `$PERFORMASURE_JDK15_OPTIONS`, according to the version of the JDK you are using, to the `value` attribute of the `java-options` tag that is enclosed in the `start-parameters` tag for each OC4J component that you want instrumented. Employ the JDK version that was used for that OC4J instance.

Example Modifications When Using JDK 1.4:

```
<category id="start-parameters">
<data id="java-options" value="-server ... $PERFORMASURE_JDK14_OPTIONS"/>
</category>
```

Example Modifications When Using JDK 1.5 or JDK 1.6:

```
<category id="start-parameters">
<data id="java-options" value="-server ... $PERFORMASURE_JDK15_OPTIONS"/>
</category>
```

5 Save *opmn.xml*.

6 If opmn is running, you must reload the modified *opmn.xml* file by launching `opmnctl reload`. If opmn is not running, start it now.

7 Stop and restart each OC4J process that was modified in step 5d.

The Oracle application servers 10g R3 is now instrumented and ready to connect to the Management Server.

8

Integrating with Spring Boot for Embedded Tomcat

You can use the Java EE Integration Agent to prepare the artifacts for the embedded Tomcat, to integrate the Java EE agent with Spring Boot startup scripts on Windows and Linux.

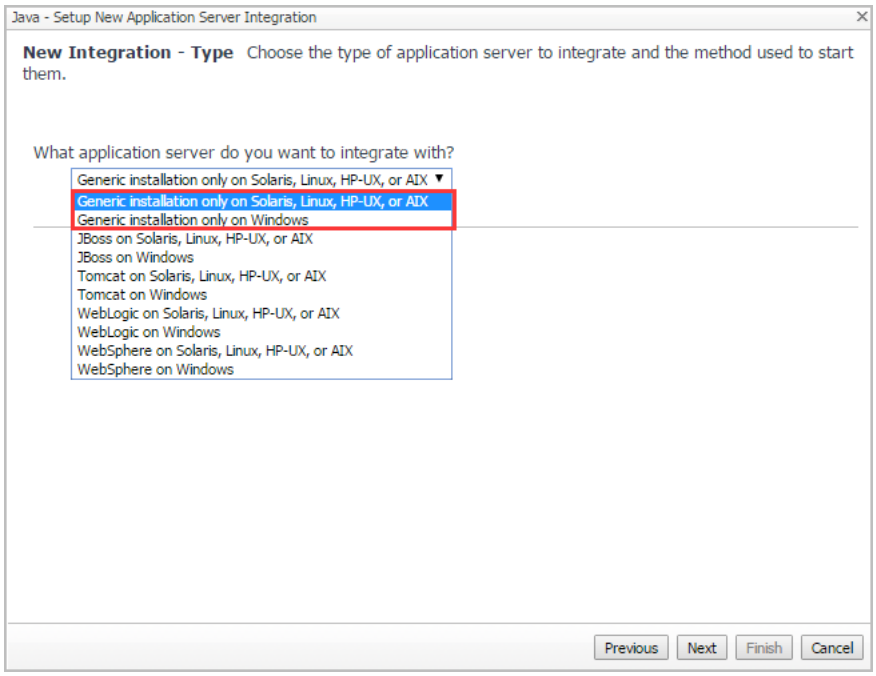
Prerequisites:

- Spring Boot version 1.4.1, version 1.5.2, or version 2.0.0 must be installed in your environment.
- Foglight for Java EE Technologies version 5.9.11 must be installed in your environment.

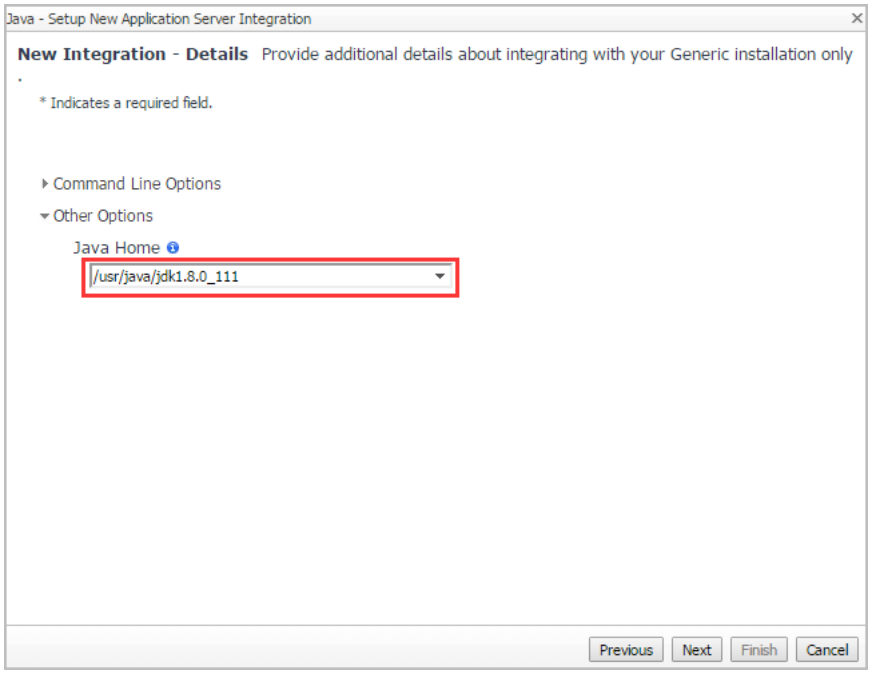
To integrate with Spring Boot on Linux® or on Windows®:

1 Create a *Generic installation only* integration on Linux or Windows. For more information, see Creating a Generic Installation for Manual Java EE Agent Integration on page 39.

- **New Integration - Type**: Select *Generic installation only on Solaris, Linux, HP-UX, or AIX* for Linux, or select *Generic installation only on Windows* for Windows.



- **New Integration - Details**: Type the same Java installation directory to run your Spring Boot application in the **Java Home** field.



2 Follow the wizard to complete the generic installation.

3 After the generic installation is complete, you can update either startup scripts or the command line tool to launch your Spring Boot application.

Consider for example you are using the `java -jar example-application.jar` command to launch your Spring Boot application. Update the original command to the following:

```
java
 -Xbootclasspath/p:<Bootstrap jar path>/<Bootstrap jar name>.jar
 -javaagent:<Performasure agent jar path>/performasure-agent.jar
 -Dquest.integrationid=<Integration ID>
 -jar example-application.jar
```

IMPORTANT: During the integration, change the variables inside *"<>"* to the actual values used in your environment.

IMPORTANT: If the JDK version or any instrument configuration has been changed, make sure to redo the integration to generate a new bootstrap JAR.

NOTE: The value of <*Integration ID*> is generated during the agent setup. In other words, the <*Integration ID*> is the value of the *QUEST_INTEGRATIONID* property in the integration file (either .sh or .cmd file) that is generated after the generic installation and located under the *<FGLAM_HOME>\agents\JavaEE\ folder*.

See the following samples for reference.

- **Linux**: Assume the Java EE agent directory is *"/opt/quest/foglightagentmanager/agents/JavaEE/5.9.11-20170410-1146"*, the "QUEST_INTEGRATIONID" value is "*8tKWIAFGo9xd82p*", and the target Spring Boot application jar is "*example-application.jar*". Then update the original command to the following:

```
java -Xbootclasspath/p:"/opt/quest/foglightagentmanager/agents/JavaEE/5.9.11-20170410-1146/
bootstrap/8tKWIAFGo9xd82p\-usr-Java-jdk1.8.0_51.jar" -javaagent:"/opt/quest/foglightagentmanager/
agents/JavaEE/5.9.11-20170410-1146/lib/performasure-agent.jar"
-Dquest.integrationid=8tKWIAFGo9xd82p -jar example-application.jar
```

- **Windows**: Assume the Java EE agent directory is *"C:\Quest\foglightagentmanager\agents\JavaEE\5.9.11-20170410-1146"*, the *"QUEST_INTEGRATIONID"* value is *"8tKWIAFGo9xd82p"*, and the target Spring Boot application jar is *"example-application.jar"*. Then update the original command to the following:

```
java -Xbootclasspath/p:"C:\Quest\foglightagentmanager\agents\JavaEE\5.9.11-20170410-1146\
bootstrap\8tKWIAFGo9xd82p\C-Java-jdk1.8.0_51.jar" -javaagent:"C:\Quest\foglightagentmanager\
agents\JavaEE\5.9.11-20170410-1146\lib\performasure-agent.jar"
-Dquest.integrationid=8tKWIAFGo9xd82p -jar example-application.jar
```

9

Creating a Generic Installation for Manual Java EE Agent Integration

Use this *Generic installation only* option in the Java agent setup wizard to create and deploy a Java EE Integration Agent to a host without a Java EE agent integration. This option is useful in situations where you need to create an installation directory and configure pre-instrumentation in preparation for manually integrating an agent.

Configuring Java Homes pre-instrumentation

Part of an application server's integration with the Java EE agent includes a step known as pre-instrumentation. Pre-instrumentation prepares a special bootstrap *jar* file required for the operation of the Java EE agent. Many application server startup routines include the pre-instrumentation step as part of the integration with the Java EE agent, but some startup routines do not. In these cases (such as, WebLogic® Windows® Services, and Oracle® Application Server), pre-instrumentation must be run as a separate step from application server startup. Create a *Generic installation only* that includes a value for the JAVA_HOME option for each JDK used in your application server environment. This task creates the initial bootstrap jar, which may be referenced in your integrations, and it can also be used to update the bootstrap jar from time to time, as needed. The Java EE Integration Agent can be configured to complete this extra step as part of its activation routine.

To create a generic installation and pre-instrument:

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, in the Monitoring Agents section, click **Setup agents..**.
3. In the menu that opens, select **Java**.
4. On the Setup Options page of the Java — Setup New Application Server Integration wizard, select **Setup a new application server integration**.

   Click **Next**.
5. On the New Integration — Type page, select either **Generic installation only on Windows** or **Generic installation only on Solaris, Linux, HP-UX, or AIX**.

   Click **Next**.
6. **For Oracle AS only**: On the New Integration — Details page, specify any agent or pre-instrumentor options that are required for integration.

   IMPORTANT: These settings are not applied to any integrations except Oracle AS. Other application servers ignore any values input here.

   a. Click the arrow beside **Command Line Options** to show the pre-instrumentor Options box.

      To set pre-instrumentor options specific to the JVM you are running, type the desired VM options in the **pre-instrumentor Options** box.

      For example, to set the memory limit for the pre-instrumentor to 256MB, type the following:
      `-Xmx256m`

   NOTE: If you need to add more than one option to the pre-instrumentor Options, separate them with spaces. Do not use quotes.

b Click the down arrow beside **Other Options** to show the Java Home box.

If you specify a value for **Java Home**, the Java EE Integration Agent instruments Java Home and creates a bootstrap jar file. If you leave the Java Home box blank, the instrumentation script is created but the bootstrap jar is not. You can use the generated instrumentation script at any time to create or recreate the bootstrap jar.

For more information about pre-instrumentation, see Configuring Java Homes pre-instrumentation on page 39.

c Click **Next**.

7 On the Nexus Connection page, select or create the connection that the agent should use to locate and communicate with a Nexus.

TIP: For more information about the Nexus, see Nexus and Agent-Nexus connections on page 43.

If you are using the default Nexus on the embedded Agent Manager, or if you have created a single remote Nexus, a default Nexus Connection is created and pre-selected for you.

Otherwise, select or define at least one Nexus connection. For more information, see Creating or editing Agent-Nexus connection settings on page 44.

Click **Next**.

8 On the Logging page, select or create a logging configuration to control how logs are written, including retention policies and rollovers.

- Select an existing log configuration from the list. The default log provides a basic log configuration that you can use to get started.
- Click **New log configuration**. The New Log Configuration dialog box opens.

TIP: If you are uncertain about how to adjust these settings, click **Help** for an overview of log targets and their purposes. For details, see Creating or editing logging properties on page 48.

Click **Next**.

9 On the Integration Hosts page, select the host that you want to integrate with. By default, only eligible hosts (that is, hosts that have an active Foglight Agent Manager of a compatible version installed and running) are listed.

Select one or more hosts and click **Next**.

10 On the Agent Configuration page, select the configuration that the agent will use. This agent configuration defines the requests to monitor and collect details.

TIP: You can create agent configurations or edit existing ones from the Java Administration dashboard. For more information, see Managing Java EE Agent Configurations on page 52.

Select a configuration and click **Next**.

11 On the Installation Directory page, define the location for the directory that contains all installation components and log files. To accept the default directory, leave the **Directory name** box empty.

IMPORTANT: If you have previously configured the host for agent installation, the existing installation directory is used.

To specify a custom directory, type a path in the **Directory name** box.

Click **Next**.

12 On the Review page, verify the details of the integration.

a By default, the instrumentation script is assigned a name based on the integration type. For example: *Generic installation only on Windows: Install Configs and pre-instrument*.

If you want to customize this name, type the new value in the **Name** box.

b Click **Finish**.

The Application Servers Administration dashboard refreshes, and the installation task appears in the Task History list, where you can monitor its progress. Wait for the Result column status to change to Success.

Next step:

- Edit your application server startup script as necessary. For more information, see Manually Integrating Application Server on page 88.

10

Managing Java EE Agent Installations, Integrations, and Configurations

The Java Administration dashboard provides a centralized location for managing agent integrations, configurations, and installations. This dashboard is sub-divided into three tabs, each of which focuses on one aspect of agent management.

To access the Java Administration dashboard:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, click the **Java** tab.

Next steps:

- To learn more about agent monitoring configurations, including how to specify custom instrumentation settings, see Managing Java EE Agent Configurations on page 52.
- To learn more about activating and deactivating the Java EE Integration Agent, viewing integration log files, or upgrading installations, see Managing Java EE Installation on page 80.
- To learn more about integration configurations or the Nexus, or to learn how to manage Nexus connections, integrations, or logging information, see the following topics:
  - Managing agent integrations on page 42
  - Nexus and Agent-Nexus connections on page 43
  - Updating integration configuration properties on page 47
  - Java EE agent logging properties on page 47
  - Reintegrating existing targets on page 49
  - Deleting and undoing an integration on page 49
  - Reusing an integration configuration on page 50

Managing agent integrations

The Java EE Integration Agent creates and manages the integration between the Java EE agent and the monitored application server. The integration agent uses an integration configuration that consists of several key pieces of information: the startup method, pre-instrumentation options, the instrumentation settings for recording data, and the Agent Manager host. You can create or edit Nexus connections and logging configurations for a particular integration, or review the integration information. All these tasks are performed through the Java Administration dashboard, Integration Configurations tab.

To access the Java Administration dashboard:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, click the **Java** tab.

3 Click the **Integration Configurations** tab.

From here you can review configured integrations, and manage Nexus connections and logging settings.

Reviewing completed integrations

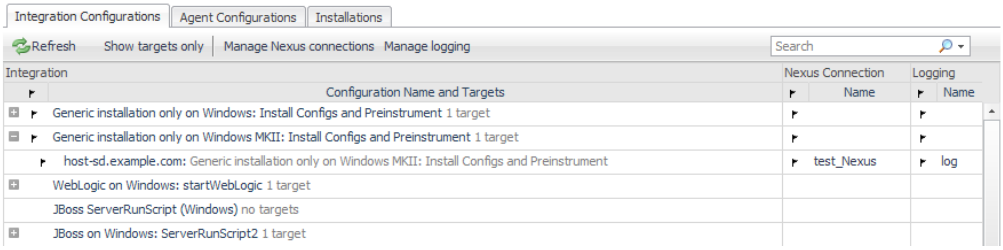
The Integration Configurations tab of the Java Administration dashboard provides an overview of completed integration tasks and targets, and their status.

To view the Integration Configurations tab:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, click the **Java** tab.

3 On the Java view, click **Integration Configurations**.



On this tab, the Configuration Name is the type of integration configuration. For example: *WebLogic on Windows: startWebLogic*, or *JBoss ServerRunScript (Windows)*.

The term *target* is the integrated host that is using the integration configuration. For example: *tor-anoorani: WebLogic on Windows: startWebLogic*.

When the target is correctly configured to connect to a Nexus and has logging properties, the name of the Nexus Connection and the Log configuration appear in the Nexus Connection and Logging columns respectively.

If you only want to see the targets and not the configurations they are using, click **Show targets only**.

Use this tab to manage integrations. From here, you can manage Nexus Connections and logging settings, reintegrate, or delete and undo integration tasks, or delete integrations targets or configurations.

Nexus and Agent-Nexus connections

Foglight for Java EE Technologies includes the Nexus, a central analytical engine that correlates data sent from multiple agents, restores and preserves the temporal order of events across all servers, and provides other basic analysis of the data. The Nexus is essential to the data management facilities of Foglight for Java EE Technologies.

Foglight creates a default Nexus as part of the installation process for Foglight for Java EE Technologies. The default Nexus controls all agent data correlation and submission unless you create another Nexus and assign agents to it instead.

TIP: For more information about the Nexus, see the topic Understanding the default Nexus in the *Foglight for Application Servers Administration and Configuration Guide* or help.

The Nexus connection is the communication link between the Java EE agent and the Nexus. It defines how the agent communicates with the Nexus and controls settings such as retry intervals and timeouts.

The initial agent-Nexus connection is created as part of the agent installation process. You can also create connections, or edit existing ones, and change which agents are using a connection by switching the agent's Nexus connection.

Managing Nexus connections

During the process of creating a Java EE agent, you assign it to an existing Nexus, and an agent-Nexus connection is created. You can also change the connection between an agent and a Nexus, or create a connection and assign it to different agents after they have been created.

Creating or editing Agent-Nexus connection settings

The process for creating an agent-Nexus connection and the process for editing an existing one are similar. The following procedure describes both options and any differences between them.

To change agent Nexus connection settings:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, click the **Java** tab.

3 On the Java view, click the **Integration Configurations** tab.

4 Click **Manage Nexus connections**.

The Manage Nexus Connections page opens.

5 To edit an existing connection, click the name of the connection. In the menu that opens, click **Edit**.

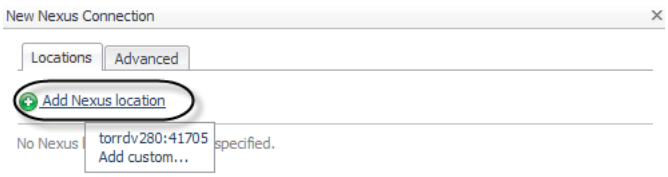
Or

To create a connection, click **New**.

In either case, the Nexus Connection dialog box opens.

6 On the Locations tab, add one or more Nexuses that the agent should attempt to connect to, in the order that you want. The agent attempts to connect to each Nexus in turn until a successful connection is established.

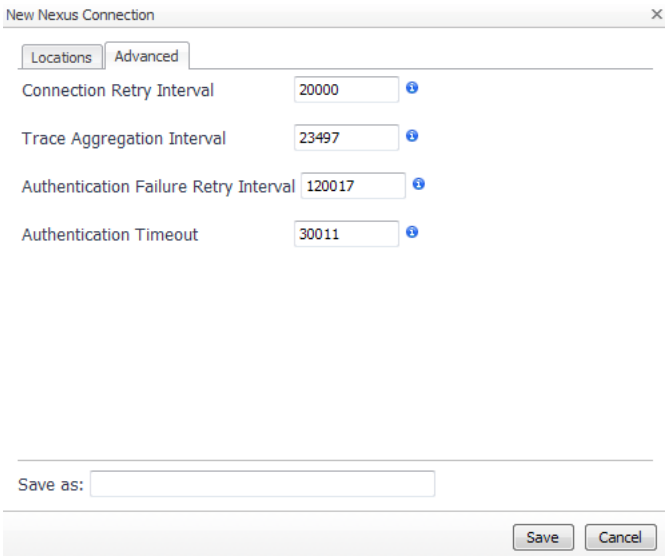
Click **Add Nexus location** and select a Nexus from the list.



TIP: Use the **Add custom** option to define a Nexus location by listen address and port.

7 Optional — Use the up and down arrows to change the order in which the agent attempts the Nexus connections.

8 Click the **Advanced** tab.



9 Edit the connection settings as necessary. Click the icon to view a detailed description of the setting and its effects.

   a Connection Retry Interval — If the connection between an agent and the Management Server is broken, or if the agent fails on its initial attempt to connect to the Nexus, the agent tries to reconnect at regular intervals. The default setting is 20,000 milliseconds (20 seconds). You can increase or decrease this interval. You can also turn off automatic reconnection attempts by setting the interval to zero.

   b Trace Aggregation Interval — When an agent attempts to synchronously connect to the Nexus, it waits for a response for the amount of time specified in this setting. The default setting is just over 23 seconds (23,497 ms). You can increase or decrease this interval.

   c Authentication Failure Retry Interval — If the agent fails on its initial attempt to connect to the Foglight Management Server due to an authentication failure (such as a build number mismatch or expired licensing) it continues trying to connect at regular intervals. The default setting is just over 120 seconds (two minutes). You can increase or decrease this interval. You can also turn off automatic reconnection attempts by setting the interval to zero.

   d Authentication Timeout — When an agent attempts to connect to the Foglight Management Server, by default it waits up to 30 seconds for an authentication response from the Nexus before dropping the connection. You can change the length of time that the agent waits for authentication.

10 For New Nexus Connections only — If you want to reuse this connection and assign it to other agents, type a unique name in the **Save as** box.

11 Click **Save**.

   The Manage Nexus Connections page refreshes. Any changes you have made to the Nexus connection information are listed in the table, along with the modification date and time.

Updating the Nexus connection for a target application server

If you changed the Nexus Connection (see Managing Nexus connections on page 44 for details), you can update any target application servers that are using that Nexus connection through the **Integration Configurations** tab of the Java Administration dashboard.

To update an application server's Nexus connection:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, click the **Java** tab.

3 On the Java view, click the **Integration Configurations** tab.

4 In the **Configuration Name and Targets** list, expand the configuration group (for example, *WebLogic on Windows: startWebLogic*) that contains the target by clicking the '+' sign.

5 Click the name of the target application server that you want to update.

6 In the menu that opens, click **Update Nexus connection**.

Switching a Nexus connection

You can change the Nexus connection between an agent and a Nexus after the agent has been created. For example, you might switch the connection for performance reasons (divide up the agents) or to organize your environment (all agents monitoring a particular application use the same Nexus). Switching the Nexus connection affects all of the agents belonging to the application server integration.

To switch a Nexus connection:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, click the **Java** tab.

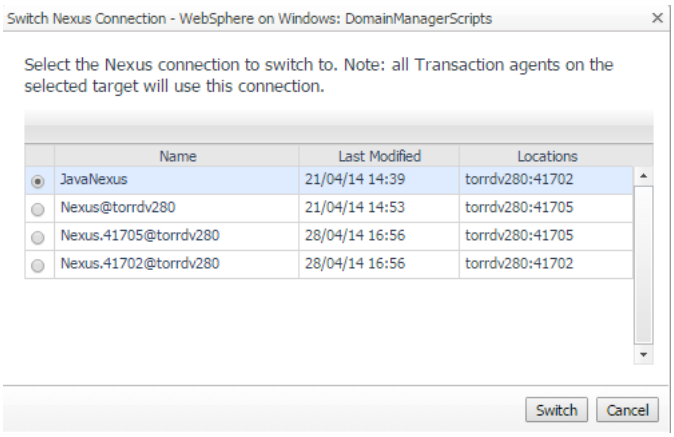
3 On the Java Administration dashboard, click the **Integration Configurations** tab.

4 In the **Configuration Name and Targets** list, expand the configuration group (for example, *WebLogic on Windows: startWebLogic*) by clicking the '+' sign.

5 Click the name of the application server (agent target) that you want to switch the Nexus connection on.

6 In the list that opens, select **Switch Nexus Connection**.

A dialog box opens, listing all the available Nexus Connections for that application server integration.



7 Select the Nexus connection that you want to switch to.

8 Click **Switch** to save your changes.

All of the agents within the application server integration now use this Nexus connection to communicate with the Nexus.

Updating integration configuration properties

Sometimes, due to monitored environment changes, it may be necessary to update the integration configuration properties, such as the target integration script location, or the agent or pre-instrumentor options.

To edit the integration configuration properties:

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, click the **Java** tab.
3. On the Java Administration dashboard, click the **Integration Configurations** tab.
4. In the **Configuration Name and Targets** list, click the name of the configuration that you want to edit (for example, *WebLogic on Windows: startWebLogic*).
5. In the menu that opens, click **Properties**.
6. In the Integration Task Properties dialog box, change the available values as necessary.
7. Click **Save**.
8. Restart the affected application servers.

Java EE agent logging properties

You can configure the following for logs that the Java EE agents produces:

- Log targets, for forwarding log messages to different outputs
- Log level (`ALL`, `DEBUG`, `VERBOSE`, `INFO`, `WARN`, `ERROR`, `FATAL`, and `OFF`) for each log target
- The directory in which log files are created (if other than the default directory, <*VERSIONED_SUBDIRECTORY*>*/logs*)
- A locale setting for changing the language in which log messages are written

You can also configure several log targets:

- Console: Outputs to the application console
- Text File: Outputs to a formatted text file
- Serialized File: Outputs to a serialized object file
- WebSphere: Forwards Agent messages to the WebSphere log system

Java EE agents normally have Console and Text File log targets active. The Serialized File target is active in all log-enabled applications, but its threshold is set to OFF normally. This can be overridden in release-mode by setting its threshold to something other than OFF, preferably ALL. WebSphere is active only for agents monitoring WebSphere.

Each target has a set of properties that can be set to modify its behavior. All targets have one common property: Threshold Log Level. Targets ignore log messages of a level below this threshold. Standard levels are ALL, DEBUG, VERBOSE, INFO, WARN, ERROR, FATAL and OFF, in descending order of verbosity. A threshold of OFF effectively disables a target. File-based targets do not create files when their threshold is OFF.

Create a basic log configuration as part of the agent installation process. You can edit that configuration, or create ones, at any other time.

Creating or editing logging properties

The process for creating a log configuration and the process for editing an existing one are similar. The following procedure describes both options and any differences between them.

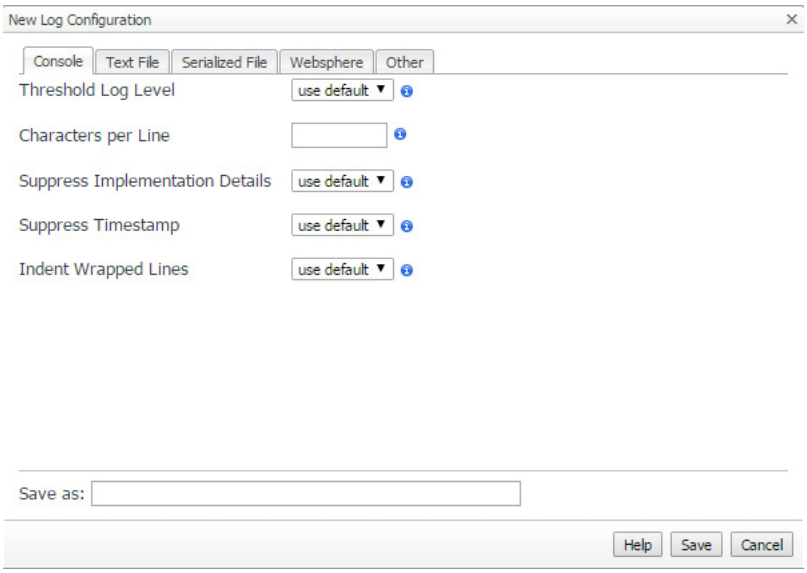
To edit or create a logging configuration:

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, click the **Java** tab.
3. On the Java Administration dashboard, click the **Integration Configurations** tab.
4. Click **Manage logging**. The Logging page opens.
5. Click the name of the log that you want to edit. In the menu that opens, click **Edit**.

   Or

   Click **New**.

   In either case, the Log Configuration dialog box opens.



> **TIP:** If you are uncertain about how to adjust these settings, click **Help** for an overview of log targets and their purposes.

6. Set the values for each of the logging target types: Console, Text File, Serialized File, WebSphere, and Other, as necessary. If you need more information about a particular setting, click the icon.
7. New configurations only — Type a unique and recognizable name in the **Save as** box.
8. Click **Save**.

The next time that you create an agent, you can assign the saved log configuration to it. You can also update any integrations that are already using the log configuration.

To update log settings for an existing configuration:

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, click the **Java** tab.
3. On the Java view, click the **Integration Configurations** tab.
4. In the list of **Configuration Name and Targets**, expand the configuration group by clicking the '+' sign, then click the name of the target you want to update.

5 In the list that opens, click **Update logging**.

Reintegrating existing targets

To keep your application servers up to date, you can edit the properties of an integration task, and then reintegrate either a single target, or all targets using that particular configuration.

To reintegrate a target:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, click the **Java** tab.

3 On the Java view, click the **Integration Configurations** tab.

4 In the list of **Configuration Name and Targets**, click the name of the integration configuration task. For example: *Generic installation only on Windows: Install Configs and Preinstrument*.

5 In the list that opens, click **Properties**.

6 In the Integration Task Properties dialog box, update the Java Home or Preinstrumentor Options as necessary.

7 Click **Save**.

8 In the Integration dialog box, click **OK** if you want to reintegrate all targets that use the configuration.

Or

Clear the check box if you want to reintegrate only certain targets, then click **OK**.

9 Optional — In the list of **Configuration Name and Targets**, do one of the following:

- To reintegrate a single target, expand the configuration group by clicking the '+' sign, then click the name of the target you want to reintegrate (for example, *appserver.example1.com: WebLogic on Windows: startWebLogic*). In the menu that opens, click **Reintegrate**.
- To reintegrate all targets that are using the same integration configuration, click the configuration name (for example, *WebLogic on Windows: startWebLogic*). In the menu that opens, click **Reintegrate targets**.

Deleting and undoing an integration

There are two options for removing integrations from targets:

- *Delete only* — use to remove integrations on decommissioned hosts or to delete stale integrations that have been superseded. Deletion permanently removes all records of the integration target.
- *Delete and undo integration* — use to restore the target host to its pre-integration state. This option is useful when the integration is still active on the target.

To delete an integration:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, click the **Java** tab.

3 On the Java view, click the **Integration Configurations** tab.

4 In the list of **Configuration Name and Targets**, expand the configuration group by clicking the '+' sign, then click the name of the target you want to delete.

5 In the list that opens, select either **Delete only**, or **Delete and undo integration**, as necessary.

Reusing an integration configuration

The first time you install a Java EE agent, create an integration configuration that is specific to the type of application server you are integrating with. During the process of creating the integration, you can choose to save it as a reusable task. Reusing a saved integration configuration allows you to quickly set up multiple agents with the same configuration.

To reuse an integration configuration:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, in the Monitoring Agents section, click **Setup agents..**.

3 In the menu that opens, select **Java**. The wizard opens.

4 On the Setup Options page of the wizard, click **Reuse previous application server integration details**.

Click **Next**.

5 On the Integration Configuration page, select a saved integration task from the list.

Click **Next**.

6 On the Nexus Connection page, select or create the connection that the agent should use to connect to the Nexus.

TIP: For more information about the Nexus, see Nexus and Agent-Nexus connections on page 43.

Select a Nexus connection from the list and click **Next**.

7 On the Logging page, select or create a logging configuration to control how logs are written, including retention policies and rollovers.

- Select an existing log configuration from the list. The default log provides a basic log configuration that you can use to get started quickly.
- Click **New log configuration**. The New Log Configuration dialog box opens.

TIP: If you are uncertain about how to adjust these settings, click **Help** for an overview of log targets and their purposes. For more information, see Creating or editing logging properties on page 48.

Click **Next**.

8 On the Integration Hosts page, select the host that you want to integrate with. By default, only eligible hosts (that is, hosts that have an active Foglight Agent Manager of a compatible version installed and running) are listed.

Select the host and click **Next**.

9 On the Agent Configuration page, select the configuration that you want the agent to use. The configuration controls request monitoring and collection details information.

TIP: For more information, see Managing Java EE Agent Configurations on page 52.

Select the configuration and click **Next**.

10 On the Installation Directory page, define the location for the directory that contains all installation components and log files. To accept the default directory, leave the Installation Directory box blank.

IMPORTANT: If you have previously configured the host for agent installation, the existing installation directory is used.

To specify a custom directory, type a path in the **Directory name** box.

Click **Next**.

11 On the Review page, verify the details of the integration.

Click **Finish**.

The Application Servers Administration dashboard refreshes, and the integration task appears in the Task History list, where you can monitor its progress.

12 Restart your application server.

The agent starts collecting data after the application server has restarted and the agent is successfully integrated.

Use the Application Servers Monitor dashboard to view the application servers status. For more information about the Application Servers Monitor dashboard, see the *Foglight for Application Servers User Guide*.

11

Managing Java EE Agent Configurations

The agent configuration controls what data the agent collects. These settings are stored in a file called *instrumentation.config*. You can edit the instrumentation settings through a GUI editor that is available on the Agent Configurations tab of the Java Administration dashboard.

To manage agent configurations:

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, click the **Java** tab.
3. On the Java view, click the **Agent Configurations** tab.

   From here, you can create an agent configuration or edit an existing one (*instrumentation.config*), or edit version-specific agent configuration files.

For more information, see:

- Managing other Java EE agent configuration files on page 78

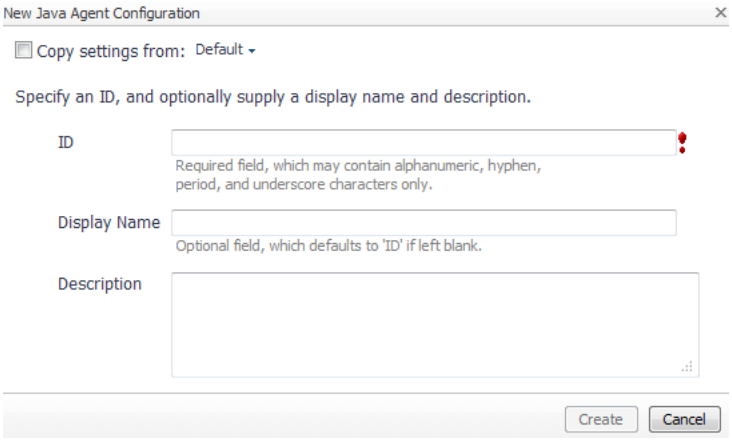
Creating agent instrumentation configurations

Foglight for Java EE Technologies includes a default, basic configuration for the Java EE agent that is defined in the *instrumentation.config* file settings. Java EE agents use the default configuration unless you explicitly create and assign a new one to them. You can create configurations by copying settings from the default configuration.

IMPORTANT: If you want to edit the default configuration, it is recommended that you create a copy of it and edit the copy instead, then assign it to the appropriate agents.

To create an agent configuration:

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, click the **Java** tab.
3. On the Java vew, click the **Agent Configurations** tab.
4. On the Agent Configurations tab, click **New configuration...**. The New Java Agent Configuration dialog box opens.



5 Optional — To use an existing configuration as the basis for your new configuration, select the **Copy settings from** check box and select an existing configuration from the list.

6 Type the identifier (**ID**) for this configuration in the first box. The ID can contain the following types of characters: alphanumeric, hyphens, periods, and underscores. No other characters are allowed.

7 Optional — Type a descriptive name for the configuration in the **Display Name** box.

> **IMPORTANT:** If you leave this box empty, the display name is the same as the ID.

8 Optional — Type a short description for the configuration in the **Description** box.

9 Click **Create**.

The Agent Configuration tab refreshes and the new configuration appears in the list.

> **TIP:** Any new configuration that was not copied from an existing configuration contains the default monitoring settings.

To view and edit the instrumentation settings, see Editing agent instrumentation configurations on page 53.

Editing agent instrumentation configurations

The agent configuration editor provides a GUI for editing instrumentation settings. This simplifies the process for users.

> **NOTE:** In previous versions of Foglight for Java EE Technologies, these settings were manually edited in the *instrumentation.config* file. If you prefer this method, see Manually editing instrumentation settings on page 55.

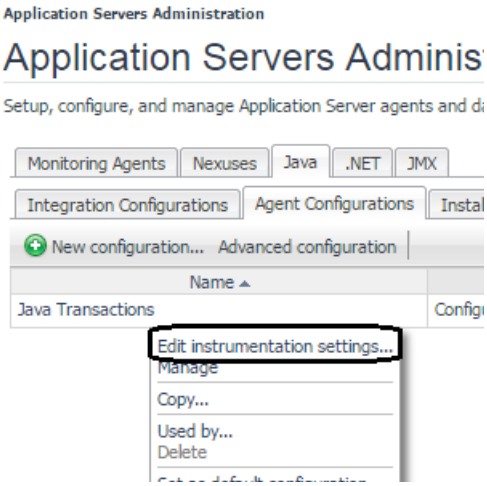
To edit an agent instrumentation configuration:

1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, click the **Java** ab.

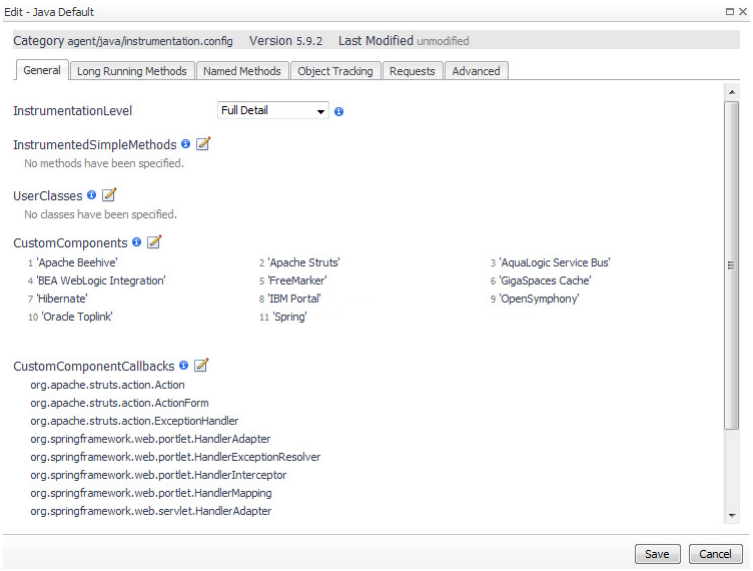
3 On the Java view, click the **Agent Configurations** tab.

4 On the Agent Configurations tab, click the name of the agent configuration that you want to edit.



5 On the menu that opens, click **Edit instrumentation settings...**

The Edit — <configuration name> dialog box opens.



For information about the settings available on each tab, see the following topics:

- General instrumentation settings on page 55
- Customizing Long Running Methods on page 66
- Named Methods settings on page 68
- Object Tracking settings on page 70
- Requests settings on page 72
- Customizing instrumented classes on page 78

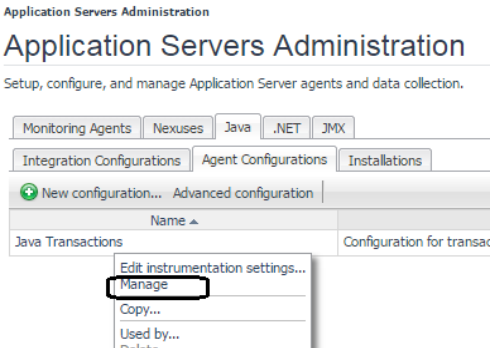
IMPORTANT: If you change the instrumentation, restart the affected application servers.

Manually editing instrumentation settings

It is possible to manually edit the settings in the *instrumentation.config* file instead of using the GUI editor. If you are comfortable and familiar with these settings from previous versions of Foglight for Java EE Technologies, you may prefer to use the text editor.

To manually edit the instrumentation settings:

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, click the **Java** tab.
3. On the Java view, click the **Agent Configurations** tab.
4. On the Agent Configurations tab, click the name of the agent configuration that you want to edit.



5. In the menu that opens, click **Manage**.

   The Java Agent Configuration: <*configuration name*> view opens.
6. From the **Version** list, select the version of the Java EE agent (for example 5.9.2 or 5.8) that you want to work with.
7. Click the name of the file that you want to edit. In this case, click **instrumentation**.
8. In the menu that opens, select **Text edit**.
9. Type the required information in the text editor.
10. After you set all the required values, click **Save**.
11. Restart the application server.

General instrumentation settings

The General instrumentation settings for a Java EE agent include instrumentation levels, simple methods, user classes, custom components, and custom component callbacks. Changing any of these settings can have performance implications. Review the provided information before changing any settings.

Changing the instrumentation level for Request Metric data collection

The Java EE agent uses instrumentation for request monitoring and to collect single traces. This instrumentation does add some overhead to your monitored servers and the information collected indirectly applies load to the Foglight Management Server. However, if you only require single traces collected at component level, or you do not require request monitoring at all, you can reduce or turn off instrumentation using the `Instrumentation`

`Level` setting. This reduces the overhead on your monitored servers and the load on the Foglight Management Server.

The default instrumentation level is `FullDetail`. At full detail level, request monitoring can be enabled and, when single traces are collected, all nodes in the call tree are included. This level provides the most flexibility in terms of request monitoring and single trace collection and should be acceptable for most customers.

If you require request monitoring, but do not require full detail single traces, instrumentation can be set to `Component Detail`. At component detail level, you can enable request monitoring, but when single traces are collected, they are collected at only the component technology detail level, reducing the complexity of the call tree. This level is more restrictive in what information can be collected for single traces, but reduces both the agent overhead on the monitored server and the load on the Management Server.

If you require request monitoring but only require request metrics for HTTP, JMS, and RMI requests, instrumentation can be set to `Basic Detail`. At this level, no breakdowns are collected and, if single traces are collected, only HTTP, JMS, and RMI nodes are included in the Request Trace call tree. No other component or full detail methods are collected, therefore reducing both the agent overhead on the monitored server and the load on the Management Server, compared to the `Component Detail` instrumentation level.

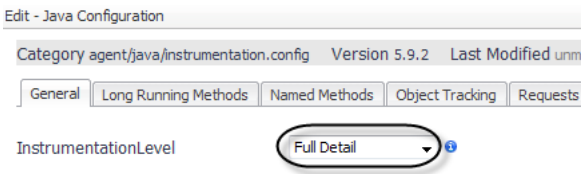
If you do not require request monitoring or the ability to capture single traces, instrumentation for this purpose can be disabled by setting the instrumentation level to `No Detail`. At this setting, request monitoring cannot be enabled and you are unable to capture single traces at any detail level. This removes instrumentation overhead on the monitored server and greatly reduces load on the Management Server.

IMPORTANT: If you want to monitor the classes or interfaces that custom components use to call customer written code, see Changing custom component callbacks settings on page 64.

To set the instrumentation level:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

1 On the General tab of the Edit dialog box, select the instrumentation level from the **InstrumentationLevel** list.



- Full Detail — Instruments at full detail level. All methods in the call tree are captured.
- Component Detail — Instruments at the component level only. Methods from defined component technology, including custom components, are captured.
- Basic Detail — Instruments at a basic level. Only request metrics for HTTP, JMS, and RMI request are collected. No other component or full detail methods are captured.
- No Detail — Turns off instrumentation that is not essential for running an agent. As a result, no request metrics are collected, and single traces call trees cannot be captured.

NOTE: As a result of setting instrumentation to this level, no data is displayed in the **Application Servers Monitor > Requests** tab.

2 Click **Save**.

3 Restart your application server.

Manual configuration — InstrumentationLevel

If you manually edit the *instrumentation.config* file, use the following syntax to set the instrumentation level:

Table 5. Instrument level syntax.

Level	Syntax
FullDetail	InstrumentationLevel = "FullDetail";
ComponentDetail	InstrumentationLevel = "ComponentDetail";
BasicDetail	InstrumentationLevel = "BasicDetail";
NoDetail	InstrumentationLevel = "NoDetail";

Configuring instrumented simple method settings

Simple methods are methods for which instrumentation would add more byte-code and overhead to the original method.

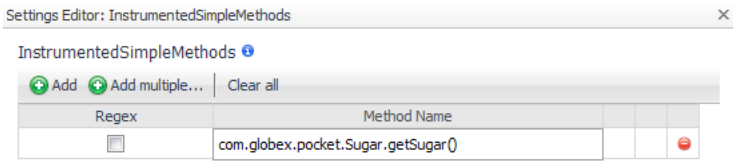
By default, simple methods are not instrumented to reduce overhead of a sampled request and the complexity of single traces. When simple methods are not instrumented, any time spent in those methods is attributed to the caller.

CAUTION: Simple methods usually run quickly, so instrumentation adds unnecessary overhead and complexity.

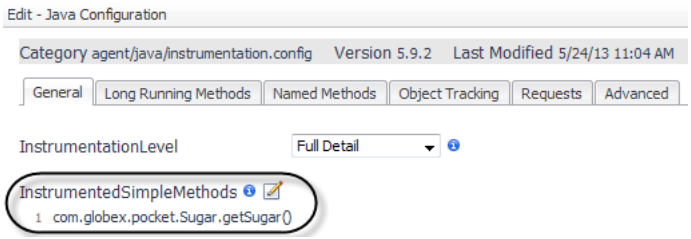
To explicitly name methods for instrumentation:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

1. On the General tab of the Edit dialog box, click **Edit** beside **InstrumentedSimpleMethods**.
2. In the Settings Editor dialog box, click **Add**.
3. If you want the method name to be treated as a regular expression, select the **Regex** check box.
4. Click in the **Method Name** box and type the name of the method you want to instrument. For example, in the following image, the `getSugar()` method is instrumented even if it is detected as a simple method.



5. Click **OK**.



6. Click **Save**.
7. Restart your application server.

Manual configuration — InstrumentedSimpleMethods

If you manually edit the *instrumentation.config* file, use the following syntax to enable simple method instrumentation:

- `InstrumentSimpleMethods = true;`

To explicitly name methods to be instrumented, even if they are detected as simple methods:

- `InstrumentedSimpleMethods = MethodList(};`

  For example, to instrument the `getSugar()` method even if it is detected as a simple method:

  ```
  InstrumentedSimpleMethods = MethodList(
  include "com.globex.pocket.Sugar.getSugar()",
  );
  ```

Changing user classes settings

Use the `User Classes` setting to customize the classes that are treated as user or application code.

Excluding a class prevents the class from being instrumented as user code. It may still be partially instrumented if it implements important application server functionality (such as RMI communications).

Including a class ensures that the class is specified and instrumented as user code.

NOTE: This setting is only valid for FullDetail, ComponentDetail, and BasicDetail instrumentation levels. Changing its value has no effect if the Instrumentation Level is set to NoDetail. For details about setting instrumentation levels, see “Changing the Instrumentation Level for Request Metric Data Collection” on page 55.

For more information about instrumented class and custom components, see: Customizing instrumented classes on page 78, and Changing custom component settings on page 60.

To exclude a user class or package:

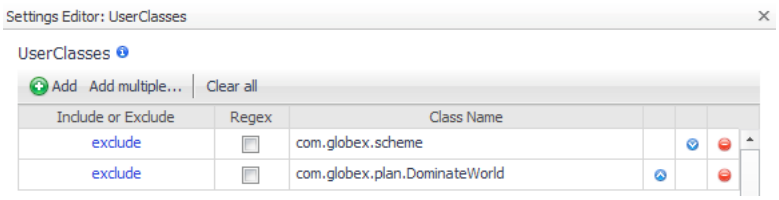
TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

1. On the General tab of the Edit dialog box, click **Edit** beside **UserClasses**.
2. In the Settings Editor dialog box, click **Add**.

   A new row appears. By default, the Include or Exclude option is set to include.
3. Click **include**. The setting switches to *exclude*.
4. If you want the class name to be treated as a regular expression, select the **Regex** check box.
5. Click the **Class Name** box and type the name of the class you want to exclude.

   For example, suppose that you have an instrumented user class named `com.globex.plan.DominateWorld`, but the instrumentation is causing excessive load on the agent or Nexus. You can exclude this class.

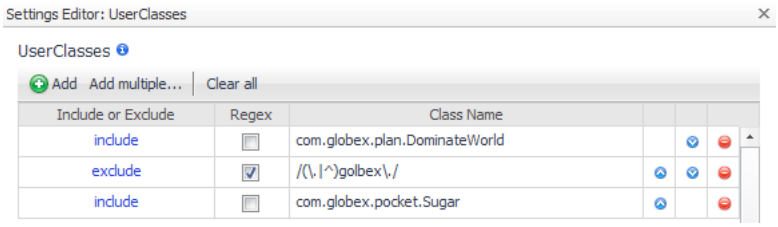
   You can also exclude entire packages. For example, `com.globex.scheme`.



The class `com.globex.plan.DominateWorld` is excluded, as well as all classes in the package "`com.globex.scheme`".

> **IMPORTANT:** Order is important. You can use multiple include and exclude elements to build a pattern, but it is the first element that matches that controls the outcome.

For example:



In this case, the class `com.globex.plan.DominateWorld` is included, but the class `com.globex.pocket.Sugar` is excluded because it matches the `exclude` pattern on the second line. Also, any classes in `net.globex.` package or the root `globex.` package are excluded since they match the `exclude` regular expression.

> **TIP:** For more information about regular expressions, see the Appendix: Regular Expressions in the *Foglight for Application Servers User Guide*.

6 In the Settings Editor dialog box, click **OK** to save your changes to the user class instrumentation.

7 In the Edit dialog box, click **Save**.

8 Restart your application server.

Manual configuration — UserClasses

If you manually edit the *instrumentation.config* file, use the following syntax to enable user class or package instrumentation:

- `UserClasses = ClassList();`

Example #1

One use for configuring user classes is to exclude a class because instrumenting it is causing excessive load on the agent or Nexus. You can also include or exclude entire packages:

```
UserClasses = ClassList(
exclude "com.globex.plan.DominateWorld",
exclude "com.globex.scheme."
);
```

The class `com.globex.plan.DominateWorld` is excluded, as well as all classes in the package "`com.globex.scheme`".

Example #2

Regular expressions can also be used to include or exclude classes. Also, multiple include and exclude elements can appear in the block, but it is the **first** element that matches that controls the outcome.

```
UserClasses = ClassList(
include "com.globex.plan.DominateWorld",
exclude /(\.|^)globex\./,
include "com.globex.pocket.Sugar"
);
```

In this case, the class `com.globex.plan.DominateWorld` is included, but the class `com.globex.pocket.Sugar` is excluded because it matches the `exclude` pattern on the third line. Also, any classes in `net.globex.` package or the root `globex.` package are excluded since they match the exclude regular expression.

For more information about regular expressions, see the Appendix: Regular Expressions in the *Foglight for Application Servers User Guide*.

Changing custom component settings

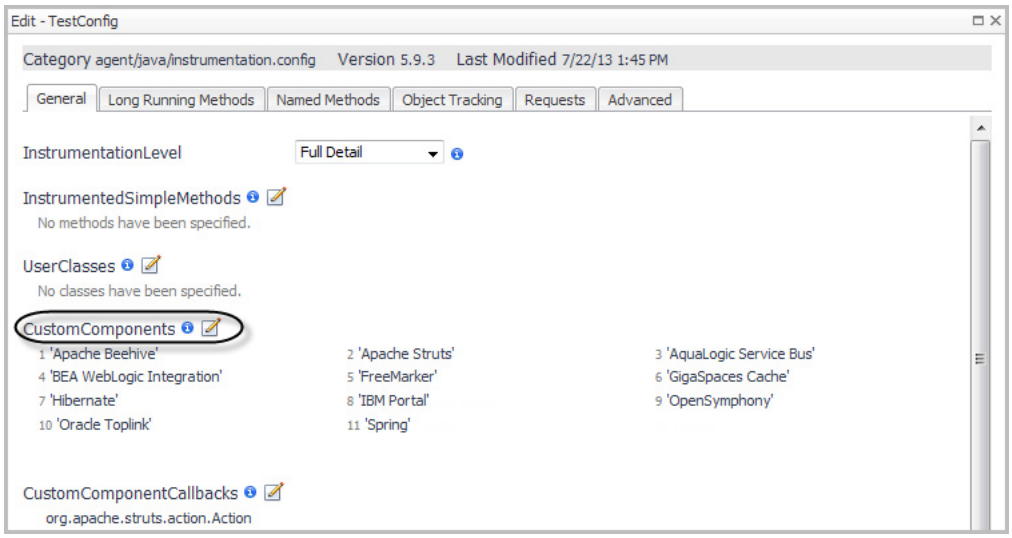
The `Custom Components` setting extends the default set of J2EE components to be included when recording a session at the Component Detail instrumentation level.

Sometimes, your distributed application may use third-party technologies that are not part of the default list of items included in the default instrumentation for the Java EE agent. If you want to include these in the component-only view, you can use custom component instrumentation.

The default custom component settings are shown in the following screenshot. Some customizations may be required to add additional classes if these frameworks have been extended. You can add package listings if necessary.

The method signature can be specified but it must be specified using the internal JVM format. For more information, see the Oracle Java SE Documentation: http://docs.oracle.com/javase/specs/#7035.

Figure 1. Default custom components



To configure custom components:

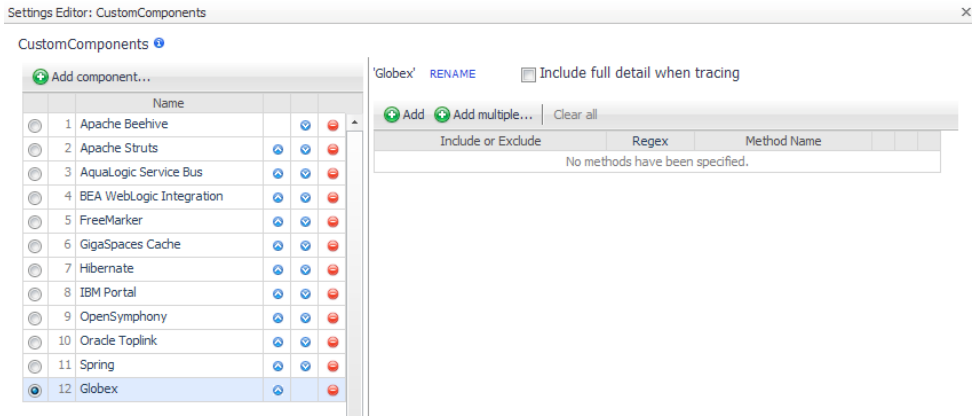
TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

1. On the General tab of the Edit dialog box, click **Edit** beside **CustomComponents**.
2. In the Settings Editor dialog box, you can add additional custom components, or include or exclude methods from the default components.

   For example, you could add the component `Globex`, and then specify methods to include or exclude.

   a. Click **Add component**.

   b. In the Add component dialog box, type a name for the component. For example, type `Globex`.



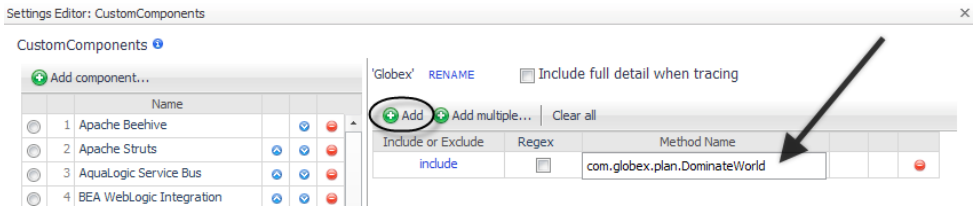
The new component appears at the bottom of the list.

c Optional — Include full detail when tracing.

By default, the Java EE agent does not collect method level detail within custom components. If you want traces for custom components you specify to include method level detail, select the **Include full detail when tracing** check box.

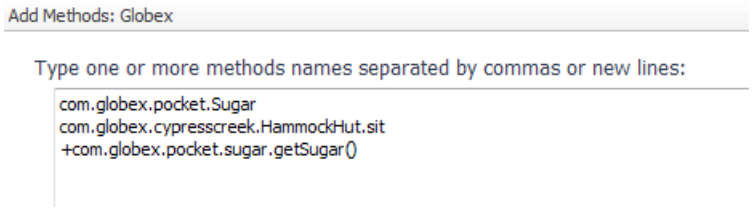
d Optional — Add methods to include or exclude.

Click **Add** to add a single method at a time.



Or

Click **Add multiple** to open a text box where you can type several methods.



3 After you have added as many methods as you want, click **OK**.

4 In the Edit dialog box, click **Save**.

Any components you added appear in the CustomComponents list and, if you enabled full detail while tracing for any methods, the text `FULL DETAIL` appears beside the custom component name.

5 Restart your application server.

Manual configuration — CustomComponents

If you manually edit the *instrumentation.config* file, use the following syntax to specify custom component instrumentation.

To manually configure custom components:

1 Edit `CustomComponents`. Specify the name of each component in a method list, either by entering the name or by forming a regular expression that matches the name or pattern. Enter the include element before each item in the list. For example:

```
CustomComponents = {
     "MQSeries": MethodList(
         include /com\.ibm\.mq\..*/
     ),
     "TopLink": MethodList(
         include "oracle.toplink."
     ),
     "Globex": MethodList(
         include "com.globex.plan.DominateWorld",
         include "com.globex.pocket.Sugar",
         include "com.globex.cypresscreek.HammockHut.sit",
         exclude /(\.|^)globex\./
         include
"com.globex.plan.DominateWorldServlet.doGet(Ljavax/servlet/http/HttpServletReq
uest;Ljavax/servlet/http/HttpServletResponse;)V",
         include
"com.globex.plan.DominateWorldHelper.dominate(Ljava/lang/String;Z[B)I",
     )
 };
```

When the Java EE agent matches method names to the methods in the list, it interprets them as custom components.

2 After editing the *instrumentation.config* settings, ensure that it is read by restarting the instrumented application servers.

Default custom component list

The default custom component settings are shown below. Some customizations may be required to add additional classes if these frameworks have been extended. In the example below, you can uncomment and add package listings if necessary.

```
CustomComponents = {


   "Apache Beehive": MethodList (
      exclude "org.apache.beehive.netui.tags.",
      include "org.apache.beehive.",
   ),


   "Apache Struts": MethodList (
      exclude "org.apache.struts.action.ActionServlet",
      exclude "org.apache.struts.taglib.",
      include "org.apache.struts.",
```

```
      include "org.apache.strutsel.",
   ),

   "AquaLogic Service Bus": MethodList (
      include "com.bea.wli.config.AppListener",
      include "com.bea.wli.sb.",
      include "stages.logging.",
      include "stages.publish.",
      include "stages.routing.",
      include "stages.transform.",
   ),

   "BEA WebLogic Integration": MethodList (
      # Consult BEA WebLogic Integration javadoc for more info.
      # http://e-docs.bea.com/wli/docs81/javadoc/index.html
      include "com.bea.wli.bpm.runtime.",
   ),

   "FreeMarker": MethodList (
      include "freemarker.template.Template",
      include "freemarker.template.Configuration",
   ),

   "GigaSpaces Cache": MethodList (
      include "com.j_spaces.",
   ),

   "Hibernate": MethodList (
      include "org.hibernate.",
      include "net.sf.hibernate.",
   ),

   "IBM Portal": MethodList (
      # Consult IBM WebSphere Portal javadoc for more info.
      # http://www-
106.ibm.com/developerworks/websphere/zones/portal/portlet/5.0api/WPS/
      #
      # include "com.ibm.wps.portlet.",
      # include "com.ibm.wps.portletservice.",
      include "org.apache.jetspeed.portlet."
```

```
),

"OpenSymphony": MethodList (
   include "com.opensymphony.",
),

"Oracle Toplink": MethodList (
   # Consult Oracle TopLink javadoc for more info.
   # http://download-east.oracle.com/docs/cd/B10464_04/web.904/b10491/index.html
   #
   include "oracle.toplink."
),

"Spring": MethodList (
   exclude "org.springframework.web.servlet.tags.",
   include "org.springframework.",
),
```

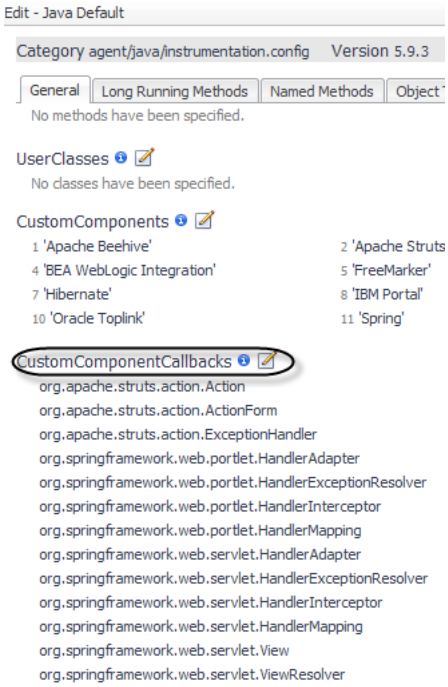
Changing custom component callbacks settings

Use the `Custom Component Callbacks` setting to specify the classes or interfaces that the custom components use to call customer written code. The default instrumentation settings include some custom component callbacks for Apache Struts and Spring.

> **NOTE:** This setting is only valid for Full Detail and Component Detail Instrumentation Levels. Changing its value has no effect if the instrumentation level is Basic Detail or No Detail. For information about instrumentation levels, see Changing the instrumentation level for Request Metric data collection on page 55.

For more information, see Appendix: Java EE Application Methods on page 102.

Figure 2. Default custom component callback settings



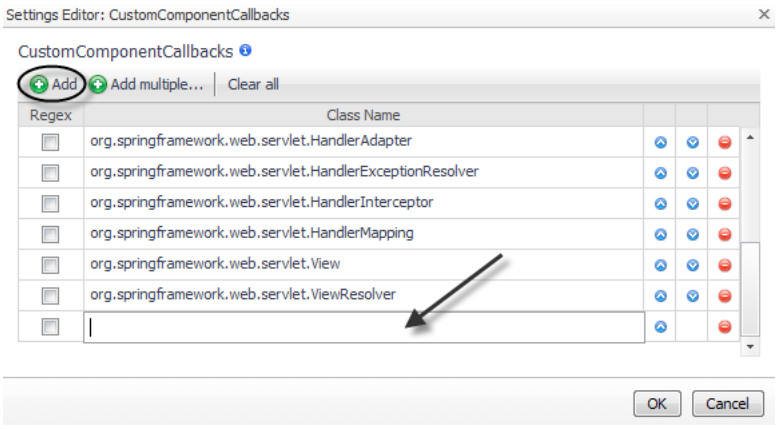
To add custom component callbacks:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

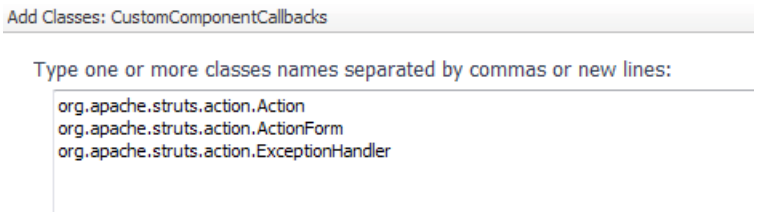
1 On the General tab of the Edit dialog box, click **Edit** beside **CustomComponentCallbacks**.

2 In the Settings Editor, you can add, remove, or change the order of callbacks.

Click **Add** to add a single class at a time.



Click **Add multiple** to open a text box where you can type several classes.



3 After you have added as many classes as you want, click **OK**.

4 In the Edit dialog box, click **Save**.

5 Restart your application server.

Manual configuration — CustomComponentCallbacks

If you manually edit the *instrumentation.config* file, use the following syntax to specify custom component callback instrumentation.

```
CustomComponentCallbacks = ClassList (
# Apache Struts 1.x
include "org.apache.struts.action.Action",
include "org.apache.struts.action.ActionForm",
include "org.apache.struts.action.ExceptionHandler",
);
```

Customizing Long Running Methods

Use the `LongRunningUserMethods` setting to customize the collection of performance data from methods that run over a long period of time. By default, the Java EE agent waits for a method to finish running before treating the collected data as complete. However, for methods such as the `run()` method of a background thread or other methods that run over a long period of time, it may be preferable to treat the data as complete at specific points in the code. This makes the data available in small pieces as the method runs instead of as one large piece when the long running method completes. Changing any of these settings can have performance implications.

The Java EE agent pauses at waypoint methods (`LongRunningUserMethodWaypoints`) to treat the data as complete before continuing. For best performance, use a minimal set of waypoint methods.

NOTE: This setting is only valid for Full Detail and Component Detail Instrumentation Levels. Changing its value has no effect if the instrumentation level is Basic Detail or No Detail. For information about instrumentation levels, see Changing the instrumentation level for Request Metric data collection on page 55.

To add long running user methods:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

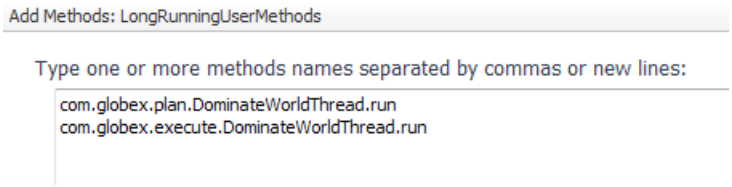
1 Click the **Long Running Methods** tab of the Edit dialog box.

2 To add long running methods, click **Edit** beside **LongRunningUserMethods**.

3 In the Settings Editor, you can add, remove, or change the order of long running methods.

Click **Add** to add a single method at a time.

Click **Add multiple** to open a text box where you can type several methods.



For example, to measure each execution through a loop in a long running `run()` method, add the method: `com.globex.plan.DominateWorldThread.run`.

4 After you have added as many methods as you want, click **OK**.

5 Next, add one or more waypoints.

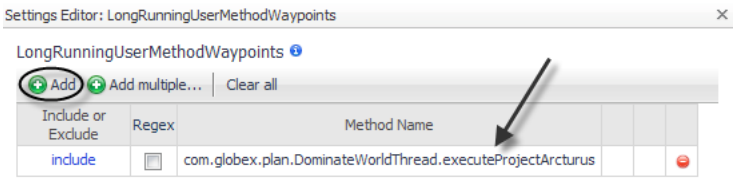
If you do not want to add any waypoints, in the Edit dialog box, click **Save**. Restart your application servers.

To add long running user method waypoints:

1 On the Long Running Methods tab of the Edit dialog box, click **Edit** beside **LongRunningUserMethodWaypoints**.

2 In the Settings Editor, you can add, remove, or change the order of long running method waypoints.

Click **Add** to add a single waypoint at a time.



For example, add the waypoint `com.globex.plan.DominateWorldThread.executeProjectArcturus` to the `com.globex.plan.DominateWorldThread.run` method from the previous procedure.

Each time the method `executeProjectArcturus()` is called from `run()` it is considered a single request.

3 After you have added as many methods as you want, click **OK**.

4 In the Edit dialog box, click **Save**.

5 Restart your application server.

Manual configuration — LongRunningUserMethods

If you manually edit the *instrumentation.config* file, use the following syntax to specify long running user methods and waypoints instrumentation.

```
LongRunningUserMethods = MethodList(
include "com.globex.plan.DominateWorldThread.run"
);
```

```
LongRunningUserMethodWaypoints = MethodList(
include "com.globex.plan.DominateWorldThread.executeProjectArcturus"
);
```

In this case, the method `"com.globex.plan.DominateWorldThread.run"` treats the data as complete each time the method `executeProjectArcturus()` is called from `run()`. The Java EE agent pauses at waypoint methods to treat the data as complete before continuing. A minimal set of waypoint methods allows for the best possible performance.

Named Methods settings

The Named Methods instrumentation settings determine how the agent tracks specific methods. Changing any of these settings can have performance implications. Review the provided information before changing any settings.

Changing Named Methods settings

The `NamedMethods` setting for a Java EE agent is used to track specific methods. Specify the names of methods by forming a regular expression that matches the name or pattern. The method signature can be specified but it must be specified using the internal JVM format, as in the "DominateWorld" example below. For more information, see http://docs.oracle.com/javase/specs/#7035.

The JVM format for constructors may not be as expected. For example, for the constructor: `com.quest.controller.AuthenticateAdmin.AuthenticateAdmin()`, the correct signature to specify would be: `com.quest.controller.AuthenticateAdmin.<init>()`.

NOTE: This setting is only valid for Full Detail and Component Detail Instrumentation Levels. Changing this value has no effect if the Instrumentation Level is set to Basic Detail or No Detail. For more information about setting instrumentation levels, see “Configuring instrumented simple method settings on page 57.

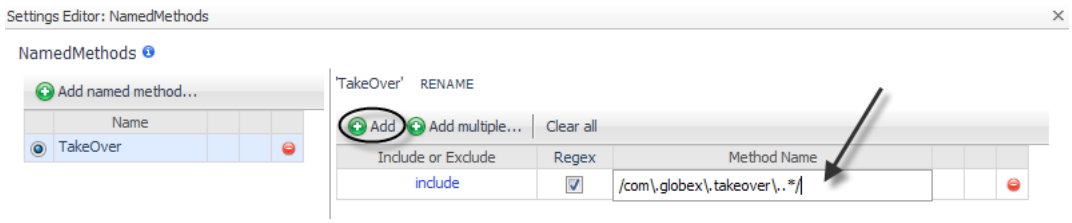
To add named methods:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

1. Click the **Named Methods** tab of the Edit dialog box.
2. To add named methods, click **Edit** beside **NamedMethods**.
3. In the Settings Editor that opens, you can add, remove, or change the order of named methods.

   Click **Add named method** to define a named method.
4. In the dialog box that opens, type the method name in the **Specify a name** box.
5. Click **OK**.
6. Add one or more methods to the named method list.

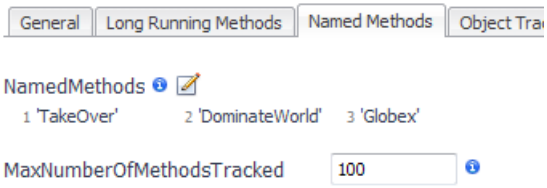
   Click **Add** to add a single method at a time.



Click **Add multiple** to open a text box where you can type several methods.

7 After you have added as many named methods as you want, click **OK**.

The Edit dialog box refreshes and displays the list of methods you added.



8 In the Edit dialog box, click **Save**.

9 Restart your application servers.

Next, specify the maximum number of named methods to track, as described in Changing the Maximum Number of Methods Tracked on page 69.

Manual configuration — NamedMethods

If you manually edit the *instrumentation.config* file, use the following syntax to specify named method instrumentation.

```
NamedMethods = {};
```

For example:

```
NamedMethods = {
"TakeOver": MethodList(
include /com\.globex\.takeover\..*/
),
"DominateWorld": MethodList(
include
"com.globex.plan.DominateWorldServlet.doGet(Ljavax/servlet/http/HttpServletRequest;
Ljavax/servlet/http/HttpServletResponse;)V",
include "com.globex.plan.DominateWorldHelper.dominate(Ljava/lang/String;Z[B)I",
),
"Globex": MethodList(
include "com.globex.pocket.Sugar",
include "com.globex.cypresscreek.HammockHut.sit",
exclude /(\.|^)globex\./
)
};
```

For more information about how to monitor specific methods, see the Monitoring Methods topic, in the *Foglight for Application Servers User Guide*.

Changing the Maximum Number of Methods Tracked

The `MaxNumberOfMethodsTracked` setting controls the maximum number of Named Methods that the Java EE agent tracks. If the Named Methods list contains a greater number of methods than the value set for `MaxNumberOfMethodsTracked`, only the first `MaxNumberOfMethodsTracked` methods are tracked.

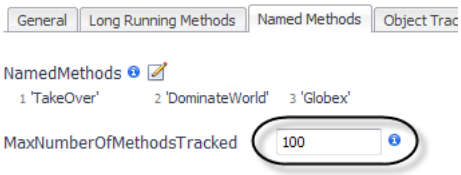
For example, if you create a named method list that contains 101 methods, but leave the maximum number of methods tracked set to the default of 100, then only the first 100 methods that are run from the named method list are tracked. The method that is not tracked is the one that runs last.

NOTE: This setting is only valid for Full Detail and Component Detail Instrumentation Levels. Changing this value has no effect if the Instrumentation Level is set to Basic Detail or No Detail. For more information about setting instrumentation levels, see Changing the instrumentation level for Request Metric data collection on page 55.

To change the maximum number of methods tracked:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

1. Click the **Named Methods** tab of the Edit dialog box.
2. Type a value for the maximum number of methods to track in the **MaxNumberOfMethodsTracked** box.



3. Click **Save**.
4. Restart your application servers.

Manual configuration — MaxNumberOfMethodsTracked

If you manually edit the *instrumentation.config* file, use the following syntax to specify the maximum number of methods tracked.

```
MaxNumberOfMethodsTracked = 100;
```

Object Tracking settings

The Object Tracking instrumentation settings for a Java EE agent determine how the agent tracks objects that are not reclaimed by the garbage collector. Changing any of these settings can have performance implications. Review the provided information before changing any settings.

Tracking object classes

Use the `ObjectTrackerClasses` setting to watch for objects that are never reclaimed by the garbage collector. You can specify groups of classes with allocated object instances that are tracked during a sampled request.

Object tracking impacts performance overhead if the number of tracked instances is too high. The simplest way to reduce imposed overhead is to track a smaller number of classes. Recording should be delayed until the application has reached a steady state (that is, after start-up work is complete and permanent objects have been created). Use the `IgnoredAllocations` setting to indicate which objects should not be tracked, even if those allocations are performed by classes in this tracked class list. For more information, see Ignoring specific objects on page 72.

NOTE: The ObjectTrackerClasses settings only take effect when object tracking is enabled. Object Tracking can be enabled and disabled on a per request basis, by setting the Enable Object Tracking option found in the **Application Servers Monitor > Server JVM** view. For more information, see the *Foglight for Application Servers User Guide*. *The* pre-instrumentor script must be run on all agent machines after changing either the ObjectTrackerClasses or IgnoredAllocations properties.

For example, the following procedure shows how to configure this object tracking:

- Track all classes in the `com.globex.plan` package.
- Do not track any other classes with `globex` in the name.
- Track the `com.acme.Widget` class.
- Track the listed collection classes.

To specify object classes for tracking:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

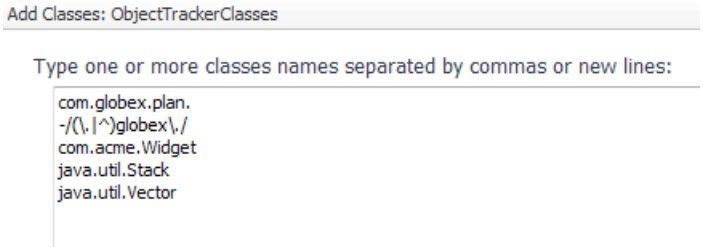
1 Click the **Object Tracking** tab of the Edit dialog box.

2 To add tracked object classes, click **Edit** beside **ObjectTrackerClasses**.

3 In the Settings Editor, you can add, remove, or change the order of tracked object classes.

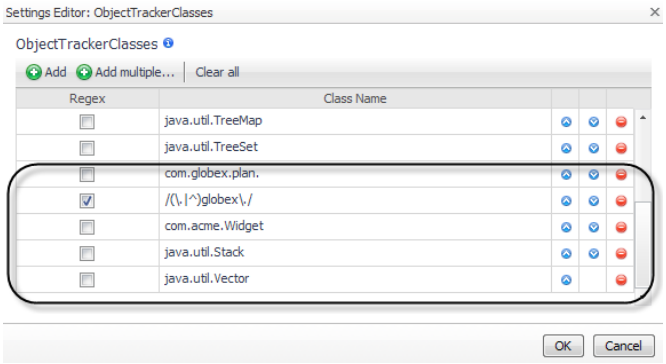
Click **Add multiple** to define several classes at once.

4 In the dialog box that opens, type the names of the classes, separated by commas or new lines.



5 Click **Add**.

The new object classes appear at the bottom of the list.



6 Click **OK**.

7 In the Edit dialog box, click **Save**.

8 Restart your application servers.

You may also want to ignore specific objects. For more information, see Ignoring specific objects on page 72.

Manual configuration — ObjectTrackerClasses

If you manually edit the *instrumentation.config* file, use the following syntax to specify the object tracking instrumentation.

```
ObjectTrackerClasses = ClassList(
include "com.globex.plan.",
exclude /(\.|^)globex\./

include "com.acme.Widget",

include "java.util.Stack",
include "java.util.Vector",
);
```

This example tracks all classes in the `com.globex.plan` package, but it does not track any other classes with `globex` in the name. It also tracks the `com.acme.Widget` class, as well as the listed collection classes.

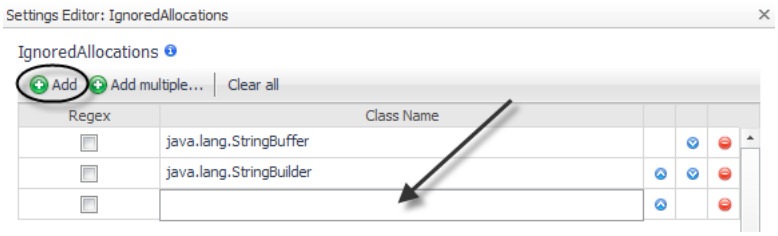
Ignoring specific objects

Even with object tracking enabled, some object allocations may be considered uninteresting. For example, the `StringBuffer` class is allocated as a temporary object by boilerplate code and is automatically generated by Java compilers. As a result of string manipulation, these objects are generated but should not cause memory issues. Use the `IgnoredAllocations` property to indicate objects that should not be tracked, even if those allocations are performed by classes in the tracked class list. For more information about tracking object classes, see Tracking object classes on page 70.

By default, `java.lang.StringBuffer` and `java.lang.StringBuilder` are ignored.

To ignore specific objects:

1. On the Object Tracking tab, click **Edit** beside **IgnoredAllocations**.
2. In the Settings Editor, click **Add**.



3. In the new row that appears, type the name in the **Class Name** box.
4. If you want to specify the name as a regular expression, select the **Regex** check box.
5. After you have added all the classes necessary, click **OK**.
6. Click **Save**.
7. Restart your application servers.

Manual configuration — IgnoredAllocations

If you manually edit the *instrumentation.config* file, use the following syntax to specify the ignored allocations.

```
IgnoredAllocations = ClassList();
```

Requests settings

The Requests instrumentation settings for a Java EE agent determine how the agent splits requests based on methods. Changing any of these settings can have performance implications. Review the provided information before changing any settings.

Setting the Maximum Request Fragment cache size

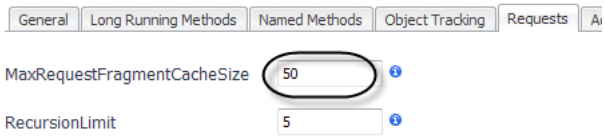
In the period between requests, the Java EE agent caches the data structures used to collect the data for request fragments. The `MaxRequestFragmentCacheSize` setting determines the maximum number of request fragments that are stored in the cache. The default value is `50`. Set the value to zero (`0`) to disable the cache.

NOTE: This setting is only valid for FullDetail, ComponentDetail and BasicDetail Instrumentation Levels. Changing its value has no effect if the Instrumentation Level is NoDetail. For more information about setting instrumentation levels, see Configuring instrumented simple method settings on page 57.

To set the maximum request fragment cache size:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

1. Click the **Requests** tab of the Edit dialog box.
2. Type a value in the **MaxRequestFragmentCacheSize** box.



3. Click **Save**.
4. Restart your application server.

Manual configuration — MaxRequestFragmentCacheSize

If you manually edit the *instrumentation.config* file, use the following syntax to specify the maximum request fragment cache size.

```
MaxRequestFragmentCacheSize = 50;
```

Setting the Request Recursion Limit

The `RecursionLimit` configuration setting adjusts the number of recursive calls for which the Java EE agent collects detailed performance information. Beyond this number, the Java EE agent collects and aggregates timing information for the entire recursive sequence.

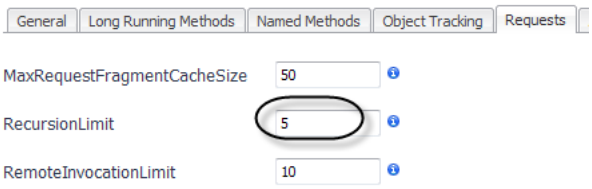
If you need to see more detail for each method call in a recursive sequence, you can increase this value. Set the value to zero (0) to see only the first method call, but none of the recursive calls. The default value is 5.

NOTE: This setting is only valid for FullDetail, ComponentDetail and BasicDetail Instrumentation Levels. Changing its value has no effect if the Instrumentation Level is NoDetail. For more information about setting instrumentation levels, see Configuring instrumented simple method settings on page 57.

To set the request recursion limit:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

1. Click the **Requests** tab of the Edit dialog box.
2. Type a value in the **RecursionLimit** box.



3 Click **Save**.

4 Restart your application server.

Manual configuration — RecursionLimit

If you manually edit the *instrumentation.config* file, use the following syntax to specify the recursion limit.

```
RecursionLimit = 5;
```

Setting the Request Remote Invocation Limit

The `RemoteInvocationLimit` determines the maximum number of calls that the Java EE agent follows from a single method to a remote system. Beyond this number, the Java EE agent only collects and aggregates overall performance information for the remote system invocations. This is done for two reasons:

- A large number calls to remote systems from a single method is highly likely to produce a performance problem.
- It is expensive for the Nexus to process Tag and Follow™ information for a large number of remote invocations on a single request.

The default value is 10. If this limit is exceeded, the Java EE agent logs a WARN message that identifies the method where the remote invocations are taking place.

If you are receiving WARN log messages and you require more detail on the remote invocations, you can increase this setting.

IMPORTANT: Increasing this value also increases the amount of CPU and memory that the Nexus uses.

Decrease this value if you are receiving the WARN log messages, but the Nexus is exhibiting symptoms of excessive memory use.

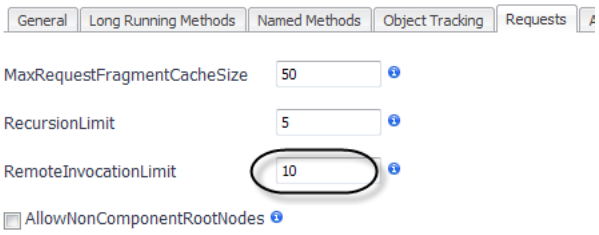
NOTE: This setting is only valid for FullDetail, ComponentDetail and BasicDetail Instrumentation Levels. Changing its value has no effect if the Instrumentation Level is NoDetail. For more information about setting instrumentation levels, see Configuring instrumented simple method settings on page 57.

To set the request remote invocation limit:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

1 Click the **Requests** tab of the Edit dialog box.

2 Type a value in the **RemoteInvocationLimit** box.



3 Click **Save**.

4 Restart your application server.

Manual configuration — RemoteInvocationLimit

If you manually edit the *instrumentation.config* file, use the following syntax to specify the remote invocation limit.

```
RemoteInvocationLimit = 50;
```

Allowing non-component root nodes

By default, the Java EE agent collects requests that start at a defined component technology (such as HTTP, JMS, RMI, EJB, or Servlet) or a custom component. Enable the `AllowNonComponentRootNodes` setting to allow the Java EE agent to collect information on requests that start at any method. This may be useful when using the X-Agent or the `StopCurrentRequestInMethods` configuration setting. For more information, see Separating requests by parent method on page 75.

By default, this setting is disabled.

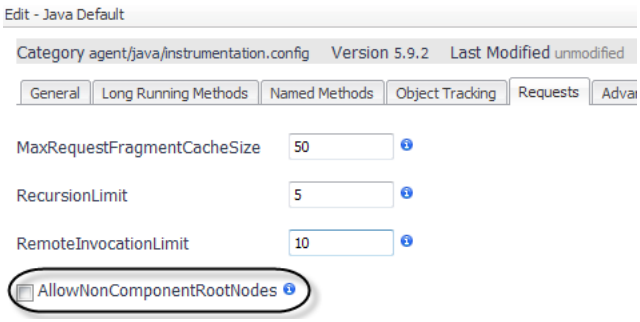
NOTE: This setting is only valid for the FullDetail instrumentation level. Changing the value has no effect if the instrumentation level is set to ComponentDetail, BasicDetail, or NoDetail. For more information about setting instrumentation levels, see Configuring instrumented simple method settings on page 57.

To enable non-component root nodes:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

1 Click the **Requests** tab of the Edit dialog box.

2 Select the check box for `AllowNonComponentRootNodes`.



3 Click **Save**.

4 Restart your application server.

Manual configuration — AllowNonComponentRootNodes

If you manually edit the *instrumentation.config* file, use the following syntax to enable non-component root nodes.

```
AllowNonComponentRootNodes = true;
```

Separating requests by parent method

Use the `StopCurrentRequestInMethods` parameter as a separate parameter when you want to stop the request in the configured method. Once stopped, the next method entered starts a new request. The data collected for the original request is not lost but has stopped collecting. The benefit of using the

`StopCurrentRequestInMethods` parameter is to organize and specify the time range during which the Java EE agent collects requests.

The `StopCurrentRequestInMethods` setting splits one request type into two or more request types where the original request ends on the specified method and new requests are started on children of the specified method.

For example, suppose that you have the following request:

```
GET /foo/bar -> service() -> doGet() -> doSomeStuff() -> doMoreStuff()
```

and you want to configure it to split on the `doGet()` method. The result is two requests:

```
GET /foo/bar -> service() -> doGet()
```

```
doSomeStuff() -> doMoreStuff()
```

The configured method becomes a leaf in the original request and methods called within the defined method become the root node in new request types. The configured method becomes a method node even if it was previously part of a component.

IMPORTANT: To prevent the split requests from being filtered out, make two additional changes:

- `AllowNonComponentRootNodes` must be set to enabled (true).
- Filtering Rules in the Recording Settings must be updated with a rule to include the new request type.

  For more information, see Allowing non-component root nodes on page 75, and the Managing Nexus Recording Settings topic in the *Managing Application Servers Administration and Configuration Guide*.

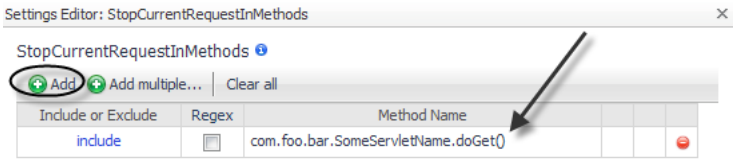
NOTE: Methods included in this list should not be included in the Named Methods list, the Custom Components list, the Long Running User Methods list, or the Start New Request In Methods list. This list takes precedence over all others. If the method is included in other lists, the agent logs a warning indicating the affected method and the ignored instrumentor.

To split a request on a parent method:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53

1. Click the **Requests** tab of the Edit dialog box.
2. To add method that the agent should stop on, click **Edit** beside **StopCurrentRequestInMethods**.

   In the Settings Editor that opens, you can add, remove, or change the order of methods that the agent should stop on.
3. Click **Add** to add a stop method.
4. In the empty row that appears, do the following:



   a. Leave the Include or Exclude option set to **include** (the default setting).

   b. This example does not use a regular expression, so leave the **Regex** check box cleared.

   c. Click in the **Method Name** box and type the name of the method that you want the agent to stop on. In this example, that is:
   `com.foo.bar.SomeServletName.doGet()`
5. Click **OK**.
6. In the Edit dialog box, click **Save**.

7 Restart your application servers.

The result is two requests:

```
GET /foo/bar -> service() -> doGet()
```

and

```
doSomeStuff() -> doMoreStuff()
```

Manual configuration — StopCurrentRequestInMethods

If you manually edit the *instrumentation.config* file, use the following syntax to specify the stop methods.

```
StopCurrentRequestInMethods = MethodList();
```

For example, if the following request

```
GET /foo/bar -> service() -> doGet() -> doSomeStuff() -> doMoreStuff()
```

is configured to split on the `doGet()` method as follows:

```
StopCurrentRequestInMethods = MethodList(
  include "com.foo.bar.SomeServletName.doGet()",
);
```

the result will be the following two requests:

```
GET /foo/bar -> service() -> doGet()
```

```
doSomeStuff() -> doMoreStuff()
```

Splitting requests during single tracing

By default, requests that have been split using the `StopCurrentRequestInMethods` setting also appear as separate requests when collecting in single-trace mode.

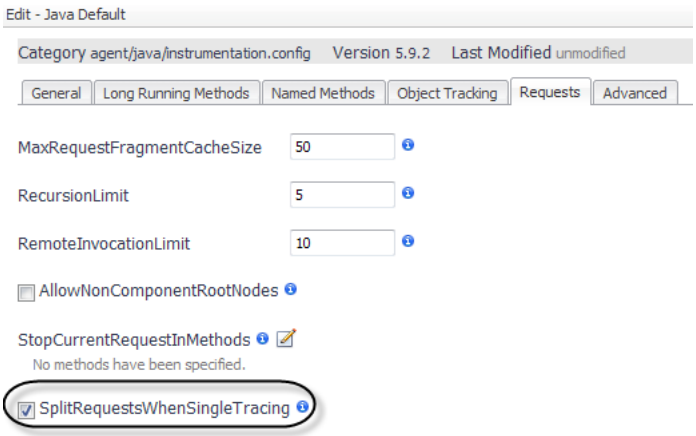
Disabling the `SplitRequestsWhenSingleTracing` setting prevents request splitting when collecting a single trace. Requests that would otherwise be separated appear as part of the same call tree. In some situations, the additional context may be helpful for more detailed analysis.

To enable non-component root nodes:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

1 Click the **Requests** tab of the Edit dialog box.

2 Clear the check box for `SplitRequestsWhenSingleTracing`.



3 Click **Save**.

4 Restart your application server.

Manual configuration — SplitRequestWhenSingleTracing

If you manually edit the *instrumentation.config* file, use the following syntax to specify that requests should be split while single tracing.

```
SplitRequestsWhenSingleTracing = true;
```

Customizing instrumented classes

The Advanced instrumentation settings for a Java EE agent control which classes are instrumented.The `InstrumentedClasses` setting customizes the classes that are instrumented. In almost all cases, tune instrumentation using the `UserClasses` or `CustomComponents`. Changing any of these settings can have performance implications. For more information, see Changing user classes settings on page 58 and Changing custom component settings on page 60.

The `InstrumentedClasses` setting is normally used to exclude a class when instrumentation causes failures or application problems.

- **Excluding** a class ensures that it is excluded from all instrumentation.
- **Including** a class causes the class to be examined and, if it implements important application server functionality, it may be instrumented appropriately.

IMPORTANT: Do not use the Instrumented Classes setting to exclude classes from instrumentation except at the recommendation of Customer Support.

To edit the list of instrumented classes:

TIP: To review how to access the instrumentation settings editor, see Editing agent instrumentation configurations on page 53.

1 Click the **Advanced** tab of the Edit dialog box.

2 Click **Edit** beside InstrumentedClasses.

3 Follow the instructions from Customer Support.

Managing other Java EE agent configuration files

The Advanced Configuration view lists all available version-specific agent configuration files. Sometimes, you may need to manually edit these files. It is strongly recommended that you do not change any of these files unless you have been told to do so by Customer Support.

To edit agent configuration files:

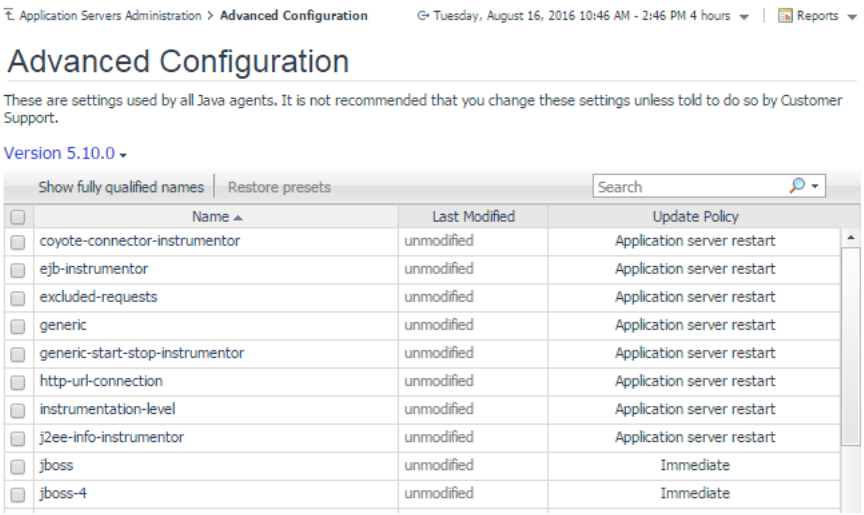
1 On the navigation panel, under Dashboards, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, click the **Java** tab.

3 On the Java Administration dashboard, click the **Agent Configurations** tab.

4 On the Agent Configurations tab, click **Advanced configuration**.

The Advanced Configuration page loads.



5 Select the version of the Java EE agent that you want to manage from the **Version** list (for example, 5.10.0).

6 Optional — To view the entire path to the file, click **Show fully qualified names**.

7 Click the name of the file that you want to edit, and in the menus that opens, click **Edit**.

8 Make any required changes, and click **Save**.

> **IMPORTANT:** The Update Policy column indicates when any changes are applied after you save the file. For some files, the change is immediate; for others, the application server must be restarted.

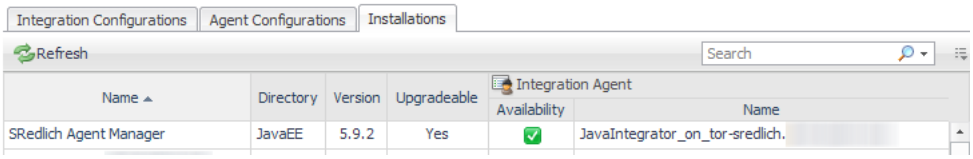
9 Restart your application server if necessary.

12

Managing Java EE Installation

The Java EE Integration Agent manages the Java EE agent installations and integrations with application servers. The Installations tab of the Java Administration dashboard provides an overview of all configured installations, including their versions, installation directories, upgrade status, and Java EE Integration Agent name and availability.

Figure 3. Installations tab on the Java view



From here, you can activate or deactivate the Java EE Integration Agent, open the log file for the agent, delete an installation, or view the installation properties. You can also check whether an installation is eligible for upgrade and, if so, perform an in-place upgrade.

Activating and deactivating the Java EE Integration Agent

The Java EE Integration Agent manages the integration of Java EE agents with application servers. The Java EE agents are the monitoring agents that collect the data. The integration agent also manages the integration script files, communication with the Nexus, and logging.

The integration agent can be activated or deactivated.

To activate or deactivate the integration agent:

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, click the **Java** tab.
3. On the Java view, click the **Installations** tab.
4. On the Installations tab, click the installation host name in the **Name** column.
5. In the menu that opens, click either **Activate** or **Deactivate**, as applicable.

Reviewing integration and agent installation log files

When you integrate a Java EE agent with an application server, the Java EE Integration Agent generates a log file detailing the process and any warning messages that occur during the installation. You can review this file to check the installation configuration, or to determine if there were any issues.

NOTE: This log file is verbose and suitable for troubleshooting. For a quick overview of the installation and integration process, use the **Result** text link for the task in the Task History table.

To open the integration agent log file:

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, click the **Java** tab.
3. On the Java view, click the **Installations** tab.
4. On the Installations tab, click the installation host name in the **Name** column.
5. In the menu that opens, click **Get log**.

   A dialog box opens, prompting you to open the log file with a text editor, or to save the file.
6. Select an option and click **OK**.

Uninstalling Java installations

IMPORTANT: Uninstalling the Java EE Integration Agent removes all files from the installation directory and deletes all the installed agents. Java agents continue to collect data until you restore any modified startup scripts to the pre-integrated versions and restart the monitored processes.

To uninstall the Java EE Integration Agent:

1. On the navigation panel, under Dashboards, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, click the **Java** tab.
3. On the Java view, click the **Installations** tab.
4. On the Installations tab, click the installation host name in the **Name** column.
5. In the menu that opens, click **Uninstall**.
6. Read the caution and click **Uninstall** to confirm that you want to remove the entire installation.

   A progress box opens.
7. Click **Close** to close the box when the process is finished.

Viewing Java installation properties

The installation properties include the location of the installation directory, and both the Java EE Integration Agent version and the Agent Manager name and version for that installation. This information is useful to have on hand when you manually integrate with an application server.

To view the installation properties:

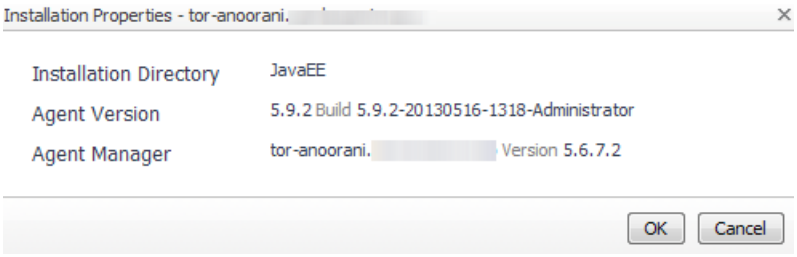
1. On the navigation panel, under **Dashboards**, click **Application Servers > Administration**.
2. On the Application Servers Administration dashboard, click the **Java** tab.

3 On the Java view, click the **Installations** tab.

4 On the Installations tab, click the installation host name in the **Name** column.

5 In the menu that opens, click **Properties**.

The Installation Properties dialog box opens.



Use this dialog box to verify the location of the installation directory, and the agent and Agent Manager version information.

6 Click **OK** to close the dialog box.

Upgrading Java installations

Upgrading an installation upgrades all agents that are managed through the installation. This is the same functionality as deploying a new *.gar* package to a monitored host.

Use the Installations tab of the Java Administration dashboard to determine the version of your installation and whether it is eligible for upgrade. This information appears in the Version and Upgradeable columns of the Installations table.

If an installation is upgradable, use the following procedure to perform an in-place upgrade.

To upgrade an installation:

1 On the navigation panel, under **Dashboards**, click **Application Servers > Administration**.

2 On the Application Servers Administration dashboard, click the **Java** tab.

3 On the Java view, click the **Installations** tab.

4 On the Installations tab, look at the **Upgradeable** column for the installation you are interested in. If the word “Yes” appears, you can upgrade the installation.

5 Click the name of the installation you want to upgrade.

6 In the menu that opens, select **Upgrade**.

7 The dashboard refreshes, and the task appears in the Task History list, with a description such as: *Upgrade Installation (<installation name>)*.

8 A progress indicator appears in the **Result** column while the task is in progress. The indicator is replaced by the text *Success* or *Failure* when the task is complete.

13

Java EE Integration Configuration FAQ and Troubleshooting

This section discusses potential issues when configuring application servers for integration with Java EE Integration Agent.

Table 6. List of FAQs/Issues

What are the supported JVM versions and what are the levels of metrics collected?

To help determine the level of metrics that are being collected for your JVM version, see the following table.

Table 7. JVM versions and collected metrics.

JVM Version	Level of Metrics Collected
IBM 1.6, 1.7, 1.8	Full
Oracle 1.6, 1.7, 1.8	Full
OpenJDK 1.6. 1.7, 1.8	Full

My integration task was a partial failure because the Java Home location was wrong. How do I correct it?

If you specify the wrong Java Home location during the creation of the Java EE Integration Agent, you can correct it from the Java Administration dashboard.

1 On the Integration Configurations tab of the Java Administration dashboard, click the name of the integration configuration (not the target) that you want to edit.

2 In the menu that opens, click **Properties**.

3 Type the correct value in the **Java Home** box.

4 Click **Save**.

5 Click **OK** to close the confirmation message box.

6 Click the name of the integration configuration again, and select **Reintegrate targets**.

7 In the Reintegrate dialog box that opens, select one or more targets and click **Reintegrate**.

Why is the Java EE agent sometimes unable to detect the correct type for the application server?

Users start their servers in various ways. Some of these ways may be different enough from the standard approaches described in this guide to cause the Java EE agent to be unable to determine the correct type for the application server. For example, if you are using a custom class to do some preliminary setup before starting the WebLogic® server, and you are passing `weblogic.Server` as a parameter to this class rather than calling `weblogic.Server` directly in the Java startup line, the Java EE agent gets the wrong name for the application server.

If the Java EE agent is unable to detect the application server type, it disables the associated agent and logs a fatal error.

Workaround

The workaround is to set the application server name and type explicitly as a system property in the Java command line that starts the application server. Add a `-D` option to the Java startup line that has this format:

```
-Dquest.agent.appserverinfo=<server-name>:<type>[:[<version>][:<cluster-name>]]
```

There are four fields, each separated by a colon-character.

The `<server-name>` and `<type>` are required parameters. The `<version>` and `<cluster-name>` are optional; if they are not supplied, the Management Server attempts to auto-detect the missing information. Valid application server type parameters are `Generic`, `WebLogic`, `WebLogicNodeManager`, `WebSphere`, `WebSphereDmgr`, `WebSphereNodeAgent`, `WebSphereAdminAgent`, `OracleAS`, and `JBoss`. The Java EE agent uses the server type and server version that are set in the command line to configure itself for that specific application server.

CAUTION: Specifying an incorrect application server type or version number results in errors later in the startup sequence. Application server type parameters are case-insensitive. The format for setting the version parameter differs based on the application server that you are configuring to work with Management Server.
Consult the examples below for the correct server-specific and version-specific format to use to set the version parameter.

This example specifies all four fields for a WebSphere® 8.5 application server named *server3* in a cluster named *my_cluster*: -
`Dquest.agent.appserverinfo=server3:WebSphere:8.5.5.9:my_cluster`

This example specifies only 3 of the fields, leaving the agent to determine the application server version number for a WebLogic server named *myserver* in a cluster named *my_cluster*: -
`Dquest.agent.appserverinfo=myServer:WebLogic::my_cluster`

Why does my application server take so long to start up after integrating it with the Management Server?

When configuring agents, the user that is running the server must have read, write, and execute permissions on all directories under and including the Installation directory (`QUEST_DEPLOYMENT_DIRECTORY`). The installation directory is specified for the Java EE Integration Agent agent that is activated on the same host.

If you have manually integrated an application server with Management Server on AIX 5.3 with JVM 1.4, and the server is taking a long time to start up, it is due to a problem with JVM 1.4 on AIX 5.3.

Workaround

Add `-Djava.net.preferIPv4Stack=true` to the Java command line that starts the server.

See `ibm.com` for more information about the problem with JVM 1.4 on AIX.

I have multiple application servers with the same name that reside on the same host. When the Java EE agents connect to the Management Server, why does only the first agent connect?

This error occurs because the Foglight Management Server does not allow duplicate agent names. Agent names are duplicated when multiple servers with the same name on the same host try to connect. If multiple Java EE agents reside on the same machine with the same server name, only one is permitted to connect.

Workaround

The workaround is to uniquely identify the agent name by using the following server startup line:

```
-Dquest.config.agent.agentname=<unique_name>
```

For example, if you have two administrative servers for two different WebLogic domains running on the same host, the startup lines can be modified to include the following (one per server):

```
-Dquest.config.agent.agentname=test_domain.admin
```

```
-Dquest.config.agent.agentname=prod_domain.admin
```

Since the agent names are unique, they will connect to the Foglight Management Server but do not modify the server name.

For more detailed information, see Java EE Integration Agent installation directory on page 11.

I received an error message about heap space shortage from the instrumented Java EE agent. What should I do?

The Java EE agent is designed to monitor and collect valuable diagnostic detail out-of-the-box with minimal impact on application server heap. The agent configuration can be tuned to reduce (or increase) the amount of detail collected which in turn will reduce (increase) the agent memory usage.

The Java EE agent runs on the application server JVM and uses CPU resources to collect data about the application server. Two types of data are collected: periodic metrics (JMX) and request metrics (instrumentation).

- The Java EE agent collects JMX metrics, such as heap usage, and thread count, regularly by calling standard JVM and application server MBean methods.
- The agent collects request metrics as the application server processes requests that match the filters specified in the agent configuration.

Both types of data are transmitted to the Nexus running on the Foglight Management Server.

The memory usage is proportional to:

- The number of metrics collected and the frequency at which they are collected
- The number of service requests monitored, the level of detail captured about those requests, and the complexity of the requests

The metric collection memory usage impact is typically fixed, while the instrumentation memory usage impact varies considerably according to the application, workload, and the agent instrumentation and collection configuration.

Workaround

Consider increasing the heap size of the instrumented application server.

Why do I see memory usage characteristic changes after enabling the Java EE agent?

The Java EE agent runs in the same JVM as the application server. As a result, the memory usage behaviors of the Java EE agent can impact the memory usage behaviors of the Application Server, which in turn can impact the memory usage behavior of the operating system.

One major contributor for the change is that objects are allocated for a cache that stores instrumented class information so the agent does not need to examine it every time. This optimization is made to improve the startup time performance. These are soft references, so the Garbage Collector can collect these objects if needed.

As a result, you may notice memory usage characteristic changes after enabling the Java agent.

Workaround

A few different strategies can be used to minimize the changes to memory usage characteristics, as follows:

- The environment may benefit from some basic heap size tuning. For example, reducing the maximum size of the heap when it is unnecessarily large may not only improve garage collection performance, but it limits the size of the process. This tuning can help alleviate problems that may arise from the JVM heap growing beyond typical or past levels.
- Turning off class caching during instrumentation greatly reduces the number of allocated objects during instrumentation and causes memory usage to more closely match the behavior that was exhibited before the Java EE agent was integrated. However, it also causes instrumentation to take longer.

  To turn off class caching, add the following to the JVM startup command line of the application server:

  ```
  -Dquest.agent.class.cache.disable=true
  ```

  IMPORTANT: By disabling the class caching, the startup time of the instrumented application server increases.

I received an error message about the max_long_data_size. What should I do?

When creating agent profiles or adding configuration file overrides to an agent profile, the Management Server may log the following error messages:

```
SQL Error: 1105, SQLState: HY000

Parameter of prepared statement which is set through mysql_send_long_data() is
longer than 'max_long_data_size' bytes
Could not synchronize database state with session
```

Workaround

Increase the value for the `max_long_data_size` or the `max_allowed_packet` system variable in your MySQL configuration. The recommended value is `8388608`.

If you still encounter the error when using the recommended value, increase the value further or reduce the number of agent profiles.

IMPORTANT: The largest usable value is 1073741824, however it is recommended you increase the value in increments, by 50% at a time.

Why does the Agent Manager support bundle not contain logs from the Java EE agent?

In order for the Agent Manager (FglAM) support bundle to include the log files from the Java EE agent, the Java EE Integration Agent (`JavaIntegrator`) must be active when the support bundle is collected.

If the Java EE Integration Agent is not active, the support bundle contains logs from the Java EE Integration Agent, but not from the Java EE agent.

14

Manually Integrating Application Server

You can use the application server agent setup wizard to configure and integrate the Java EE agent and the Java EE Integration Agent to work with most application servers. If for any reason you need to manually integrate, follow the instructions in this section that are specific to the application server you want to integrate with.

JBoss remote agent integration from the command-line

As an alternative to configuring agent integration through the agent setup wizard in the Foglight browser interface, you can use the `fglcmd` command-line scripts that are available on the Management Server.

If you are using the command-line interface and

- the JBoss *.jar* files are in a `lib/endorsed` directory: add the `-Dquest.agent.instrument.bootstrap=true` option to the `QuestAgentOpts` parameter
- you want to specify preinstrumentor options: add them to the `QuestPreinstOpts` parameter

NOTE: Use a single space to separate options on the command line. Do not use quotes.

Manual integration with Wildfly and JBoss EAP

This section contains instructions for manually integrating the Java EE agent with a Wildfly or JBoss® Enterprise Application Platform (EAP).

IMPORTANT: Foglight for Java EE Technologies version 5.9.11 is the last major release that supports JBoss AS (all versions), JBoss EAP 4.x, and JBoss EAP 5.x.

IMPORTANT: Review the following exceptions before configuring your server.

Exception list

The following exceptions are not mutually exclusive. If all exceptions apply to your environment, complete the exception configurations for all of them.

- **If JBoss JARs are moved to a lib/endorsed directory:**

  A system property must be added to the application server command line to properly instrument classes. Add the following line before the code block documented in Step 3 of Manually integrating the Wildfly and JBoss EAP on page 89.

  Windows®:

  ```
  set PERFORMASURE_OPTIONS=-Dquest.agent.instrument.bootstrap=true
  ```

  UNIX®:

  ```
  PERFORMASURE_OPTIONS=-Dquest.agent.instrument.bootstrap=true
  ```

- **If you want to set preinstrumentor options specific to the JVM you are running:**

  Set the environment variable `QUEST_PREINST_OPTS` with the desired VM options. The environment variable `QUEST_PREINST_OPTS` is passed to the Java EE agent *pre-instrumentor* script.

  For example, to set the memory limit for the preinstrumentor to 256MB, specify the following: `QUEST_PREINST_OPTS=-Xmx256m`

  IMPORTANT: Use a single space to separate multiple options. Do not use quotes.

Manually integrating the Wildfly and JBoss EAP

Use the following procedure to manually configure the Wildfly and JBoss® server startup script for integration with the Java EE agent.

To manually configure the Wildfly and JBoss EAP:

1. Make a backup copy of your original server startup script.

   Standalone mode:

   Wildfly and JBoss EAP: *<jboss-home>\bin\standalone.bat* (Windows®) or *<jboss-home>/bin/standalone.sh* (UNIX®)

   Domain mode:

   Wildfly and JBoss EAP: *<jboss-home>\bin\domain.bat* (Windows®) or *<jboss-home>/bin/domain.sh* (UNIX®)

2. Complete a *Generic installation only on Windows* or a *Generic installation only on Solaris, Linux, HP-UX, or AIX*, as described in Creating a Generic Installation for Manual Java EE Agent Integration on page 39.

3. Copy the following block of code into your server startup script. Insert the block just before the Java command line that starts the Wildfly or JBoss application server, but after any commands that set or modify the `JAVA_HOME` and `JAVA_OPTS` environment variables.

   Windows (standalone mode):

   ```
   set QUEST_DEPLOYMENT_DIRECTORY=* SET JavaEE Installation directory *
   set QUEST_JAVA_ENV_OPTS=JBOSS:SERVER:JAVA_OPTS
   call "%QUEST_DEPLOYMENT_DIRECTORY:"=%\<integrate.cmd>" %*
   if ERRORLEVEL 1 echo Java EE agent NOT enabled
   ```

   Windows (domain mode):

   ```
   set QUEST_DEPLOYMENT_DIRECTORY=* SET JavaEE Installation directory *
   set QUEST_JAVA_ENV_OPTS=JBOSS:DOMAIN
   ```

```
call "%QUEST_DEPLOYMENT_DIRECTORY:"=%\<integrate.cmd>" %*
if ERRORLEVEL 1 echo Java EE agent NOT enabled
```

UNIX (standalone mode):

```
QUEST_DEPLOYMENT_DIRECTORY=* SET JavaEE Installation directory *
if [ -f "$QUEST_DEPLOYMENT_DIRECTORY/<integrate.sh>" ]
then
     QUEST_JAVA_ENV_OPTS=JBOSS:SERVER:JAVA_OPTS
     . "$QUEST_DEPLOYMENT_DIRECTORY/<integrate.sh>"
else
    echo Java EE agent NOT enabled
fi
```

UNIX (domain mode):

```
QUEST_DEPLOYMENT_DIRECTORY=* SET JavaEE Installation directory *
if [ -f "$QUEST_DEPLOYMENT_DIRECTORY/<integrate.sh>" ]
then
     QUEST_JAVA_ENV_OPTS=JBOSS:DOMAIN
     . "$QUEST_DEPLOYMENT_DIRECTORY/<integrate.sh>"
else
    echo Java EE agent NOT enabled
fi
```

4 Set *<integrate.cmd/sh>* to the name of the integration task you created in step 2. For example: `integrate-Generic_installation_only_on_Windows_Install_Configs_and_pre-instrument.cmd`.

5 Set the `QUEST_DEPLOYMENT_DIRECTORY` variable to the Installation Directory specified by the Java EE Integration Agent instance activated on this host. In Unix, quote this path if it contains spaces.

6 Ensure that your environment does not meet any of the exceptions in the Exception list and then save the new script.

7 If it is not already running, start the Management Server. The instrumented Wildfly or JBoss application server attempts to connect to Foglight for Java EE Technologies when started using the new script.

8 Start the Wildfly or JBoss application server using the new script.

Foglight for Java EE Technologies starts a Java EE agent for each instrumented server. When the Java EE agent connects to the Management Server, the Management Server console confirms that a Java EE agent has connected by displaying a message , for example:

```
<timestamp> INFO Accepted connection from agent “<agent name> (JBoss)”, (JBoss
<version>) on host: <hostname>
```

In addition, the log file for the Java EE agent contains an INFO message stating that it has connected to the Management Server.

Manual integration with Tomcat

This section contains instructions for integrating the Java EE agent with a Tomcat Application Server on UNIX® and Windows® operating systems:

TIP: Make a backup copy of your original Tomcat startup script before proceeding with the steps listed in this section.

The manual integration instructions are OS-specific for the target system:

- Integrating with Tomcat on UNIX
- Integrating with Tomcat on Windows

Integrating with Tomcat on UNIX

The following exceptions are not mutually exclusive. If all exceptions apply to your environment, complete all of the following exceptions.

Exception list — UNIX

- **If you have spaces in the UNIX Installation directory path:**

  Edit the startup script and insert the following code in the Java command line before `org.apache.catalina.startup.Bootstrap "$@" start:`

  ```
  "${PERFORMASURE_BOOTSTRAP}" "${PERFORMASURE_JAVA5}" ${PERFORMASURE_OPTIONS}
  ```

  The *catalina.sh* script uses six different versions of the startup command line. Sometimes, you must modify all these scripts.

- **If either of the following conditions are true:**

  - You are using JRockit
  - The Java command line includes `-Djava.endorsed.dirs=...<tomcat_home>\bin`

  Then this exception applies and you must do the following:

  Edit the startup script and insert the following code in the Java command line before `org.apache.catalina.startup.Bootstrap "$@" start`:

  ```
  ${PERFORMASURE_BOOTAGENT}
  ```

  The *catalina.sh* script uses six different versions of the startup command lines. Sometimes, you must modify all these commands.

- **If you want to set preinstrumentor options that are specific to the JVM you are running:**

  Set the environment variable `QUEST_PREINST_OPTS` with the desired VM options. The environment variable `QUEST_PREINST_OPTS` is passed to the Java EE Agent *pre-instrumentor* script.

  For example, to set the memory limit for the preinstrumentor to 256MB, specify the following: `QUEST_PREINST_OPTS="-Xms256m"`

Integrating on a UNIX operating system

To integrate the Java EE agent with Tomcat on UNIX:

1 Make a copy of your original *catalina.sh* script.

  By default, you can find this script in the *<tomcat_home>/bin/* directory.

2 Complete a *Generic installation only on Windows*, as described in Creating a Generic Installation for Manual Java EE Agent Integration on page 39.

3 Open the *catalina.sh* script in a text editor and change the following:

Immediately before the line:

```
if [ "$1" = "debug" ] ; then
```

Insert the following block of code:

```
# QUEST_PREINST_OPTS=* OPTIONAL pre-instrumentor Options *
# export QUEST_PREINST_OPTS
QUEST_DEPLOYMENT_DIRECTORY=* SET JavaEE Installation Directory *
if [ "$1" != "stop" ]; then
if [ -f "$QUEST_DEPLOYMENT_DIRECTORY/<integrate.sh>" ]; then
QUEST_JAVA_ENV_OPTS=TOMCAT
. "$QUEST_DEPLOYMENT_DIRECTORY/<integrate.sh>"
else
echo Java EE agent not enabled
fi
fi
```

4 Set *<integrate.sh>* to the name of the integration task you created in step 2. For example: `integrate-Generic_installation_only_on_UNIX_Install_Configs_and_pre-instrument.cmd`.

5 Set the `QUEST_DEPLOYMENT_DIRECTORY` variable to the Deployment Directory specified by the Java EE Integration Agent instance that is activated on this host.

IMPORTANT: Enclose this path in double quotes if it contains spaces.

6 Save the startup script.

7 If it is not already running, start the Foglight Management Server. The instrumented Tomcat Server attempts to connect to the Foglight Management Server when the Tomcat Server is started using the modified script.

8 Use the modified script to start Tomcat.

The Foglight console confirms that a Tomcat Agent has connected by displaying a message similar to:

```
INFO Accepted connection from Tomcat Agent on <hostname>
```

In addition, the log file for the Tomcat Agent contains an `INFO` message stating that it has connected to the Foglight Management Server.

IMPORTANT: By default, Agent log files are created in the logs subdirectory of the Foglight home directory on the Agent machine.

Integrating with Tomcat on Windows

Follow these instructions to integrate Tomcat with Foglight for Java EE Technologies by modifying your Tomcat startup script.

Review the following exception before you configure your server. If this exception applies to your environment, complete these instructions.

Exception list — Windows®

- **If you want to set preinstrumentor options that are specific to the JVM you are running:**

  Set the environment variable `QUEST_PREINST_OPTS` with the desired VM options. The environment variable `QUEST_PREINST_OPTS` is passed to the Java EE agent *pre-instrumentor* script.

  For example, to set the memory limit for the pre-instrumentor to 256MB, specify the following: `QUEST_PREINST_OPTS=-Xms256m`.

Integrating on a Windows operating system

To integrate the Java EE agent with Tomcat on Windows:

1. Make a copy of the *catalina.bat* script.

   By default, you can find this script in the *<tomcat_home>\bin* directory.

2. Complete a *Generic installation only on Windows*, as described in Creating a Generic Installation for Manual Java EE Agent Integration on page 39.

3. Open the *catalina.bat* script in a text editor and update the script.

   Immediately before the line:

   ```
   if ""%1"" == ""debug"" goto doDebug
   ```

   Insert the following block of code:

   ```
   @rem set QUEST_PREINST_OPT=* OPTIONAL pre-instrumentor Options *
   set QUEST_DEPLOYMENT_DIRECTORY=* SET JavaEE Installation Directory *
   if ""%1"" == ""debug"" set QUEST_JAVA_ENV_OPTS=TOMCAT
   if ""%1"" == ""run"" set QUEST_JAVA_ENV_OPTS=TOMCAT
   if ""%1"" == ""start"" set QUEST_JAVA_ENV_OPTS=TOMCAT
   if defined QUEST_JAVA_ENV_OPTS (
   call "%QUEST_DEPLOYMENT_DIRECTORY:"=%\<integrate.cmd>"
   )
   if ERRORLEVEL 1 echo Quest Agent NOT enabled for Tomcat server
   ```

4. Set *<integrate.cmd>* to the name of the integration task you created in step 2. For example: `integrate-Generic_installation_only_on_Windows_Install_Configs_and_pre-instrument.cmd`.

5. Set the `QUEST_DEPLOYMENT_DIRECTORY` variable to the Installation directory specified by the Java EE Integration Agent instance activated on this host.

   IMPORTANT: Do not place quotation marks around the directory name even if it contains spaces. Doing so causes a conflict with existing quotation marks in commands that use the QUEST_DEPLOYMENT_DIRECTORY variable.

6. Save the modified startup script.

7. If it is not already running, start the Foglight Management Server. The instrumented Tomcat Server attempts to connect to the Management Server when the Tomcat Server is started using the modified script.

8. Use the modified script to start Tomcat.

   The Foglight console confirms that a Tomcat Agent has connected by displaying a message similar to:

   ```
   INFO Accepted connection from Tomcat Agent on <hostname>
   ```

   In addition, the log file for the Tomcat Agent contains an `INFO` message stating that it has connected to the Management Server.

   IMPORTANT: By default, agent log files are created in the logs subdirectory of the Foglight home directory on the agent machine.

Configuring a Tomcat Windows Service that requires manual integration

If you have chosen not to specify the Java home, or you have set the Startup mode to *jvm* and Java to *Use Default*, complete the following procedure after an installation-only setup.

To manually integrate a Tomcat Windows® Service:

1 Complete a *Generic installation only on Windows*, as described in Creating a Generic Installation for Manual Java EE Agent Integration on page 39. Specify the JVM used by the service in the Java_Home box in the setup wizard.

The generic installation creates a *bootstrap.jar* file and an `integrationid` that are needed later in this integration procedure.

2 Set the following Java options:

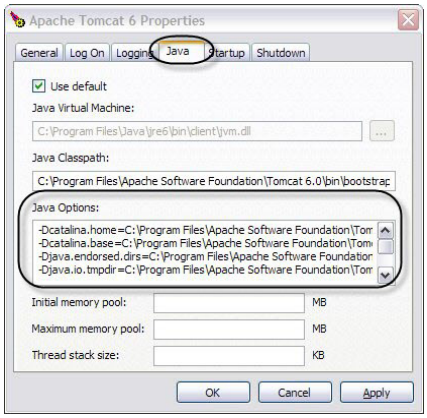
a In a command prompt, open the Tomcat installation *bin* directory.

b Type `./tomcat#w.exe //ES//<WindowsServiceIDname>`.

For example:

`./tomcat5w.exe //ES//Tomcat_Toronto_Host3`.

c In the Apache Tomcat Properties dialog box, open the **Java** tab.



d Add the following options to the end of the list of existing **Java Options**:

```
-Dquest.debug=0

-Dquest.integrationid=<INTEGRATIONID>

-
Xbootclasspath/p:<VERSIONED_SUBDIRECTORY>\bootstrap\<INTEGRATIONID>\<BOOT
STRAP_FILENAME>.jar

-javaagent:<VERSIONED_SUBDIRECTORY>\lib\performasure-agent.jar
```

e Replace `<VERSIONED_SUBDIRECTORY>` with the absolute path of the deployment directory and the current agent directory version.

Example of a <*VERSIONED_SUBDIRECTORY*>:

```
C:\Quest Software Inc.\Foglight for Java EE
Technologies_Agent_Manager\agents\JavaEE\5.7.0-570-20101210-1400
```

The Installation Directory is specified in the Java EE Integration Agent Startup Parameters, which you can see in the **Edit Properties** interface. If the Installation Directory path is specified as a relative path, as it is by default, you may need to examine the agent machine to determine the absolute path.

TIP: The current version of the subdirectory is indicated within the jee_builds.properties file. It is the value of the set.QUEST_AGENT_HOME parameter.This value already includes the installation directory. The version of the subdirectory portion is the last directory specified in the value.

f Replace both instances of `<INTEGRATIONID>` with the `integrationid` produced in Step 1. The `integrationid` can be found in the integrate scripts generated by the generic installation, which are placed in the Installation Directory. Locate the scripts corresponding to your generic installation, open the `.cmd` script, and look for a command similar to the following:

```
@set QUEST_INTEGRATIONID=YourCustomizedId
```

where the word after the '=' is your integrationid. In this example, the integrationid is *YourCustomizedId*.

g Replace <BOOTSTRAP_FILENAME> with the name of the pre-instrumentation *.jar* filename that was created in step 1. corresponds to the JRE/JDK that the Tomcat service uses. This name can be found in your agent machine *bootstrap* directory, in the <VERSIONED_SUBDIRECTORY>.

Here is an example where <INTEGRATIONID>, <VERSIONED_SUBDIRECTORY> and <BOOTSTRAP_FILENAME> are updated:

```
-Dquest.debug=0

-Dquest.integrationid=KEo3JDr6ALaybTu

-
Xbootclasspath/p:C:\Quest_Software\Foglight_Agent_Manager\agents\JavaEE\5
.9.3-20130815-0923\bootstrap\KEo3JDr6ALaybTu\C--Program_Files_-x86--Java-
jdk1.6.0_20.jar

-javaagent:C:\Quest_Software\Foglight_Agent_Manager\agents\JavaEE\5.9.3-
20130815-0923\lib\performasure-agent.jar
```

h Click **OK** to save the changes to the Windows Service made within *tomcat#w.exe*.

3 Ensure that the Foglight Management Server is started and the Java EE agent is activated.

4 For the changes to take effect, stop and restart the Tomcat Windows Service.

IMPORTANT: If the JVM that the Tomcat Windows Service uses is changed (for example, by installing an updated version of the JRE or JDK on the host), this manual integration must be done again. If Foglight for Java EE Technologies is updated on the Tomcat host, this manual integration must be done again to take advantage of the new release.

Manual integration with WebLogic

This section contains instructions for integrating the Java EE agent with a WebLogic® Server. You must modify your WebLogic startup script by adding lines to set the variables that the Java EE agent requires.

Manually configuring WebLogic for integration

Use the steps in this section to manually configure a WebLogic® Server or a WebLogic Express Startup for integration with the Java EE agent.

To configure WebLogic server or WebLogic Express startup:

1 Make a backup copy of your WebLogic Server startup script before proceeding with the steps below.

2 Complete the *Generic installation only on Windows*, as described in Creating a Generic Installation for Manual Java EE Agent Integration on page 39.

3 Copy the following block of code to your WebLogic Server startup script. Insert the block just **before** the WebLogic startup line.

Windows®*:*

```
@rem set QUEST_PREINST_OPTS=* OPTIONAL pre-instrumentor Options *
set QUEST_DEPLOYMENT_DIRECTORY=* SET JavaEE Installation Directory *
set QUEST_JAVA_ENV_OPTS=WEBLOGIC:SERVER
call "%QUEST_DEPLOYMENT_DIRECTORY:"=%\<integrate.cmd>" || (
echo Java EE agent not enabled )
```

UNIX®:

```
QUEST_PREINST_OPTS=* OPTIONAL pre-instrumentor Options *
export QUEST_PREINST_OPTS
QUEST_DEPLOYMENT_DIRECTORY=* SET JavaEE Installation Directory *
if [ -f "$QUEST_DEPLOYMENT_DIRECTORY/<integrate.sh>" ]
then
QUEST_JAVA_ENV_OPTS=WEBLOGIC:SERVER
. "$QUEST_DEPLOYMENT_DIRECTORY/<integrate.sh>"
else
echo Java EE agent not enabled
fi
```

4 Set *<integrate.cmd/sh>* to the name of the integration task you created in step 2. For example: `integrate-Generic_installation_only_on_Windows_Install_Configs_and_pre-instrument.cmd`.

5 Set the `QUEST_DEPLOYMENT_DIRECTORY` variable to the Installation Directory specified by the Java EE Integration Agent instance activated on this host. This agent instance was created in step 2.

IMPORTANT: In Unix, quote this path if it contains spaces.

6 Before saving this new startup script, ensure that your environment does not meet any of the exceptions listed at the beginning of this section, then save the new script.

7 If it is not already running, start the Management Server. The instrumented WebLogic Server attempts to connect to it when starting using the new script.

8 Restart the WebLogic Server using the new script.

The Management Server starts a Java EE agent for each instrumented server. When the Java EE agent connects to the Management Server, the log file for the Java EE agent contains an INFO message stating that it has connected to the Management Server. For example:

```
INFO [Multiplexer: 0] com.quest.pas.agent.nexus.PsychoNexusStalker - Connected
to Nexus at APMFoglightServer:41705.
```

Manual integration with IBM WebSphere Application Server

IMPORTANT: Review the section Understanding what should be instrumented on page 24 before you begin manually integrating WebSphere®.

This section contains instructions for integrating the Java EE agent with WebSphere Server. You either need to create Windows® services that start instrumented servers, Admin agents, Node agents, and WebSphere Deployment Managers; or modify the startup scripts.

If you want to instrument the servers that a Node agent starts, configure integration with the Node agent.

If you want to collect cell-wide configuration information and application server states, configure the integration with the WebSphere Deployment Manager.

Using Foglight for Java EE Technologies with your WebSphere servers requires three steps:

1 WebSphere environments on page 97.

2 Creating WebSphere services on page 98.

3 Modifying the WebSphere startup scripts on page 99.

Other topics include:

- Running WebSphere with security enabled on page 25

- Setting the PMI levels on WebSphere on page 25

CAUTION: Before you attempt to integrate WebSphere, ensure that the Disable JIT check box is not selected on the Java Virtual Machine page in the WebSphere Administrative Console. Disabling the JIT drastically slows down the server's startup and operation. Disabling metrics or changing metric sets while you are collecting data is not recommended. If you do so, Foglight may be unable to collect the necessary WebSphere metrics. In addition, if you disable metrics or change metric sets while data collection is in progress, Foglight cannot restore the original set of metrics that you configured.

WebSphere environments

WebSphere® are configured to use profiles. Therefore, configuration with the Java EE agent involves two steps. The first is to edit to the global *setupCmdLine* script. These changes only take effect if the environment variable `QUEST_DEPLOYMENT_DIRECTORY` has been set in your user profile *setupCmdLine* script. All other profiles ignore the code in the global *setupCmdLine* script.

Before you begin

Review the following exceptions before you configure your server.

Exception list

- **If you want to set pre-instrumentor options specific to the JVM you are running:**

  Set the environment variable `QUEST_PREINST_OPTS` with the desired VM options. The environment variable `QUEST_PREINST_OPTS` is passed to the Java EE agent *pre-instrumentor* script.

  For example, to set the memory used for the pre-instrumentor to 256MB, specify the following:

  Windows®:

  ```
  QUEST_PREINST_OPTS=-Xms256m -Xmx256m
  ```

  UNIX®:

  ```
  QUEST_PREINST_OPTS="-Xms256m -Xmx256m"
  ```

Configuring WebSphere environments

Complete the following steps to configure a WebSphere® environment.

To configure a WebSphere environment:

1. Make a backup copy of your original global setupCmdLine script (.bat or .sh).

   By default, this script can be found in the *<websphere_home>/bin/* directory.

2. Create a *Generic Installation Only* integration on the host. For more information, see Creating a Generic Installation for Manual Java EE Agent Integration on page 39.

3. Open the global *setupCmdLine* script in a text editor and make the following modifications:

   - **Windows® Only** — Add the following code to the end of the script:

   ```
   if %1==-Dquest.agent.websphere.server set QUEST_JAVA_ENV_OPTS=WEBSPHERE:SERVER
   if %1==-Dquest.agent.websphere.nodeagent set
   QUEST_JAVA_ENV_OPTS=WEBSPHERE:NODEAGENT
   if %1==-Dquest.agent.websphere.dmgr set QUEST_JAVA_ENV_OPTS=WEBSPHERE:DMGR
   ```

   Add the following code before the line `goto :EOF`:

```
if defined QUEST_DEPLOYMENT_DIRECTORY if defined QUEST_JAVA_ENV_OPTS (
    call "%QUEST_DEPLOYMENT_DIRECTORY%\<integrate.cmd>"
if ERRORLEVEL 1 echo Java EE agent NOT enabled
)
```

- **UNIX®** — Add the following code to the end of the script:

```
if [ -f "$QUEST_DEPLOYMENT_DIRECTORY/<integrate.sh>" -a -n "$QUEST_JAVA_ENV_OPTS"
];
then
    . "$QUEST_DEPLOYMENT_DIRECTORY/<integrate.sh>"
else
    echo Java EE agent is NOT enabled
fi
```

4 Set *<integrate.cmd/sh>* to the name of the integration task you created in step 2. For example: `integrate-Generic_installation_only_on_Windows_Install_Configs_and_pre-instrument.cmd`.

5 Save your modified *setupCmdLine* script file.

6 Locate your profile *setupCmdLine* script. The default installation places it in *WAS_HOME/profiles/<profile_name>/bin*. Set the `QUEST_DEPLOYMENT_DIRECTORY` at the end of the profile script to the Installation directory defined in step 2 to enable monitoring for that profile.

Windows:

```
rem Enable Java EE agent monitoring for this profile
set QUEST_DEPLOYMENT_DIRECTORY=* SET JavaEE Installation Directory *
```

Unix:

```
# Enable Java EE agent monitoring for this profile
QUEST_DEPLOYMENT_DIRECTORY=* SET JavaEE Installation Directory *
export QUEST_DEPLOYMENT_DIRECTORY
```

7 Save the changes to your profile-specific *setupCmdLine* script.

8 Restart your application server.

Creating WebSphere services

If you are starting your servers, Node agents, or WebSphere® Deployment Managers as a service, create a Windows® Service to use the Java EE agent instrumentation. See the IBM® documentation for instructions on creating and installing a Windows Service with the *WASService.exe* tool. You can keep your old Windows Service, but ensure that only one service for a particular WebSphere server, Node agent, or WebSphere Deployment Manager has a startup type of *Automatic*. All others should be set to *Manual* or *Disabled*.

When creating a Windows Service to integrate with the Java EE agent, provide an extra argument to the *-startArgs* option:

For WebSphere server use:

```
" -Dquest.agent.websphere.server"
```

For WebSphere Node agent use:

```
" -Dquest.agent.websphere.nodeagent"
```

```
For WebSphere Deployment Manager use:
```

```
" -Dquest.agent.websphere.dmgr"
```

NOTE: Quotes are required. The space between the first quote and the dash is also required.

A caret (^) character is required at the end of a line (other than the last) to correctly wrap the DOS command on multiple lines. A space is a required before the caret character to ensure proper spacing when copying and pasting the following code blocks.

Examples:

WebSphere Server:

```
WAS_HOME\bin\WASService.exe -add "BASE server1 - with agent" ^
  -servername server1 ^
  -profilePath "C:\WebSphere\profiles\AppSrv01" ^
  -startArgs " -Dquest.agent.websphere.server" ^
  -startType automatic
```

WebSphere Node agent:

```
WAS_HOME\bin\WASService.exe -add "ND nodeagent - with agent" ^
  -servername nodeagent ^
  -profilePath "C:\WebSphere\profiles\AppSrv01" ^
  -startArgs " -Dquest.agent.websphere.nodeagent" ^
  -startType automatic
```

WebSphere Deployment Manager:

```
WAS_HOME\bin\WASService.exe -add "ND dmgr - with agent" ^
 -servername dmgr ^
 -profilePath "C:\WebSphere\profiles\Dmgr01" ^
 -startArgs " -Dquest.agent.websphere.dmgr" ^
 -startType automatic
```

Modifying the WebSphere startup scripts

Use the steps in this section to modify the WebSphere startup scripts.

NOTE: The following procedures are not required if you are starting your servers, Node agent, or Deployment Manager using Windows services.

Before you begin

Review the following exceptions before you configure your server.

Exception list

- **If you want to set pre-instrumentor options specific to the JVM you are running:**

  Set the environment variable `QUEST_PREINST_OPTS` with the desired VM options. The environment variable `QUEST_PREINST_OPTS` is passed to the Java EE agent *pre-instrumentor* script.

  For example, to set the memory used for the pre-instrumentor to 256MB, specify the following:

  Windows:

  ```
  QUEST_PREINST_OPTS=-Xms256m -Xmx256m
  ```

  Unix:

  ```
  QUEST_PREINST_OPTS=”-Xms256m -Xmx256m”
  ```

Modifying WebSphere startup scripts

Complete the following steps to modify a WebSphere startup script.

To modify the WebSphere startup scripts:

1 Locate your profile-specific startServer, startNode, or startManager script. Save a backup copy before continuing.

2 Open your profile-specific startServer, startNode, or startManager script. Before the call to the WebSphere Installation startServer, startNode, or startManager script, add the following environment variable as appropriate:

Windows*:*

a In startServer add:

```
set QUEST_JAVA_ENV_OPTS=WEBSPHERE:SERVER
```

b In startNode add:

```
set QUEST_JAVA_ENV_OPTS=WEBSPHERE:NODEAGENT
```

c In startManager add:

```
set QUEST_JAVA_ENV_OPTS=WEBSPHERE:DMGR
```

Unix:

a In startServer add:

```
QUEST_JAVA_ENV_OPTS=WEBSPHERE:SERVER
export QUEST_JAVA_ENV_OPTS
```

b In startNode add:

```
QUEST_JAVA_ENV_OPTS=WEBSPHERE:NODEAGENT
export QUEST_JAVA_ENV_OPTS
```

c In startManager add:

```
QUEST_JAVA_ENV_OPTS=WEBSPHERE:DMGR
export QUEST_JAVA_ENV_OPTS
```

3 Restart your application server. See, Restarting modified WebSphere servers.

Restarting modified WebSphere servers

After you have modified the startup script, restart the WebSphere® server.

To start the WebSphere modified server:

1 If it is not already running, start the Foglight Management Server.

2 Stop and restart the WebSphere server using the instructions for the server type you modified:

- If you are using a standalone WebSphere server, stop the server and restart it with the modified script or the new Windows Service.
- When starting WebSphere 8.x/9.0.0.x servers managed by a WebSphere Deployment Manager, stop the WebSphere node agent and restart the node agent with the modified script or with the new Windows Service. Restart any managed WebSphere Application Server instances after restarting the modified WebSphere node agent.
- When starting a standalone WebSphere 8.x/9.0.0.x Server managed by a WebSphere Admin Agent, stop the WebSphere Admin Agent and restart the Admin Agent with the modified script or with the new Windows

Service. Restart any managed WebSphere Application Server instances after restarting the modified WebSphere Admin Agent.

CAUTION: For any server with a corresponding Windows service: Attempts to start a server by using a startup script may launch a Windows service instead. If the startup script is instrumented with the Foglight for Java EE Technologies, then the resulting WebSphere process is instrumented as well. Otherwise, the integration properties of the Service are used. This release changes the behavior of services launched as a result of a start script invocation.

CAUTION: Foglight starts a Java EE agent for each instrumented server. When the Java EE agent connects to the Management Server, the log file for the Java EE agent contains an INFO message stating that it has connected to the Nexus which is deployed on the Management Server. For example:

```
INFO [Multiplexer: 0]
com.quest.pas.agent.nexus.PsychoNexusStalker - Connected to Nexus at
APMFoglightServer:41705.
```

A

Appendix: Java EE Application Methods

This appendix provides information on methods collected for the Full detail instrumentation level and Component detail instrumentation level.

Full detail instrumentation level

The Java EE Agent instruments all the classes and methods in your application unless you explicitly exclude some.

At the package or class level, you can control which packages or classes are included or excluded by editing `UserClasses` in the *agent_instrumentation.config* file. For more information, see Changing user classes settings on page 58.

Within a class, you can control the number of recursive calls that are fully instrumented by editing the `RecursionLimit` in the *agent_instrumentation.config* file. For more information, see Setting the Request Recursion Limit on page 73.

Component detail instrumentation level

Focusing on a few crucial methods allows you to localize a potential trouble spot. Then you can add instrumentation targets to expand your search. Afterwards, by switching to the full detail instrumentation level, you can collect performance data on each method in your application.

Component detail instrumentation also reduces overhead. By collecting data only on a limited set of important classes and methods, you can run Foglight for Java EE Technologies with little degradation to your system's response time.

To implement component detail instrumentation, Foglight for Java EE Technologies measures selected methods in Servlets, JSPs, EJBs, JMS, JNDI, HTTP, and JDBC. You can include your own choice of custom components by editing the `CustomComponents` configuration section in the file *agent_instrumentation.config*. For more information, see Changing custom component settings on page 60.

Application methods in component detail instrumentation level

When you choose component detail, only the classes listed in the following sections are instrumented. Therefore, any of your application's classes that do not implement or extend the interfaces listed in this section are not tracked using Quest Software's Tag and Follow™ technology.

When you need specific information about all the classes in your application, record in full detail. When you want to gather data on key Java EE classes, record in component detail. You can also extend the component detail view to include custom components.

IMPORTANT: Depending on how your application server implements them, certain methods listed below may not appear in a session recorded with component detail.

Component detail classes for servlets

For classes that implement the `javax.servlet.Servlet` interface, Foglight for Java EE Technologies records:

```
void destroy()
void init(ServletConfig)
void service(ServletRequest, ServletResponse)
```

For classes that implement the `javax.servlet.http.HttpServlet` interface, Foglight for Java EE Technologies records:

```
void doDelete(HttpServletRequest, HttpServletResponse)
void doGet(HttpServletRequest, HttpServletResponse)
void doHead(HttpServletRequest, HttpServletResponse)
void doOptions(HttpServletRequest, HttpServletResponse)
void doPost(HttpServletRequest, HttpServletResponse)
void doPut(HttpServletRequest, HttpServletResponse)
void doTrace(HttpServletRequest, HttpServletResponse)
long getLastModified(HttpServletRequest)
void service(ServletRequest, ServletResponse)
void service(HttpServletRequest, HttpServletResponse)
```

For classes that implement the `javax.servlet.Filter` interface, Foglight for Java EE Technologies records:

```
void doFilter(ServletRequest, ServletResponse, FilterChain)
```

Component detail dlasses for JSPs

For classes that implement the `javax.servlet.jsp.HttpJspPage` interface, Foglight for Java EE Technologies records:

```
void _jspService(HttpServletRequest, HttpServletResponse)
```

Component detail classes for EJBs

Session beans, entity beans, and message-driven beans are instrumented. As well, all classes that implement the `javax.ejb.EJBLocalHome` interface or the `javax.ejb.EJBHome` interface are recorded, excluding RMI stub classes.

WebLogic® or WebSphere® EJB wrapper classes are excluded from component detail instrumentation. For example, in a PetStore session, the `productdetails` tree has a node for the class named `InventoryEJB_ripg5n_Impl`. This class contains methods such as `__WL_setBusy(Boolean)` and `__WL_isBusy()`. This class is a WebLogic-generated wrapper class for the InventoryEJB and is not instrumented in component detail.

Generic EJB classes

For classes that implement the `javax.ejb.EJBLocalHome` interface, or the `javax.ejb.EJBHome` interface, except for the RMI stub classes, Foglight for Java EE Technologies records:

```
<unknown> create<METHOD>(args)
void remove(java.lang.Object)
<unknown> find<METHOD>(args)
```

WebLogic and WebSphere wrapper classes are excluded from component detail instrumentation.

Component detail classes for session EJBs

For classes that implement the `javax.ejb.SessionBean` interface, Foglight for Java EE Technologies records all public methods, including:

```
void ejbCreate<METHOD>(args)
void ejbActivate()
void ejbPassivate()
void ejbRemove()
```

Foglight for Java EE Technologies records all the public methods that your application has added to the bean to support your custom business logic.

Foglight for Java EE Technologies does not record:

```
void setSessionContext(SessionContext)
```

Component detail classes for entity EJBs

For classes that implement the `javax.ejb.EntityBean` interface, Foglight for Java EE Technologies records all public methods, including:

```
void ejbCreate<METHOD>(args)
void ejbPostCreate<METHOD>(args)
void ejbRemove()
void ejbLoad()
void ejbStore()
void ejbFind<METHOD>(args)
void ejbSelect<METHOD>(args)
void ejbPassivate()
```

Foglight for Java EE Technologies does not record:

```
void setEntityContext(EntityContext)
void unsetEntityContext()
```

Component detail classes for message-driven EJBs

For classes that implement the `javax.ejb.MessageDrivenBean` interface, and the `javax.jms.MessageListener` interface, Foglight for Java EE Technologies records all public methods, including:

```
void onMessage(javax.jms.Message)
void ejbRemove()
```

Foglight for Java EE Technologies does not record:

```
void setMessageDrivenContext(MessageDrivenContext)
```

Component detail annotations for EJB3

For classes that are annotated with `javax.ejb.Stateless`, Foglight for Java EE Technologies records all public methods, including methods that are annotated with the following:

```
javax.annotation.PostConstruct
javax.annotation.PreDestroy
```

For classes that are annotated with `javax.ejb.Stateful`, Foglight for Java EE Technologies records all public methods, including methods that are annotated with the following:

```
javax.annotation.PostConstruct
javax.annotation.PreDestroy
javax.ejb.PostActivate
javax.ejb.PrePassivate
javax.ejb.Init
javax.ejb.Remove
```

For classes that are annotated with `javax.ejb.MessageDriven`, Foglight for Java EE Technologies records all public methods, including methods that are annotated with the following:

```
javax.annotation.PostConstruct
javax.annotation.PreDestroy
```

For classes that are annotated with `javax.interceptor.Interceptors`, Foglight for Java EE Technologies records all public methods, including methods that are annotated with the following:

```
javax.annotation.PostConstruct
javax.annotation.PreDestroy
javax.ejb.PostActivate
javax.ejb.PrePassivate
```

For classes that are annotated with `javax.persistence.Entity`, Foglight for Java EE Technologies records all public methods, including methods that are annotated with the following:

```
javax.persistence.PrePersist
javax.persistence.PostPersist
javax.persistence.PreRemove
javax.persistence.PostRemove
javax.persistence.PreUpdate
javax.persistence.PostUpdate
javax.persistence.PostLoad
```

For classes that are annotated with `javax.persistence.EntityListeners`, Foglight for Java EE Technologies records all public methods, including methods that are annotated with the following:

```
javax.persistence.PrePersist

javax.persistence.PostPersist

javax.persistence.PreRemove

javax.persistence.PostRemove

javax.persistence.PreUpdate

javax.persistence.PostUpdate

javax.persistence.PostLoad
```

For classes that are annotated with `javax.persistence.MappedSuperclass`, Foglight for Java EE Technologies records all public methods, including methods that are annotated with the following:

```
javax.persistence.PrePersist
javax.persistence.PostPersist
javax.persistence.PreRemove
javax.persistence.PostRemove
javax.persistence.PreUpdate
javax.persistence.PostUpdate
javax.persistence.PostLoad
```

Component detail classes for JDBC

For classes that implement the `java.sql.Connection` interface, Foglight for Java EE Technologies records:

```
void close()

void commit()
```

```
DatabaseMetaData getMetaData()
CallableStatement prepareCall(String)
CallableStatement prepareCall(String, int, int)
CallableStatement prepareCall(String, int, int, int)
PreparedStatement prepareStatement(String)
PreparedStatement prepareStatement(String, int)
PreparedStatement prepareStatement(String, int[])
PreparedStatement prepareStatement(String, int, int)
PreparedStatement prepareStatement(String, int, int, int)
PreparedStatement prepareStatement(String, String[])
void releaseSavepoint(Savepoint)
void rollback()
void rollback(Savepoint)
Savepoint setSavepoint()
Savepoint setSavepoint(String)
```

For classes that implement the `java.sql.Driver` interface, Foglight for Java EE Technologies records:

```
Connection connect(String, Properties)
```

For classes that implement the `java.sql.PreparedStatement` interface, Foglight for Java EE Technologies records:

```
void addBatch()
boolean execute()
ResultSet executeQuery()
int executeUpdate()
ResultSetMetaData getMetaData()
ParameterMetaData getParameterMetaData()
```

NOTE: If you are using a `ResultSet` implementation for your application with “scrollable” and “sensitive to changes” (`TYPE_SCROLL_SENSITIVE`) enabled, you may see unexpected database activity in the single trace view, if your users employed cursor movement methods such as `last()` or `next()`. This is because a `ResultSet` marked as “scrollable” and “sensitive” can require internal calls back to the database, to fetch data, as a result of method calls that move the cursor.

For classes that implement the `java.sql.ResultSet` interface, Foglight for Java EE Technologies records:

```
void deleteRow()
void insertRow()
void refreshRow()
void updateRow()
void moveToCurrentRow()
void moveToInsertRow()
boolean absolute(int)
void afterLast()
void beforeFirst()
boolean first()
```

```
boolean last()
boolean next()
boolean previous()
boolean relative(int)
```

For classes that implement the `java.sql.Statement` interface, Foglight for Java EE Technologies records:

```
boolean relative(int)
void addBatch(String)
boolean execute(String)
boolean execute(String, int)
boolean execute(String, int[])
boolean execute(String, String[])
int[] executeBatch()
resultSet executeQuery(String)
int executeUpdate(String)
int executeUpdate(String, int)
int executeUpdate(String, int[])
int executeUpdate(String, String[])
```

For classes that implement the `javax.sql.ConnectionPoolDataSource` interface, Foglight for Java EE Technologies records:

```
PooledConnection getPooledConnection()
PooledConnection getPooledConnection(String, String)
```

For classes that implement the `javax.sql.DataSource` interface, Foglight for Java EE Technologies records:

```
Connection getConnection()
Connection getConnection(String, String)
```

For classes that implement the `javax.sql.PooledConnection` interface, Foglight for Java EE Technologies records:

```
void close()
Connection getConnection()
```

For classes that implement the `javax.sql.XADataSource` interface, Foglight for Java EE Technologies records:

```
XAConnection getXAConnection()
XAConnection getXAConnection(String, String)
```

Classes for JMS, JNDI, HTTP, and RMI

Foglight for Java EE Technologies records the same information in component detail as for the full detail level.

JMS

Foglight for Java EE Technologies records the following calls to methods in JMS interfaces:

- `javax.jms.QueueSender`–all `send(...)` methods
- `javax.jms.TopicPublisher`–all `publish` methods
- `javax.jms.MessageConsumer`–all `receive` methods
- `javax.jms.MessageListener`–all `onMessage` methods
- `javax.jms.Message`–all methods

- `javax.jms.Destination`—all methods
- `javax.jms.Queue`—all methods
- `javax.jms.Topic`—all methods
- `javax.jms.TopicSubscriber`—all methods
- `javax.jms.QueueReceiver`—all methods

JNDI

Foglight for Java EE Technologies records calls to all methods of these interfaces:

- `javax.naming.Context`
- `javax.naming.NamingEnumeration`
- `javax.naming.directory.DirContext`

HTTP

Foglight for Java EE Technologies tracks all the service requests that enter your instrumented application server, unless you explicitly choose to exclude some. You also have the option of splitting one service request and tracking each part separately.

RMI

Foglight for Java EE Technologies instruments all the RMI calls that enter your instrumented system.

B

Appendix: Managing Permissions for the Java EE Integration Agent

To manually adjust the permissions for the Java EE Integration Agent, use the Foglight for Java EE Technologies Agent Properties page.

Before you make any changes, review the following topics:

- Managing permissions on remote filesystems on page 109
- Windows Services permissions on page 110
- Permissions required by the application server user on page 111

To edit Java EE Integration Agent permissions:

1. On the navigation panel, under Dashboards, click **Administration > Agents > Agent Properties**.
2. On the Agent Properties dashboard, expand the **ApplicationServers** Namespace.
3. Click **JavaIntegrator**. The Properties page for the Java EE Integration Agent opens.
4. Set the permissions as necessary and click **Save**.

Managing permissions on remote filesystems

Before creating and activating an instance of the Java EE Integration Agent on a remote host, the user running the Agent Manager process must have permission to create the Installation directory (`DEPLOYMENT_DIRECTORY`).

Before submitting an integration task to a Java EE Integration Agent, the Java EE Integration Agent must be granted permission to make the necessary changes to application server files and to create backup copies of the files being changed. The following permissions must be granted to the user running the Agent Manager process (for the application server being integrated):

Table 8. User permissions required.

Remote File System directory	Directory/App Server file	Required permission
`DEPLOYMENT_DIRECTORY`	`.`	`(managed)`
	`..`	`ugo+rwx`
`(JBoss)` `JBOSS_HOME/bin`	`.`	`ugo+rwx`
	`standlone.bat, domain.bat, run.bat`	`ugo+rw`
	`standalone.sh, domain.sh, run.sh`	
`(Tomcat)` `CATALINA_HOME/bin`	`.`	`ugo+rwx`
	`catalina.bat, catalina.sh`	`ugo+rw`

Table 8. User permissions required.

Remote File System directory	Directory/App Server file	Required permission
(WebLogic)	.	ugo+rwx
WL_HOME/server/bin	startNodeManager.cmd, startManagedWebLogic.cmd, startWLS.cmd	ugo+rw
	startNodeManager.sh, startManagedWebLogic.sh, startWLS.sh	
(WebLogic)	.	ugo+rwx
DOMAIN_HOME/common/bin	startWebLogic.cmd, startWebLogic.sh	ugo+rw
(WebSphere)	.	ugo+rwx
PROFILE_HOME/bin	startServer.bat, startNode.bat, startManager.bat, setupCmdLine.bat	ugo+rw
	startServer.sh, startNode.sh, startManager.sh, setupCmdLine.sh	
(WebSphere)	.	ugo+rwx
PROFILE_HOME/properties	server.policy	ugo+rw
(WebSphere)	.	ugo+rwx
WAS_HOME/bin	setupCmdLine.bat	ugo+rw
	setupCmdLine.sh, startServer.sh, startNode.sh, startManager.sh,	

If the Agent Manager and Application Server processes use the same user, the "g" and "o" permissions can be omitted.

If the Agent Manager user is in the same group as the Application Server files being modified, the "u" and "o" can be omitted.

If the Agent Manager is not the same as the Application Server user, or in the same group, then the "o" must be included.

Windows Services permissions

For integration of Windows Services, the Agent Manager user requires permission to run the *regedit.exe* executable in export or import modes on one or more registry keys, under the HKEY_LOCAL_MACHINE hive.

For Tomcat Windows Services:

Table 9. Tomcat Windows Registry keys.

HKLM Windows Registry Keys (Tomcat)	Read	Write
System\CurrentControlSet\Services	Y	N
Software \Apache Software Foundation\Procrun 2.0\<service>\ Parameters\Java	Y	Y

Table 9. Tomcat Windows Registry keys.

HKLM Windows Registry Keys (Tomcat)	Read	Write
Software\JavaSoft\Java Runtime Environment	Y	N
Software\JavaSoft\Java Development Kit	Y	N
Software \Wow6432Node\Apache Software Foundation\Procrun 2.0\<service>\ Parameters\Java	Y	Y
Software \Wow6432Node\JavaSoft\Java Runtime Environment	Y	N
Software\Wow6432Node\JavaSoft\Java Development Kit	Y	N

For WebLogic or WebSphere Windows Services:

Table 10. WebLogic or WebSphere Windows Services.

HKLM Windows Registry Keys (WebLogic, WebSphere)	Read	Write
System\CurrentControlSet\Services\<service>\Parameters	Y	Y

Permissions required by the application server user

The Java EE Integration Agent automatically grants permissions on the files and directories within the `DEPLOYMENT_DIRECTORY` to be readable by any other user on the remote filesystem. By default, only users in the same group as the Agent Manager (and the Agent Manager user itself) can create files within the `DEPLOYMENT_DIRECTORY`.

If the Application Server user is the same as the Agent Manager user, or belongs to the group owning the Agent Manager files, these category permissions do not need to be changed.

Otherwise, the permissions for Dynamic Directories should be changed to `rwx` for the `For Other (default r-x)` setting.

The Java EE Integration Agent manages file permissions using a set of six categories to apply particular permissions to particular types of files. Review the categories and the files they represent in the following table.

Table 11. Permission categories and files.

Permission Category	Included Files and Directories
Stock Directories	• `config`, `lib`, `scripts`, and their subdirectories within versioned homes • All parent directories of `DEPLOYMENT_DIRECTORY` created by the Agent Manager
Stock Files	• All files within `config` and `lib` and their subdirectories
Stock Scripts	By default, stock scripts are not set executable as they are sourced instead of run. If manual execution of `pre-instrumentor.sh` is required, it may be provided as an argument to `/bin/sh` instead of changing the permissions for this category. Includes: • Non-customized copies of `integrate.cmd`, `integrate.sh` • Files within versioned `scripts` directory

Table 11. Permission categories and files.

Permission Category	Included Files and Directories
Dynamic Directories	This is the most important category, as it affects the ability of the Application Server user to create files within the `DEPLOYMENT_DIRECTORY` Includes: • `DEPLOYMENT_DIRECTORY` • `bootstrap`, `logs` and `state` sub-directories • config file parent directories (except `config`, which is created using stock directory permissions) • `DEPLOYMENT_DIRECTORY/exports` (this is the WORKING_SUBDIR) • Each versioned home folder • Integration-specific bootstrap folders • log directories that the Java EE Integration Agent creates
Dynamic Files	• `jee_builds.properties` • Agent-local config files (*agent.config*, *log.config*, *agenttype.config*) • Backups of integrated application server files • Backups of Windows Service exports
Dynamic Scripts	Customized integration scripts (for example, `integrate-MyTask.sh`)

Each category provides a set of permissions for the file or directory owner, group, and everyone else. Also provided are the abilities to set the `setuid`, `setgid` and `sticky` bits.

The `setuid` bit can be set on script files to have the launched process take on the user ID of the script file itself. This ability is not needed in stock integrations.

The `setgid` bit can be set on script files and directories. When set on script files, the effect is similar to the `setuid` bit, except the launched process takes on the group ID of the script file, instead of the user ID. When set on a directory, the `setgid` bit results in files created in that directory having the same group ownership as the directory itself, rather than the group of the user who creates the file. By default, the `setgid` bit is set on Dynamic Directories, so that the Java EE Integration Agent can maintain these directories regardless of which user creates files within them.

The `sticky` bit can be used on directories with `other` write permission to prevent a user from deleting another user's file.

With the lone exception noted above, these file permissions should not need to be changed during stock integrations. The options exist to aid integrations where extraordinary circumstances require them.

About Us

We are more than just a name

We are on a quest to make your information technology work harder for you. That is why we build community-driven software solutions that help you spend less time on IT administration and more time on business innovation. We help you modernize your data center, get you to the cloud quicker and provide the expertise, security and accessibility you need to grow your data-driven business. Combined with Quest's invitation to the global community to be a part of its innovation, and our firm commitment to ensuring customer satisfaction, we continue to deliver solutions that have a real impact on our customers today and leave a legacy we are proud of. We are challenging the status quo by transforming into a new software company. And as your partner, we work tirelessly to make sure your information technology is designed for you and by you. This is our mission, and we are in this together. Welcome to a new Quest. You are invited to Join the Innovation™.

Our brand, our vision. Together.

Our logo reflects our story: innovation, community and support. An important part of this story begins with the letter Q. It is a perfect circle, representing our commitment to technological precision and strength. The space in the Q itself symbolizes our need to add the missing piece—you—to the community, to the new Quest.

Contacting Quest

For sales or other inquiries, visit www.quest.com/contact.

Technical support resources

Technical support is available to Quest customers with a valid maintenance contract and customers who have trial versions. You can access the Quest Support Portal at https://support.quest.com.

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. The Support Portal enables you to:

- Submit and manage a Service Request.
- View Knowledge Base articles.
- Sign up for product notifications.
- Download software and technical documentation.
- View how-to-videos.
- Engage in community discussions.
- Chat with support engineers online.
- View services to assist you with your product.