

Oracle® Waveset 8.1.1 System Administrator's Guide

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related software documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

| | |
|---|----|
| Preface | 9 |
| 1 Configuring Waveset | 15 |
| Using Waveset.properties to Configure Waveset | 15 |
| ▼ To Edit Waveset.properties | 16 |
| Using the Administrator Interface to Configure Waveset | 16 |
| ▼ To Configure Waveset Settings from the Administrator Interface | 16 |
| Reconciler Settings | 17 |
| ▼ To Configure Reconciler Settings | 17 |
| ▼ To View Reconciler Status | 18 |
| Scheduler Settings | 18 |
| Email Template Server Settings | 19 |
| Configuring JMX Monitoring | 19 |
| Enabling JMX | 20 |
| To Configure JMX Polling Settings | 21 |
| To Set Java Security Manager MBean Permissions | 21 |
| Filtering Resource MBeans | 22 |
| Viewing JMX Data | 23 |
| Data Exposed in JMX | 24 |
| Editing Default Server Settings | 25 |
| ▼ To Edit the Default Server Settings | 25 |
| 2 Working with Firewalls, Load Balancers, or Proxy Servers | 27 |
| Servlet APIs | 27 |
| Configuring Logs to Work with Load Balancers | 28 |

| | | |
|----------|--|----|
| 3 | Working with Logs | 29 |
| | Preventing Audit Log Tampering | 29 |
| | ▼ To Configure Tamper-Resistant Logging | 30 |
| | Removing Records from the System Log | 31 |
| | ▼ To Schedule a Task to Remove Old Records From the System Log | 32 |
| | | |
| 4 | Performance Tuning | 33 |
| | Before You Begin Tuning | 34 |
| | Important Notes | 34 |
| | Related Documentation and Web Sites | 34 |
| | Tuning Roadmap | 36 |
| | Tuning Your Deployment Environment | 37 |
| | Tuning Your Java EE Environment | 38 |
| | Tuning Your Application Server | 39 |
| | Tuning Your Repository Database | 41 |
| | Tuning Waveset Performance | 54 |
| | General Performance Tunings | 55 |
| | Tuning Active Sync Adapter Performance | 56 |
| | Tuning Admin Cache Initialization | 56 |
| | Tuning Bulk Loading | 57 |
| | Tuning Configurable XML Objects | 58 |
| | Working with Waveset Wizards | 64 |
| | Updating Database Statistics | 66 |
| | Tuning Data Exporter | 67 |
| | Tuning the General XML | 67 |
| | Tuning Oracle Waveset Service Provider | 68 |
| | Tuning the Waveset Web Interface | 68 |
| | Tuning Initial Loads | 68 |
| | Tuning Memory Requirements | 69 |
| | Tuning Operating System Kernels | 69 |
| | Tuning the Waveset Profiler | 70 |
| | Tuning Provisioner | 70 |
| | Tuning Reconciliation | 72 |
| | Tuning Resource Queries | 75 |
| | Tuning the Scheduler | 76 |

| | |
|--|------------|
| Tuning Sessions | 80 |
| Tuning the Oracle Waveset Gateway | 80 |
| Tuning the Task Bar | 83 |
| Debugging Performance Issues | 83 |
| Working With Waveset Debug Pages | 84 |
| Working with the Waveset Profiler | 89 |
| Working With Other Debugging Tools | 108 |
| 5 Tracing and Troubleshooting | 117 |
| Before You Begin Tracing or Troubleshooting | 117 |
| Intended Audience | 117 |
| Important Notes About Tracing and Troubleshooting Waveset | 118 |
| Before Calling Support | 118 |
| Related Documentation and Web Sites | 125 |
| Process Overview | 126 |
| Tracing Waveset Objects and Activities | 127 |
| How To Configure Tracing | 127 |
| How to View Trace Files | 132 |
| Tracing the Waveset Server | 133 |
| Tracing Adapters | 133 |
| Tracing Auditor | 141 |
| Tracing Custom Code | 142 |
| Tracing Exceptions | 142 |
| Tracing Forms | 143 |
| Tracing Global XPRESS | 144 |
| Tracing PasswordSync | 145 |
| Tracing Rule-Driven Members Caches | 148 |
| Tracing Oracle Waveset Service Provider Delegated Administration | 149 |
| Tracing Reconciliation | 150 |
| Tracing the setRepo Command | 152 |
| Tracing SPML | 152 |
| Tracing the Task Scheduler | 156 |
| Tracing Workflows | 157 |
| Locating Version Information | 159 |
| Tracing the Waveset Gateway Objects and Activities | 160 |

| | |
|--|------------|
| How to Configure Gateway Tracing | 160 |
| How to Capture Encrypted Information | 163 |
| How to Configure Tracing for the PowerShellExecutor.dll Add-On | 164 |
| How to Capture Dr. Watson Logs | 166 |
| Troubleshooting and Fixing Common Problems | 166 |
| Working with Debugging Tools | 167 |
| Debugging Errors Displayed in the Browser | 172 |
| Troubleshooting Adapters | 173 |
| Troubleshooting Auditor | 179 |
| Troubleshooting ClassNotFound Exceptions | 180 |
| Troubleshooting Form Problems | 180 |
| Troubleshooting the Gateway | 181 |
| Troubleshooting Java Code Problems | 182 |
| Troubleshooting Low Memory Conditions | 183 |
| Troubleshooting PasswordSync Problems | 186 |
| Troubleshooting Reconciliation Problems | 188 |
| Troubleshooting Repository Connection Problems | 189 |
| Troubleshooting Server-Related Problems | 191 |
| Troubleshooting Service Provider Problems | 192 |
| Troubleshooting an SPML Configuration | 192 |
| Troubleshooting Upgrades | 193 |
| 6 Errors and Exceptions | 195 |
| Before Working with Waveset Messages | 195 |
| Important Notes About Waveset Messages | 195 |
| Related Documentation and Web Sites | 196 |
| About Waveset Errors and Exceptions | 197 |
| Where Messages Are Stored | 197 |
| How Messages Are Displayed | 198 |
| Error Severity Levels | 198 |
| Viewing Errors in the System Log Report | 199 |
| ▼ To Run a System Log Report From the Administrator Interface | 199 |
| ▼ To Run a System Log Report From the Command-Line Interface | 202 |
| Customizing a Default Error Message | 202 |
| ▼ To Modify the ErrorUIConfig Object: | 203 |

Index 207

Preface

This *Oracle Waveset 8.1.1 System Administrator's Guide* contains information that you must understand to administer the systems running Oracle Waveset (Waveset) software.

Oracle Waveset was formerly known as Sun Identity Manager. This release of Waveset follows Oracle's acquisition of Sun Microsystems. Some parts of the product and documentation may still refer to the original Sun Identity Manager product name.

Before You Read This Book

Waveset is one component of a software infrastructure that supports enterprise applications distributed across a network or Internet environment. You should be familiar with the documentation that can be accessed online at <http://docs.sun.com/app/docs/prod/ent.comput?|en#hic> and <http://docs.sun.com/app/docs/prod/ident.mgmt?|en#hic>.

Because Waveset Directory Server is used as the data store in an Waveset deployment, you should be familiar with the documentation provided with that product. You can access the Directory Server documentation online at http://docs.sun.com/coll/DirectoryServer_04q2.

Who Should Use This Book

This book is for application server and database administrators, front-line support engineers, and partners who administer systems running Waveset and are interested in optimizing Waveset performance and responsible for maintaining Waveset in a deployment environment.

Waveset administrators must understand the following technologies:

- Lightweight Directory Access Protocol (LDAP)
- Java technology
- JavaServer Pages (JSP) technology
- Hypertext Transfer Protocol (HTTP)
- Hypertext Markup Language (HTML)
- Extensible Markup Language (XML)

How This Book is Organized

This guide is organized into the following chapters:

[Chapter 1, “Configuring Waveset,”](#) describes how to set Waveset properties and configure Waveset servers to run specific tasks.

[Chapter 2, “Working with Firewalls, Load Balancers, or Proxy Servers,”](#) describes Waveset uses Uniform Resource Locators (URLs) and contains information about using Waveset when firewalls, load balancers, or proxy servers are in place.

[Chapter 3, “Working with Logs,”](#) describes tasks you might perform when you are working with audit or system logs.

[Chapter 4, “Performance Tuning,”](#) describes tools, methodologies, and references you can use to improve Waveset performance and to debug performance-related issues.

[Chapter 5, “Tracing and Troubleshooting,”](#) explains how to use Waveset tracing to fix problems, solve performance issues, and understand how things flow in the product components.

[Chapter 6, “Errors and Exceptions,”](#) contains information about Waveset error and exception messages.

Related Documentation

Other publications in the Oracle Waveset library include:

| Primary Audience | Title | Description |
|------------------|--|--|
| All Audiences | Oracle Waveset 8.1.1 Overview | Provides an overview of Waveset features and functionality. Provides product architecture information. |
| | Oracle Waveset 8.1.1 Release Notes | Describes known issues, fixed issues, and late-breaking information not already provided in the Waveset documentation set. |

| Primary Audience | Title | Description |
|-------------------------|--|---|
| System Administrators | <i>Oracle Waveset Installation</i> | Describes how to install Waveset and optional components such as the Oracle Waveset Gateway and PasswordSync. |
| | <i>Oracle Waveset 8.1.1 Upgrade</i> | Provides instructions on how to upgrade from an older version of Sun Identity Manager to Oracle Waveset. |
| | <i>Oracle Waveset 8.1.1 System Administrator's Guide</i> | Contains information and instructions to help system administrators manage, tune, and troubleshoot their Waveset installation. |
| Business Administrators | <i>Oracle Waveset 8.1.1 Business Administrator's Guide</i> | Describes how to use Waveset's provisioning and auditing features. Covers the user interfaces, user and account management, reporting, and more. |
| System Integrators | <i>Oracle Waveset 8.1.1 Deployment Guide</i> | Describes how to deploy Waveset in complex IT environments. Topics covered include working with identity attributes, data loading and synchronization, configuring user actions, applying custom branding, and so on. |
| | <i>Oracle Waveset 8.1.1 Deployment Reference</i> | Contains information about workflows, forms, views, and rules, as well as the XPRESS language. |
| | <i>Oracle Waveset 8.1.1 Resources Reference</i> | Provides information about installing, configuring, and using resource adapters. |
| | <i>Oracle Waveset Service Provider 8.1.1 Deployment</i> | Describes how to deploy Oracle Waveset Service Provider, and how views, forms, and resources differ from the standard Waveset product. |
| | <i>Oracle Waveset 8.1.1 Web Services</i> | Describes how to configure SPML support, which SPML features are supported (and why), and how to extend support in the field. |

Documentation Updates

Corrections and updates to this and other Oracle Waveset and Sun Identity Manager publications are posted to the Identity Manager Documentation Updates website:

<http://blogs.sun.com/idmdocupdates/>

An RSS feed reader can be used to periodically check the website and notify you when updates are available. To subscribe, download a feed reader and click a link under Feeds on the right side of the page. Starting with version 8.0, separate feeds are available for each major release.

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Oracle is not responsible for the availability of third-party web sites mentioned in this document. Oracle does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Oracle will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Documentation, Support, and Training

Information about the following additional resources is available on the Internet.

- [Documentation](http://docs.sun.com) (<http://docs.sun.com>)
- [Support](http://www.sun.com/support/) (<http://www.sun.com/support/>)
- [Training](http://education.oracle.com) (<http://education.oracle.com>)

Oracle Welcomes Your Comments

Oracle is interested in improving its documentation and welcomes your comments and suggestions.

To share your comments, go to <http://docs.sun.com> and click the Feedback link. In the online form, enter the document title and part number. The part number is a seven-digit or nine-digit number that can be found on the title page of the book or at the top of the document.

For example, the title of this book is *Oracle Waveset 8.1.1 System Administrator's Guide*, and the part number is 820-7976-10.

Typographic Conventions

The following table describes the typographic changes used in this book.

TABLE P-1 Typographic Conventions

| Typeface | Meaning | Examples |
|--------------------------------------|--|--|
| AaBbCc123 (Monospace) | API and language elements, HTML tags, web site URLs, command names, file names, directory path names, onscreen computer output, sample code. | Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>% You have mail.</code> |
| AaBbCc123 (Monospace bold) | What you type, when contrasted with onscreen computer output. | <code>% su Password:</code> |
| <i>AaBbCc123</i> (Italic) | Book titles, new terms, words to be emphasized. A placeholder in a command or path name to be replaced with a real name or value. | Read Chapter 6 in the <i>Oracle Waveset 8.1.1 System Administrator's Guide</i> . These options are called <i>class</i> options. <i>Do not</i> save the file. The file is located in the <i>install-dir/bin</i> directory. |

Shell Prompts in Command Examples

The following table shows the default UNIX system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P-2 Shell Prompts

| Shell | Prompt |
|---|----------------------------|
| C shell | <code>machine_name%</code> |
| C shell for superuser | <code>machine_name#</code> |
| Bourne shell and Korn shell | <code>\$</code> |
| Bourne shell and Korn shell for superuser | <code>#</code> |

Note – The Windows command-line prompt is `c:\.`

Configuring Waveset

This chapter describes how to set Waveset properties and configure Waveset servers to run specific tasks.

The information in this chapter is organized into the following topics:

- “Using `Waveset.properties` to Configure Waveset” on page 15
- “Using the Administrator Interface to Configure Waveset” on page 16
- “Reconciler Settings” on page 17
- “Scheduler Settings” on page 18
- “Email Template Server Settings” on page 19
- “Configuring JMX Monitoring” on page 19
- “Editing Default Server Settings” on page 25

Using `Waveset.properties` to Configure Waveset

Some Waveset settings can only be updated by editing `Waveset.properties` in a text editor. The `Waveset.properties` file is located in the `config` directory in your base Waveset installation directory (`$WSHOME/config` or `%WSHOME%/config`).

The `Waveset.properties` file is loosely organized into the following sections:

- Misc Options
- Authentication Options
- Object Cache Options
- Provisioning Options
- Task Scheduler Options
- Resource Adapter Options
- List Cache Properties
- Rule Driven Members Cache Properties
- Session Settings
- Server Listeners

- UI Options
- Global Options for the JNDI Context Pooling
- Task Executor Options
- XML Parsing Options

▼ **To Edit** `Waveset.properties`

- 1 **Open the `Waveset.properties` file in a text editor.**

You can find the `Waveset.properties` file in the `config` directory in your base Waveset installation directory (`$WSHOME/config` or `%WSHOME%/config`).

- 2 **Modify the setting (or settings) that you want to change and save the file.**
- 3 **Repeat as needed on other Waveset instances.**

Using the Administrator Interface to Configure Waveset

This section provides basic instructions for using the Administrator tool to configure server-specific settings for reconciler, scheduler, JMX and other tasks.

▼ **To Configure Waveset Settings from the Administrator Interface**

To edit server-specific settings that enable Waveset servers to run only specific tasks

- 1 **In the Administrator interface, click `Configure` → `Servers`.**
- 2 **When the `Configure Servers` page displays, click a server name in the list.**
- 3 **When the `Edit Server Settings` page displays, modify the settings that you want to change, and then click `Save`.**

Reconciler Settings

The *reconciler* is the Waveset component that performs reconciliation. To learn about reconciliation, see “[Account Reconciliation](#)” in *Oracle Waveset 8.1.1 Business Administrator’s Guide*.

This section contains instructions to help you perform the following tasks:

- “[To Configure Reconciler Settings](#)” on page 17
- “[To View Reconciler Status](#)” on page 18

Note – For information about tuning and troubleshooting the reconciler, see [Chapter 5](#), “[Tracing and Troubleshooting](#).”

▼ To Configure Reconciler Settings

Use the following steps to configure the Reconciler:

- 1 **In the Administrator interface, click [Configure](#) → [Servers](#).**
- 2 **When the [Configure Servers](#) page displays, click a server name in the list.**
- 3 **Select the [Reconciler](#) tab.**
Reconciler settings display on the [Edit Server Settings](#) page by default.
- 4 **Accept the default values or deselect the [Use the default option to specify custom values](#).**

Note – To change the default reconciler settings used by Waveset servers, see “[Editing Default Server Settings](#)” on page 25.

- 5 **Configure the following settings.**
 - **Parallel Resource Limit.** Specify the maximum number of resource threads that the reconciler can process in parallel. Resource threads allocate work items to worker threads, so if you add additional resource threads, you may also need to increase the maximum number of worker threads. For new installations, the default value is 3.
 - **Minimum Worker Threads.** Specify the number of processing threads that the reconciler will always keep alive. For new installations, the default value is 2.
 - **Maximum Worker Threads.** Specify the maximum number of processing threads that the reconciler can use. The reconciler will only start as many threads as the workload requires. This places a limit on that number. Worker threads automatically close if they are idle for a short duration. For new installations, the default value is 6.

▼ To View Reconciler Status

Use the following steps to view reconciler status information:

- 1 **Open the Reconciler Status debug page by typing the following URL into your browser:**

`http://<AppServerHost>:<Port>/idm/debug/Show_Reconciler.jsp`

Where AppServerHost is a host that has the reconciler enabled.

Note – You must have the Debug capability to view /idm/debug/ pages. For information about capabilities, see “[Assigning Capabilities to Users](#)” in *Oracle Waveset 8.1.1 Business Administrator’s Guide*.

- 2 **Refresh the Reconciler Status page to view updated reconciler status information. For additional information about this page, click Help.**

Scheduler Settings

The Scheduler component controls task scheduling in Waveset.

To configure scheduler settings on a particular server,

1. Follow the first two steps under “[Using the Administrator Interface to Configure Waveset](#)” on page 16.
2. Select the Scheduler tab.

The following Use default options are enabled by default, which allows the scheduler to behave more efficiently when the Waveset server and repository are very busy. You can accept the default settings or deselect the Use default option to specify custom values.

- **Use default (Automatic).** Sets the scheduler start up mode to Automatic.
- **Use default (false).** Disables scheduler debug tracing.
- **Use default (100).** Allows the scheduler to run a maximum of 100 tasks at any one time.
- **Use default (Allow any task).** Allows any task to execute on the server.

To specify custom settings for the scheduler, deselect one or more of the Use default options and configure the following options:

- **Scheduler Startup.** Select a startup mode for the scheduler on this server:
 - **Automatic.** Starts when the server is started. This setting is the default startup mode.
 - **Manual.** Starts when the server is started, but remains suspended until manually started.
 - **Disabled.** Does not start when the server is started.

- **Tracing Enabled.** Select this option to activate scheduler debug tracing to standard output on this server.
 - **Maximum Concurrent Tasks.** Select this option to specify the maximum number of tasks, other than the default, that the Scheduler will run at any one time. Requests for additional tasks above this limit will either be deferred until later or run on another server.
 - **Task Restrictions.** Specify the set of tasks that can execute on the server. To do this, select one or more tasks from the list of available tasks. The list of selected tasks can be an inclusion or exclusion list depending on the option you select. You can choose to allow all tasks except those selected in the list (the default behavior), or allow only the selected tasks.
3. Click Save to save your changes to the server settings.

To change the default scheduler settings for Waveset servers, see [“Editing Default Server Settings” on page 25](#). For information about tuning and troubleshooting the scheduler, see [“Tuning the Scheduler” on page 76](#) and [“Tracing the Task Scheduler” on page 156](#).

Email Template Server Settings

To configure SMTP server settings, follow the steps under [“Using the Administrator Interface to Configure Waveset” on page 16](#). Select the Email Template tab.

Specify the default email server by clearing the Use Default selection and entering the mail server to use, if other than the default. The text you enter is used to replace the *smtpHost* variable in Email Templates.

Simple Mail Transfer Protocol (SMTP) is the standard for email transmissions across the Internet.

To change the default SMTP settings for Waveset servers, see [“Editing Default Server Settings” on page 25](#).

Configuring JMX Monitoring

Java Management Extensions (JMX) is a Java technology that allows for managing and/or monitoring applications, system objects, devices, and service oriented networks. The managed/monitored entity is represented by objects called MBeans (for Managed Bean).

This section describes how to configure JMX on an Waveset server so that a JMX client can monitor the system for changes.

Note – You can also configure Waveset to make audit events available using JMX. For information, see “The JMX Publisher Type” in *Oracle Waveset 8.1.1 Business Administrator’s Guide*.

Enabling JMX

The JRE did not provide a JMX MBean server prior to Java 1.5, so using JMX required support from the application server. Starting with the Java 1.5 specification, the JRE must provide a built-in MBean server. Consequently, JMX is available from any application server running JRE version 1.5 or later.

Each application server that provides direct JMX support has a different technique for enabling JMX and providing access to the MBeans. You must consult the application server documentation for those instructions.

The MBean server in the JVM is enabled by default, but requires some additional options to be accessed by JConsole or other MBean viewing tools. In most cases, enabling JMX means the application server allows communication on a separate channel with a security profile that is different from that of the HTTP channel. When deciding to enable this channel, you must consider the need to access the MBean data (without restarting the server) against the additional security exposure. Oracle strongly recommends enabling remote JMX access on development and test servers.

For Local Access to MBeans

To enable local access to JMX, where JConsole running on the same host can access MBeans but JConsole running on other hosts cannot, add the following option to the JVM startup:

```
-Dcom.sun.management.jmxremote
```

You must decide whether to use this option on production servers based on your security requirements and the need to access the MBean data without restarting the server.

For Remote Access to MBeans

To enable remote access to JMX, where JConsole running on a remote host can access the MBeans but JConsole running on other hosts cannot, add the following options to the JVM startup:

```
-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=<port num>
```

When you enable remote JMX access, you must carefully consider the security ramifications. For a test server, you can add the following option, which allows non-SSL access to the JMX server without providing any authentication data:

```
-Dcom.sun.management.jmxremote.authenticate=false
```

Using this option is very insecure. You should never use this option on a production server, but it might be appropriate for a test server. For more information, consult the following web site:

<http://download.oracle.com/>

[docs/cd/E17476_01/javase/1.5.0/docs/guide/management/agent.html#jmxagent](http://download.oracle.com/docs/cd/E17476_01/javase/1.5.0/docs/guide/management/agent.html#jmxagent).

To Configure JMX Polling Settings

The Cluster MBean provided by Waveset has both passive and active operational modes. In active mode, the MBean polls the Waveset repository for additional data and exposes that data through JMX.

Currently, enabling Cluster polling is not recommended because the operational cost of polling is greater than the information provided by the polling. However, if you choose to enable Cluster polling, you can use the following process:

1. Follow the steps described in “[Using the Administrator Interface to Configure Waveset](#)” on [page 16](#). Select the JMX tab.
2. Enable JMX cluster polling and configure the interval for the polling threads.

Use the following options:

- **Enable JMX Cluster Polling.** Enables or disables the polling thread for the JMX Cluster MBean. Because system resources are used for polling cycles, enable this option only if you plan to use JMX.
- **Polling Interval (ms).** Changes the default interval at which the server polls the repository for changes, when JMX is enabled. Specify the interval in milliseconds.

The default polling interval is set to 60000 milliseconds. To change it, enter the new value in the entry field provided.

3. Click Save to save changes to the server settings.

With the exception of the Cluster MBean, all other Waveset MBeans are passive, meaning they do not do any additional processing on their own. Most MBeans reflect the values of counters or other variables in the Waveset server. Because they are passive, they add very little processing load to the server.

To Set Java Security Manager MBean Permissions

If you deploy Waveset on an application server that has Java Security Manager enabled, you must add several MBean-related permissions to the application server security policy file, as follows:

```
grant codeBase "file:${application_server_instance}
  /Oracle Waveset_web_deployment_directory/" {
  permission javax.management.MBeanServerPermission "*";
  permission javax.management.MBeanPermission "*", "*";
  permission javax.management.MBeanTrustPermission "*";
};
```

These permissions enable the entire set of Waveset MBean monitoring actions.

Note – If you have a security requirement to disable certain MBean actions, refer to the following web site for more information, http://www.jcp.org/aboutJava/communityprocess/maintenance/jsr003/JMX_1.2_Security_Proposal.pdf.

Filtering Resource MBeans

By default, each Waveset resource has a corresponding JMX MBean that captures the operational statistics of Waveset's interaction with that resource. When an installation has a lot of resources, this may be too much of a good thing, so there is a way to exclude resources and particular operations on resources.

To disable JMX from reporting on different resources or operations on resources follow these steps:

1. Log in to the Waveset admin console.
2. Select the Configuration tab.
3. Select the Servers subtab.
4. Click the Edit Default Server Settings button.
5. Select the JMX tab.

This page contains two text fields:

- Use the top field to exclude one or more resources from being monitored by JMX.

To exclude a resource from JMX monitoring, you must provide a Java regular expression that matches the resource to be excluded. The pattern being compared is in the following form:

```
<resource type>:<resource version>:<resource name>
```

For example, a regular expression of `Database Table:*:Oracle*` excludes all resources with a name that starts with *Oracle*.

- Use the bottom field to exclude one or more operations (on a set of resources) from being monitored by JMX. To exclude a specific operation from being monitored, you must provide a Java regular expression that matches the operation to be excluded. Operation names are in the following form:

```
ACCOUNT_<Operation>
```

```
OBJECT_<Operation>
```

```
ACTION_<Operation>
```

The regular expression patterns must match those names. For example:

- To exclude all operations for non-account objects, enter

```
^OBJECT.*
```

- To exclude updates and deletes, enter

```
.*UPDATE$
```

```
.*DELETE$
```

JMX can monitor these operations:

```
ACCOUNT_CREATE
ACCOUNT_UPDATE
ACCOUNT_DELETE
ACCOUNT_GET
ACCOUNT_AUTHENTICATE
OBJECT_CREATE
OBJECT_UPDATE
OBJECT_DELETE
OBJECT_GET
OBJECT_LIST
ACTION_RUN
```

Viewing JMX Data

Because JMX is a Java standard, there are multiple consoles you can use to interact with MBeans. The Java 1.5 JDK provides the JConsole utility, which is a graphical MBean console that is well suited for displaying Waveset MBean data. JConsole can show the current static value of MBean attributes and it can chart individual (numeric) attributes over time. JConsole charts MBean attributes by periodically polling the MBean for its current value and displaying the attributes as a line chart.

Using JConsole Locally

Using JConsole to view a local server is as simple as starting JConsole and selecting the application server process from a list. When JConsole displays the new connection dialog, it scans the local host for any Java processes that offer an MBean server. Local JConsole connections connect to a local process. These connections are only possible when JConsole is running on the same host as the application server. If your application server does not show up in JConsole's local process list, you probably have not enabled JMX (see [“Enabling JMX” on page 20](#)).

Using JConsole Remotely

Assuming you started the JVM with the settings that allow remote MBean access, you can access a remote server's MBean by starting JConsole and connecting to a remote process. The exact Uniform Resource Identifier (URI) used to connect to the remote server depends on how you enabled remote JMX access. The URI is typically one of the following:

- `hostname:port` (if remote access was granted with the options shown above)
- `service:jmx:rmi://host:port/jndi/...`

Again, refer to your application server documentation for more information about how to remotely connect to the MBean server. If you have problems connecting JConsole to the application server, check the following URL for more information: http://download.oracle.com/docs/cd/E17476_01/javase/1.5.0/docs/guide/management/faq.html

Note – You can view the structure (but not the data) of the Waveset-provided MBean by going to the following debug page ([http://host:port/Oracle Waveset/debug](http://host:port/Oracle%20Waveset/debug)) and clicking the Show MBean button.

For more information on JMX, visit this website:

<http://www.oracle.com/technetwork/java/javase/tech/index.html>

Data Exposed in JMX

So what kind of data does Waveset expose using JMX MBeans? The following table contains a list of MBeans, describes the data they provide, and explains how to use that data.

| MBean | Description | Use |
|------------------|---|---|
| Cluster | Names of the servers in the Waveset cluster, including which servers Waveset considers active. | Provides a quick status of the cluster from the perspective of one Waveset server. |
| ObjectRepository | Waveset repository configuration data, including database name and version, JDBC driver version, and connection pooling settings | Shows everything you need to know about how Waveset talks to the repository. |
| Performance | Contains sub-beans for DataExporter, DataQueue, FormConverter, ObjectChangeNotification, Reconcile, Rule, TaskInstanceCache, ViewMaster, and WorkItemCache execution. | Identifies GUI performance problems and provides visibility into how these subsystems perform. You can enable these beans to capture data that can quickly isolate why a particular page loads slowly on a production server. |

| MBean | Description | Use |
|-----------------|---|--|
| Scheduler | Internal processing counters for the Scheduler | Provides detailed visibility into what the Scheduler is doing at any given point in time. Making use of this bean requires an in-depth understanding of how the Scheduler works. |
| Server | Current status of this Waveset server | Shows server status |
| <resource name> | Operational statistics for each resource and connector used by Waveset. | Shows the count and maximum, minimum, and average time for each resource operation. Gives visibility into the performance of the resource, the Waveset interface, and the Waveset activity on each resource. |

Note – Every application server also provides a number of useful MBeans. These MBeans typically supply information about the application server's processing elements such as HTTP processing, DataSource operations, classloading, and JSP compilation. If you have never browsed the MBeans provided by your application server, consider doing so.

Editing Default Server Settings

The Default Server Settings feature lets you set the default settings for all Waveset servers. The servers inherit these settings unless you select differently in the individual server settings pages.

▼ To Edit the Default Server Settings

- 1 In the Administrator interface, click Configure → Servers.**

The Configure Servers page opens.

- 2 Click Edit Default Server Settings.**

The Edit Default Server Settings page opens.

The Edit Default Server Settings page displays the same options as the individual server settings pages. For help, refer to the documentation for the individual server settings pages.

Changes you make to each default server setting is propagated to the corresponding individual server setting, unless you have deselected the Use default option for that setting.

Click Save to save changes to the server settings.

Working with Firewalls, Load Balancers, or Proxy Servers

This chapter describes how Waveset uses Uniform Resource Locators (URLs) and how to configure Waveset to obtain accurate URL data when firewalls, load balancers, or proxy servers are in place.

Servlet APIs

The Web-based Waveset user interface is highly dependent on Uniform Resource Locators (URLs) to specify the location of pages to be retrieved by the Web client.

Waveset depends on the Servlet APIs provided by an application server (such as Glassfish, Apache Tomcat, IBM WebSphere, or BEA WebLogic) to determine the fully qualified URL in the current HTTP request so that a valid URL can be placed in the generated HTML and HTTP response.

Some configurations prevent the application server from determining the URL the Web client uses for an HTTP request. Examples include:

- A port-forwarding or Network Address Translation (NAT) firewall placed between the Web client and Web server, or between the Web server and application server
- A proxy server (such as Tivoli Policy Director WebSEAL) placed between the Web client and Web server, or between the Web server and application server

For instances in which the Servlet APIs do not provide accurate URL data from an HTTP request, the correct data can be configured in the `Waveset.properties` file (located in your Waveset installation `config` directory).

The following attributes control Waveset's Web-based documentation root and whether Waveset uses the HTML BASE HREF tag.

- `ui.web.useBaseHref` (Default value: `true`). Set this attribute to one of the following values:
 - `true`. Waveset uses the HTML BASE HREF tag to indicate the root of all relative URL paths.
 - `false`. All URLs placed into HTML contain fully qualified paths; including scheme, host, and port.
- `ui.web.baseHrefURL`. Set this attribute to a non-empty value to define the BASE HREF used in generated HTML, which overrides the value that is calculated using servlet APIs.

Overriding this calculated value can be useful when those APIs do not return the whole truth, which occurs when:

- The application server is behind a firewall using port forwarding or NAT
- The connector between the application server and Web server does not provide accurate information
- The application server is front-ended by a proxy server

Configuring Logs to Work with Load Balancers

You can configure Waveset to automatically log the client IP address contained in the `x-Forwarded-For` HTTP request header, which is the standard header for identifying the originating IP address of a client connecting to a web server through a load balancer or HTTP proxy. If necessary, you can also configure Waveset to use a custom HTTP header.

To force Waveset to log IP addresses contained in a custom HTTP request header, use the following steps:

1. Open `Waveset.properties` in a text editor.
2. Search for `client.headerIPVariable=` and uncomment the line.
3. Type the name of the HTTP request header that you want Waveset to use instead.
Alternatively, you can disable this feature by setting the property equal to zero.
4. Save `Waveset.properties`.
5. Restart Waveset.

Note – You can apply this configuration to both audit logs and system logs.

If you do not want Waveset to automatically log the client IP address, you can disable this feature by uncommenting the `client.headerIPVariable` in the `Waveset.properties` file.

Working with Logs

This chapter describes tasks that you might have to perform when working with Audit logs or System logs.

These topics include:

- [“Preventing Audit Log Tampering” on page 29](#)
- [“Removing Records from the System Log” on page 31](#)

Preventing Audit Log Tampering

You can configure Waveset to prevent the following forms of audit log tampering:

- Adding or inserting audit log records
- Modifying existing audit logs records
- Deleting audit log records or the entire audit log
- Truncating audit logs

All Waveset audit log records have unique, per-server sequence numbers and encrypted hash of records and sequence numbers.

When you create a Tamper Detection Report, it scans the audit logs per server for:

- Gaps in the sequence number (indicating a deleted record)
- Hash mismatches (indicating a modified record)
- Duplicate sequence numbers (indicating a copied record)
- Last sequence number that is less than expected (indicating a truncated log)

▼ To Configure Tamper-Resistant Logging

- 1 Create a tampering report by selecting **Reports > New > Audit Log Tampering Report**.
- 2 When the **Define a Tampering Report** page displays, as shown in [Figure 3–1](#), enter a title for the report and then **Save** it.

FIGURE 3–1 Configuring an Audit Log Tampering Report

You can also specify the following optional parameters:

- **Report Summary.** Enter a descriptive summary of the report.
 - **Starting sequence for server** '`<server_name>`'. Enter the starting sequence number for the server.
 - This option enables you to delete old log entries without having them flagged as tampering and limits the report's scope for performance reasons.
 - **Email Report.** Enable to email report results to a specified email address.
 - When you select this option, the page refreshes and prompts you for email addresses. However, keep in mind that email is not safe for text content-sensitive information (such as account IDs or account history) may be exposed.
 - **Override default PDF options.** Select to override the default PDF options for this report.
 - **Organizations.** Select organizations that should have access to this report.
- 3 Next, select **Configure** → **Audit** to open the **Audit Configuration** page, as shown in [Figure 3–2](#).

FIGURE 3-2 Tamper-Resistant Audit Logging Configuration

Audit Configuration

Click a box next to an audit group name to record successful and failed events in that group. Click **All Successes** or **All Failures** to store successful or failed events for all groups. To edit which events are enabled by a group, click the group name. To use custom publishers, check the **Use Custom Publishers** option and use the drop-down list to configure new audit publishers.

Enable auditing

All Successes All Failures

| Audit Group Name | Success | Failure |
|---------------------------------|-------------------------------------|-------------------------------------|
| Account Management | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Logins/Logoffs | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Password Management | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Resource Management | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Role Management | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Security Management | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Task Management | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Changes Outside Identity System | <input type="checkbox"/> | <input type="checkbox"/> |
| Configuration Management | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Service Provider Edition | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Compliance Management | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |

Use custom publisher

Save Cancel

- 4 Select Use Custom Publisher, and then click on the Repository publisher link.
- 5 Select Enable tamper-resistant audit logs, and then click OK.
- 6 Click Save to save the settings.

You can turn this option off again, but unsigned entries will be flagged as such in the Audit Log Tampering Report, and you must reconfigure the report to ignore these entries.

Removing Records from the System Log

The system log captures errors generated by Waveset. Periodically, the system log should be truncated to keep it from growing too large. Use the System Log Maintenance Task to remove old records from the system log.

▼ **To Schedule a Task to Remove Old Records From the System Log**

- 1 In the Administrator interface, click Server Tasks → Manage Schedule.**
- 2 In the Tasks Available for Scheduling section, click the System Log Maintenance Task.**
The Create New System Log Maintenance Task Task Schedule page opens.
- 3 Complete the form and click Save.**

Performance Tuning

You can significantly improve the performance of Oracle Waveset (Waveset) software across nearly all activities with the proper tuning. In addition to changing settings within the software, you can make performance improvements by tuning the application server, the Java Virtual Machine (JVM machine), hardware, operating system, and network topology.

Additionally, you can use several tools to diagnose and monitor performance. Several of these tools exist within Waveset, such as trace and method timers. You can also use other Oracle and third-party tools to debug performance issues with Waveset.

This chapter describes tools, methodologies, and references you can use to improve performance and to debug performance-related issues.

Note – The tuning process spans many entities and is specific to your deployment environment. This chapter describes different tuning methods for optimizing Waveset performance, but these methods are *only intended as guidelines*. You might have to adapt these methods for your deployment environment.

This chapter covers the following topics:

- “Before You Begin Tuning” on page 34
- “Tuning Your Deployment Environment” on page 37
- “Tuning Waveset Performance” on page 54
- “Debugging Performance Issues” on page 83

Before You Begin Tuning

Review all of the information in this section before you start tuning Waveset.

Important Notes

Note – The tuning methods described in this chapter are only provided as *guidelines*. You might have to modify some of these tunings for your deployment. In addition, Be sure to validate tunings *before* applying changes in a production environment.

Before you can tune Waveset, you must:

- Be familiar with tuning application servers
- Be familiar with Java 5.0 (required for Oracle Waveset 8.1.1)
- Understand performance limitations within your deployment environment
- Be able to identify areas needing performance improvements
- Understand the checklists provided in this chapter

Related Documentation and Web Sites

In addition to the information provided in this chapter, consult the documents and web sites listed in this section for information related to tuning Waveset.

Recommended Reading

See the following documents for information related to performance tuning.

TABLE 4-1 Related Documentation

| Document Title | Description |
|--|---|
| <i>Java Tuning White Paper</i> | Contains information, techniques, and pointers related to Java performance tuning. |
| <i>Oracle MetaLink: Note:114671.1: Gathering Statistics for the Cost Based Optimizer</i> | Explains how to use system statistics and Oracle's Cost-Based Optimizer (CBO). Note: This document is available to Oracle Metalink subscribers. Registration is required. |
| <i>Solaris Dynamic Tracing Guide</i> | Explains how to use DTrace to observe, debug, and tune your system's behavior. |

TABLE 4-1 Related Documentation (Continued)

| Document Title | Description |
|--|---|
| <i>Sun Java System Application Server Performance Tuning Guide</i> | Describes how to obtain optimal performance from your Sun Java System Application Server. Download the necessary version of this book from the Oracle documentation web site . (http://docs.sun.com/app/docs) |
| <i>Tuning Garbage Collection with the 5.0 Java Virtual Machine</i> | Describes how to tune your garbage collection application by using JVM. |
| <i>Turbo-charging Java HotSpot Virtual Machine, v1.4.x to Improve the Performance and Scalability of Application Servers</i> | Explains how to download and use the <code>PrintGCStats</code> script and how to collect statistics to derive optimal JVM tunings. |
| <i>Using JConsole to Monitor Applications</i> | Describes how to use JConsole to monitor applications that run on the Java platform. |

Useful Web Sites

The following table describes some web sites that you might find useful when trying to tune Waveset performance.

TABLE 4-2 Useful Web Sites

| Web Site URL | Description |
|---|--|
| http://sunsolve.sun.com | Oracle web site containing diagnostic tools, forums, features and articles, security information, and patch contents. Note: The information on this site is divided into three areas: <ul style="list-style-type: none"> ■ Internal. Available only to Oracle employees ■ Contract. Available only to customers with contract access ■ Public. Available to everyone |
| http://forums.sun.com | Oracle Developer Network (SDN) web site where you can browse forums and post questions. |
| http://jrat.sourceforge.net/ | JRat web site that describes how to use the Java Runtime Analysis Toolkit, an open source performance profiler for the Java platform. |

TABLE 4-2 Useful Web Sites (Continued)

| Web Site URL | Description |
|--|---|
| https://metalink.oracle.com/ | Oracle's internal forum site that contains information about tuning Oracle databases. Note: You must be an Oracle Metalink subscriber to access the information provided on this site. |
| http://performance.netbeans.org/howto/jvmswitches/index.html | NetBeans web site containing information about tuning JVM switches for performance. |
| https://sharespace.sun.com/gm/folder-1.11.60181? | Waveset link on Oracle's Share Space. Note: This space is only available to Oracle employees and Oracle partners. In addition, these individuals must have a Share Space ID to access the information on this site. |
| https://sharespace.sun.com/gm/document-1.26.2296 | Waveset FAQ on Oracle's Share Space. Note: This space is only available to Oracle employees and Oracle partners. In addition, these individuals must have a Share Space ID to access the information in this FAQ. |
| http://www.slamd.com/ | SLAMD Distributed Load Generation Engine web site. |
| http://hub.opensolaris.org/bin/view/Community+Group+dtrace/WebHome | OpenSolaris Community: DTrace web page. |
| http://www.solarisinternals.com/ http://docs.sun.com/app/docs/prod/solaris.10 | Web sites containing information related to tuning the Solaris OS. |

Tuning Roadmap

How well your Waveset solution performs can depend on the following deployment-specific settings:

- Resource configuration
- How many resources are connecting
- What type of resources are connecting
- How attributes are mapped on the resources
- Exact resource version
- Network topology
- Number (and distribution) of domain controllers
- Number of installed Waveset Gateways
- Number of concurrent settings

- Number of concurrent processes (running workflows)
- Number of concurrent users
- Number of concurrent Waveset administrators
- Total number of users under management

When you are trying to debug performance problems, start by analyzing and describing the problem. Ask yourself the following questions:

- Where do you see the performance issue? In reconciliation, workflows, custom workflows, GUI page loading, provisioning, Access Review?
- Are you CPU bound, memory bound, resource bound, or network bound?
- Have you examined your configuration (hardware, network, parameters, and so forth) for problems?
- Have you recently changed anything in your deployment environment?
- Have you tried profiling troublesome resources natively to see if the problem is on the resource side and not with Waveset?
- What size are the views?
- Are you provisioning to several resources?
- Are resources on slow networks connecting to Waveset?
- Are you running additional applications on the server with Waveset?
- Do your organizations have a Rule-Driven Members rule?
- Have you run a series of thread dumps to see if there is a consistent pattern?

Note – Looking at just a single thread dump can be misleading.

- Have you recently turned on tracing?
- Have you checked your JVM garbage collection?
- Have you added organizations that are adding load to memory management?

Tuning Your Deployment Environment

This section provides information about tuning your deployment environment, including:

- [“Tuning Your Java EE Environment” on page 38](#)
- [“Tuning Your Application Server” on page 39](#)
- [“Tuning Your Repository Database” on page 41](#)

Tuning Your Java EE Environment

This section describes some tuning suggestions you can use to optimize your Java Platform, Enterprise Edition (Java EE platform) environment.

These tuning suggestions were derived from a series of experiments in which a considerable increase in throughputs was observed for the use cases tested. The increases were attributed to JVM sizing and to switches that affected garbage collector behavior.

Note – For more information about tuning Java, JConsole, or JVM, visit the web sites noted in [Table 4–1](#) and [Table 4–2](#).

The following sections provide information about tuning Java and the JVM in your Java EE environment.

Tuning Java

For information, best practices, and examples related to Java performance tuning, see the [Java Tuning White Paper](#) at:

<http://java.sun.com/performance/reference/whitepapers/tuning.html>

Tuning the JVM

The following tuning scripts were used to derive the tuning suggestions noted in this section. These scripts were added to the `domain.xml` file (located in the domain configuration directory, which is typically `domain-dir/config`) on a Sun Java System Application Server.

- `PrintGCStats` – A data mining shell script that collects data from `verbose:gc` logs and displays information such as garbage collection pause times, parameter calculations, and timeline analyses over the application's runtime by sampling the data at user-specified intervals.

Note – For more information about how to use this script and garbage collection statistics to derive optimal JVM tunings, see the following web site:

<http://java.sun.com/developer/technicalArticles/Programming/turbo/#PrintGCStats>

- `PrintGCDetails` – A shell script that can provide more verbose garbage collection statistics.
- `PrintGCTimeStamps` – A shell script that adds time-stamp information to the garbage collection statistics collected by using the `PrintGCDetails` script.

To help ensure the best JVM performance, verify the following:

- Be sure that you are using the required Java version noted in the “Supported Software and Environments” section of the *Oracle Waveset 8.1.1 Release Notes* to ensure you are using the most current features, bug fixes, and performance enhancements.
- Be sure that you are using a newer version of garbage collection.

Frequently, customers do not remove the older, default garbage collection scheme when installing an application server. Running Waveset with an older garbage collector creates many objects, which forces the JVM to constantly collect garbage.

- If you deployed Waveset on Sun Java System Application Server, you can increase throughput by adding garbage collection elements to the deployed Waveset instance `server.xml` file.

If you expect a peak load of more than 300 users, try modifying the following settings to increase performance:

- For HTTP listeners configured for the deployed Waveset instance, edit the listener definition element in the `server.xml` file and set the number of acceptor threads to the *Number of active CPUs on the host* divided by the *Number of active HTTP listeners on the host*.

For example:

```
<http-listener id="http-listener-1" \address="0.0.0.0" port="80"
\acceptor threads="Calculated Acceptor Threads" ...>
```

- Because the static content of most Waveset deployments is not projected to change frequently, you can edit the File Cache settings (on the File Cache Configuration page) for static content. Specify a high number (such as the number of seconds in 24 hours) for the maximum age of content within the file cache before the content is reloaded.

To access the File Cache Configuration page, click the File Caching tab on the web-based Administrative Console for the HTTP server node. (See the latest *Sun Java System Web Server Administrator's Guide* for detailed instructions.)

Note – Sun Java System Application Server exposes tunables that affect the size of various thread pools and connection queues that are maintained by the HTTP container.

By default, most of these tunables are set for a concurrent user load of 300 users or less.

Tuning Your Application Server

The following guidelines are provided to help you tune your application server:

- “Tuning a Sun Java System Application Server” on page 40
- “Tuning a WebSphere Application Server” on page 41

Note – Other than heap size, you can use the default parameter settings for most application servers. You might want to modify the server’s heap size, depending on the release being used.

Tuning a Sun Java System Application Server

The “Tuning the Application Server” chapter, in the latest *Sun Java System Application Server Performance Tuning Guide*, contains information about tuning a Sun Java System Application Server. This document is available from the following URL at <http://docs.sun.com/app/docs>.

In addition, if you are using Sun Java System Application Server 8.2 Enterprise Edition, the following changes solve “concurrent mode failures,” and should give you better and more predictable performance:

- If you are constantly running old generation collections, review your application’s heap footprint and consider increasing the size. For example:
 - 500 Mbytes is considered a modest size, so increasing this value to 3 Gbytes might improve performance.
 - With a 2 Gbyte young generation collection, each scavenge promotes about 70 Mbytes. Consider giving this 70 Mbytes at least one more scavenge before promoting.

For example, you might need the following SurvivorRatio:

$$2 \text{ GB} / 70 \text{ M} \times 2 = 4096 / 70 = 55$$

Where:

`-XX:SurvivorRatio=32 -XX:MaxTenuringThreshold=1`

This ratio prevents premature promotion and the added problem of “nepotism,” which can degrade scavenge performance.

- If you specified `-XX:CMSInitiatingOccupancyFraction=60`, and the CMS collections are still starting before they reach that threshold. For example:

```
56402.647: [GC [1 CMS-initial-mark: 265707K(1048576K)] 1729560K(3129600K),
3.4141523 secs]
```

Try removing `-XX:CMSInitiatingOccupancy=60` (using the default value of 69 percent), and add the following line:

`-XX:UseCMSInitiatingOccupancyOnly`

- If your young generation collection is 2 Gbytes and the old generation collection is 1 Gbyte, this situation might also be causing premature CMS collections. Consider reversing this ratio. Use a 1 Gbyte young generation collection and a 2 Gbyte old generation collection, as follows:

`-Xms3G -Xmx3G -Xmn1G`

Also, remove `-XX:NewRatio`. This ratio is redundant when you have explicitly specified young generation and overall heap sizes.

- If you are using a 5uXX version of the Java Development Kit (JDK software), and have excessively long “abortable preclean” cycles, you can use `-XX:CMSMaxAbortablePrecleanLoops=5` as a temporary workaround.

You might have to adjust this value further.

- Add the following line to view more information about garbage collector performance:
`-XX:+PrintHeapAtGC`

Note – Use this command with caution because it increases how much verbose garbage collection data is produced. Be sure that you have enough disk space on which to save the garbage collector output.

- If you are using the Sun Fire T2000 server, large-heap data Translation Look-aside Buffers (DTLBs) can become a scarce resource. Using large pages for the Java heap often helps performance. For example:
`-XX:+UseLargePages`

Tuning a WebSphere Application Server

If you are tuning Waveset on an IBM WebSphere application server, consider limiting how much memory is allocated for the heap because heap memory can affect the memory used by threads.

If many threads are created simultaneously and the heap size increases, the application’s space limit can be quickly impacted and the following error results:

```
JVMCI015:OutOfMemoryError
```

Tuning Your Repository Database

Waveset relies on a repository database to store and manage its identity and configuration data. For this reason, database performance can greatly influence Waveset’s performance.

Note – Detailed information about performance tuning and managing databases is beyond the scope of this document because this information is dataset-specific and vendor-specific. In addition, customer database administrators (DBAs) should already be experts on their own databases.

This section characterizes the Waveset application and provides general information about the nature of Waveset data and its typical usage patterns to help you plan, tune, and manage your databases.

This information is organized into the following sections:

- “Understanding the Waveset Repository” on page 42
- “Configuring the Waveset Repository” on page 47
- “General Guidelines for Tuning a Repository Database” on page 51
- “Vendor-Specific for Tuning a Repository Database” on page 52

Understanding the Waveset Repository

You must understand the Waveset repository database scripts and how Waveset uses that content before you can effectively tune the Waveset repository.

This section provides an overview of the database and the information is organized into the following topics:

- “Repository Table Types” on page 42
- “XML Columns” on page 43
- “Data Classes” on page 44
- “Object IDs” on page 46
- “Prepared Statements” on page 46
- “Character Sets and Encodings” on page 47

Repository Table Types

The Waveset repository contains three types of tables, and each table has slightly different usage characteristics.

- **Attribute Tables.** These tables enable you to query for predefined single-valued or multi-valued object attributes.

For most object types, stored attributes are hard-coded.

Note – The `User` and `Role` object types are exceptions to this rule. The inline attributes that are stored in the object table for `User` and `Role` are configurable, so you can configure additional custom attributes as queryable.

When you search for objects based on attribute conditions, Waveset accesses attribute tables in joins with the corresponding object tables. Some form of join (such as a `JOIN`, an `EXISTS` predicate, or a `SUB-SELECT`) occurs for each attribute condition.

The number of rows in the attribute table are proportional to the number of rows in the corresponding object table. The values distribution might exhibit *skew*, where multi-valued attributes have a row per value and some objects might have more attributes or more attribute values than others. Typically, there is a many-to-one relation between rows in the attribute table and rows in the object table.

Attribute tables have `ATTR` in the table name.

- **Change Tables.** Waveset uses a change table to track changes made to a corresponding object table. These table sizes are proportional to the rate of object change, but the tables are not expected to grow without bound. Waveset automatically truncates change tables.

Change tables can be highly volatile because the lifetime of a row is relatively short and new rows can be created frequently.

Access to a change table is typically performed by a range scan on the time-stamp field.

Change tables have CHANGE in the table name.

- **Object Tables.** The Waveset repository uses object tables to hold serialized data objects, such as Large Objects (LOBs). Object tables can also hold commonly queried, single-valued object attributes.

For most object types, stored attributes are hard-coded.

Note – The User and Role object types are exceptions to this rule. The inline attributes that are stored in the object table are configurable, and you can configure additional custom attributes as queryable for User and Role.

The number of rows in an object table equals the number of objects being stored. The number of objects stored in each object table depends on which object types are being stored in the table. Some object types are numerous, while other types are few.

Generally, Waveset accesses an object table by object ID or name, though Waveset can also access the table by using one of the attributes stored in the table. Object IDs and names are unique across a single object type, but attribute values are not unique or evenly distributed. Some attributes have many values, while other attributes have relatively few values. In addition, several object types can expose the same attribute. An attribute may have many values for one object type and few values for another object type. The uneven distribution of values might cause an uneven distribution of index pages, which is a condition known as *skew*.

Object tables are tables that *do not* have ATTR or CHANGE suffixes in the table name.

XML Columns

Every object table contains an XML column, which is used to store each serialized object except the LOG table-set. Certain LOG table-set optional attributes are stored in the XML column if these attributes are present. For example, if digital signing is enabled.

Data Classes

You can roughly divide Waveset data into a number of classes that exhibit similar properties with respect to access patterns, cardinality, lifetime, volatility, and so forth. Each of the following classes corresponds to a set of tables in the repository:

- **User Data.** User data consists of user objects.

You can expect this data to grow quite large because there is an object for each managed identity. After an initial population phase, you can expect a proportionally small number of creates because the majority of operations will be updates to existing objects.

User objects are generally long-lived and they are removed at a relatively low rate.

User data is stored in USEROBJ, USERATTR, and USERCHANGE tables.

- **Role Data.** Role data consists of Role objects, including Roles subtypes such as Business Roles, IT Roles, Applications, and Assets.

Role data is similar to organization data, and these objects are relatively static after a customer deploys Waveset.

Note – An exception to the preceding statement is a deployment that is integrated with an external source containing an authoritative set of roles. One integration style might be to feed role changes into Waveset, which causes Waveset Role data to be more volatile.

Generally, the number of role objects is small when compared to the number of identity objects such as users (assuming that multiple users share each role), but this depends on how each enterprise defines its roles.

Role data is stored in ROLEOBJ, ROLEATTR, and ROLECHANGE tables.

- **Account Data.** Account data solely consists of account objects in the Account Index.

As with user data, account data can become rather large, with an object for each known resource account. Account objects are generally long-lived, removed at a relatively low rate, and after initial population, are created infrequently. Unless you frequently add or remove native accounts, or enable native change detection, account object modifications occur infrequently.

Waveset stores account data in ACCOUNT, ACCTATTR, and ACCTCHANGE tables.

- **Compliance Violation Data.** Compliance Violation data contains violation records that indicate when the evaluation of an Audit Policy failed. These violation records exist until the same Audit Policy is evaluated against the same User and the policy passes. Violation records are created, modified, or deleted as part of an Audit Policy Scan or as part of an Access Review.

The number of violation records is proportional to the number of Audit Policies that are used in scans and the number of Users. An installation with 5000 users and 10 Audit Policies might have 500 violation records ($5000 \times 10 \times 0.01$), where the 0.01 multiplier depends on how strict the policies are and how user accounts are changed.

Waveset stores Compliance Violation records in OBJECT, ATTRIBUTE, and OBJCHANGE tables.

- **Entitlement Data.** Entitlement data predominately consists of user entitlement objects, which are only created if you are doing compliance access reviews.

Entitlement records are created in large batches, modified slowly (days) after initial creation, and are then untouched. These records are deleted after an Access Review is deleted.

Waveset stores entitlement data in ENTITLE, ENTATTR, and ENTCHANGE tables.

- **Organization Data.** This data consists of object group or organization objects.

Object group data is similar to configuration data, and this data is relatively static after being deployed. Generally, the number of objects is small (one for each defined organization) when compared to task objects or to identity objects such as users or accounts, however, the number can become large compared to other configuration objects.

Organization data is stored in ORG, ORGATTR, and ORGCHANGE tables.

- **Task Data.** Task data consists of objects that are related to tasks and workflows, including state and result data.

The data contained in these tables is short-lived compared to other classes because objects are created, modified, and deleted at a high rate. The volume of data in this table is proportional to the amount of activity on the system.

Task data is stored in TASK, TASKATTR, and TASKCHANGE tables.

- **Configuration Data.** Configuration data consists of objects related to Waveset system configuration, such as forms, roles, and rules.

Generally, configuration data is:

- Relatively small compared to other classes
- Only expected to change during deployment and upgrade, and changes occur in large batches
- Not expected to change much after being deployed

Waveset stores configuration data in ATTRIBUTE, OBJCHANGE, and OBJECT tables.

- **Export Queue Data.** If you enable Data Exporting, some records are queued inside Waveset until the export task writes those records to the Data Warehouse. The number of records that are queued is a function of Data Exporting configuration and the export interval for all queued types.

The following data types are queued by default, and all other data types are not:

- ResourceAccount
- WorkflowActivity
- TaskInstace
- WorkItem

The number of records in these tables grows until the export task drains the queue. The current table size is visible through a JMX Bean.

Records added to this table are never modified. These records are written during other Waveset activities, such as reconciliation, provisioning, and workflow execution. When the Data Exporter export task runs, the task drains the table.

Waveset stores Export Queue data records in QUEUE, QATTR, and QCHANGE tables.

- **Log Data.** Log data consists of audit and error log objects. Log data is write-once only, so you can create new audit and error log objects, but you cannot modify these objects.

Log data is long-lived and can potentially become very large because you can only purge log data by explicit request. Access to log data frequently relies on attributes that are stored in the object table instead of in the attribute table. Both the distribution of attribute values and queries against the log specifically depend on how you are using Waveset.

For example, the distribution of attribute values in the log tables depends on the following:

- What kind of changes are made
- Which Waveset interface was used to make the changes
- Which types of objects were changed

The pattern of queries against the log table also depends on which Waveset reports, which custom reports, or which external data mining queries a customer runs against the log table.

Waveset stores audit log records in LOG and LOGATTR tables, and error log records in SYSLOG and SLOGATTR tables. This data does not have corresponding change tables.

Object IDs

Waveset generates globally unique identifiers (GUIDs) for objects by using the VMID class provided in the JDK software.

These GUID values exhibit a property that gets sorted by its string representations, based on the order in which the objects are created. For example, when you create new objects with Waveset, the newer objects have object IDs that are greater than the older objects. Consequently, when Waveset inserts new objects into the database, the index based on object IDs can encounter contention for the same block or blocks.

Prepared Statements

Generally, Waveset uses prepared statements for activities such as inserting and updating database rows, but does not use prepared statements for queries.

If you are using Oracle, this behavior can create issues with the library cache. In particular, the large number of statements versions can cause contention on the library cache latch.

To address this contention, change the Oracle `CURSOR_SHARING` parameter value from `EXACT` to `SIMILAR`. Changing this value causes Oracle to replace literals in SQL statements with bind variables, thereby reducing the number of versions.

Character Sets and Encodings

Because Waveset is a Java application that generally reads and writes character data rather than bytes, it does not restrict which encoding the database uses.

Waveset only requires that the data is sent and returned correctly. For example, the data does not become corrupted when written or reread. Use an encoding that supports multi-byte characters and is appropriate for the customer's data. Generally, UTF-8 encoding is sufficient, but enterprises with a large number of true multi-byte characters, such as Asian or Arabic, might prefer UTF-16.

Most database administrators prefer to use an encoding that supports multi-byte characters because of the following:

- Their deployments often grow to support international characters.
- Their end users cut-and-paste from a Microsoft application's text containing characters that look like ASCII but are actually multi-byte, such as em dashes (–).

Configuring the Waveset Repository

This section describes how to configure some commonly configured properties in the Waveset repository.



Caution – Do not modify properties in the Waveset repository unless you are very familiar with repository databases and understand the consequences of making these changes.

Disabling Automatic Connection Pooling

If you are using a `DataSource`, set the `connectionPoolDisable` attribute to `true` in the `RepositoryConfiguration` object to disable automatic internal connection pooling in the Waveset repository.

For example, setting `<RepositoryConfiguration connectionPoolDisable='true'>` allows you to avoid having two connection pools (one for Waveset and one for your application server).

Note – You can see the current `connectionPoolDisable` setting in the `ObjectRepository JMX MBean`.

Using Inline Attributes

You can edit the `RepositoryConfiguration` object to enhance the performance of searches against specific, single-valued attributes. For example, you might edit this object to add an extended attribute, such as `employeeID`, that is used to search for `Users` or as a correlation key.

The default `RepositoryConfiguration` object looks like the following example:

```
<RepositoryConfiguration ... >
  <TypeDataStore Type="User" ... attr1="MemberObjectGroups",
attr2="lastname" attr3="firstname" attr4="" attr5="">
  </TypeDataStore>
</RepositoryConfiguration>
```

Note – The ellipses represent XML attributes that are not relevant here.

Each of the `attr1`, `attr2`, `attr3`, `attr4`, and `attr5` XML attributes specifies a single-valued attribute to be copied into the `waveset.userobj` table. The `waveset.userobj` table can contain up to five inline attributes. The attribute value named by `attr1` in `RepositoryConfiguration` will be copied into the “`attr1`” database column in this table.

Inline attributes are stored in the base object table for a `Type` (rather than as rows in the child attribute table).

Using inline attributes improves the performance of repository queries against those attributes. (Because inline attributes reside in the main “object” table, queries against inline attributes are faster than those against non-inline attributes, which are stored in the child “attribute” table. A query condition against a non-inline attribute requires a “join” to the attribute table.)

By default, `Waveset` uses the `MemberObjectGroups`, `lastname`, and `firstname` inline attributes.

- You can add two more attributes to enable faster searching, as long as those attributes are queryable.
For example, if your deployment contains an `employeeID` extended attribute, adding that attribute inline will improve the performance of repository searches against that attribute.
- If you do not need `lastname` or `firstname`, you can remove them or replace them with other attributes.
- *Do not* remove `MemberObjectGroups`. `Waveset` uses this attribute internally to speed up authorization checks.

Limiting Concurrent Connections

Every public repository method acquires a connection when the method begins and then releases that connection when the method exits. This connection is almost always a JDBC connection because the repository almost always accesses a DBMS. The method releases the connection whether it completes successfully or unsuccessfully.

You can configure the number of concurrent connections by modifying the `maxConcurrentConnections` value in the `RepositoryConfiguration` object.

- The number of concurrent JDBC connections is unlimited by default (`maxConcurrentConnections=0`). The Waveset repository can request as many connections as necessary to serve each concurrent repository request.
- You specify a maximum number of concurrent connections that the repository can use at one time by changing the `maxConcurrentConnections` value to a positive integer. In effect, this setting limits the number of calls to the repository that can be actively executing at any one time.

If the repository reaches the configured limit, calls to the repository will wait until the number of connections being used by the repository falls back below the limit. The repository might appear to slow down, or even stop, if this is necessary to stay within the specified number of connections.

Note – Any limit on the number of concurrent connections applies whether the repository pools connections or not. In other words, `maxConcurrentConnections` is unaffected by `connectionPoolDisable`.

To configure the `maxConcurrentConnections` value, you must edit the `RepositoryConfiguration` object from the System Settings debug page (<http://host:port/idm/debug/session.jsp>),

1. Select Configuration from the menu located next to the List Objects button.
2. Click the List Objects button to view all of the Configuration-type objects.
3. Locate `RepositoryConfiguration` and click Edit.
4. When the Checkout Object: Configuration, `#ID#REPOSITORYCONFIGURATION` page displays (similar to the following example), locate `maxConcurrentConnections` and modify the existing value as needed.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Configuration PUBLIC 'waveset.dtd' 'waveset.dtd'>
<!-- MemberObjectGroups="#ID#Top" extensionClass="RepositoryConfiguration"
id="#ID#RepositoryConfiguration" name="RepositoryConfiguration"-->
<Configuration id='#ID#RepositoryConfiguration' name='RepositoryConfiguration'
lock='Configurator#1251229174812' creator='getRepositoryConfiguration(boolean)'
createDate='1249052436640' repoMod='1249048790203' type=
'RepositoryConfiguration'> <Extension>
```

```

    <RepositoryConfiguration instanceOf=
      'com.waveset.repository.RepositoryConfiguration'
      lockTimeoutMillis='300000' maxConcurrentConnections='0' blockRowsGet='1000'
      blockRowsList='10000' maxAttrValLength='255' maxLogAcctAttrChangesLength='4000'
      maxLogMessageLength='255' maxLogXmlLength='2147483647'
      maxLogAcctAttrValueLength='128' maxLogParmValueLength='128'
      maxXmlLength='2147483647' maxSummaryLength='2048' optimizeReplaceAttributes=
      'true' maxInList='1000' maxDelSet='1000' mcDBCcall='10' mcDeleteAttrVal='3'
      mcInsertAttrVal='15' mcUpdateAttrVal='20' connectionPoolDisable='false'
      changeScanInterval='5000' changePropagationDelay='3000' changeAgeOut='60000'>
      <RelocatedTypes>
        <TypeDataStore typeName='User' deleteDestroyInterval='1'
          attr1='MemberObjectGroups' attr2='lastname' attr3='firstname'>
        </TypeDataStore>
      </RelocatedTypes>
    </RepositoryConfiguration>
  </Extension>
<MemberObjectGroups>
  <ObjectRef type='ObjectGroup' id='#ID#Top' name='Top' />
</MemberObjectGroups>
</Configuration>L

```

5. Save your changes.

Note – For more information about which object types are stored in each set of tables, see [“Data Classes” on page 44](#).

Using preferPreparedStatement

JDBC supports execution of two types of SQL statements:

- Statement
- PreparedStatement

Generally, a PreparedStatement is more efficient for the DBMS because the DBMS server can cache and reuse a PreparedStatement for multiple executions, avoiding the cost of SQL parsing and optimizer selections. To get the benefit of this caching, Waveset associates a PreparedStatement with a JDBC connection. When Waveset is going to issue the same SQL statement multiple times using the same connection, it automatically uses a PreparedStatement.

However, most of the JDBC operations performed by Waveset do not share a connection between JDBC calls. Most JDBC operations are the result of the application server processing an HTTP request through a JSP, so that the processing thread acquires a JDBC connection when needed and releases that connection when it is done. Consequently, you must do some additional work to get the benefits of using PreparedStatements.

Many application servers provide DataSource implementations that provide both connection pooling and *implicit statement caching*. Implicit statement caching is the ability of the DataSource or connection pool to implicitly cache JDBC statements and associate them with the connection.

If you configure Waveset to use an application server DataSource, you must have both of these features to benefit from using Waveset's `preferPreparedStatement` repository option. If you are not using an application server DataSource, if the DataSource is not configured to use a connection pool, or if the connection pool does not support implicit statement caching, using `preferPreparedStatement` causes decreased performance because the DBMS cannot reuse the statement.

Waveset code is not structured to associate a statement with a connection, except in a few circumstances, so Waveset depends on the DataSource or connection pool to perform this caching. If you configure the Waveset repository to use a DataSource or connection pool that performs implicit statement caching, and you enable `preferPreparedStatement=true` in the `RepositoryConfiguration` configuration object, Waveset can use `PreparedStatement` for object lock/unlock operations and most get operations, which potentially results in performance improvements. The `WavesetObjectRepository` has a JMX MBean that indicates the setting of the `preferPreparedStatement` switch and the number of `Statement/PreparedStatement` requests made by the Waveset server.

Note – You can see the current `preferPreparedStatement` setting in the `ObjectRepository` JMX MBean.

General Guidelines for Tuning a Repository Database

This section describes some general guidelines for tuning a repository database:

- Update statistics frequently to monitor what is happening with the repository database.
- Defragment indexes and tables more frequently.
- Report top queries.
- Preallocate the user table to be sure it is large enough.
Estimate how many users you have and make the table big enough to accommodate the data. If you just allow the table to extend itself, loading users can become a very slow process. Periodically the table will stop to double its size, reformat itself, and push all the data back in.
- Similarly, preallocate the accounts table to be sure it is large enough.
Estimate how many accounts you have per user and make the table large enough to accommodate the data.

Note – Oracle's Professional Services group and Partners have estimator tools that you can use to estimate user table and accounts table sizes.

- Periodically run Waveset's built-in Audit Log Maintenance Task and System Log Maintenance Task to configure log record expirations. Log records can grow without bound, so use these tasks to prevent the repository database from running out of space. For information, see the *Oracle Waveset 8.1.1 Business Administrator's Guide*.
- Keep in mind that the tasks are volatile. DBAs should consider defragmenting the database more frequently, consider what type of storage to use, and leave plenty of free space in the database.

Vendor-Specific for Tuning a Repository Database

This section describes some vendor-specific guidelines for tuning Oracle and SQL Server repository databases.

Note – Currently, MySQL databases are only supported in development and for demonstrations.

Oracle Databases

This section describes guidelines for tuning Oracle repository databases:

- The Waveset application does not require Oracle database features or options.
- If you are using an Oracle repository database and Oracle Waveset Service Provider or Waveset, you might encounter problems with object table fragmentation because Waveset uses LONG, rather than LOB, data types by default. Using LONG data types can result in large amounts of “unallocated” extent space, which cannot be made into usable space.

To mitigate this problem, do the following:

- Take EXPORT dumps of the Object table and re-import them to free up unallocated extent space. After importing, you must stop and restart the database.
- Use LOB data types and DataDirect Technologies' Merant drivers, which provide a standard LOB implementation for Oracle.
- Use Locally Managed Tablespaces (LMTs), which offer automatic free space management. LMTs are available in Oracle 8.1.5.

Waveset does not require Oracle `init.ora` parameter settings for SGA sizing, buffer sizing, open cursors, processes, and so forth.

- While the Waveset repository is a general-purpose database, it is best described as an object database.

Of the Waveset tables, the TASK table-set comes closest to having transaction-processing characteristics. The LOG and SYSLOG table-sets are also exceptional because these tables do not store serialized objects.

See “[Repository Table Types](#)” on page 42 and “[Data Classes](#)” on page 44 for descriptions of the tables, the object types stored in each table, and the general access pattern for each table.

- If you have performance issues with the Oracle database, check for issues related to poor query plans being chosen for what Waveset expects to be relatively efficient queries. For example, Waveset is configured to perform a full table-scan when an index is available for use. These issues are often visible in Automated Workload Repository (AWR) reports provided in the SQL by the `buffer gets` table. You can also view issues in the Enterprise Manager tool.

Performance problems typically appear to be the result of bad or missing database table statistics. Addressing this problem improves performance for both the database and Waveset.

The following articles (available from Oracle) are a good source of information about the cost-based optimizer (CBO) in Oracle:

- [Oracle MetaLink: Note:114671.1: Gathering Statistics for the Cost Based Optimizer](#)
- [Cost Control: Inside the Oracle Optimizer](#)
- You might also investigate using SQL Profiles, which are another method for choosing the best query plans. You can use the SQL Advisor within Enterprise Manager to create these profiles when you identify poorly performing SQL.

If you detect unexpected growth in the Oracle redo log, you might have workflows that are caught in an infinite loop with a manual action. The loop causes constant updates to the repository, which in turn causes the size of each `TaskInstance` to grow substantially. The workflow errors are caused by improper handling of `WF_ACTION_TIMEOUT` and by users closing their browser in the middle of a workflow.

To prevent problematic workflows, preview each manual action before a production launch and verify the following:

- Have you set a timeout?
- Have you created appropriate transition logic to handle a timeout for the activity with the manual action?
- Is the manual action using the `exposed variables` tag when there is a large amount of data in the `TaskInstance`?

Frequently, you can significantly improve Waveset performance if you change the `CURSOR_SHARING` parameter value from `EXACT` to `SIMILAR`.

Waveset uses prepared statements for some activities (such as inserting and updating database rows), but does not use these statements for most queries.

When you use Oracle, this behavior can cause issues with the library cache. In particular, the large number of statement versions can create contention on the library cache latch. Changing `CURSOR_SHARING` to `SIMILAR` causes Oracle to replace literals in SQL statements with bind variables, which greatly reduces the number of versions.

See “[Prepared Statements](#)” on page 46 for more information.

SQL Server Databases

Some customers who used an SQL Server 2000 database as a repository reported that as concurrency increased, SQL Server 2000 reported *deadlocking* problems that were related to SQL Server's internal use of pessimistic locking (primarily lock escalation).

These deadlock errors display in the following format:

```
com.waveset.util.IOException:  
  ==> com.microsoft.sqlserver.jdbc.SQLServerException: Transaction (Process ID 51)  
was deadlocked on lock | communication buffer resources with another  
process and has been chosen as the deadlock victim. Rerun the transaction.
```

To prevent or address deadlocking problems, do the following:

1. Use the SQL Server 2005 database.
2. Configure the READ_COMMITTED_SNAPSHOT parameter by formatting the command as follows:

```
ALTER DATABASE waveset SET READ_COMMITTED_SNAPSHOT ON
```

Enabling the READ_COMMITTED_SNAPSHOT parameter does the following:

- Removes contention during the execution of SELECT statements that can cause blocks, which greatly reduces the potential for deadlocks internal to SQL Server.
- Prevents uncommitted data from being read and guarantees that SELECT statements receive a consistent view of committed data.

For more information about the READ_COMMITTED_SNAPSHOT parameter, see:<http://msdn.microsoft.com/en-us/library/ms188277.aspx>.

Tuning Waveset Performance

Suggestions for optimizing Waveset's performance are organized into the following areas:

- "General Performance Tunings" on page 55
- "Tuning Active Sync Adapter Performance" on page 56
- "Tuning Admin Cache Initialization" on page 56
- "Tuning Bulk Loading" on page 57
- "Tuning Configurable XML Objects" on page 58
- "Working with Waveset Wizards" on page 64
- "Updating Database Statistics" on page 66
- "Tuning Data Exporter" on page 67
- "Tuning the General XML" on page 67
- "Tuning Oracle Waveset Service Provider" on page 68
- "Tuning the Waveset Web Interface" on page 68
- "Tuning Initial Loads" on page 68

- “Tuning Memory Requirements” on page 69
- “Tuning Operating System Kernels” on page 69
- “Tuning the Waveset Profiler” on page 70
- “Tuning Provisioner” on page 70
- “Tuning Reconciliation” on page 72
- “Tuning Resource Queries” on page 75
- “Tuning the Scheduler” on page 76
- “Tuning Sessions” on page 80
- “Tuning the Oracle Waveset Gateway” on page 80
- “Tuning the Task Bar” on page 83

General Performance Tunings

In general, you can optimize Waveset performance if you do the following:

- Turn off tracing (such as Java class, userform, and workflow tracing). Tracing can add substantial overhead.
- Run the Waveset built-in Audit Log Maintenance Task and System Log Maintenance Task to configure log record expirations. Log records can grow without bound, so use these tasks to prevent the repository database from running out of space. For information, see the *Oracle Waveset 8.1.1 Business Administrator’s Guide*.
- Check the README file in Waveset updates (formerly called *service packs* or *installation packs*) to see if any performance improvements have been made to the product. If so, schedule an upgrade.
- Consider the performance impact when fetching data from one or more remote systems, including the Waveset repository.
- Increase the number of application server instances running Waveset, either on the same server or by adding servers, and use a load-balancing tool to distribute the requests between instances.
- Keep the size of files referenced in a binary attribute as small as possible. Loading extremely large graphics files, for example, can decrease Waveset performance.
- Write robust and readable XML code that minimizes duplication (for example, refactored), that uses memory efficiently, and that mitigates the impact to overall system performance.
- Configure Waveset system monitoring to track events in real time.

You can view these events in dashboard graphs to quickly assess system resources, spot abnormalities, understand historical performance trends (based on the time of day, the day of week, and so forth), and interactively isolate problems before looking at audit logs. Dashboards do not provide as much detail as audit logs, but can provide hints about where to look for problems in the logs.

For more information about dashboards, see [Chapter 8, “Reporting,” in *Oracle Waveset 8.1.1 Business Administrator’s Guide*](#).

Tuning Active Sync Adapter Performance

Because synchronization is a background task, how you configure an Active Sync adapter can affect server performance.

Use the Resources list to manage Active Sync adapters. Choose an Active Sync adapter and access start, stop, and status refresh control actions from the *Synchronization* section of the Resource Actions list.

To improve Active Sync adapter performance, do the following:

- Evaluate and adjust polling intervals based on the type of activity being performed.
The polling interval determines when the Active Sync adapter will start processing new information. For example, if the adapter reads in a large list of users from a database and updates these users in Waveset each time, you could run this process in the early morning every day. Some adapters have a quick search for new items to process and can be set to run every minute.
- Edit the synchronization file for the resource to specify the host where the adapters will run.
You can configure Active Sync adapters that require more memory and CPU cycles to run on dedicated servers to help load balance the systems.
- If you have the appropriate administrator capability, you can change Active Sync resources to disable, manually start, or automatically start Active Sync adapters.

When you set an adapter to automatic, the adapter restarts when the application server starts. When you start an adapter, it runs immediately and executes at the specified polling interval. When you stop an adapter, it stops the next time the adapter checks for the stop flag.

- Adjust the level of detail captured by the synchronization logs.
Synchronization logs capture information about the resource that is currently processing. Each resource has its own log file, path, and log level. The amount of detail captured by the adapter log depends on the specified logging level. You specify these values in the Logging section of the Synchronization Policy for the appropriate user type (Waveset or Service Provider).

Tuning Admin Cache Initialization

Waveset provides two different queries that you can use to determine which Waveset users have Administrative rights. One of these queries can be slow to execute when there are lots of users in the Oracle Waveset repository. Waveset executes this slow query during AdminCache initialization, which occurs during Waveset startup.

Waveset now uses the faster query by default; resulting in much faster server start-up times for installations with large numbers of users in the repository. However, under certain conditions

the faster query may not produce the correct results. If you upgraded your installation from version 5.0, some Administrative users may not have the attribute used by the new query correctly set.

- Refresh all Administrative users by importing the following XML, which sets the new query attribute properly:

```
<ImportCommand type='refreshType' targetType='User'>
<List>
  <AttributeCondition attrName='MemberAdminGroups' operator='isPresent'
    operand='' />
  <AttributeCondition attrName='ControlledObjectGroups' operator='isPresent'
    operand='' />
  <AttributeCondition attrName='AdminRoles' operator='notPresent' operand='' />
</List>
</ImportCommand>

<ImportCommand type='refreshType' targetType='User'>
<List>
  <AttributeCondition attrName='AdminRoles' operator='isPresent' operand='' />
</List>
</ImportCommand>
```

- Disable the new query, and continue to use the old query by adding the following line to `Waveset.properties` on each Waveset server: `admincache.fastinit=false`.

Tuning Bulk Loading

To improve performance during bulk load operations, do the following:

- Simplify default workflows to improve processing time (especially for bulk processing actions such as Active Sync, bulk actions, and reconciliation) by removing the callout to the Approval subprocess.
- Keep user forms that are assigned to administrators as simple as possible. For example:
 - When creating a form for data loading, remove any code that is designed to display data.
 - When using bulk add actions, be sure that your CSV file defines basic attributes such as `firstname` and `lastname`. You can then remove these attributes from the administrator user form.

Note – Do not modify the default forms provided with Waveset. Instead, make a copy of the form, give the copy a unique name, and modify the renamed copy. This approach prevents your customized forms from being overwritten during upgrades and product updates.

See [Chapter 2, “Waveset Forms,” in *Oracle Waveset 8.1.1 Deployment Reference*](#) for more information about creating and editing forms.

- Implement the following features in deployment environments where you have NIS (Network Information Service) implemented:

- Add an account attribute named `user_make_nis` to the schema map and use this attribute in your reconciliation or other bulk provisioning workflow. Specifying this attribute causes the system to bypass the step of connecting to the NIS database after each user update on the resource.
- To write the changes to the NIS database after provisioning has completed, create a ResourceAction named `NIS_password_make` in the workflow.

Tuning Configurable XML Objects

Configurable XML objects offer a broad spectrum of user interface specifications that enable you to define how data is presented to users for different tasks and to automate complex business processes. However, this same flexibility can affect efficiency, performance, and reliability.

This section describes some guidelines for tuning Waveset’s configurable XML objects, which consist of forms, rules, and workflows. The information is organized into the following sections:

- [“Tuning Forms” on page 58](#)
- [“Tuning Rules” on page 60](#)
- [“Tuning Workflows” on page 60](#)

Tuning Forms

You can use Waveset forms to define interfaces to interact with views or variable contexts in an executing task. Forms also provide an execution context for business and transformation logic on a set of data elements. Although you can create very powerful, dynamic forms that perform a variety of tasks, reducing the complexity of forms increases efficiency.

The following sections describe some methods for improving the performance of your customized forms:

- [“Optimizing New Forms” on page 58](#)
- [“Optimizing Administrator Forms” on page 59](#)
- [“Optimizing End-User Forms” on page 59](#)
- [“Optimizing Expressions in Form Fields” on page 60](#)

Optimizing New Forms

When designing new Waveset forms, system integrators can optimize a form’s performance by doing the following:

- Performing “expensive” queries only one time, wherever possible. To minimize these queries,
 - Use `<Field> <Default>` elements to execute and store query results.
 - Use field names to reference values in later fields.

- For custom tasks
 - ☐ Calculate the value in the task before a `ManualAction`, then store that value in a task variable.
 - ☐ Use `variables.tmpVar` to reference variables in the form.
 - ☐ Use `<setvar name='tempVar' />` to clear the variable after a `ManualAction`.
- Using `<defvar>` for calculations that are performed for the initial display and with each refresh.

Optimizing Administrator Forms

To improve the performance of administrator forms, do the following:

- Specify `TargetResources` that only fetch specific resources for editing. (See [“Tuning Workflows” on page 60](#) for more information.)
- Use `cacheList` and `cacheTimeout` caching parameters for objects that change infrequently if you are working with `FormUtil.getResourceObjects` or `FormUtil.listResourceObjects`.
- Store the results of time-consuming calculations and fetches in `<Field>` elements and evaluate in the `<Default>` expression to help ensure that an operation occurs only one time.
- Use `update.constraints` to limit which resources are fetched at runtime (see [“Dynamic Tabbed User Form” in *Oracle Waveset 8.1.1 Deployment Reference*](#)).
- Use background approval (`ManualAction` with different owners and one-second timeouts) for faster page submissions.
- Be aware that Waveset refreshes *all fields* defined on *all panels* of a Tab Panel Form when the page reloads, regardless of which panel is selected.

Optimizing End-User Forms

To improve the performance of end-user forms, do the following:

- Use `TargetResources` to limit view checkouts to just those resource accounts of interest, which reduces fetch time for view and the memory consumed by `TaskInstance` and `WorkItems`.
- Consider using `Session.getObject(Type, name)` to return a `WSUser` if just the view properties and attributes of the Waveset user object are of interest (useful for managing multiple deferred task triggers).
- Be aware that end-user tasks typically have more `WorkItems` than Provisioning tasks, so end user tasks are especially susceptible to `WorkItem` size.
- Consider using temporary generic objects for “view” editing that is constructed on view check-out then merged back into a full view for check-in.
- Consider using scalable forms instead of the default Create and Edit User interfaces.

When you use the default User forms to edit a user, Waveset fetches the resources owned by that user the moment you start editing the user's account. In deployment environments where users have accounts on many resources, this potentially time-intensive operation can result in performance degradation.

Optimizing Expressions in Form Fields

Some activities performed in forms call resources that are external to Waveset. Accessing these resources can affect Waveset performance, especially if the results contain long lists of values, such as compiling a list of groups or email distribution lists.

To improve performance during these calls, follow the guidelines in "Using a Java Class to Obtain Field Data" in *Oracle Waveset 8.1.1 Deployment Reference*.

Also, avoid using JavaScript in performance-critical expressions such as `<Disable>` expressions. Short XPRESS expressions are easier to debug if you use the built-in tracing facilities. Use JavaScript for complex logic in workflow actions.

If a form is slow to display, you can use the `debug/Show_Timings.jsp` page to determine the problem. Look for calls to `Formconverter.convertField()`. This method shows how long each field took to compute its value.

Note – You can use the `FormConverter JMX MBean` to identify specific fields in a form that are slow to compute or render to HTML.

Tuning Rules

You use Waveset rules to encapsulate constants and XPRESS logic that can be reused in forms, workflows, and other configurable components in the product.

When writing rules, use the following guidelines (as applicable) to obtain optimal performance:

- Use static declarations to return a constant value.
- Use `defvar` methods to implement algorithms with temporary values for incremented values or for values that are referenced only one time.
- Use `putmap`, `setlist`, or `setvar` methods for complex or expensive calculations whose value must be returned multiple times. Be sure to eventually set the value to `<null>`.

Note – You can use the `Rule JMX MBean` to identify rules that are executing slowly.

Tuning Workflows

You customize Waveset workflows to facilitate and automate complex business processes with various human and electronic touchpoints.

You can use the following methods to improve custom workflow performance:

- Simplify default workflows to improve processing time (especially for bulk processing actions such as Active Sync, bulk actions, and reconciliation) by removing the callout to the Approval subprocess.
- Ensure that no infinite loops exist in your workflows. In particular, be sure that break flags are updated and properly checked in the loops that exist in approval subprocesses.
- Put fetched objects into a variable for use later if you must contact the repository for the same object multiple times.

Using a variable is necessary because Waveset does not cache all objects.

- Specify TargetResources options in WorkflowServers checkoutView to restrict the number of resources that are queried for account information.

The following example shows how to restrict the number of resources being queried for account information.

```
<Argument name='TargetResources'>
  <list>
    <string>resource name[| #]</string>
  </list>
</Argument>
```

Note – In the preceding example, [| #] is an optional parameter that you can use when more than one account exists on a particular resource. In most cases, the resource name is sufficient.

- Clear unnecessary view variables left by forms, especially large maps and lists. For example:

```
<setvar name='myLargeList'></null></setvar>
```

The view is copied multiple times in a TaskInstance object, so large views greatly increase the size of each TaskInstance and corresponding TaskResult.

- Use resultLimit (in seconds) in the TaskDefinition, or set this option during task execution to quickly dispose of completed tasks. Large numbers of TaskInstances impact the following:
 - How taskResults.jsp in footers and some JSP tasks in the Administrator interface are displayed
 - How JSP tasks are displayed
 - Querying each TaskInstance for task renaming
 - Database size

Set the following options as needed:

- (*Preferred selection*) delete — Causes an older TaskInstance of the same name to be deleted before the new task begins execution.

- `wait` — Suspends the current `TaskInstance` until the older `TaskInstance` is deleted or expired due to reaching its `resultLimit`.
- `rename` — Inserts a time stamp into the `TaskInstance` name to avoid naming collisions.
- `terminate` — Deletes an older `TaskInstance` of the same name. Each currently executing `TaskInstance` of the same name is terminated.

Note – Using the `SaveOnlyOnError` attribute (see the following description) with the `resultLimit` attribute can significantly improve performance.

- Use the `SaveOnlyOnError` attribute in `TaskDefinitions` and `TaskInstances` to configure a task that is persisted only when errors or warnings occur.
 - If you set `SaveOnlyOnError` to `true`, and the `resultLimit` value is non-zero, Waveset persists the task if there is an error or a warning. If the `resultLimit` value is 0, Waveset will not persist the task, even if an error or warning occurs.
 - If you set `SaveOnlyOnError` to `false`, Waveset just considers the `resultLimit` value. (The default value is `false`.)

Tuning WorkItems (ManualActions)

The number and size of `WorkItems` (indicated by `ManualActions` in a workflow) can affect memory and system performance *significantly*. By default, Waveset copies an entire workflow context into a `WorkItem`, then writes the workflow context back out after submission.

To improve performance for `WorkItems` and `ManualActions` do the following:

- Reduce the size of `WorkItems`.

By default, `ManualAction` creates a `WorkItem`, then copies each variable in the task context into `WorkItem.variables`. Limiting task context variables prevents overwrites from parallel approvals.

 - Use `ExposedVariables` to limit which variables are copied back into `WorkItem`. For example:

```
<ExposedVariables><List><String>user ...
```
 - Use `EditableVariables` to limit the variables assimilated back into the executing task from `WorkItem`. For example:

```
<EditableVariables><List><String>user ...
```

Remember to include an approval flag, a form button value, and the actual approver's name.
- Change the confirmation page and background processing to improve user interface response time.

- Create a confirmation `ManualAction` or background `ManualAction`, owned by another user such as `Configurator`.
- Set `timeout='5'` (5 seconds) and `ignoreTimeout='true'` to prevent an error message if a user submits an action after the task is executed and the `WorkItems` are deleted.
- Optimize memory use by setting large attribute values, such as value maps and lists, to `null` on submission or instantiate them as `Activity`-scoped variables that quickly pass out of scope.
- Shorten the lifetime of finished tasks.
 - Prevent dead-end tasks by ensuring that each `WorkItem` specifies a `Timeout` and that the workflow anticipates a `Timeout` for each `WorkItem`.
 - Consider using the `resultLimit` and `resultOption` options in the `TaskDefinition` to determine how the Scheduler handles a task after the task completes.
 - Use `resultLimit` to control how many seconds a task is allowed to live after the task has completed. The default is zero (0), which means that the task instance will be deleted immediately after task completion.
 - Use `resultOption` to specify what action to take when repeated instances of a task are started (such as `wait`, `delete`, `rename`, or `terminate`). The default is `delete`.
If you want to immediately delete tasks that complete successfully, but you also want to keep tasks containing errors long enough to debug, you can conditionally delete finished tasks.
Set a `resultLimit` in the `TaskDefinition` to a sufficient time period to debug issues. You can set `resultLimit` to zero (or a small value) if no errors are reported at runtime (such as `WF_ACTION_ERROR` is `<null/>`) after a `WorkflowServices` call.
- Evaluate and fix poorly scoped variables. Scope variables according to where they are declared, as follows:
 - **Global variables** are values that must be used across many activities (such as the case owner, view) and as approval flags in subprocesses.
If a variable is declared as an element of `<TaskDefinition>`, scope the variable globally. If a variable is declared `external`, its value is resolved up the call stack.
 - **Activity variables** of expensive values (such as those variables that require a resource fetch or that store a large list or map of values) can be referenced in a `WorkItem`.
If a variable is declared as an element of `<Activity>`, ensure that the variable is visible to actions and transition elements in `Activity`.

Note – Beginning with Waveset Version 2005Q3M1 SP1, use `<Activity>` variable values in Forms, rather than in workflows, to avoid copying values on `WorkItem` creates.

- **Activity variables** are values used in transition logic.

If a variable is declared as an element of `<Action>`, the variable should pass out of scope on completion of the action. Action variables are typically used in `WorkflowApplication` invocations to “change” the names of variables set by the application (such as `View → User`).

- Do not specify synchronous execution (`syncExec='true'`) for the last page in a wizard workflow.

If set to `true`, the task will run to completion when the user executes the task. The interface will hang until the task completes or encounters another stopping point (such as another `ManualAction`).

- Remove unnecessary approval checks.

Note – For Active Sync, use a streamlined provisioning task in place of the system-specified provisioning task specified by `viewOptions.Process`.

- Do not modify the provisioning tasks provided with Waveset.

You must create a new task, then identify that task in the form and in the process mappings configuration (unless the task is not listed).

Working with Waveset Wizards

Sometimes automating a process requires a *wizard*. A wizard is a multi-step GUI operation where each step presents the user with a page used to capture or display data. Waveset provides two techniques for building a wizard:

- **Form-Based wizards.** These wizards use form processing to change what the user sees, allowing sets of fields to be visible or invisible based on data held in the view.
- **Workflow-Based wizards.** These wizards require task execution and suspension to provide page transitions.

A Form-Based wizard is four-times more efficient than the best Workflow-Based wizard. Workflow-Based wizards can vary by up to 10x in performance, depending on how they are constructed. Wizard efficiency is typically not noticeable until many wizards are run concurrently.

Because view processing does not necessarily require object repository access, a single Waveset server can process many Form-Based wizards concurrently without contention. However, Form-Based wizards are limited in what they can do between steps. The only processing these wizards can perform between steps is done by normal form or view processing. Form derivation and expansion, which are done when the view is being refreshed, is more limiting than the processing that is possible in a workflow. If a Form-Based wizard's processing limits preclude its use, you must use a Workflow-Based wizard. However, you should consider using a Form-Based wizard first.

Because Workflow-Based wizards require task execution and suspension to provide page transitions, and because each page in a Workflow-Based wizard corresponds to a workflow `ManualAction` object, these wizards are less efficient than Form-Based wizards. A problem occurs when the wizard has hundreds of concurrent invocations because all of the task start, suspend, and resume operations must contend for the object repository task table.

The object repository is being accessed synchronously with each HTTP request; consequently, a Workflow-Based wizard does not scale to large numbers of concurrent executions due to repository contention. Each start, suspend, and resume operation involves several reads and writes to the task table in the repository, and under a large concurrent load results in the page-to-page response of the GUI slowing down. Unlike a Form-Based wizard that scales by adding more Waveset servers and balancing the HTTP load between them, Workflow-Based wizards slow down due to repository contention. Adding more Waveset servers actually makes the problem worse because the repository is shared between all servers.

If you must use a Workflow-Based wizard because you need processing between page transitions, consider using transient `ManualAction`. A `ManualAction` in a Workflow-Based wizard is the mechanism used to display pages to the user. A five-step wizard typically has at least five `ManualActions`.

If you construct the wizard so that the user must either complete or abort the entire flow, then you can mark the `ManualActions` with the `transient='true'` attribute. Adding this attribute allows Waveset to bypass the repository instead of keeping the task in memory and sequencing it without accessing the object repository. This construction decreases the load the wizard puts on the repository, allowing it to scale to higher concurrency loads. However, using this attribute has some drawbacks. If you set `transient` to `true` (it is `false` by default), you cannot restart the workflow after the task enters the transient section because the in-memory state of the workflow does not match the repository state. Also, each HTTP request for the wizard must use the same server because the true state of the wizard is now only kept in memory on the initiating server. As soon as Waveset encounters a `ManualAction` with `transient=false`, Waveset writes the workflow to the repository normally, and normal workflow behavior resumes.

Consider using a Workflow-Based wizard with the following structure:

```
Begin Activity 1
ManualAction 1 (transient = true)
Activity 2
ManualAction 2 (transient = true)
ManualAction 3 (transient = true)
ManualAction 4 (transient = true)
ManualAction 5 (transient = false)
Activity 3
...
```

When launched, this wizard creates a task that is initially stored in the object repository. All processing between `ManualAction 1` and `ManualAction 5` is done without any further repository work (the transient section). When Waveset executes `ManualAction 5`, the task is again stored in the repository (normal behavior), which is a significant performance savings because each normal suspend/resume for a `ManualAction` does the following:

1. Creates and stores a `WorkItem`
2. Saves the task state
3. Reads/writes the `WorkItem`
4. Locks the task
5. Resumes the task
6. Deletes the `WorkItem`

Each suspend/resume pair (when in wizard mode) results in more than 20 repository read/writes, all on the task table. The state of the task in the repository is *running*, so if the server crashes or shuts down during the execution of the wizard, the task will be deleted from the repository within a few minutes by another server or when the crashed server restarts, whichever comes first.

Note – If you use a transient `ManualAction`, you can observe the effects by looking at the JMX `TaskInstanceCache` and `WorkItemCache` MBeans. These MBeans show the number of `Store` (repository write) operations compared to `Cache` (memory only) operations occurring. Each `Cache` operation means a `Store` was avoided, thus reducing object repository contention.

Updating Database Statistics

As a database administrator, you should frequently run statistics to monitor your repository database.

Performance problems are often caused by bad or missing database table statistics. Fixing this problem improves performance for both the database and Waveset performance.

See the following Oracle articles for more information:

- [Oracle MetaLink: Note:114671.1: Gathering Statistics for the Cost Based Optimizer](#)
- [Cost Control: Inside the Oracle Optimizer](#)

Also consider using SQL Profiles, which is another method for choosing the best query plans. You can use the SQL Advisor within Enterprise Manager to create these profiles when you identify poorly performing SQL.

Tuning Data Exporter

Data Exporter enables you to export new, changed, or deleted Waveset data to an external repository that is suitable for reporting or analytic work. The actual exporting of data is done in batches, where each type of data to be exported is able to specify its own export cycle. The data to be exported comes from the Waveset repository and, depending on the length of the export cycle and the amount of changed data, the volume of exported data can be large.

Some Waveset data types are queued into a special table for later export. Specifically, `WorkflowActivity` and `ResourceAccount` data is queued because this data is not persisted otherwise. Any persisted data type can also be queued if the warehouse needs to see all changes to the type, or if the type has a lifecycle that does not correspond to the export cycle, such as `TaskInstance` and `WorkItem` data.

To maximize performance, only queue and export the types of data that you require in the warehouse. Data exporting is disabled by default, but if you enable data exporting, it exports all data types. Turn off any data types that you do not need.

When the export task exports data, the task attempts to complete the export as quickly as possible, using multiple threads to achieve as much throughput as possible. Depending on the I/O speed of the Waveset repository and the warehouse, the export task can fully utilize the processors on the Waveset server, which causes any interactive performance to degrade. Ideally, the export should occur on a machine dedicated to that task or at least occur during periods when there is no interactive activity on the machine.

The export task supports the following tuning parameters:

- Queue read block size
- Queue write block size
- Queue drain thread count

The drain thread count is the most important throughput. If a large number of records are in the queue table, increasing the number of threads (up to 24) tends to increase throughput. However, if the queue is dominated by one type of record, fewer drain threads might actually be faster. The export task attempts to divide the queue table contents into as many sets as there are threads allocated, and to give each thread a set to drain. Note that these threads are in addition to the drain threads that are draining the other repository tables.

Tuning the General XML

You can usually optimize the general XML by using static `XMLObject` declarations wherever possible. For example, use:

- `<List>` instead of `<list>`
- `<String>` instead of `<s>`
- `<Map><MapEntry . . .></Map>` instead of `<map>`

Also, depending on the context, you might have to wrap objects instead of using the `<0></0>` element.

Tuning Oracle Waveset Service Provider

You can use Waveset dashboard graphs to quickly assess the current system, spot abnormalities, and understand historical trends (such as concurrent users or resource operations over a time period) for Oracle Waveset Service Provider (Service Provider).

Note – Service Provider does not have an Administrator interface. You use the Waveset Administrator interface to perform almost all administrative tasks (such as viewing dashboard graphs).

For more information about tuning Service Provider see *Oracle Waveset Service Provider 8.1.1 Deployment*.

Tuning the Waveset Web Interface

When you are working with the Waveset Web Interface, you can optimize performance by using the OpenSPML toolkit that is co-packaged with Waveset.

Note – Using the `openspml.jar` file from the <http://openspml.org/> web site might cause memory leaks.

Tuning Initial Loads

To improve performance during a large, initial user load, follow this procedure:

1. Disable all Audit Events from the Waveset Administrator interface.

Note – Audit Logging can add several records per operation, making future audit reports perform more slowly.

- a. Choose Configure → Audit.
 - b. On the Audit Configuration page, deselect the Enable auditing box and click Save.
2. Disable the list cache by shutting down the web server or by changing the `ChangeNotifier.updateSearchIntervalCount` property (on the `debug/Show_WSPop.jsp` debug page) to `0`.

The list cache keeps lists of users in frequently accessed organizations in memory. To maintain these lists, the list cache searches for and checks all newly created users.

3. Clear the current list cache on the `debug/Clear_List_Cache.jsp` page.
4. Ensure that the workflow being used to process the users does not contain approvals.
5. Use alternative load methods, which include:
 - Splitting the load and running the data in zones
 - Using bulk loads, which are much faster
 - Loading from a file
6. Disable Data Exporter for the `WorkflowActivity` type.

Tuning Memory Requirements

You must determine your memory needs and set values in your application server's JVM by adding maximum and minimum heap size to the Java command line. For example:

```
java -Xmx512M -Xms512M
```

To improve performance do the following:

- Set the maximum and minimum heap size values to the same size.
- Depending on your specific implementation, you might want to increase these values if you run reconciliation.
- For performance tuning purposes, you may also set the following in the `waveset.properties` file:

```
max.post.memory.size value
```

Note – The `max.post.memory.size` specifies the maximum number of bytes that a posted file (for example by using an `HTML FileSelect` control) might contain without being spooled to the disk. For cases where you do not have permission to write to temporary files, increase the `max.post.memory.size` to avoid having to spool to the disk. The default value is 8 Kbytes.

For additional information about system requirements, see the [Oracle Waveset 8.1.1 Release Notes](#).

Tuning Operating System Kernels

For information about tuning Solaris and Linux operating system kernels, see the “Tuning the Operating System” chapter in the *Sun Java System Application Server Enterprise Edition Performance Tuning Guide*.

For information about tuning Oracle operating system kernels, see the product documentation provided with your Oracle system.

Tuning the Waveset Profiler

Each Waveset server captures profiling data by default. You can use this data with the Waveset IDE to diagnose a large range of performance problems. However, capturing and storing this profiling data adds a measurable load to the server, which consumes both memory and CPU. In a stable production environment, disable the profiler in production servers. Enable the profiler only when you are investigating a performance problem.

Note – For information about using the Waveset IDE, see [“Identity Manager IDE Frequently Asked Questions \(FAQ\)”](#) in *Oracle Waveset 8.1.1 Release Notes*.

EXAMPLE 4-1 Modifying the SystemConfiguration Configuration Object

The following example shows how to disable the Profiler from capturing data.

1. Import the following XML to disable the profiler.

Note – Setting the attribute value to **true** *disables* the profiler, and setting it to **false** *enables* the profiler.

2. Use the `lh import` command to import the following XML

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Waveset PUBLIC 'waveset.dtd' 'waveset.dtd'>

  <Waveset>
    <ImportCommand class='com.waveset.session.SystemConfigurationUpdater' >
      <Object>
        <Attribute name='server.default.disableProfiling'>
          <Boolean>true</Boolean>
        </Attribute>
      </Object>
    </ImportCommand>
  </Waveset>
```

3. Restart all Waveset servers.

Tuning Provisioner

Network latency tends to be a common cause for performance issues when dealing with view provisioning. Tracing individual resource adapters can help you determine what is causing performance problems.

To improve provisioner performance, do the following:

- Set the provisioner `maxThreads` parameter in the `Waveset.properties` file to limit the maximum number of threads that are started to perform parallel resource provisioning operations each time a user is created, modified, or deleted.

The default value is 10, which generally provides optimal performance. Specifying a value greater than 20 significantly degrades the provisioner's performance.

For example, if a user has 15 resource accounts, a maximum of ten provisioner threads are started to simultaneously perform resource provisioning operations on ten resource accounts. The remaining five resource accounts will not be modified until the first ten threads have completed resource provisioning operations.

In a different example, if you are modifying two users, and each user has two resource accounts, four provisioner threads might be used to update those resources.

Increasing the `provisioner.maxThreads` value can help throughput when users have many resource accounts. However, specifying a very large value might create many provisioner threads, which can degrade performance on the server as a whole.

- Configure quota settings in the `Waveset.properties` file to control the number of concurrent operations (such as reprovisioning) a user can execute for a specific task. Increasing the number of concurrent actions can help more operations complete faster, but trying to process too many actions at once might cause bottlenecks.

You can create configuration sets on a per-pool basis. For example, if you create configuration A, configuration B, and configuration C, when you create a `TaskDefinition` (workflow), you can assign a specific pool configuration to the workflow from the configurations that you defined.

The following example shows the quota settings that limit user bob to running one reprovisioning task at a time:

```
Quota.poolNames=ReProvision,Provision
Quota.pool.ReProvision.defaultLimit=1
Quota.pool.ReProvision.unlimitedItems=Configurator
Quota.pool.ReProvision.items=bob,jan,ted
Quota.pool.ReProvision.item.bob.limit=1
```

To enforce the task quota, reference `poolName` in a `TaskDefinition`. The format is as follows:

```
<TaskDefinition ... quotaName='{poolName}'...>
```

Most users start only one task at a time. For proxy administrators who perform reconciliation or Active Sync tasks, set the task quota higher.

Note – Avoid using the Configurator user for reconciliation and Active Sync tasks. The Configurator has access to unlimited tasks and can monopolize available resources, which adversely affects concurrent processes.

Tuning Reconciliation

The Reconciler is the Waveset component that performs reconciliation. This section suggests methods for improving Reconciler performance, including:

- [“General Suggestions for Tuning Reconciliation” on page 72](#)
- [“Tuning the Reconciler Server Settings” on page 73](#)
- [“Tuning Reconciliation for Multiple Resources” on page 74](#)

General Suggestions for Tuning Reconciliation

In general, you can improve Reconciler performance if you do the following:

- Avoid using the Configurator user for reconciliation tasks. The Configurator has access to unlimited tasks and can monopolize available resources, which adversely affects concurrent processes.
Instead, use a streamlined, minimal user for reconciliation and Active Sync tasks. Because the subject executing the task is serialized as part of the task, a minimal user takes less space, or overhead, for each task and update in the repository.
- Use information on the Reconciler status page (`debug/Show_Reconciler.jsp`) to decide which settings to adjust based on queue sizes, available system resources, and performance benchmarks. Be aware that these settings are dependent on the environment.

Note – The Reconciler JMX MBean provides much of the information in `Show_Reconciler.jsp`, and shows a processing rate estimate.

- Use the System Memory Summary page (`debug/Show_Memory.jsp`) to see how much total and free memory is available. Reconciliation is a memory-intensive function, and you can use this information to determine whether there is sufficient memory allocated to the JVM. You can also use this page to launch garbage collection or to clear unused memory in the JVM for investigating heap usage.
- When you assign user forms to proxy administrators who are performing reconciliations, keep the user forms as simple as possible and only use essential fields. Depending on the schema map, including a field that calculates the `waveset.organization` attribute is generally sufficient.

Note – Administrators who need to view or edit the Waveset schema for Users or Roles must be in the IDM Schema Configuration AdminGroup and must have the IDM Schema Configuration capability.

- Use per-account workflows judiciously. The reconciliation process does not start provisioning tasks for performance reasons by default.
If you must use a per-account workflow task, edit the reconciliation policy to limit the Reconciler's automatic responses to events of interest only. (See the Situation area of the Edit Reconciliation Policy page.)

Reconciliation of a resource goes through two phases. In the first phase, Waveset gets a list of all users in its internal repository that are known to have accounts on the resource. This first phase does not involve the physical resource at all, and typically happens very quickly.

The second phase requests a list of all accounts from the resource, and then processes those accounts; potentially linking them to users or even creating new users. Performance of the second phase, indicated as reconciling accounts in the resource status message, is proportional to the speed of the resource and the number of worker threads. You can compensate for a slow resource by adding more worker threads; assuming the resource can handle more concurrent AccountGet operations.

Note – There is a JMX MBean for each resource that shows the average, minimum, and maximum response times for each resource operation. Reconciliation phase two involves lots of Account_Get operations, so the average time for each Account_Get strongly influences the overall reconciliation performance. To compensate for resources with longer Account_Get times, use more worker threads. However, because the same number of worker threads are used for all resources, setting a maximum worker thread too high might overwhelm the Waveset object repository on faster resources.

Tuning the Reconciler Server Settings

Although the default settings are usually adequate, you can sometimes improve Reconciler performance if you adjust the following settings on the Edit Server Settings page:

- **Parallel Resource Limit.** Specifies the maximum number of resource threads that the Reconciler can process in parallel.
Resource threads allocate work items to worker threads, so if you add additional resource threads, you might also have to increase the maximum number of worker threads.
- **Minimum Worker Threads.** Specifies the number of processing threads that the Reconciler always keeps open.

- **Maximum Worker Threads.** Specifies the maximum number of processing threads that the Reconciler can use. The Reconciler starts only as many threads as the workload requires, which places a limit on that number. Worker threads automatically close if they are idle for a short duration.

During idle times, the threads stop if they have no work to do, but only down to the minimum number of threads specified. As the load increases, the Reconciler adds more threads until the maximum number of threads is reached. The Reconciler never has less than the minimum number of threads or more than the maximum.

Generally, more threads allow more concurrency. However, at some point, too many threads can put too much load on the machine or just do not provide additional benefit. Because the worker threads are typically reading and writing User and Account objects, having too much concurrency might overload the Waveset repository RDBMS.

Note – Recommending generic, optimal settings is not possible because deployments are so different. Reconciler settings must be adjusted differently for each deployment environment.

▼ To Change the Reconciler Server Settings

Perform the following steps to change the Reconciler server settings:

- 1 **Log into the Administrator interface.**
- 2 **Click the Configure → Servers → Reconciler tabs.**
- 3 **When the Edit Server Settings page is displayed, adjust the settings as necessary.**
See [“Editing Default Server Settings” on page 25](#) for more information.

Tuning Reconciliation for Multiple Resources

If you are configuring reconciliation for multiple resources in Waveset, you have several options:

- All of the resources on the same server, all at the same time.
This option is the most efficient from the Waveset perspective, but if you have many resources (for example more than 20), you are likely to experience Java resource issues.
- All of the resources on the same server, each at a different time.
This option is easier on Java resource loading, but puts a significant burden on your schedule configuration.
- Each resource on a different server, all at the same time.
This option minimizes elapsed time, but increases the number of servers.

An ideal solution does not exist for this configuration because deployments are so different. You might have to mix and match these options to find an acceptable solution for your deployment.

Preparing a usage survey, based on the business reasons behind this functionality, might help you decide how to proceed.

Address these questions:

- Why are you reconciling these resources?
- Do you have the same the goal for each of these resources?
- Are each of these resources equally important or critical?
- Must all resources be reconciled on the same schedule, or can you spread out the reconciliations?
- How often must each resource be reconciled?

Also, remember that the reconciliation server does not have to be one of the pools that handles web traffic. You can add a server that you never interact with directly because this server exists solely for transaction processing. Having a server dedicated to transaction processing might make the first option more attractive for very large systems.

Tuning Resource Queries

Note – Network latency tends to be a common cause of performance issues during view provisioning. Tracing individual resource adapters can help you determine what is causing performance problems.

You can improve resource query performance if you use `FormUtil.getResourceObjects` to implement the query.

Use one of the following methods to cache query results:

- `getResourceObjects(Session session, String objectType, String resID, Map options, String cacheList, String cacheTimeout, String cacheIfExists)`
- `getResourceObjects(String subjectString, String objectType, String resID, Map options, String cacheList, String cacheTimeout, String clearCacheIfExists)`

Note –

- Set `cacheTimeout` in milliseconds.
 - Restrict searches to specific `searchContext`, if applicable.
 - Return the minimum number of attributes in `options.searchAttrsToGet`.
-

Tuning the Scheduler

The Scheduler component controls task scheduling in Waveset.

This section suggests methods for improving Scheduler performance, including:

- [“General Suggestions for Tuning the Scheduler” on page 76](#)
- [“Tuning the Scheduler Server Settings” on page 77](#)
- [“Tuning Scheduler Attributes in the SystemConfiguration Object” on page 78](#)

General Suggestions for Tuning the Scheduler

The following `TaskDefinition` options determine how the Scheduler handles tasks after they are completed:

- `resultLimit` — Controls how many seconds task results are kept after the task has completed. The default setting varies for different tasks. A setting of zero immediately removes tasks after completion.
- `resultOption` — Controls what action is taken when repeated instances of a task are started. The default setting is `delete`, which removes previous task instances.

These default settings are designed to optimize memory by shortening the lifetime of finished Scheduler tasks. Unless there is a compelling reason to change these settings, use the defaults.

If you want to immediately delete tasks that completed successfully, but you also want to keep tasks containing errors long enough to debug, you can do the following:

- Set the `resultLimit` to a small value, such as 3600 seconds.
- Set the `saveOnlyOnError` value to **true**.

With these two settings, Waveset will only store the task results if the task has an error or warning `ResultItem`. This configuration can improve the performance of some tasks by allowing them to bypass being stored in the repository when they complete.

Tuning the Scheduler Server Settings

You can sometimes improve Scheduler performance by adjusting the following settings on the Edit Server Settings page:

- **Maximum Concurrent Tasks.** Specifies the maximum number of tasks that the Scheduler can run at one time.

When more tasks are ready to run than the Maximum Concurrent Tasks setting allows, the extra tasks must wait until there is room available or until they are run on another server.

If too many tasks are being swapped out of memory and sharing CPU time, the overhead slows down performance. Alternatively, setting the maximum too low results in idle time. The Scheduler checks for available tasks every minute, so a waiting task waits at least a minute before being run.

The default Maximum Concurrent Tasks setting (100) is usually adequate. You can decide whether to adjust this setting up or down based on which tasks are being run in the deployment and by profiling the runtime behavior after the deployment is otherwise complete.

Before you change this setting, you may want to observe the Scheduler using JMX. The `Scheduler.executingTaskCount` attribute shows how many tasks the Scheduler is running.

In some cases, you might want to suspend or disable the Scheduler. For example, if you want a server dedicated to handling the End User interface, disabling the Scheduler will prevent tasks from running on that server. The server would only serve the End User interface pages and store launched tasks for other servers to execute.

- **Task Restrictions.** Specifies the set of tasks that can execute on the server.

The Task Restrictions setting can provide a finer granularity of control over what tasks are allowed to run on a server. You can restrict tasks individually or through the server settings.

Note – Recommending generic, optimal settings is not possible because deployments are so different. Scheduler settings must be adjusted differently for each deployment environment.

▼ To Change the Scheduler Server Settings

- 1 Log in to the Administrator interface.
- 2 Click the **Configure** → **Servers** → **Scheduler** tabs.
- 3 When the **Edit Server Settings** page is displayed, adjust the settings as necessary.

See “[Editing Default Server Settings](#)” on page 25 for more information.

Tuning Scheduler Attributes in the `SystemConfiguration` Object

To improve Scheduler performance when the component is under a heavy backlog, you must modify the `SystemConfiguration` configuration object to enable all optimizations. The Scheduler can pick up any changes to the `SystemConfiguration` object while the server is running.

The Waveset Scheduler is responsible for executing scheduled tasks, resuming suspended tasks, and cleaning up task results for completed tasks. The Scheduler is single-threaded, meaning it processes its work with a single control thread. However, when the Scheduler starts or resumes a task, it runs that task in a new thread and can have several tasks running at the same time.

You can directly start tasks without any Scheduler processing. For example, workflow and report tasks are often started as a result of a HTTP request. Starting, suspending, and resuming a task causes a modest amount of object repository processing. The repository can become congested if lots of concurrent HTTP requests are starting or resuming tasks. Because an application server may service hundreds of HTTP requests simultaneously, it is easy to create a large backlog for the Scheduler.

For example, if hundreds of HTTP requests started workflows that were subsequently suspended, the Scheduler would be responsible for resuming those workflows.

Because the Scheduler has a single control thread, but many different tasks to perform, it might sometimes seem like the Scheduler is not keeping up with one task or another. Waveset uses the `sleepingTaskLimit` and `readyTaskLimit` attributes to control how long the Scheduler can spend processing sleeping or ready tasks during each control loop. These limits ensure that a Scheduler, when presented with thousands of ready tasks, does not spend too much time starting these tasks and ignoring sleeping tasks.

When you have multiple Waveset servers, each server typically runs a Scheduler. By default, these Schedulers compete for the same work by polling the object repository. Configuring `blockProcessing` allows each Scheduler to process a different block of tasks, resulting in somewhat less object repository contention for specific records. The `blockProcessing` attribute is enabled by default, but all servers process the same slot. To enable cooperative (rather than competitive) task processing, you can assign each server a different slot (starting with 0). When you assign slots (other than 0) to the servers, they segment the tasks into buckets and only process the tasks in their assigned bucket.

The Scheduler has a JMX MBean that is very useful in diagnosing what is perceived as slow Scheduler performance. The `ExecuteTime` attribute in the MBean is often the key to understanding the Scheduler's performance. `ExecuteTime` is the time (in milliseconds) it took the Scheduler to start or resume the last task it processed. On a healthy system, this time should be less than 150 milliseconds. When this value starts to get large, the server is having trouble starting tasks, typically because there is congestion on the task table in the object repository, or because of internal synchronization in the Waveset code itself. Viewing the Scheduler's thread stack in the JMX console usually reveals the problem.

The `ExecutingTaskCount` attribute in the MBean shows how many tasks the Scheduler is currently managing. By default, the task limit is 100, which is almost always sufficient unless the tasks being executed run for a long time without suspending (such as report tasks). The `ExecutingTaskCount` value does not reflect all tasks running on the server. Remember that HTTP requests can also start tasks, so the total number of tasks running on the server is unknown to the Scheduler.

One of the Scheduler's many jobs is to resume tasks. Because each Waveset server can have a Scheduler running, the Schedulers also periodically look to process work that was being handled by another server that is not currently running. When a server goes into the recovered state, Schedulers on other servers attempt to process or clean up work that was being done on the recovered server. Schedulers go into a recovered state when one server observes another server has not issued a heartbeat within the last five minutes.

This check compares the timestamp of the last heartbeat message in the repository to the current server's clock. If Waveset servers have more than five minutes of system clock skew, the server with the clock that is farthest ahead marks servers with clocks that are behind as recovered. To avoid this situation, keep the system clocks on your Waveset servers synchronized.

Note – If necessary, review “Editing Waveset Configuration Objects” in *Oracle Waveset 8.1.1 Business Administrator’s Guide*.

To control the Scheduler's behavior, add one or more of the following attributes to the `SystemConfiguration` configuration object. Use the `serverSettings.default.scheduler.attribute` path to create these attributes.

- `blockProcessing` (Boolean). Indicates whether servers should work on independent blocks of tasks or not. Defaults to true, which means Scheduler will work on independent blocks of tasks.

Note – `blockProcessing` does not apply to task deletion.

- `blockProcessingSize` (Integer). Represents how many tasks a Scheduler will process in its block. Defaults to 50.
- `slots` (List of Strings). Indicates slots of Waveset server instance names. Defaults to an empty list.
 - If the `blockProcessing` attribute is *true*, each Scheduler processes the block at the server's position in this list.
 - If you do not provide a list of slots, or do not include a server in the list, every Scheduler assumes it has slot 0 and each Scheduler processes the same block of tasks concurrently.

Note – slots do not apply to task deletion.

- `fastResume` (Boolean). Instructs the Scheduler to immediately execute tasks that are sleeping with an expired `restart` time. Defaults to `true`.
 - If `true`, the Scheduler executes sleeping tasks without first placing them in the ready state, which reduces load on the Identity Manager repository.
 - If `false`, the Scheduler first puts the task in a ready state, and then executes the task on the next Scheduler poll.
- `threadPriority` (Boolean). Indicates the Scheduler thread should be run at an elevated priority. Defaults to `false`.
 - If `true`, the Scheduler threads run with an elevated priority.
 - If `false`, the Scheduler threads are only read at server startup.
- `sleepingTaskLimit` (Integer). Indicates how many milliseconds the Scheduler can spend processing sleeping tasks before reentering the polling loop. Defaults to 60 seconds.

The Scheduler is single-threaded and does not perform any other operations when it is processing sleeping tasks. If a large number of tasks are sleeping and ready to run, the Scheduler only processes as many tasks as it can within this time limit.
- `readyTaskLimit` (Integer). Indicates how many milliseconds the Scheduler can spend processing ready tasks before reentering the polling loop. Defaults to 60 seconds.

The Scheduler is single-threaded and does not perform any other operations when it is processing ready tasks. If a large number of tasks are ready to run, the Scheduler only processes as many tasks as it can within this time limit.

Tuning Sessions

Waveset maintains a least recently used (LRU) cache of authenticated sessions for use by authenticated users. By using existing authenticated sessions, you can speed up repository access for objects and actions that require a session.

To optimize the authentication pool size, change the `session.userPoolSize` value in the `Waveset.properties` file to the maximum number of expected, concurrent user sessions on the server.

Tuning the Oracle Waveset Gateway

The Oracle Waveset Gateway generates a thread for each connection, and uses a different pool for each unique combination of resource type, Gateway host, and Gateway port. The Gateway checks for idle connections every five minutes. When a connection has been idle for 60 seconds, the Gateway closes and removes that connection from the pool.

When the Gateway receives a request, it does the following:

- If there are no idle connections in the corresponding pool, the Gateway creates a new connection.
- If an idle connection exists in the pool, the Gateway retrieves and reuses that connection.

You must configure the maximum number of connections on the resource, and you must configure these connections the same way for all resources of the same type, that are using the same Gateway. For that resource type, the first connection made to the Gateway on a given host and port uses that resource's maximum connections value.

Note – When you change the maximum number of connections on a resource, you must start and stop the server for the change to take effect.

The following example shows how connections, requests, and Gateway threads are related.

If you set the maximum number of connections to 10 on an Active Directory resource, and you are using two Waveset servers, then you can have up to 20 simultaneous connections (10 from each Waveset server) to the Gateway for that Active Directory resource. The Gateway can have 10 simultaneous requests outstanding from each server, and the Gateway processes each request on a different thread. When the number of simultaneous requests exceeds the maximum number of Gateway connections, additional requests are queued until the Gateway completes a request and returns the connection to the pool.

Note – Although the Gateway code is multi-threaded, this characteristic does not address the APIs or services being used by the Gateway. For Active Directory, the Gateway uses the ADSI interface provided by Microsoft. No investigation has been done to determine whether this interface handles Gateway requests in parallel.

Other methods for improving Gateway performance, include:

- Locating the Gateway near (from a network connectivity perspective) the domain controllers of the managed domain
- Increasing the block size on a Gateway resource can increase throughput during reconciliation or load operations

Increased throughput results have been noted for basic reconciliations with no custom workflows and in which no attribute reconciliations are being performed. Initially, the Gateway does consume more system memory, but this memory is eventually released.

Be aware that there is a diminishing return. At some point, larger block sizes do not result in proportionately increased performance. For example, the following data shows the speed observed for a Load from Resource of 10,000 users from an Active Directory resource. Also, the peak memory usage for the Gateway process during the load is included.

| Block Setting | Users Created Per Hour | Peak Gateway Memory Usage |
|---------------|------------------------|---------------------------|
| 100 | 500 | 20 MB |
| 200 | 250 | 25 MB |
| 500 | 9690 | 60 MB |
| 1000 | 10044 | 92 MB |

- For Exchange Server 2007, the PowerShellExecutor performs actions for Exchange Server 2007. You can modify the following registry settings to change the behavior of the PowerShellExecutor inside the Gateway.

Note – Both settings can have a large impact on the behavior and memory usage of the Gateway. Changes to these parameters should only be considered after careful testing.

– powerShellTimeout

Content. Timeout for PowerShell actions (registry type REG_DWORD)

Default. 60000 ms (1 minute)

When the powerShellTimeout setting times out, any RunSpace actions are interrupted and canceled to prevent runaway actions in the PowerShell environment that cause the Gateway to become unresponsive.

Decreasing the powerShellTimeout value to a small value can prematurely cancel actions, and can prevent the RunSpace initialization from finishing correctly. Observed startup times for the first RunSpace in the pool range from 2—5 seconds.

The powerShellTimeout value is read-only on startup, and you cannot change it without restarting the gateway.

– runSpacePoolSize

Content. Number of RunSpaces in the pool (registry type REG_DWORD)

Default. 5

Minimum. 5

Maximum. 25

The number of RunSpaces in the pool allow for parallel execution of PowerShell actions by the gateway. One provisioning action or update of a user in Exchange 2007 can result in multiple PowerShell actions being executed.

A started RunSpace can consume a large amount of memory. For the first RunSpace, the typical size is approximately 40 MB. Subsequent RunSpaces normally use between 10—20 MB.

Note – The preceding figures can differ in specific environments and are only given as *guidelines*, so be careful when changing this value.

The `runSpacePoolSize` value is read-only on startup, and you cannot change the pool size value without restarting the Gateway.

Tuning the Task Bar

The Administrator interface task bar displays links to previously performed provisioning tasks, which causes the interface to render more slowly when there are a large number of tasks.

To improve interface performance, remove the `taskResults.jsp` link from each JSP by deleting the `<List>...</List>` element from the `UserUIConfig` object.

The following example shows `<List>...</List>` entries within `<TaskBarPages>`.

EXAMPLE 4-2 Modifying the UserUIConfig Object

```
<TaskBarPages>
  <List>
    <String>account/list.jsp</String>
    <String>account/find.jsp</String>
    <String>account/dofindexisting.jsp</String>
    <String>account/resourceReprovision.jsp</String>
    <String>task/newresults.jsp</String>
    <String>home/index.jsp</String>
  </List>
</TaskBarPages>
```

Debugging Performance Issues

This section describes the different Waveset and third-party debugging tools you can use to debug performance issues.

The information is organized into the following sections:

- “Working With Waveset Debug Pages” on page 84
- “Working with the Waveset Profiler” on page 89
- “Working With Other Debugging Tools” on page 108

Working With Waveset Debug Pages

Note – Tracing affects system performance. To help ensure optimal performance, specify the minimum tracing level or turn tracing off after debugging the system.

This section provides instructions for accessing the Waveset Debug pages and describes how to use these pages to identify and debug Waveset performance issues.

See the following sections for information:

- “Accessing the Debug Pages” on page 84
- “Control Timings (callTimer.jsp)” on page 85
- “Edit Trace Configuration (Show_Trace.jsp)” on page 86
- “Host Connection Pool (Show_ConnectionPools.jsp)” on page 86
- “List Cache Cleared (Clear_XMLParser_Cache.jsp)” on page 86
- “Method Timings (Show_Timings.jsp)” on page 86
- “Object Size Summary (Show_Sizes.jsp)” on page 87
- “Provisioning Threads for Administrator Configurator (Show_Provisioning.jsp)” on page 88
- “System Cache Summary (Show_CacheSummary.jsp)” on page 88
- “System Memory Summary (Show_Memory.jsp)” on page 88
- “System Properties (SysInfo.jsp)” on page 88
- “System Threads (Show_Threads.jsp)” on page 89
- “User Session Pool Cleared (Clear_User_Cache.jsp)” on page 89
- “Waveset Properties (Show_WSProp.jsp)” on page 89
- “XML Resource Adapter Caches Flushed and Cleared (Clear_XMLResourceAdapter_Cache.jsp)” on page 89

Accessing the Debug Pages

Note – You must have the *Debug*, *Security Administrator*, or *Waveset Administrator* capabilities to access and execute operations from the Waveset Debug pages. Administrators and Configurator are assigned this capability by default.

If you do not have the Debug capability, an error message results.

▼ To Access the Waveset Debug Pages

1 Open a browser and log in to the Administrator interface.

2 Type the following URL:

`http://host:port/idm/debug`

where:

- *host* is the application server on which you are running Waveset.
- *port* is the number of the TCP port on which the server is listening.

3 When the System Settings page displays, type the .jsp file name for the debug page you want to open.

For example:

```
http://host:port/idm/debug/pageName.jsp
```

Note – Some debugging utilities are not linked from the System Settings page, but you can use them to enhance your ability to gather data for product performance and usability. For a complete list of debug pages, open a command window and list the contents of the `idm/debug` directory.

Control Timings (`callTimer.jsp`)

Use the Control Timings page to collect and view call timer statistics for different methods. You can use this information to track bottlenecks to specific methods and invoked APIs. You can also use options on the Call Timings page to import or export call timer metrics.

Note – Call timing statistics are only collected while trace is enabled.

▼ To View Call Timer Statistics

- 1 Open the Control Timings page, and click Start Timing & Tracing to enable trace and timing.**
- 2 To stop the timing, click Stop Timing & Tracing or click Stop Timing.**

The page re-displays and populates the Show Timings table with a list of methods for which statistics are available and the methods' aggregate call timer statistics (*not broken down by caller*).

This table contains the following information:

- Method name (Click a method name to see which methods it calls)
- Total time
- Average time
- Minimum time
- Maximum time
- Total calls
- Total errors

- 3 To clear the list, click Clear Timing.**

Note – You can also use the `callTimer` command to collect call timer data from the Console. This command is useful when you are debugging performance issues during an upgrade or in other situations where Waveset is not running on the application server.

Edit Trace Configuration (Show_Trace.jsp)

Use the Edit Trace Configuration page to enable and configure tracing for the Java classes provided with your Waveset installation.

Specifically, you can use this page to configure the following trace settings:

- Choose methods, classes, or packages to trace and specify the level of trace you want to capture.
- Send trace information to a file or to standard output.
- Specify the maximum number of trace files to be stored and the maximum size for each file.
- Specify how dates and times are formatted in the trace output file.
- Specify the maximum number of methods to be cached.
- Indicate how to write data to the trace file.

Write data to the trace file as the data is generated, or queue the data and then write it to the file.

Host Connection Pool (Show_ConnectionPools.jsp)

If you are not using a data source, you can use the Host Connection Pool page to view connection pool statistics. These statistics include the pool version, how many connections were created, how many are active, how many connections are in the pool, how many requests were serviced from the pool, and how many connections were destroyed.

You can also use the Host Connection Pool page to view a summary of the connection pools used to manage connections to the Gateway. You can use this information to investigate low-memory conditions.

List Cache Cleared (Clear_XMLParser_Cache.jsp)

Use the List Cache Cleared page to clear recently used XML parsers from the cache and to investigate low memory conditions.

Method Timings (Show_Timings.jsp)

Use the Method Timings page to quickly detect and assess hotspots at a method level.

The following information is gathered from Waveset methods and displayed on the Method Timings page:

- Method names
- How many times the methods were called
- How many times the methods exited with an error status
- Average time consumed by the methods
- Minimum and maximum times consumed by invocations of each method

The Method Timings page also contains a table with the following links. You can click these links to view additional information.

- **Details.** Shows call stack information.
- **History.** Shows a graph of call duration compared with the time of the most recent calls.
- **History data.** Shows a list of the most recent calls, showing what time the call was made and the duration of the call.

To keep stack history and to control its depth, edit `Waveset.properties` and look at the `MethodTimer` keys.

Note – Waveset does not keep stack history by default. Keeping stack history has a large negative impact on CPU and memory use.

Note – The Clear ALL option on the Method Timings page clears all results. This option is enabled by default.

Object Size Summary (Show_Sizes.jsp)

Use the Object Size Summary page to detect problematically large objects that can affect your system.

The Object Size Summary page shows information about the size of objects (in characters) stored in the repository. These objects are listed by type, along with the total number of objects of each type, and the objects' total combined size, average size, maximum size, and minimum size.

Click an entry in the Type column to view additional size information about that object type. For example, click Configuration to view the ID, name, and size of the largest configuration objects in the repository.

You can also access this size information from the Console command line.

▼ To Access Object Size Information from the Command Line

- 1 Open the console.
- 2 At the command prompt, type:

```
showSizes [ type[limit ] ]
```

Note – For upgrades, existing objects will report a size of 0 until they have been updated or otherwise refreshed.

Provisioning Threads for Administrator Configurator (Show_Provisioning.jsp)

Use the Provisioning Threads for Administrator Configurator to view a summary of the provisioning threads in use by the system. This summary is a subset of the information available in Show_Threads.jsp.

Note – Looking at just a single thread dump can be misleading.

System Cache Summary (Show_CacheSummary.jsp)

Use the System Cache Summary page to view information about the following items to help you investigate low-memory conditions:

- Administrator-associated object caches
- System object cache
- User login sessions
- XML parser cache

System Memory Summary (Show_Memory.jsp)

Use the System Memory Summary page to view how much total and free memory you have available in Mbytes. When you are using memory-intensive functionality such as Reconciliation, this information can help you determine whether there is sufficient memory allocated to the JVM.

You can also use this page to launch garbage collection or to clear unused memory in the JVM for investigating heap usage.

System Properties (SysInfo.jsp)

The System Properties page provides information about your environment, including software versions, paths and environmental variables.

System Threads (Show_Threads.jsp)

Use the System Threads page to see which processes are running so you can verify that automated processes (such as reconciliation or Active Sync) are running.

This page includes information about the process type, process name, its priority, if the process is a daemon, and if the process is still running.

Note – Looking at just a single thread dump can be misleading.

User Session Pool Cleared (Clear_User_Cache.jsp)

Use the Session Pool Cleared page to clear all of the cached sessions for users who have recently logged in and to investigate low memory conditions.

Waveset Properties (Show_WSProp.jsp)

Use the Waveset Properties page to view and *temporarily* edit properties in the `Waveset.properties` file. You can test different property settings for a particular server on which the `Waveset.properties` file resides without having to restart the server to pick up the changes. The edited property settings only remain in effect until the next time you restart the server.

XML Resource Adapter Caches Flushed and Cleared (Clear_XMLResourceAdapter_Cache.jsp)

Use the XML Resource Adapter Caches Flushed and Cleared page to clear test XML resource adapters from the cache and to investigate low memory conditions.

Working with the Waveset Profiler

Waveset provides a Profiler utility to help you troubleshoot performance problems with forms, Java, rules, workflows, and XPRESS in your deployment.

Customized forms, Java, rules, workflows, and XPRESS can cause performance and scale problems. The Profiler profiles how much time is spent in these different areas, enabling you to determine whether these forms, Java, rules, workflows, or XPRESS objects are contributing to performance and scale problems and, if so, which parts of these objects are causing the problems. You must use the Waveset IDE to view the profiled data.

Note – When enabled, the Profiler has both a memory and a performance impact because this feature captures a significant amount of information across a wide range of services. If you are not having performance problems in your production system and you do not need the Profiler data, you can disable the Profiler as described in [“Disabling the Profiler” on page 94](#).

This section explains how to use Waveset’s Profiler and provides a tutorial to help you learn how to troubleshoot performance issues in your deployment.

The information is organized into the following topics:

- [“Overview” on page 90](#)
- [“Getting Started” on page 93](#)
- [“Using the Profiler” on page 95](#)
- [“Tutorial: Troubleshooting Performance Problems” on page 103](#)

Note – Waveset Profiler is only supported on version 7.1 Update 1 and later.

Overview

The section provides an overview of the Waveset’s Profiler’s features and functionality. The information is organized as follows:

- [“Major Features” on page 90](#)
- [“How the Profiler Locates and Manages Source” on page 91](#)
- [“Statistics Caveats” on page 92](#)

Note – The Profiler provides helpful information, but has a performance cost on its own. If you do not need the Profiler data, you can disable this utility using the instructions provided in .

Major Features

You can use the Profiler utility to

- Create “snapshots” of profiling data.
A *snapshot* is the cumulative result of profiling since the last time you reset all of your collected profile results.
- Display snapshot results in four, different data views:
 - **Call Tree view** provides a tree table showing the call timing and invocations counts throughout the system.
 - **Hotspots view** provides a flattened list of nodes that shows the aggregate call timings regardless of parent.

- **Back Traces view** provides an inverted call stack showing all the call chains from which that node (known as the *root node*) was called.
- **Callees view** provides an aggregate call tree of the root node, regardless of its parent chain.
- Specify what kinds of information to include in your snapshot:
 - You can include every element of form, workflow, and XPRESS or restrict the content to a set of specific elements.
 - You can pick specific Java methods and constructors to include or exclude from the instrumentation. Instrumentation of Identity Manager classes and custom classes is supported.
- Manage your project snapshots as follows.
 - Save the snapshot in your project's `nbproject/private/idm-profiler` directory or to an arbitrary location outside of your project.

Note – You can view a list of all saved snapshots in the Saved Snapshots section of the IDM Profiler view.

- Open snapshots from your project or load them from an arbitrary location outside your project.
- Delete snapshots.
- Search for specific nodes, by name.

How the Profiler Locates and Manages Source

This section describes how the Profiler looks up and manages the source for the following Waveset objects:

- **Forms, Rules, Workflows, and XPRESS objects:** When you take a snapshot with the Profiler, the server evaluates all of the profiling data and discovers on which sources the data depends. The server then fetches all of these sources from the repository and includes them in the snapshot. Consequently, you can be sure that the Waveset objects displayed in the snapshot are accurately reflecting the point at which the snapshot was captured.
This process adds to the size of the snapshot, but the source size is actually a relatively small fraction of the total size. As a result, you can send a snapshot to Sun's Customer Support without having to send your source files separately.
- **Java source:** When you take a snapshot of Java source, the client downloads the snapshot and then goes through the snapshot to capture all referenced Java sources from the project. When you save the snapshot, the client zips the sources and attaches them to the end of the snapshot.

Then, when you view the snapshot and go to the Java source, the client first checks the content of the snapshot. If the client cannot find the content there, it checks the project's content. This process allows you to send a snapshot containing profiling data from both your custom Java code and Waveset code.

Note – In a Java source snapshot, do not assume the source is up-to-date with the server or always available.

Tip – In Call Tree view or Hotspots view, you can double-click any node that corresponds to a Java method, workflow, form, rule, or XPRESS to view the source for that node.

Statistics Caveats

The following sections contain information to consider when you evaluate results provided by the Profiler:

- **Self Time Statistics:** To compute a root node's Self Time statistic, the Profiler subtracts the times of all children nodes from the root node's total time. Consequently, an uninstrumented child node's time is reflected in the root node's self time. If a root node has a significant self time, you should certainly investigate why. You might not have the proper methods instrumented and so you are looking in the wrong place. For example, assume method A calls method B.

Method A takes a total time of 10 seconds (where total time includes the call to B) and the call to B takes a total time of 10 seconds.

If both A and B are instrumented, the call stack reflects that information. You will see that A has a self-time of 0 seconds and that B has a self-time of 10 seconds (where 10 seconds was actually spent in B). If, however, B is not instrumented, you only see that the call to A takes 10 seconds and that A's self-time is 10 seconds. Consequently, you might assume the problem lies directly in A rather than in B.

In particular, you might notice large self times on JSPs during their initial compile. If you reset the collected results and then re-display the page, the self time value will be much less.

- **Constructor Calls:** Because there are limitations in the Java instrumentation strategy, initial calls to `this()` or `super()` will appear as a sibling to the constructor call, rather than as a child. See the following example:

```
class A
{
    public A()
    {
        this(0);
    }
    public A(int i)
    {
```

```

    }
}

and:

class B
{
    public static void test()
    {
        new A();
    }
}

```

The call tree will look like this:

```

B.test()
  -A.<init>(int)
  -A.<init>()

```

Rather than this:

```

B.test()
  -A.<init>()
    -A.<init>(int)

```

- **Daemon Threads:** Do not be misled by the seemingly large amount of time spent in a number of Waveset's daemon threads, such as `ReconTask.WorkerThread.run()` or `TaskThread.WorkerThread.run()`. Most of this time is spent sleeping, while waiting for events. You must explore these traces to see how much time is actually spent when they are processing an event.

Getting Started

This section describes how to start the Profiler and how to work with various features of the Profiler's graphical user interface. This information is organized as follows:

- [“Before You Begin” on page 93](#)
- [“Starting the Profiler” on page 94](#)
- [“Disabling the Profiler” on page 94](#)

Before You Begin

Because the Profiler is very memory intensive, you should significantly increase the memory for both your server and the Netbeans Java Virtual Machine (JVM).

To increase your server's memory,

1. Open the Netbeans window and select the Runtime tab.
2. Expand the Servers node, right-click Bundled Tomcat, and select Properties from the menu.
3. When the Server Manager dialog displays, clear the Enable HTTP Monitor box on the Connection tab.
4. Select the Platform tab, set VM Options to **-Xmx1024M**, and then click Close.

To increase the Netbeans JVM memory,

1. Open the `netbeans-installation-dir\etc\netbeans.conf` file and locate the following line:
`netbeans_default_options="-J-Xms32m -J-Xmx ...`
2. Change the `-J-Xmx` value to `-J-Xmx 1024M`.
3. Save, and then close the file.

When you are finished, start the Profiler using the instructions provided in the [“Starting the Profiler” on page 94](#) section.

Starting the Profiler

You can use any of the following methods to start the Profiler from the Waveset IDE window:

- Click the Start Identity Manager Profiler on Main Project icon located on the menu bar.
- Select Window → IDM Profiler from the menu bar.

The Identity Manager Profiler window appears in the Explorer. From this window, select an Waveset project from Current Project drop-down menu, and then click the Start Identity Manager Profiler icon located in the Controls section.

Note – The Start Identity Manager Profiler on Main Project icon is enabled when the main Waveset project is version 7.1 Update 1 or later.

- Right-click a project in the Projects window, and then select Start Identity Manager Profiler from the pop-up menu.
- Select a project in the Projects window, and then select IdM → Start Identity Manager Profiler from the menu bar.

When you start the Profiler, the Profiler Options dialog displays so you can specify which profiling options you want to use. Instructions for setting these options are provided in [“Specifying the Profiler Options” on page 95](#).

Disabling the Profiler

To disable the Waveset Profiler, import the following configuration update:

```
lh import file
```

where *file* contains

```
<?xml version='1.0' encoding='UTF-8'?>  
<!DOCTYPE Waveset PUBLIC 'waveset.dtd' 'waveset.dtd'>  
  
<Waveset>
```

```
<ImportCommand class='com.waveset.session.SystemConfigurationUpdater'>
<Object>
<Attribute name='serverSettings.default.disableProfiling'>
<Boolean>true</Boolean>
</Attribute>
</Object>
</ImportCommand>

</Waveset>
```

Using the Profiler

This section describes the features of the Profiler graphical user interface, and how to use these features. The information is organized as follows:

- “Specifying the Profiler Options” on page 95
- “Working with the IDM Profiler View” on page 99
- “Working with the Snapshot View” on page 100
- “Using the Pop-Up Menu Options” on page 101
- “Searching a Snapshot” on page 102
- “Saving a Snapshot” on page 102

Specifying the Profiler Options

The Profiler Options dialog consists of the following tabs:

- **Mode**

The Mode tab provides the following options:

- **IDM Objects Only:** Select to profile form, rule, workflow, and XPRESS objects. Excludes Java objects from the profile.
- **Java and IDM Objects:** Select to profile form, Java, rule, workflow, and XPRESS objects.

Note –

- The Java and IDM Objects option is not available if you are using a regular Waveset project with an *external* Identity Manager instance or using a remote Waveset project.
 - You cannot change the Mode option while the Profiler is running. You must stop the Profiler to change the option.
-

- **IDM Object Filters:**

The IDM Object Filters tab provides the following options:

- **Show IDM Object details**
 - Select this box to include every executed form, workflow, and XPRESS element in the snapshot.
 - Clear this box to include only the following elements in the snapshot:
 - <invoke>
 - <new>
 - <Rule>
 - <Form>
 - <WFProcess>
 - <ExScript>
 - <ExDefun>
 - <FieldRef>
 - <Action> (for workflow application callouts)

- **Include Anonymous Sources**

Note – *Anonymous sources* are forms (or portions of a form) that are generated on the fly (such as Login forms and MissingFields forms) and do not correspond to a persistent form that resides in the Waveset repository.

- Select this box to include Anonymous sources in the snapshot.
 - Clear this box to exclude Anonymous sources from the snapshot.
- Java Filters

Select the Java Filters tab to

- Include or exclude Java filters
- Create new filters
- Delete existing filters
- Restore the default filters

Java filters are given in terms of method patterns, and they are expressed in patterns that include or exclude based on *canonical method name*. Where a canonical method name is:

fully-qualified-class-name.method-name(parameter-type-1, parameter-type-2, ...)

Note – For constructors, *method-name* is <init>.

Here are a few examples:

- To exclude all constructors, enable the Exclude box and add the following filter:
 - *.<init>(*)

- To exclude all constructors with a single `org.w3c.dom.Element` parameter, enable the Exclude box and add the following filter:

```
*.<init>(org.w3c.dom.Element)
```

- To exclude all Identity Manager classes, enable the Exclude box and add the following filters:

```
"com.waveset.*"  
"com.sun.idm.*"
```

- To instrument your custom code only, disable the Exclude box, remove the initial `*` include filter, and then add the following filter:

```
"com.yourcompany.*"
```

Note – The last two examples are currently equivalent because the filters are applied only to your custom classes and Waveset classes.

If necessary, you can instrument other jars by modifying the following lines in `build.xml` as appropriate. For example,

```
<instrument todir="${lighthouse-dir-profiler}/WEB-INF"  
verbose="${instrumentor.verbose}" includeMethods="${profiler.includes}"  
excludeMethods="${profiler.excludes}">  
  <fileset dir="${lighthouse-dir}/WEB-INF">  
    <include name="lib/idm*.jar"/>  
    <include name="classes/**/*.*class"/>  
  </fileset>  
</instrument>
```

By default, the configuration includes all your custom classes and most Waveset classes. A number of Waveset classes are forcibly excluded— because enabling them would break the Profiler.

For example, classes from the workflow, forms, and XPRESS engines are excluded or the Profiler would produce an unintelligible snapshot when profiling Java and Waveset objects.

Note that Java filters provide much more filtering granularity than IDM Object Filters. Java instrumentation adds significant overhead to the execution time, which can drastically skew the profiling results. Because Waveset objects are interpreted rather than compiled, the instrumentation overhead is negligible. So for example, there is basically no reason to exclude workflow A and include workflow B, and so forth.

Note – You cannot modify Java filters while the Profiler is running. You must stop the Profiler before changing Java filters.

- Miscellaneous

The Miscellaneous tab provides the following options:

- **Prune snapshot nodes where execution time is 0:**
 - Disable this option (default) if you want the snapshot to include invocation information for all executed entities— even those whose execution time is zero.
It might be useful to have the number of invocations, even for nodes where there is no execution time.
 - Enable this option to remove these nodes, which allows you to focus on the most relevant profiling data. In addition, enabling this option can provide a large savings in Profiler snapshot size.
- **Automatically Open Browser Upon Profiler Start:**
 - Enable this option (default) when you launch the Profiler to automatically open a browser that points to the Identity Manager instance being profiled.
 - Disable this option if you do not want to open a browser.
- **Include Java Sources in Snapshot:**
 - Enable this option (default) to include Java sources for any Java methods referenced by the profiling data in the Snapshot. You should always use this setting for snapshots in the field. Custom Java is relatively small and it is very valuable to have for support.
 - Disable this option only if you are profiling Waveset and have the complete Waveset source available.
In this situation, you do not want to include the Waveset source because it can create extremely large snapshots. (See “[How the Profiler Locates and Manages Source](#)” on [page 91](#) for more information.)

Use the options on these tabs to indicate which objects to profile and which elements to display in the profile.

After specifying the Profiler options, click OK to start the Profiler. Depending on your project configuration, the Profiler does one of two things:

- If you are using a regular Waveset project with an *Embedded* Identity Manager Instance, the Profiler performs a full build, deploys into the NetBean’s application server, and starts the Profiler.
- If you are using a regular Waveset project with an *External* Identity Manager Instance or the remote Waveset project, the Profiler attaches to the Identity Manager instance configured for the project.

Note – You can select IdM → Set Identity Manager Instance to control the Identity Manager Instance action for the project.

Working with the IDM Profiler View

The IDM Profiler view consists of the following areas:

- **Current Project Area:** Consists of a drop-down menu that lists all of your current projects. Use this menu to select the project you want to profile.
- **Controls Area:** Contains four icons, as described in the following table:

| Icon | Name | Purpose |
|---|-------------------------|---|
|  | Start Waveset Profiler | Starts the Profiler and opens the Profiler Options dialog. |
|  | Stop Waveset Profiler | Stops the Profiler. |
|  | Reset Collected Results | Resets all of the profile results you collected to this point. |
|  | Modify Profiling | Reopens the Profiler Options dialog so you can change any of the settings to modify your current profile results. |

- **Status Area:** Reports whether you are connected to the Host and provides Status information as the Profiler is starting up, running, and stopping.
- **Profiling Results Area:** Contains two icons, which are described in the following table:

| Icon | Name | Purpose |
|---|-------------------------|--|
|  | Start Waveset Profiler | Starts the Profiler and opens the Profiler Options dialog. |
|  | Reset Collected Results | Resets all of the profile results you collected to this point. |

- **Saved Snapshots Area:** Provides a list of all saved snapshots.

Note – Instructions for saving snapshots are provided in “Saving a Snapshot” on page 102.

In addition, you can use the following buttons to manage these snapshots:

- **Open:** Click to open saved snapshots in the Snapshot View window.

Tip – You can also double-click a snapshot in the Saved Snapshots list to open that snapshot.

- **Delete:** Select a snapshot in the Saved Snapshots list, and then click this button to delete the selected snapshot.
- **Save As:** Select a snapshot in the list and then click this button to save that snapshot externally to an arbitrary location.
- **Load:** Click to open a snapshot from an arbitrary location into the Snapshot View window.

Working with the Snapshot View

When you open a snapshot, the results display in the Snapshot View window, located on the upper right side of Waveset IDE.

A snapshot provides the following views of your data:

- **Call Tree view:** Consists of a tree table showing the call timing and invocation counts throughout your system.

This tree table contains three columns:

- **Call Tree column:** Lists all nodes.

Top-level nodes are one of the following:

- `Thread.run()` methods for various background threads in the system

For example, if you enabled Java profiling, you will see the `ReconTask.WorkerThread.run()` method.

- Request timings

For example, if you viewed the `idm/login.jsp` URL, you will see a top-level entry for `idm/login.jsp`. The data displayed in the Time column for this entry represents the total time for that request (or requests). The data displayed in the Invocations column represents the total number of invocations to that page. You can then explore further into that data to see what calls contributed to its time.

Note – The Call Tree also contains Self Time nodes. Self Time values represent how much time was spent in the node itself. (For more information, see [“Statistics Caveats” on page 92.](#))

- **Time column:** Lists the time spent in each node when that node was called from its parent. The percentages are given relative to parent time.
- **Invocations column:** Lists how many times each node was invoked from its parent.
- **Hotspots view:** Provides a flattened list of nodes that shows aggregate call timings regardless of parent.

This view contains the following columns:

- **Self Time:** Lists the total amount of time spent in each node.
- **Invocations:** Lists the total number of times each node was invoked from its parent.
- **Time:** Lists the total amount of time spent in each node and in all of its children.
- **Back Traces view:** Provides an inverted call stack showing all the call chains from where each node was called.

You can use these statistics to answer the question— How much time would I save if I eliminated this particular call chain from this node?

You can access the Back Traces view from any of the other snapshot views by right-clicking a node (known as the *root node*) and selecting Show Back Traces from the pop-up menu.

Note –

The Time and Invocations data values mean something different in Back Traces view:

- **Time:** The values in this column represent the time spent in the root node when it is called from a given call chain.
 - **Invocations:** The values in this column represent how many times the root node was invoked from a given call chain.
-

- **Callees view:** Provides an aggregate call tree for a node (known as the *root node*), regardless of its parent chain.

These statistics are helpful if you have a problem area that is called from many places throughout the master call tree and you want to see the overall profile for that node.

You can access the Callees view from any of the other snapshot views by right-clicking a node (known as the *root node*) and selecting Show Callees from the pop-up menu.

Note – The Time and Invocations data values used in Callees view have the same meaning as those used in Call Tree view.

Using the Pop-Up Menu Options

Right-click any node in Call Tree view or in Hotspots view and a pop-up menu displays with the options described the following table:

| Menu Options | Description |
|--------------|---|
| GoTo Source | Select this option to view the XML source for a node that corresponds to a Java method, workflow, form, rule, or XPRESS. For detailed information about this view, see “How the Profiler Locates and Manages Source” on page 91 . |

| Menu Options | Description |
|---------------------------------------|---|
| Show Back Traces | Select this option to access the Back Traces view. For detailed information about this view, see “Working with the Snapshot View” on page 100 . |
| Show Callees | Select this option to access the Callees view. For detailed information about this view, see “Working with the Snapshot View” on page 100 . |
| Find In Hotspots | Select this option to find a node in the Hotspots view. For detailed information about this view, see “Working with the Snapshot View” on page 100 . |
| List Options → Sort → | Select this option to <ul style="list-style-type: none"> ■ None ■ Call Tree ■ Time ■ Invocations ■ Ascending ■ Descending |
| List Options → Change Visible Columns | Select this option to change the columns displayed in the Call Tree or Hotspots list. <p>When the Change Visible Columns dialog displays, you can select one or more of the following options:</p> <ul style="list-style-type: none"> ■ Call Tree: Call Tree ■ Invocations: Invocations ■ Time: Time |

Searching a Snapshot

Use the Search icon , located at the top of the Snapshot View window to search for nodes by name the Call Tree view or Hotspots tree.

Alternatively, right-click any node in Call Tree view or Hotspots view and select Find in Call Tree or Find in Hotspots (respectively) from the pop-up menu to search for a node.

Saving a Snapshot

The Profiler provides several options for saving a snapshot. See the following table for a description of these options:

| Icon | Name | Purpose |
|--|--|---|
|  | Save the Snapshot in the Project icon (located at the top of the Snapshot View window) | Saves the snapshot in the nbproject/private/idm-profiler directory of your project. Snapshots saved in your project are listed in the Saved Snapshots section of the Profiler view. |

| Icon | Name | Purpose |
|---|---|--|
|  | Save the Snapshot Externally icon (located at the top of the Snapshot View window) | Saves a snapshot to an external, arbitrary location. |
|  | Save As button (located in the Saved Snapshots area) | Saves a snapshot to an external, arbitrary location. |

Tutorial: Troubleshooting Performance Problems

Waveset provides a tutorial (`profiler-tutorial.zip`) to help you learn how to use the Profiler to troubleshoot forms, Java rules, workflows, and XPRESS.

Use the following steps to complete the tutorial.

▼ Step 1: Create an Waveset Project

- 1 Select File → New Project.
- 2 When the New Project wizard displays, specify the following, and then click Next:
 - a. In the Categories list, select Web to indicate what type of project you are creating.
 - b. In the Projects list, select Waveset Project.

Note – You must create a regular Waveset project for a fully featured development environment. Do not select the Waveset Project (Remote) option.

- 3 Complete the following fields on the Name and Location panel, and then click Next:
 - **Project Name:** Enter `Idm80` as the project name.
 - **Project Location:** Use the default location or specify a different location.
 - **Project Folder:** Use the default folder or specify a different folder.
- 4 When the Waveset WAR File Location panel displays, enter the location of the Waveset 8.1.1 war file. Typically, unzipping this file creates an `idm.war` file in the same directory.
- 5 Click Next to continue to the Repository Setup panel.

You should not have to change the default settings on this panel, just click Finish. When you see the BUILD SUCCESSFUL message in the Waveset IDE Output window, you can extract the Profiler tutorial files. See [“Step 2: Unzip the Profiler Tutorial” on page 104](#) for instructions.

▼ Step 2: Unzip the Profiler Tutorial

- 1 Unzip `profiler-tutorial.zip` in the project root. The extracted files include:

```
<project root>/custom/WEB-INF/config/ProfilerTutorial1.xml  
<project root>/custom/WEB-INF/config/ProfilerTutorial2.xml  
<project root>/src/org/example/ProfilerTutorialExample.java  
<project root>/PROFILER_TUTORIAL_README.txt
```

- 2 Start the Profiler. Proceed to [“Step 3: Start the Profiler” on page 104](#).

▼ Step 3: Start the Profiler

- 1 Use the instructions provided in [“Before You Begin” on page 93](#) to increase the memory for your server and Netbeans JVM.
- 2 Use any of the methods described in [“Starting the Profiler” on page 94](#) to start the Profiler.
- 3 When the Profiler Options dialog displays, you can specify profiling options.
- 4 Continue to [“Step 4: Set the Profiler Options” on page 104](#)

Note – For detailed information about all of the different Profiler options, see [“Specifying the Profiler Options” on page 95](#).

▼ Step 4: Set the Profiler Options

For the purposes of this tutorial, specify the following Profiler options:

- 1 On the Mode tab, select Java and IDM Objects to profile form, Java, rule, workflow, and XPRESS objects.
- 2 Select the Java Filters tab.
Use the following steps to disable all Waveset Java classes *except* your custom Java classes (in this case, `org.example.ProfilerTutorialExample`):
 - a. Click New and a new, blank field appears at the bottom of the Filter column.
 - b. Enter `com.waveset.*` into the new field, and then select the Exclude box.
 - c. Click New again.
 - d. Enter `com.sun.idm.*` into the new field, and then select the Exclude box.
- 3 Click OK to run the Profiler.

Note – The Profiler takes a few minutes to complete the first time you run it on a project or if you have recently performed a Clean Project action.

When the Profiler finishes processing, you are prompted to Log In.

- 4 Enter the password configurator, select the Remember Password box, and then click OK to continue.**
- 5 When the Waveset window displays, log in.**

Note – Typically, you should log in to Waveset as a different user instead of logging in as configurator again. You are already logged into the Profiler as configurator, and the Waveset session pool only allows one entry per user. Using multiple entries can result in the appearance of a broken session pool and might skew your profiling results for finer-grained performance problems.

However, for this simple example the session pool is of no consequence so you can login as configurator/configurator.

- 6 In Waveset, select Server Tasks → Run Tasks, and then click ProfilerTutorialWorkflow1.**

The tutorial might take a few moments to respond.

- 7 Although you could take a snapshot now; you are going to reset your results instead, run the Profiler, run it again, and then take a snapshot.**

Note – It is a best practice to run the Profiler a couple of times before taking a snapshot to be sure all the caches are primed, all the JSPs are compiled, and so forth.

Running the Profiler several times enables you to focus on actual performance problems. The only exception to this practice is if you are having a problem populating the caches themselves.

- a. Return to the IDM Profiler view in the Waveset IDE. Click the Reset Collected Results icon in the Profiling Results section (or in the Controls section) to reset all of the results collected so far.**
- b. In Waveset, select Server Tasks → Run Tasks again, and click ProfilerTutorialWorkflow1.**
- c. When the Process Diagram displays, return to the Waveset IDE and click Take Snapshot in the Profiling Results section.**

- 8 The Waveset IDE downloads your snapshot and displays the results on the right side of the window.**

This area is the *Call Tree* view. At the top of the Call Tree, you should see a `/idm/task/taskLaunch.jsp` with a time listed in the Time column. The time should indicate that the entire request took six+ seconds.

- 9 Expand the `/idm/task/taskLaunch.jsp` node, and you can see that `ProfilerTutorialWorkflow1` took six seconds.**

- 10 Expand the `ProfilerTutorialWorkflow1` node. Note that `activity2` took four seconds and `activity1` took two seconds.**

- 11 Expand `activity2`.**

Note that `action1` took two seconds and `action2` took two seconds.

- 12 Expand `action1` and note that the `<invoke>` also took two seconds.**

- 13 Double-click the `<invoke>` to open `ProfilerTutorialWorkflow1.xml` and highlight the following line:**

```
<invoke name='example' class='org.example.ProfilerTutorialExample'/>
```

You should see that a call to the `ProfilerTutorialExample` method took two seconds.

Note – You are actually browsing XML source that was captured in the snapshot, rather than source in the project. Snapshots are completely self-contained. (For more information, see [“How the Profiler Locates and Manages Source” on page 91.](#))

- 14 Select the CPU:`<date><time>` tab to return to your snapshot.**

- 15 Expand the `<invoke>` node, and note that the Profiler spent two seconds in the Java `ProfilerTutorialExample.example()` method.**

- 16 Double-click the method name to open the `ProfilerTutorialExample.java` source and highlight the following line:**

```
Thread.sleep(2000);
```

There’s the problem! This method contains a two-second thread sleep.

- 17 If you return to the Call Tree, you can see that all of the two second paths lead to this method. (You should see three paths; for a total of six seconds.)**

- 18 Select the Hotspots tab (located at the bottom of the Call Tree area) to open the Hotspots view. Notice that `ProfilerTutorialExample.example()` has a total self time of six seconds.**

(For more information about Hotspots, see [“Working with the Snapshot View” on page 100.](#))

- 19 **Right-click `ProfilerTutorialExample.example()` and select Show Back Traces from the pop-up menu.**

A new Back Traces tab displays at the bottom of the area.
- 20 **Expand the `ProfilerTutorialExample.example()` node on the Back Traces tab to see that this method was called from three places, and that the method took two seconds when it was called from each place.**

(For more information about Back Traces, see [“Working with the Snapshot View” on page 100.](#))
- 21 **Click the Save the snapshot in the project icon to save your snapshot and close it.**

If you check the Saved Snapshots section on the IDM Profiler tab, you should see your snapshot. (You might have to scroll down.)
- 22 **Select the saved snapshot, and then click Open to re-open it.**

Note – You can use the Save As button to save your snapshots externally and use the Load button to load a snapshot from outside your project.

- 23 **Close the snapshot again.**

▼ **Step 5: Profile a ManualAction Workflow**

The next part of this tutorial illustrates how to profile a workflow `ManualAction`.

- 1 **In Waveset, select Server Tasks → Run Tasks, and then click `ProfilerTutorialWorkflow2`.**

After a few moments, an empty form displays.
- 2 **Click Save and the process diagram displays.**
- 3 **Select Server Tasks → Run Tasks again.**
- 4 **Return to the Waveset IDE IDM Profiler view and click the Reset Collected Results icon in the Profiling Results section.**
- 5 **Now click `ProfilerTutorialWorkflow2` in Waveset.**
- 6 **When the blank form displays again, click Save.**
- 7 **In the IDM Profiler view, click Take Snapshot.**

After a few seconds, a snapshot should display in the Call Tree area. You should see that `/idm/task/workItemEdit.jsp` took six+ seconds. (This result corresponds to the manual action in the workflow.)

- 8 **Expand the `/idm/task/workItemEdit.jsp` node and note that running all Derivations in the ManualAction form took a total of six seconds.**
- 9 **Expand the `Derivation, displayNameForm, variables.dummy`, and `<block>` nodes.**

You should see that the `<block>` took six seconds and, of that time, the Profiler spent two seconds in each of the three invokes to the `ProfilerTutorialExample.example()` method.
- 10 **You can double-click `<block>` to view the source.**

Working With Other Debugging Tools

You can use the following Oracle and third-party tools to identify potential performance bottlenecks:

- “Using DTrace” on page 108
- “Using JMX” on page 110
- “Using JConsole” on page 114
- “Using JRad” on page 115

These tools can be particularly useful if your deployment uses custom Java classes.

Using DTrace

The DTrace facility is a dynamic tracing framework for the Solaris 10 operating system that enables you to monitor JVM activity.

DTrace contains more than 30,000 probes and uses integrated user-level and kernel-level tracing to give you a view into your production system. You can also trace arbitrary data and expressions by using the D language, which is similar to C or awk. The DTrace facility also includes special support for monitoring the JVM, and enables you to watch your whole system and span outside the JVM.

DTrace is easiest to use with Java 6 because probes are built into the JVM. The facility also works with Java 1.4 and Java 5, but you must download JVM PI or JVM TI agents from the following URL:

<https://solaris10-dtrace-vm-agents.dev.java.net/>

The following example shows how to write a DTrace script.

EXAMPLE 4-3 Example DTrace Script

```
#!/usr/sbin/dtrace -Zs
#pragma D option quiet
hotspot$I:::
{
```

EXAMPLE 4-3 Example DTrace Script (Continued)

```
printf("%s\n", probename);
}
```

In this example, you would replace *\$1* with the first argument to the script, which is the PID of the Java process you want to monitor. For example:

```
# ./all-jvm-probes.d 1234
```

The following table describes the commands you can use to enable different DTrace probes.

TABLE 4-3 DTrace Commands

| Command | Description |
|--|---|
| -XX:+DTraceMonitorProbes | Enables JVM support in Java 6 (patches for Java 1.4 and Java 5) |
| -XX:+ExtendedDTraceProbes | Provides the following information: <ul style="list-style-type: none"> ■ JVM startup (begin and end) and shutdown ■ Thread starting and stopping ■ Class loading and unloading ■ Garbage collection (several options available) ■ JIT compilation begin and end ■ Compiled method loading and unloading ■ Monitor contention, wait, and notify ■ Method entry, method return, and object allocation |
| /usr/sbin/dtrace -n 'hotspot*:::' | Enables all JVM probes for all Java processes on the system |
| /usr/sbin/dtrace -n 'hotspot1234:::' | Enables all JVM probes for only the Java process with PID <i>1234</i> |
| /usr/sbin/dtrace -n 'hotspot1234:::gc-begin' | Enables only the probe that starts when garbage collection for process <i>1234</i> begins |

Note – Because DTrace causes additional work in the system, enabling this facility affects system performance. The effect is often negligible, but can become substantial if you enable many probes with costly enablings.

Instructions for minimizing the performance effect of DTrace are provided in the “Performance Considerations” chapter of the *Solaris Dynamic Tracing Guide*.

For more information about DTrace, see `/usr/demo/dtrace` and `man dtrace`.

Using JMX

Waveset enables you to use Java Management Extensions (JMX) to capture and expose operational statistics for certain resource adapter operations. You can use this data for diagnostic and predictive purposes, such as to monitor system health and reports.

This statistical data includes the following:

- The number of times the action was performed
- The minimum, maximum, and average duration of the operations

| Objects | Actions Monitored |
|-------------------|---|
| For Accounts | <ul style="list-style-type: none"> ■ Create ■ Update ■ Delete ■ Get ■ Authenticate |
| For Actions | Run |
| For Other Objects | <ul style="list-style-type: none"> ■ Create ■ Update ■ Delete ■ Get ■ List |

JMX creates MBeans for each resource adapter, by server, and registers these beans with names that match the following pattern:

```
serverName=server name, resourceAdapterType=Resource Adapter Type,
resourceAdapterName=Resource Adapter Name
```

Waveset records statistics for all completed operations, whether they completed successfully or with errors. However, Waveset does not record statistics for incomplete operations, such as any operations that throw exceptions.

You can configure excludes as follows:

1. From the Administrator interface, select Configure → Servers.
2. On the Configure Servers page, perform one of the following tasks:
 - Click the Edit Default Server Settings button to edit the default server settings.
 - Click a server link to edit the policy for that server.
3. Click the JMX tab and enable the JMX Enable Resource Adapter Monitor box to turn on resource monitoring.

- To exclude specific resources, add regular expressions to the JMX Resource Adapter Monitor Excludes list.
- To exclude monitoring specific actions, add regular expressions to the JMX Resource Adapter Monitor Operation Excludes list.

All excludes use regular expressions. For excluding certain resources, JMX just matches on the resource name. For example, if you have adapters named

```
resource1
resource2
resource3
resource10
resource11
```

and you specify the following pattern

```
.*1$
```

which means, match any 0 or more of any character (.*) until something that ends with a 1 (1\$). JMX will exclude resource1 and resource11.

For operations, the process is similar. If your operations have the following names, the patterns must match against those names.

```
ACCOUNT_CREATE
ACCOUNT_UPDATE
ACCOUNT_DELETE
ACCOUNT_GET
ACCOUNT_AUTHENTICATE
OBJECT_CREATE
OBJECT_UPDATE
OBJECT_DELETE
OBJECT_GET
OBJECT_LIST
ACTION_RUN
```

For example, the ^ACCOUNT.* pattern excludes all operations that start with ACCOUNT. Or, using this pattern excludes updates and deletes:

```
.*UPDATE$
.*DELETE$
```

Note – For more information about configuring and using JMX, see “[Configuring JMX Monitoring](#)” on page 19 and “[The JMX Publisher Type](#)” in *Oracle Waveset 8.1.1 Business Administrator’s Guide*.

Waveset supplies some JMX MBeans that provide information about the following:

- Data Exporter
- Object Repository
- Performance
- Reconciliation
- Resource Adapters and Connectors
- Scheduler
- Waveset Server Cluster

There are nine MBeans in the Performance category:

- DataExporter
- DataQueue
- FormConverter
- ObjectChangeNotification
- Reconcile
- Ruler
- TaskInstanceCache
- ViewMaster
- WorkItemCache

These MBeans can capture the same performance data that is captured by the Waveset Profiler, but the MBeans have a lower runtime performance cost.

- You enable the Profiler on a per-server basis. When enabled, the Profiler impacts both memory (to capture and hold the performance data) and performance.
- Performance MBeans are disabled by default. When enabled, MBeans have a significantly lower operational impact, however they capture very specific performance data.

When enabled, the FormConverter and ViewMaster MBeans each have a configurable threshold and when processing takes longer than the threshold, a JMX Notification is produced. This notification indicates which view or form element was involved and how long it took to process that element. If there are no JMX notification listeners registered with the MBean server the notification is discarded, otherwise Waveset delivers the notification to the listener.

These MBeans are useful for tracking down performance problems with Waveset's GUI. Because you can customize Waveset's GUI, and rendering parts of the GUI can involve large amounts of data or significant computations. The FormConverter and ViewMaster MBeans can help you quickly identify whether the performance problem is caused by view processing or by processing a specific form field.

To use the FormConverter or ViewMaster MBeans, perform the following steps from a JMX console:

1. Enable the MBean by setting the `Enabled` attribute to **true**.
2. Specify an appropriate value (in mSecs) for the `Limit` attribute.

3. Subscribe to the MBean for notifications by using the Notifications tab.

After completing these steps, any View/Form processing that takes longer than the configured limit will cause a JMX notification to display in the JMX console. The notification will specify the Form, Field, or View ID and how long the operation took.

If the `ViewMaster` MBean indicates delays in processing a view, check these additional items:

- For delays caused by `runDerivations` or `runExpansions`, the problem may still be in the form associated with the view.
- For delays caused by `getView`, `checkoutView`, `checkinView`, `refreshView`, or `getForm`, the problem is probably in the viewer. You must turn on tracing in the viewer to further isolate the problem.

The `FormConverter` MBean shows processing delays for any field that is rendered to HTML during form processing. Only fields that have a `<Display>` element are candidates. Common reasons for fields taking a long time to process are:

- The field has a lot of data
- The field executes XPRESS or script code to compute its value

Note – The `FormConverter` MBean indicates which Form/Field and how long it took to render. If a form has a lot of fields, the form might display slowly but no single field will exceed the limit.

The `Rule` MBean emits a notification any time the execution of a rule exceeds the configured limit. Rules can be executed in many places (in tasks, forms, or workflows), so having a single place to capture any rule execution that exceeds a specified time is useful for a high-level performance analysis.

The `TaskInstanceCache` and `WorkItemCache` MBeans show which cache operations are used when workflows contain `ManualActions` that are marked as *transient*. When a workflow contains a transient `ManualAction`, changes to the corresponding `WorkItem` and workflow are made in memory, bypassing the repository but removing the assurance that a workflow will survive a server restart. These MBeans are useful when diagnosing self-service workflow wizard performance.

The `ObjectChangeNotification` MBean issues a JMX notification any time the server exceeds the `execLimit` value by delivering a notification to Identity Manager code indicating that an object has changed. This MBean provides a useful diagnostic when the server appears to be processing tasks too slowly. If this MBean is producing JMX notifications with an `execLimit` of 50 milliseconds, the server is running slowly and you should capture both the JMX notification and the JVM thread dumps for analysis.

The `Reconcile` MBean shows how much data `Reconcile` has already processed, how much data is still queued, and the current processing rate. This bean is useful when measuring the impact

of changing Reconcile processing thread counts. You can use this MBean with resource-specific MBeans to assess Reconciliation tunings. If the `Reconcile.processRate` is low and the resource account get is high, you can add more Reconciliation worker threads to increase throughput.

The `DataExporter` and `DataQueue` MBeans indicate both the size of the internally queued data and the rate at which the queue is being filled. The `Data Exporter Queue` provides an in-memory queue to buffer data that needs to be written to the repository, allowing separate threads to drain the queue without blocking the code that queued the data.

When looking at notifications using `JConsole`, you can hover the mouse pointer over the message field in the notification to see the details (such as `viewId`, `form`, `field`, and so on).

Using JConsole

The Java Monitoring and Management Console (*JConsole*) is a Java Management Extension (JMX) technology-compliant graphical management tool that is co-packaged with at least JDK 5. `JConsole` connects to a running JVM and gathers information from the JVM MBeans in the connected JMX agent.

Specifically, you can use `JConsole` to perform the following tasks:

- Detect low memory and deadlocks
 - `JConsole` accesses the memory system, memory pools, and MBeans garbage collector to provide information about memory use, including memory consumption, memory pools, and garbage collection statistics.
- Enable or disable garbage collection
- Enable or disable verbose tracing
- Monitor local and remote applications
- Monitor and manage MBeans including current heap memory use, non-heap memory use, and how many objects are pending for finalization
- View information about performance, resource consumption, and server statistics
- View summary information about the JVM and monitored values, threads running on the application, and loaded classes
- View information about operating system resources (Waveset's platform extension), such as:
 - CPU process time
 - How much total and free physical memory is available
 - The amount of committed virtual memory (how much virtual memory is guaranteed to be available to the running process)
 - How much total and free swap space is available
 - The number of open file descriptions (UNIXonly)

Note – For more information about using JConsole to monitor applications on the Java platform, see the Oracle Developer Network (SDN) article titled *Using JConsole to Monitor Applications*, which is available from the following URL:

<http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html>

Using JRat

You can use the Java Runtime Analysis Toolkit (JLat), an open-source performance profiler for the Java platform, to identify potential performance bottlenecks, especially if your deployment uses custom Java classes. JRat monitors your application's execution and persists the application's performance measurements.

For example, if you have a custom workflow for provisioning, you can use JRat to see which classes are being invoked and how much time is required to run your workflow compared to the default Waveset provisioning workflow.

For more information about JRat, see <http://jrat.sourceforge.net>.

Tracing and Troubleshooting

This chapter explains how you can use tracing to help fix errors, solve performance issues, and understand the flow within Waveset. This chapter also describes methods for troubleshooting problems that might occur with different Waveset components.

This chapter covers the following topics:

- “Before You Begin Tracing or Troubleshooting” on page 117
- “Tracing Waveset Objects and Activities” on page 127
- “Tracing the Waveset Gateway Objects and Activities” on page 160
- “Troubleshooting and Fixing Common Problems” on page 166

Before You Begin Tracing or Troubleshooting

Review the information in the following sections before tracing or troubleshooting Waveset:

- “Intended Audience” on page 117
- “Important Notes About Tracing and Troubleshooting Waveset” on page 118
- “Before Calling Support” on page 118
- “Related Documentation and Web Sites” on page 125
- “Process Overview” on page 126

Intended Audience

This chapter is intended for application server and database administrators, front-line support engineers, and partners who are responsible for maintaining Waveset in a deployment environment.

Before troubleshooting problems with Waveset, you must

- Be familiar with Java 5.0 (required for Oracle Waveset 8.1.1)
- Understand the components that you are trying to troubleshoot

Important Notes About Tracing and Troubleshooting Waveset

Before you start tracing or troubleshooting Waveset, note the following:

- Tracing affects system performance. To ensure optimal performance, use the minimum tracing level needed or turn off tracing after debugging the systems.
- Do not enable tracing for the `com.waveset` class. The `com.waveset` class is verbose and has many classes, so tracing this class might cause your server to hang.
- Do not configure Waveset to trace at the advanced level unless instructed to do so by Oracle Support.
- If you set `exception.trace` in the `Waveset.properties` file to debug a specific problem, do not use it for an extended period of time and be sure you disable it after debugging. Setting `exception.trace` in the `Waveset.properties` file can significantly affect system performance.
- By default, Waveset returns stack trace information as HTML comments when certain errors occur. Generally, a stack trace is not visible to the end-user, but the trace can be seen by viewing the page source in the browser.

Stack trace information can be very useful when diagnosing a problem, but exposes information about the execution of the web application that could be considered a security risk. Production deployments should consider turning off stack trace information rendering by configuring the following setting in `Waveset.properties`:

```
ui.web.disableStackTraceComments=true
```

You must perform this change manually on each Waveset server.

Note – Beginning with Waveset *version 8.1.1*, the default setting for `ui.web.disableStackTraceComments` is **true**. In previous releases, the default setting for `ui.web.disableStackTraceComments` was **false**.

However, because version 8.1.1 is an upgrade and not an install, the new setting does not automatically take place because `Waveset.properties` is not overridden during an upgrade.

Before Calling Support

Before calling Oracle Technical Support or Sustaining for assistance, you must prepare for the call by collecting as much information as possible about your environment and the problem. Instructions are provided in the following sections:

- [“Identify the Problem Area” on page 119](#)
- [“Gather Required Information” on page 119](#)

- “Gather Additional Information for Specific Problems” on page 120

Note – Unless specifically noted otherwise, the commands described in the following sections will work from a properly sourced terminal on UNIX or Windows.

Identify the Problem Area

Try using one or more of the following methods to narrow the problem area:

- Use web browser search tools to research the problem.
- Use resource-specific support options when available. Your problem might be related to a resource and fixed by using a resource patch.
- Remove Waveset from the equation by using an appropriate client tool when possible. For example, by using a Java LDAP browser.

Gather Required Information

You must have the following information about your environment and Waveset installation available for all calls to your Support or Sustaining representative.

| Information to Collect | How to Get this Information |
|--|---|
| Product version, including <ul style="list-style-type: none"> ▪ A list of all installed patches, hotfixes, and e-fixes ▪ A list of customizations | Use the following commands from the WavesetConsole: <ul style="list-style-type: none"> ▪ <code>installed</code> ▪ <code>inventory</code> |
| Waveset topography, including: <ul style="list-style-type: none"> ▪ How your Waveset cluster is configured ▪ Environment localization of the issue and any extra information about your environment ▪ A list of servers | The information <ul style="list-style-type: none"> ▪ Must be manually determined ▪ Must be manually determined ▪ Can be obtained by selecting <code>Configure</code> → <code>Servers</code> from the Waveset Administrator interface on a running Waveset server |
| Recent changes to your environment | Must be manually determined |
| Java version and type | Use the following Java command: <pre>java -version</pre> |
| Application server version and type | Depends on the Application Server being used, but must be manually determined |
| Operating system level and information | Must be manually determined |
| Default XML output, which is a list of all repository objects | Use the <code>export default.xml default</code> command from the Console |

| Information to Collect | How to Get this Information |
|---|--|
| Task instance data, including <ul style="list-style-type: none"> ▪ A list of all running tasks ▪ The size of all current task instances | Use the following commands from the Console: <ul style="list-style-type: none"> ▪ <code>listTasks</code> ▪ <code>showSizes TaskInstance</code> |
| System Logs | Note – You might be asked to provide additional system logs based on the type of issue you are reporting. |

Gather Additional Information for Specific Problems

If you are experiencing issues related to the following, specific problem areas, you must gather some additional information. Be sure to read the pertinent sections before calling Support.

- [“Installation Problems” on page 120](#)
- [“Hung Systems” on page 121](#)
- [“Performance Issues” on page 121](#)
- [“System Crashes” on page 124](#)

Installation Problems

If you are having problems with your Waveset installation, you must have the following information available for Support:

- **Information about your repository configuration**
You can type the `lh setRepo -c -v` command in a terminal window to view detailed information about your repository configuration.
- **Database information**
You should be able to answer the following questions about your database:
 - What type of database is installed?
 - What database version, including any patches, are you using?
 - What JDBC driver is being used to connect?

Note – If you do not know this information, consult your database administrator.

Note –

For *upgrade issues*, Support asks for this information on a case-by-case basis:

- **Current level.** What version of Waveset are you upgrading from?
 - **Target level.** What version of Waveset are you upgrading to?
-

Hung Systems

If your system is hung, try to collect following information:

- **Thread dump.** A thread dump creates a stack trace for every running thread. Support can use thread dumps to determine the state of the system.

To collect a thread dump, send a SIGQUIT signal to the Java process running Waveset, as follows:

- **In UNIX:** Type `kill -3 pid`.

Where *pid* is the process ID of the JVM running Waveset (usually the same as Application Server).

- **In Windows:** Press CTRL-BREAK in the JVM console.

- **Heap dump.** A heap dump contains the entire status of what is in memory for a Java process.

To force a heap dump, type the `jmap -dump:file=filenamepid` command.

Where:

- *filename* is the heap dump file to be created. Be aware that this file can be very large.
- *pid* is the process ID of the JVM running Waveset (usually the same as Application Server).
- **Screen shots or saved HTML pages.** Use these tools to preserve information about the last action you performed before the system hung. You might capture information about the state of the system that is worth noting and saving for review.
- **Recent activity report.** If you can restart Waveset, you can retrieve an activity report that might help you determine what was happening in the system before the hang occurred. Select the Reports tab in Waveset; and generate a Recent Systems Messages report and a Today's Activities report.

Performance Issues

Note – Instructions for accessing the Waveset debug pages mentioned in this section are described in “[Accessing the Debug Pages](#)” on page 84.

If you are having performance problems, try gathering the following information for Support:

- **Call Timer data.** Waveset's Call Timer debug page (`debug/callTimer.jsp`) enables you to collect call timer statistics for different methods, which can be useful performance data.

Use the following steps to collect the most-relevant data:

1. Open the Clear Timing Statistics page and click Start Timing and Tracing to enable tracing and timing.

Note – Waveset can only collect call timing statistics when tracing is enabled.

2. Return to Waveset and perform the action that is causing performance issues.
3. Refresh the Call Timer debug page.
4. Verify that you captured performance data.

The Show Timings table should list the methods for which statistics are available and the methods' aggregate call timer statistics. (These statistics are *not* broken down by caller).

5. Click the Export button to export the data to a `calltimer.xml` file.
6. Save `calltimer.xml` and deliver the file to Support for analysis.

Note – For more information about the Call Timer debug page, see [“Control Timings \(callTimer.jsp\)” on page 85](#).

- **Java Profiler data.** Java profilers are third-party applications that attach to running Java processes and track all operations.

Support might ask you to run a memory profiler if you are having extreme or low-level performance issues in Waveset or in your custom code. Running Java profilers can cause additional performance degradation, but they provide useful insight about what is happening at a low level. If running a profiler is necessary, your Support representative will give you detailed instructions.

- **Main/Run Loop Tracing data.** You can use Waveset Trace to capture the state of your system but to be useful, the trace must be targeted where the problem occurs. Identifying this location can be difficult, so consult your Support team if you need help.

For common performance issues, it's likely that Support already knows which area is affected. For example, following are several major Waveset systems and which areas to trace:

- **Scheduler**
 - `com.waveset.task.Scheduler`
 - `com.waveset.task.TaskManager`
- **Caching**
 - `com.waveset.server.AdminCache`
 - `com.waveset.server.ListCache`
 - `com.waveset.server.SubjectAuthCache`
- **Reconciler**

- `com.waveset.recon.ReconResults`
 - `com.waveset.recon.ReconTask`
- **Process Level Profiling data.** You must sometimes gather profiling data using native tools on the server or resource having issues.
 - **For UNIX:** Invoke the `top` command from the command line to view a list of all processes with their CPU and memory consumption.
 - **For Windows:** Type Control—Alt—Del, click Task Manager to open the Windows Task Manager dialog. Select the Processes tab to view a list of all processes with their CPU and memory consumption. Alternatively, you can launch the Windows Task Manager by right-clicking the Task Bar and selecting Task Manager.
- **Query Timer data.** This utility facilitates connecting to Waveset's back-end repository and allows you to issue JDBC SQL statements directly to the database. As the name suggests, this utility times statements issued to the repository SQL database.

When you have performance issues, the Query Timer can check the performance of specific SQL statements or check connectivity using the JDBC connection that Waveset uses from a command line. The Query Timer uses the JDBC driver, user, and password configured for Waveset.

Note – The Query Timer utility has been hard coded to use the default Configurator authentication. If you modified Configurator, you will not be able to run the Query Timer in your environment.

You invoke the Query Timer from the command line, by using a `$WSHOME/bin/lh` script. The following example is for a UNIX system. If necessary, modify the script appropriately for Windows.

```
sh ${WSHOME}/bin/lh com.waveset.repository.QueryTimer [ sql statement ]
```

Issue this script with no options to perform a query that summarizes the statistics of each object type in the repository. You can use this query to determine the size of the objects in the database.

Note –

If you are running Waveset 7.0, this script outputs additional information for every 1000th object of each object type in the repository, such as:

- Type
- numObjects
- numAttrVals
- objectBytes
- attrBytes
- objIdxBytes

- `attrIdxBytes`
-

- **Show Timings data.** The Method Timings page (`debug/Show_Timings.jsp`) enables you to determine at a glance which processes are being run most frequently. You might use this page instead of the Call Timings page if you are only interested in a quick view of the processes being run, and where an in-depth investigation is not likely to be necessary. For the best results, click the Clear ALL button and then perform the action that is causing poor performance. After checking the page results, save the HTML page from the JSP and send the page to Support.

Note – See “[Method Timings \(Show_Timings.jsp\)](#)” on page 86 for more information.

System Crashes

If you are reporting a system crash, try to collect the following information for Support:

- **Stack traces.** When Java crashes, a stack trace is sometimes printed to Standard Error. If a stack trace is available, forward the trace to Oracle Support.
- **Java core files.** If the JVM dumps a core file, the file might be useful to Oracle Support or Sustaining. Although core files are large, collect and store them in a safe place so you can deliver them to Support upon request.
- **Memory issues.** If your system is running out of memory, try to gather the following information for Support before your program crashes:
 - **Heap dump.** You can automate heap dump generation by adding the `-XX:+HeapDumpOnOutOfMemoryError` option to the application server's JVM options. You can also force a heap dump on a running process as described in “[Hung Systems](#)” on page 121.
 - **Histogram.** A histogram is a list of class objects that are occupying memory for the Java application.

You must collect histograms at set intervals over a period of time to track memory growth over time. For example, starting a reconcile consumes all system memory in ten hours, so collecting a histogram every hour results in ten histograms, or ten points of data from which to track memory growth. Save the output from each histogram to a different file, with an appropriate timestamp, and deliver this information to Support.

To collect histograms for Support, type `jmap -histo pid` in a terminal window. Where *pid* is the Process ID of the JVM running Waveset (usually the same as application server).
- **Exception tracing.** Support can frequently use an exception trace to determine the cause of a fatal error. However, Waveset's exception tracing is disabled by default. To collect a useful exception trace for Support, you must enable exception tracing and then coerce Waveset into crashing again in the same way.

To enable exception tracing, use the instructions in “Tracing Exceptions” on page 142.

Related Documentation and Web Sites

In addition to the information provided in this chapter, consult the documents and web sites listed in this section for information related to tracing and troubleshooting Waveset.

Recommended Reading

See the following documents for information related to tracing and troubleshooting Waveset:

- “Testing Your Customized Form” in *Oracle Waveset 8.1.1 Deployment Reference* for the recommended method of tracing XPRESS functions.
- The article titled *Using JConsole to Monitor Applications* for information about using JConsole to monitor applications that run on the Java platform. This article is available from the following URL, <http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html>.

Useful Web Sites

The following table describes web sites related to troubleshooting Waveset.

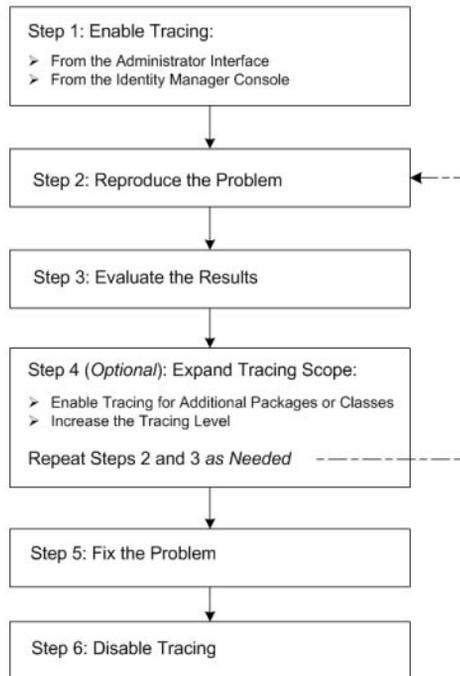
TABLE 5-1 Useful Web Sites

| Web Site URL | Description |
|---|--|
| http://sunsolve.sun.com/ | Oracle web site containing diagnostic tools, forums, features and articles, security information, and patch contents. Note: The information on this site is divided into three areas: <ul style="list-style-type: none"> ▪ Internal. Available only to Oracle employees ▪ Contract. Available only to customers with contract access ▪ Public. Available to everyone |
| http://sunsolve.sun.com | Oracle Technical Support web site. |
| https://sharespace.sun.com/gm/folder-1.11.60181? | Waveset folder on Oracle’ Share Space, which contains Waveset’s FAQ, links to forums, features and articles, and more. Note: This space is only available to Oracle employees and Oracle partners. In addition, these individuals must have a Share Space ID to access the information on this site. |
| https://identitymanageride.dev.java.net | Open source Oracle Waveset Integrated Development Environment (Waveset IDE) project. Includes instructions for installing and configuring Waveset IDE. |

Process Overview

Generally, you can identify and resolve problems in your deployment if you follow the steps shown in the following illustration.

FIGURE 5-1 Tracing and Troubleshooting a Problem



See [“Tracing Waveset Objects and Activities”](#) on page 127 for information about how to enable tracing for different Waveset objects and activities.

See [“Troubleshooting and Fixing Common Problems”](#) on page 166 for information about how to troubleshoot problems that commonly occur for those objects and activities.

Tracing Waveset Objects and Activities

Trace output can be very helpful when you are trying to identify and resolve problems, or when you are developing custom resource adapters.

This section describes how to enable tracing for a variety of Waveset objects and activities. The information is organized as follows:

- [“How To Configure Tracing” on page 127](#)
- [“How to View Trace Files” on page 132](#)
- [“Tracing the Waveset Server” on page 133](#)
- [“Tracing Adapters” on page 133](#)
- [“Tracing Custom Code” on page 142](#)
- [“Tracing Exceptions” on page 142](#)
- [“Tracing Forms” on page 143](#)
- [“Tracing Global XPRESS” on page 144](#)
- [“Tracing PasswordSync” on page 145](#)
- [“Tracing Oracle Waveset Service Provider Delegated Administration” on page 149](#)
- [“Tracing Reconciliation” on page 150](#)
- [“Tracing the setRepo Command” on page 152](#)
- [“Tracing SPML” on page 152](#)
- [“Tracing the Task Scheduler” on page 156](#)
- [“Tracing Workflows” on page 157](#)
- [“Locating Version Information” on page 159](#)

Note – Tracing affects system performance. To ensure optimal performance, specify the minimum tracing level or turn tracing off after debugging the system.

How To Configure Tracing

This section describes how to enable and configure tracing from several locations within Waveset. Instructions are organized as follows:

- [“To Enable Tracing from the System Settings Page” on page 128](#)
- [“To Edit the Default Configuration Object” on page 128](#)
- [“To Create a New Trace Configuration Object” on page 131](#)
- [“To Configure Tracing From Individual Debug Pages” on page 131](#)
- [“To Enable Tracing From the Waveset Console” on page 132](#)
- [“To View Sensitive Data in the Trace Output” on page 132](#)

▼ To Enable Tracing from the System Settings Page

The System Settings page is the primary Waveset Debug page.

From this page, you can

- View and edit objects in the repository
- Clear caches
- Set specialized traces
- Reload the `Waveset.properties` file

Use the following steps to enable tracing from the Waveset System Settings page:

- 1 Open a browser and log in to the Waveset Administrative interface.**
- 2 Type the following URL:**
`http://hostport/idm/debug`
where:
 - *host* is the local server on which you are running Waveset.
 - *port* is the number of the TCP port on which the server is listening.
- 3 When the System Settings page displays, click Show Trace (located near the bottom of the page) to manage a single trace configuration, where you can create, modify, or delete up to ten trace settings.**

Note – The remaining instructions assume you have only one trace configuration.

To manage multiple trace configurations, click the Show Trace List button instead. When the Trace Configuration page displays, click the configuration name to edit the current settings.

Waveset supplies one configuration, called *Global*, by default. However, if you have multiple servers in an Waveset instance, defining different configurations for particular servers might be useful. If the name of a trace configuration matches the name of the current host, the host configuration overrides the Global configuration.

Next Steps After you configure tracing, you can view and edit the default global trace configuration object or create a new configuration object as described in the following sections:

- [“To Edit the Default Configuration Object” on page 128](#)
- [“To Create a New Trace Configuration Object” on page 131](#)

▼ To Edit the Default Configuration Object

- 1 From the Edit Trace Configuration page, click the Trace Enabled box to enable tracing.**

Note – Deselecting this box stops tracing, but keeps your configuration. You can turn tracing on and off without having to remember and retype the classes you were tracing.

2 Specify which classes, packages, or methods to trace by typing their names into the table.

For example:

- To trace all classes in the `waveset.repository` package, enter `com.waveset.repository`.
- To trace the `AbstractDataStore` class in the `waveset.repository` package, enter `com.waveset.repository.AbstractDataStore`.
- To trace the `list` method in the `AbstractDataStore` class in the `waveset` package, enter `com.waveset.repository.AbstractDataStore#list`.

Do not enable trace for the `com.waveset` class. The `com.waveset` class is verbose and has many classes, so tracing this class might cause your server to hang.

3 In the same table, choose a Method/Class tracing level from the Level menu.

Each level captures different types of information, as described in the following table:

| Trace Level | Description |
|-------------|--|
| 0 | Minimum debugging output, traces exceptions and error information only |
| 1 | Trace Level 0 events, plus entries and exits for public methods |
| 2 | Trace Level 1 events, plus entries and exits for non-public methods |
| 3 | Trace Level 2 events, plus decision points and significant variables |
| 4 | Maximum debugging output |

Note – Method/Class tracing produces a predictable, but possibly very large volume of trace output. Try to be as specific as possible when specifying methods and classes to trace.

4 (Optional) To enable subcall tracing, choose a level from the Subcall Trace menu. This menu uses the same trace numbering scale described in the previous table.

Note –

- The default Subcall Tracing level is *none*, which disables subcall tracing on a per-method or per-class basis.
 - Subcall Tracing levels are independent of the Method/Class tracing levels you specified in the previous step.
-

When you enable Subcall Tracing for a particular method that supports subcall tracing, you are automatically setting the tracing level for methods that are *called by* this method. Subcall Tracing enables you to produce a short, but detailed burst of trace output that is triggered by the entry into a specified method and terminated by the method's exit.

For example, assume you created a trace configuration setting for the `com.waveset.adapter.NewRes#init` method, set Method/Class tracing to level one and set Subcall tracing to level three.

Also, assume that the `init` method calls two other methods:

- `NewRes#subcallA`
- `NewRes#subcallB`

When the `init` method runs, the `com.waveset.adapter.NewRes#init` method produces trace output at level one until reaching `subcallA`. When `subcallA` begins executing, the trace level changes to three and continues at that level until `subcallA` exits. The `com.waveset.adapter.NewRes#init` method returns to the `init` method and restores the trace level to one. Later, when `init` calls `subcallB`, there is another burst of level three trace detail that lasts until `subcallB` exits. Finally, when `init` exits, level one tracing stops.

5 Send the trace results to a specified file location or to `stdout`.

If you choose output to a file, the Trace File field displays. Use this field to specify an alternate location and file name for trace output files. The default location and file name is

`path_to_idm_install\export\pipeline\config\WSTrace.log`

6 Specify the maximum number of trace files to store (default is 2).**7 Specify the maximum size for each file (default is 512K).****8 Specify whether to write the trace output file as generated (synchronously) or to queue the data and then write it to the trace file (asynchronously).****9 Save your changes.**

▼ To Create a New Trace Configuration Object

Use the following steps to create a new trace configuration object:

1 Decide which package or method you want to trace.

Generally, you specify a resource adapter name or use information that was revealed in an error message.

2 Log in to the Waveset Administrator interface and open the System Settings page as described in [To Enable Tracing from the System Settings Page](#).

3 On the Systems Setting page, click Show Trace List.

4 When the Waveset Trace Configuration page displays, click New.

5 Enable tracing from the Edit Trace Configuration page.

Choose one of the following options from the Trace Configuration menu:

- **Global.** Select to enable tracing for all servers.
- **Server_name.** Select a server name to enable tracing for a particular server.

6 Select the Trace Enabled box to enable tracing for this object and configure the remaining parameters on this page as described in [To Edit the Default Configuration Object](#).

7 Save your changes.

▼ To Configure Tracing From Individual Debug Pages

This section describes how to enable tracing from an Waveset Debug page.

1 Open a browser and log in to the Waveset Administrative interface.

2 Type the following URL:

```
http://host:port/idm/debug/pageName.jsp
```

where:

- *host* is the local server on which you are running Waveset.
- *port* is the number of the TCP port on which the server is listening.
- *pageName.jsp* is the particular Debug page you want to open.

For example, to trace adapter classes and methods for a custom adapter, you can open the Edit Trace Configuration page by typing the following URL:

```
http://host:port/idm/debug/Show_Trace.jsp
```

▼ To Enable Tracing From the Waveset Console

This section explains how to enable tracing from the Waveset console.

1 Set `$SHOME`.

For example, to set the variable to the default installation directory:

```
set WSHOME=C:\Program Files\tomcat\webapps\idm
```

2 Type `lh console` to open the Waveset console from the `bin` directory.

3 From the console, type `trace` to see a detailed summary of available trace options, including `enable` and `disable`.

The syntax for this command is:

```
trace [ -s server ] $subcommand
```

▼ To View Sensitive Data in the Trace Output

By default, Waveset “sanitizes” trace information by replacing each occurrence of *sensitive data* with a single asterisk. Sensitive data includes any data that is inbound to encryption methods, outbound from decryption methods, and cipher-related metadata such as data lengths, padding lengths, and so forth. However, you can configure a property in the `Waveset.properties` file to include this information in your trace output for debugging purposes.

Use the following steps to enable sensitive cipher data in your trace logs:

1 From a command line, open the `Waveset.properties` file.

2 Locate the `cipher.data.trace` property file comments, and uncomment the following line:

```
cipher.data.trace=true
```

3 Restart your server for this change to take effect.

How to View Trace Files

By default, Waveset sends tracing information to a file named `WSTrace#.log` that is stored in the `path_to_idm_install\export\pipeline\config\` directory. If necessary, you can specify alternate file names and locations when you configure tracing for an object.

Each log file is numbered sequentially, up to the maximum number of files specified on the Edit Trace Configuration page. For example, if you specify a maximum of three files, the files are named `WSTrace1.log`, `WSTrace2.log`, and `WSTrace3.log`.

Use one of the following methods to view these log files:

- If you are sending trace information to a file, open the trace file from the specified location. For example:

```
path_to_idm_install \export\pipeline\config\WSTrace2.log
```

- If you are sending trace information to stdout, open your application server's stdout file in a text editor to view the trace output logs.

Tracing the Waveset Server

Waveset is a Java-based product whose executables consist of Java classes grouped as packages. When the code is implemented, many classes can supply trace output.

Tracing the Waveset server can provide helpful information such as where a server is failing, having problems, or not running. You can use the Waveset Debug pages to enable package-level and method-level tracing in a running Waveset server.

Note – Configure Waveset to trace at this advanced level *only* if instructed by Oracle Support.

Tracing Adapters

You can use trace information to verify that a resource adapter has started, to verify that all setting changes were saved, and to diagnose problems with the adapter.

This section contains information to help you configure adapter tracing, and is organized into the following topics:

- [“General Notes About Tracing Adapters” on page 133](#)
- [“Instrumenting Code with Trace Points” on page 134](#)
- [“Other Tracing Guidelines” on page 140](#)

General Notes About Tracing Adapters

When you enable tracing for an adapter, you must identify the methods that you want to trace by typing:

```
com.waveset.adapter.sample.MyResourceAdapter
```

You must also provide calls to create log entries for any new methods in your custom resource adapter so the adapter can write its resource settings to a log file.

In some cases, you can specify additional logging parameters for the resource instance, such as:

- Maximum Log Archives
- Maximum Active Log Age
- Log File Path
- Maximum Log File Size
- Log Level

Note –

- To further debug the synchronization process, you can configure synchronization logging for an Active Sync adapter. Instructions are provided in the Waveset online help.
- Tracing custom Java code can be useful when you are writing your own resource adapter. See [“Tracing Custom Code” on page 142](#) for more information.
- You can instrument code with trace points to help assess and resolve problems more effectively. See [“Instrumenting Code with Trace Points” on page 134](#) for more information.
- For more information about tracing specific adapter methods, see the [Oracle Waveset 8.1.1 Resources Reference](#).
- The following tasks were moved from the adapter to the task package in Identity Manager version 8.0. You must update the paths if you have tracing enabled for any of these tasks, or if you have customized task definitions referencing these packages.

| Old Package Name | New Package Name |
|---|--|
| com.waveset.adapter.ADSyncFailoverTask | com.waveset.task.ADSyncFailoverTask |
| com.waveset.adapter.ADSyncRecoveryCollectorTask | com.waveset.task.ADSyncRecoveryCollectorTask |
| com.waveset.adapter.SARunner | com.waveset.task.SARunner |
| com.waveset.adapter.SourceAdapterTask | com.waveset.task.SourceAdapterTask |

Instrumenting Code with Trace Points

The Waveset trace facility enables you to instrument code with trace points in the adapters component (resource adapters). This section contains some guidelines to help you to consistently use trace points to assess and resolve adapter problems.

Some guiding principles for instrumenting code with trace points are as follows:

- Make trace points as useful as possible for troubleshooting in both production and development environments by displaying relevant information.
- Create one entry point and one exit or exception trace point for every significant method.

- Use trace points as efficiently as possible during normal operations and when tracing is enabled by
 - Minimizing the number of trace level checks
 - Minimizing object creation when tracing is enabled

You can specify a trace level for a trace point to control how much information displays. The following table describes each of the trace levels that are defined in the `com.sun.idm.logging.TraceManager` interface.

TABLE 5-2 Defined Trace Levels

| Trace Level | Trace Variable | Usage |
|-------------|---------------------------|--|
| 1 | <code>Trace.LEVEL1</code> | Entry and exit points of public methods of the resource adapter interface |
| 2 | <code>Trace.LEVEL2</code> | Entry and exit points of non-public methods or methods not included in the Waveset component external interface |
| 3 | <code>Trace.LEVEL3</code> | Decision points or significant variables |
| 4 | <code>Trace.LEVEL4</code> | Very detailed information, such as significant variables within a loop |
| n/a | <code>Trace.ALWAYS</code> | Do not perform a trace-level check Note: Use this option if you have already specified the trace level in a condition. |

Note – Use the `com.sun.idm.logging.Trace` class to prevent circular dependencies on the `com.sun.idm.logging` package that implements the `com.sun.idm.logging.TraceManager` interface.

When adding trace points to code, be aware of the following information:

- Trace arguments to the method and return values from the method.

If the arguments “print large,” where the text representation is not brief, specify whether the Object is null or not. If the argument or return value is an array or a list, and the value could print large, showing the array size or list size is sufficient. You can use the `com.waveset.util.Util.length()` utility method to guard against null values.
- To format trace information or to create objects that present trace information in a meaningful way, put an `if` statement around the trace statements, using the `Trace.getLevel` method and the `isLogging(level)` method, so that Waveset does not marshal the trace point data unless the specified trace level is enabled.

When using this construct, specify the `Trace.ALWAYS` trace level for trace points inside the condition. This trace level prevents an unnecessary check for the trace level when that check has already been performed in the condition.

- If you are tracing an object value that requires a method call to get a printable representation of the object, such as `obj.getName()`, guard against a null object to prevent a `NullPointerException` occurring if trace becomes enabled.

Note – In general, do not use trace points for simple getter and setter methods or for any other trivial methods.

- When tracing a `WSUser` object, use the `toString()` method instead of the `getName()` method to present the value of the object. The `toString()` method is more robust and prints more useful information than the `getName()` method. Instead of tracing `user.getName()`, trace `user`, which calls `toString()` implicitly.
- If a method or constructor is just a wrapper, where that method or constructor does nothing significant except call another method or constructor with the same name but a different signature, just put trace points into the destination method or constructor.
- If performance is critical and the method is called many times, use information trace points to indicate errors. Do not add entry or exit trace points to the method. Use this option carefully.
- Before adding a trace point, ask yourself if that trace point will provide useful information when determining problems in the field.

The following sections describe specific trace levels in greater detail and provide examples showing how to use trace points in code.

- [“Using Entry and Exit Trace Points” on page 136](#)
- [“Using Information Trace Points” on page 138](#)
- [“Using Exception Trace Points” on page 139](#)
- [“Using Trace Point Templates” on page 140](#)

Using Entry and Exit Trace Points

Consider these guidelines before adding entry and exit trace points to your code:

- Use `Trace.Level1` entry or exit trace points for all Waveset component external interface methods, which are the public methods declared in the resource adapter interface.
- Use `Trace.Level2` entry or exit trace points for significant non-public methods including constructors and static methods or methods not included in the Waveset component external interface.
- Specify arguments in the trace points.

If the argument is not a primitive type or `java.lang.String`, and the argument requires any kind of object creation or method calls, conditionalize the entry trace point so that the formatting work is only done when tracing is enabled.

- When specifying an exit trace point, show the return value if the method is a primitive type or `java.lang.String`. If the return value requires some sort of formatting, conditionalize the object formatting as specified in these guidelines. Use the following:

```
int getLevel()
int getLevel (Method)
boolean isLogging(level,method)
boolean level1 (method)
boolean level2 (method)
boolean level31 (method)
boolean level4 (method)
```

- If you are tracing information that requires appending several `java.lang.Strings` together, use `java.lang.StringBuffer` instead of `java.lang.String`. Appends are faster using `java.lang.StringBuffer`. Also, conditionalize as in the preceding bullet item.
- When adding entry or exit trace points for significant constructors, remember that you cannot place a trace point before a `this()` or `super()` method call within a constructor. Put the entry trace point immediately after the `this()` or `super()` method call.
- Do not use an exit trace point to trace an exception condition. If the method throws an exception, use an exception trace point at the same trace level as the entry or exit trace point just before the exception is thrown. See [“Using Exception Trace Points” on page 139](#) for more information.

If a method contains an entry trace point, that method also must contain a code path that executes either a throwing method, a caught method, or an exit.

Following are some simple entry and exit trace statements. For these examples, assume the following CLASS variables are declared for each statement.

```
private static final String CLASS =
"com.waveset.adapter.MyResourceAdapter";
protected static Trace _trace = Trace.getTrace();
```

EXAMPLE 5-1 Entry Trace Point Example

```
final String METHOD = methodName;
_trace.entry(_trace.LEVEL1, CLASS, METHOD);
_trace.entry(_trace.LEVEL1, CLASS, METHOD, user);
if (_trace.level1(CLASS, METHOD)) {
_trace.entry(_trace.ALWAYS, CLASS, METHOD,
user= + user);
}

// Show the size of an array argument
// ASSUME: password is an argument to the method
// Note the use of the Util.length() method. It is
// a convenience method that guards against null.
if (_trace.level1(CLASS, METHOD)) {
StringBuffer sb = new StringBuffer(32);
```

EXAMPLE 5-1 Entry Trace Point Example *(Continued)*

```

        sb.append(password length=);
        ab.append(Util.length(password));
        _trace.entry(_trace.ALWAYS, CLASS, METHOD, sb.toString());
    }

```

EXAMPLE 5-2 Exit Trace Point Example

```

_trace.exit(_trace.LEVEL1, CLASS, METHOD);

_trace.exit(_trace.LEVEL1, CLASS, METHOD, returnValue);
if (_trace.level1(CLASS, METHOD)) {
    _trace.exit(_trace.ALWAYS, CLASS, METHOD,
        returnValue != null ? returnValue.getName() : returnValue);
}

// Show the size of an array
String[] accounts = ...
if (_trace.level1(CLASS, METHOD)) {
    StringBuffer sb = new StringBuffer(32)
    sb.append(accounts length=);
    ab.append(accounts.length);
    _trace.exit(_trace.ALWAYS, CLASS, METHOD, sb.toString());
}

```

Using Information Trace Points

Consider these guidelines before adding information trace points to your code:

- Generally, use `Trace.Level3` or `Trace.Level4` for information, variable and data trace points. The trace level depends upon the significance, expense, and the verbosity of the information presented.
- Use `Trace.Level3` and `Trace.Level4` information and variable trace points to trace significant conditions, branches, and other information as they occur in the code path.
- A variable trace point is a convenience method that is the same as an information trace point, except that a variable trace point prints out `variable=<variable>` instead of just printing the argument value. In addition, this method does not format the information unless the trace level specified in the argument is matched.
- Do not use an information trace point if the method contains an exception condition but does not rethrow the exception. Use the caught method. For an example, see [“Using Exception Trace Points” on page 139](#).
- Use `Trace.Level3` and `Trace.Level4` data trace points to show information in byte arrays. You can use a lower trace level if the information is appropriate for that trace level and the information presented is brief.
- You can use `Trace.Level1` and `Trace.Level2` information, variable, and data trace points to trace significant, but brief, information. However, you should rarely use these trace points at these levels.

Following is an example of a simple information trace statement. For this example, assume the following CLASS variables are declared.

```
private static final String CLASS =
"com.waveset.adapter.MyResourceAdapter";
protected static Trace _trace = Trace.getTrace();
```

EXAMPLE 5-3 Information Trace Point Example

```
_trace.info(_trace.LEVEL3, CLASS, METHOD, Some Message);
WavesetResult result = new WavesetResult();
try {
    someMethod();
} catch(Exception e) {
    String msg = Some Error Message;
    WavesetException we = new Waveset(msg, e);
    _trace.caught(_trace.LEVEL3, CLASS, METHOD, e);
    result.addException(we);
}
```

Using Exception Trace Points

Consider these guidelines before adding exception trace points to your code:

- Use exception trace points in all methods where entry or exit trace points are specified, and in methods that catch and rethrow exceptions. Waveset sometimes throws exceptions in which a new exception wraps the original exception. Use the same trace level being used by the entry and exit trace points.
- Use an exception trace point if an exception is created and thrown from the current method.
- Use an exception trace point caught method if an exception is caught and handled. For example, when the exception is not thrown from the current method. This situation typically occurs when Waveset catches and handles the exception by placing it in a container, such as a `WavesetResult`, to be examined later.
- If the exception thrown from the current method is not a *checked* exception, which is an exception that extends `java.lang.RuntimeException` or `java.lang.Error`, the method exit point will not be traced unless other measures are taken. One way to set an exception trace point in this situation is to use a `try/catch` block around the critical section of the method and rethrow the exceptions that occur. In general, this means that you should use `Trace.Level3` and higher when it is critical to know whether a method has succeeded or failed.

Following is an example of a simple exception trace statement. For this example, assume the following CLASS variables are declared:

```
private static final String CLASS =
"com.waveset.adapter.MyResourceAdapter";
protected static Trace _trace = Trace.getTrace();
```

EXAMPLE 5-4 Exception Trace Point Example

```
try {
    someMethod();
} catch(Exception e) {
    _trace.throwing(_trace.ALWAYS, CLASS, METHOD, e);
    throw e;
}

try {
    someMethod();
} catch(Exception e) {
    _trace.throwing(_trace.ALWAYS, CLASS, METHOD, e);
    WavesetException we = new WavesetException(Some Message, e);
    throw we;
}

if (error) {
    WavesetException e = new WavesetException(Some Error Message.;
    _trace.throwing(_trace.LEVEL3, CLASS, METHOD, e);
    throw e;
}

try {
    someMethod();
} catch(Exception e) {
    _trace.caught(_trace.LEVEL1, CLASS, METHOD, e);
}
// execution continues.
someOtherMethod();
```

Using Trace Point Templates

Using trace point templates can help you provide consistent and useful trace points in Waveset resource adapters. Waveset provides Eclipse templates in the following location:

```
src/wps/doc/eclipse-trace-templates.xml
```

To import these templates into Eclipse, select Window → Preferences → Java → Editor → Templates.

Note – You can create similar templates if you are using Emacs or IDEA.

Other Tracing Guidelines

This section describes some additional tracing guidelines, including the following:

- “Tracing Inner Classes” on page 141
- “Tracing Static Initializers” on page 141

Tracing Inner Classes

Remember to trace significant methods in inner classes. Be sure to declare a final static CLASS variable if there are any trace methods in the inner class.

EXAMPLE 5-5 Example of Using a CLASS Variable in Inner Classes

```
private final static String CLASS =  
com.waveset.adapter.SomeResourceAdapter$AdapterInnerClass;
```

Tracing Static Initializers

In general, do not use the trace facility in static initializers. Waveset executes these initializers when the class is loaded, which can occur when the repository is initialized. Use the Debug class to trace anything significant in static initializers.

Tracing Auditor

You can trace the following methods to troubleshoot issues with Identity Auditor:

- `com.sun.idm.auditor.policy` to trace issues with Audit Scans.
- `com.sun.idm.auditor.accessreview` to trace issues with Access Reviews.
- `com.sun.idm.auditor.report` to trace issues with Audit Reports.
- `com.sun.idm.auditor.view` to trace issues with Auditor Views.

▼ To Enable Tracing:

- 1 Open a browser and log in to the Administrator interface.
- 2 Select **Configure** → **Servers**.
- 3 When the **Configure Servers** page displays, click the server name in the **Server** column to edit the settings.
- 4 On the **Edit Server Settings** page, click the **Scheduler** tab.
- 5 Select the **Tracing Enabled** box to activate Scheduler debug tracing and write the results to stdout.
- 6 Save your changes.

Tracing Custom Code

You can use the Waveset tracing facility in custom-written code to provide seamless integration when troubleshooting your deployment.

Use the following classes to provide this tracing facility:

- Use `com.sun.idm.logging.trace.Trace` objects to interface with the tracing subsystem.
- Use `com.sun.idm.logging.trace.TraceManager` as a factory for these objects.

You can also use these `com.sun.idm.logging.trace` facilities to record trace output. See the Javadoc for more information about these facilities and classes.

Note – Be aware that the `com.sun.idm.logging.trace` facilities were *not* available prior to the Waveset Version 6.0 SP1 release. For earlier versions of Waveset, you must use the `com.waveset.util.Trace` class instead.

Tracing Exceptions

Exception logs are stack traces that you can view from the Waveset Debug pages or from the `config/Waveset.properties` file. Trace data does not include all exceptions by default, but exception logging can be an important and informative troubleshooting tool.

Use one of the following methods to enable exception logging:

- Open the Waveset Properties page (`debug/Show_WSProp.jsp`) in the Waveset Administrator interface. Locate the `exception.trace` key and change the Value to **true**.
- Open the `config/Waveset.properties` file in a text editor, and change the `exception.trace` key value to **true**. The following figure shows an example exception trace.

```

WavesetException: Validation errors detected in form.
com.waveset.exception.ValidationException: Validation errors detected in form.
at com.waveset.util.WavesetException.checkException(WavesetException.java:513)
at com.waveset.util.WavesetException.<init>(WavesetException.java:114)
at com.waveset.exception.FormValidation.<init>(FormValidation.java:139)
at com.waveset.object.UserEditor.checkInUse(Editor.java:747)
at com.waveset.server.UserEditor.runExpressionContext(Editor.java:483)
at com.waveset.view.UserEditor.runExpressionContext(Editor.java:438)
at com.waveset.view.UserEditor.checkInUse(Editor.java:1133)
at com.waveset.session.LocalSession.checkInUse(LocalSession.java:611)
at com.waveset.ui.util.GenericUISource.checkInUse(GenericUISource.java:522)
at com.waveset.ui.util.GenericEditForm.processGenericEditForm.java:613)
at org.apache.jsp.modif_jsp_jspServiceCmd419_jsp.java:395)
at org.apache.jasper.runtime.HttpServletService(HttpServlet.java:92)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:889)
at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:162)
at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:146)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:889)
at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:208)
at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:237)
at org.apache.catalina.core.StandardPipeline$StandardPipelineCtx.invokeNext(StandardPipeline.java:596)
at org.apache.catalina.core.StandardPipeline.invoke(StandardPipeline.java:433)
at org.apache.catalina.core.ContainerBase.invoke(ContainerBase.java:948)
at org.apache.catalina.core.StandardPipeline$StandardPipelineCtx.invokeNext(StandardPipeline.java:596)
at org.apache.catalina.core.StandardPipeline.invoke(StandardPipeline.java:433)
at org.apache.catalina.core.ContainerBase.invoke(ContainerBase.java:948)
at org.apache.catalina.core.StandardPipeline$StandardPipelineCtx.invokeNext(StandardPipeline.java:596)
at org.apache.catalina.core.StandardPipeline.invoke(StandardPipeline.java:433)
at org.apache.catalina.core.ContainerBase.invoke(ContainerBase.java:948)
at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:116)
at org.apache.catalina.core.StandardPipeline$StandardPipelineCtx.invokeNext(StandardPipeline.java:594)
at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:118)
at org.apache.catalina.core.StandardPipeline$StandardPipelineCtx.invokeNext(StandardPipeline.java:594)
at org.apache.catalina.core.StandardPipeline.invoke(StandardPipeline.java:433)
at org.apache.catalina.core.ContainerBase.invoke(ContainerBase.java:948)
at org.apache.coyote.tomcat4.CoyoteAdapter.service(CoyoteAdapter.java:152)
at org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:797)
at org.apache.coyote.http11.Http11Protocol$Http11ConnectionHandler.processConnection(Http11Protocol.java:785)
at org.apache.tomcat.util.net.NioSelectorThread.run(NioSelectorThread.java:577)
at org.apache.tomcat.util.thread.ThreadPool$ControlRunnable.run(ThreadPool.java:683)
at java.lang.Thread.run(Thread.java:595)

```

Waveset sends exception logs to `stdout` on the web application instance, which is often the application server console.

Note – When you are finished, disable exception logging to stop unnecessary output to the application server logs. To disable exception logging, set the `exception.trace` key value back to `false`.

Tracing Forms

You can enable tracing to troubleshoot edited forms and to check for expression statement errors within your form fields.

Use either of the following methods to enable tracing:

- Open the Waveset Properties page (`debug/Show_WSProp.jsp`) in the Waveset Administrator interface. Locate the `form.trace` key and change the Value to `true`.
- Open the `config/Waveset.properties` file in a text editor, and change the `form.trace` key value to `true`.

Waveset reports any problems with form expression syntax to standard output.

Note – The `form.trace` key is disabled by default because it produces trace information for every field on every page, including the Accounts List page, which affects system performance. Consider using a more targeted form and field tracing method.

When you are finished troubleshooting your forms, remember to disable tracing by changing the `form.trace` key value back to **false**.

Global XPRESS tracing might also be helpful while you are developing and updating forms and form processes. Although Global XPRESS tracing produces a large amount of output that affects system performance, this tracing method shows XPRESS output and might expose where problems are happening in your form.

For more information, see

- “Tracing Global XPRESS” on page 144
- “Testing Your Customized Form” in *Oracle Waveset 8.1.1 Deployment Reference*

Tracing Global XPRESS

While not generally recommended, you can use global XPRESS tracing to trace any and all XPRESS code, wherever the code is located. For example, you can trace XPRESS code in forms, views, and workflows. The resulting trace shows XPRESS output that can expose potential problems.

Note – XPRESS tracing is disabled by default because it produces a large amount of output, which affects system performance.

See “Testing Your Customized Form” in *Oracle Waveset 8.1.1 Deployment Reference* for more information about tracing XPRESS functions.

▼ To Enable Global XPRESS Tracing:

- 1 **Open a command window.**
- 2 **Change directories to `config/Waveset.properties` in the default Waveset installation directory.**
- 3 **Open the `config/Waveset.properties` file and edit the `xpress.trace` line to read:**
`xpress.trace=true`
- 4 **Save the `Waveset.properties` file.**

- 5 **Restart your application server or reload the `Waveset.properties` file from the Waveset debug pages.**
- 6 **Replicate the XPRESS trace output to a file by adding this line to the `Waveset.properties` file:**
`xpress.traceFile=FileName.txt`

```
xpress.traceFileOnly=true
```

When you set `xpress.traceFileOnly=true` in `Waveset.properties`, all XPRESS statement evaluations will generate trace messages to a file specified by `xpress.traceFile`. Otherwise, when `xpress.traceFile` has a value, trace messages are redirected to both the console and a file.

Tracing PasswordSync

This section describes how to enable tracing for PasswordSync and how to configure tracing in Direct access or JMS modes.

To Enable Trace for PasswordSync

You can use the following methods to configure tracing for Waveset's PasswordSync feature:

- “Using the PasswordSync Configuration Tool” on page 145
- “Editing the Registry Keys” on page 146

Using the PasswordSync Configuration Tool

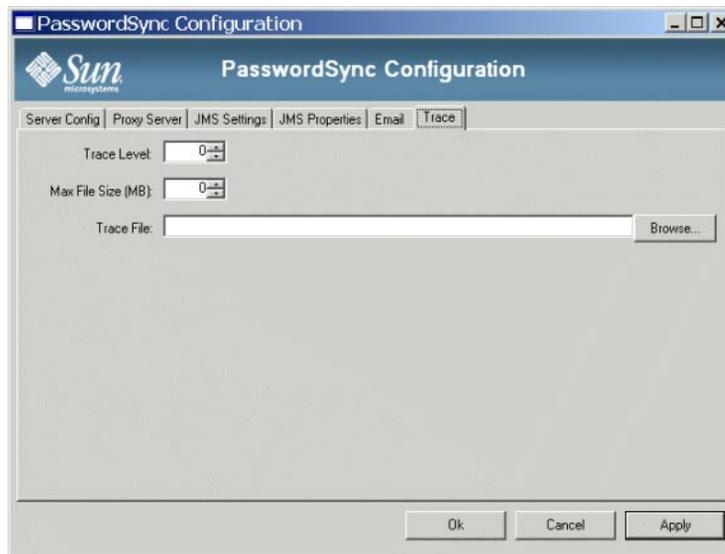
This section describes how to configure tracing from the PasswordSync Configuration Trace tab.

Note – For more information about installing and configuring PasswordSync, see [Chapter 11, “PasswordSync,”](#) in *Oracle Waveset 8.1.1 Business Administrator's Guide*.

The first time you run the configuration tool, the wizard does not allow you to configure tracing. Subsequently, when you run the configuration tool, the wizard displays a Trace tab where you can configure tracing.

The following figure shows the PasswordSync Configuration tool Trace tab.

FIGURE 5–2 Trace Tab in the PasswordSync Configuration Tool



From this tab, you can specify the following:

- Use the Trace Level field to specify the level of detail you want PasswordSync to provide when writing to the trace log. A value of 0 turns tracing off, while a value of 4 shows full detail.
- Use the Max File Size field to specify a maximum size for the log file.
When the trace file exceeds the size specified in the Max File Size (MB) field, PasswordSync starts a new trace file and appends .bk to the old trace file name. For example, if your trace level is set to 100 Mbytes, and your trace file writes to C:\logs\pwicsvc.log, when the trace file exceeds 100 Mbytes PasswordSync renames the file to C:\logs\pwicsvc.log.bk. PasswordSync then creates a new C:\logs\pwicsvc.log file where new trace file messages are written.
- Use the Trace File field to specify a location for the PasswordSync trace file.

Editing the Registry Keys

To enable additional PasswordSync configuration settings, edit the following PasswordSync registry keys using the PasswordSync configuration tool.

Note – Using PasswordSync configuration tool is the safest method for editing PasswordSync registry keys. Editing these keys directly in the Windows Registry is discouraged.

TABLE 5-3 Registry Keys

| Key Name | Type | Description |
|--------------|--------|--|
| dumpFilebase | REG_SZ | <p>Set this registry key to enable Windows to generate a dump file if the PasswordSync DLL displays an exception.</p> <p>You must set this registry key to the fully qualified directory path where you want to write the memory dump. For example: c:\temp</p> <p>Set the registry value to write the memory dump each time Waveset catches an exception during password processing.</p> <p>Note: On Windows 2000 server (any service pack), you also must install in the configured directory DbgHelp.dll, which is available from Microsoft. The minimum release version for the DbgHelp.dll file must be Version 5.1. Download the DbgHelp.dll file here:</p> <p>http://www.microsoft.com/whdc/DevTools/Debugging/default.mspx</p> <p>If DbgHelp.dll is not installed, no dump files will be generated on Windows 2000.</p> <p>The format for dump file names is</p> <p>lhpwic-YYYYMMDD-HHMM-xxxxx.dmp</p> <p>In this name, YYYYMMDD will be the date of the dump, HHMM is the time of the dump (24-hour clock), and xxxxx is the thread number of the application.</p> <p>You must manually remove dump files! Dump files range in size from 20 MB to more than 100 MB, depending on the size of the Windows Local Security Authority Subsystem (LSASS) process. Over time, systems with limited disk space could fill up if these dump files are not removed.</p> |
| installdir | REG_SZ | Directory where the PasswordSync application is installed. |

The PasswordSync registry keys are located in the following location:

HKEY_LOCAL_MACHINE\SOFTWARE\Sun Microsystems\Sun Identity Manager\Passwordsyc

Other keys are present in this location.

To Collect Logs for the Different Modes

PasswordSync trace logs are the same, whether you are using a direct access mode or JMS mode configuration. However, these trace logs might only provide partial information. You must configure different classes for each configuration to collect logs on the server side, as described in the following sections.

Tracing in Direct Mode

When using PasswordSync with a direct access mode configuration, the trace logs show failures, but not all logged failures are real failures. For example, in some circumstances the view check-in takes a long time, which shows as a failure in the log. You must trace on the server side to see this information.

In Direct mode, PasswordSync talks to the servlet that generates the view to be checked into the repository. You can trace the `com.waveset.rpc.GenericMessageHandler` class at level 4 to view all phases of password synchronization, from receiving the password change to the response generated and returned to the servlet. Level 4 is the only level that supplies enough detail for troubleshooting.

Tracing in JMS Mode

When using PasswordSync with a JMS mode configuration, the logs only show successful or failed deliveries to the JMS server. From this point on, you must rely on server side logs. JMS tracing is a little more complex.

You can trace the `com.waveset.rpc.PasswordSyncHandler` class at level 4 to convert the messages generated by the PasswordSync dll into a JMS message and add those messages to the JMS queue. Limited tracing is available in this class, and only level 4 can provide enough information to help with troubleshooting.

If PasswordSync successfully delivers the JMS message to the JMS queue, the tracing will not help you find the cause of a problem. The next, and final step is to trace the JMS adapter. See the [Oracle Waveset 8.1.1 Resources Reference](#) for instructions.

Tracing Rule-Driven Members Caches

You can trace rule-driven members caches and use the results to tune cache properties in `Waveset.properties`.

If you trace `com.waveset.server.RuleDrivenMembersCache` at Level 1, the resulting information includes the number of adds, removes, hits to the cache, and so forth. Use this information to evaluate cache sizes and decide whether tuning the cache properties in `Waveset.properties` is necessary.

You can use the following properties in `Waveset.properties` to control the rule-driven members cache:

- Use `ruledrivenmemberslistcache.size = value` to specify the maximum number of object lists to be cached per subject. (Default is 20.)
- Use `ruledrivenmemberslistcache.rowlimit = value` to specify the maximum total number of objects to be cached per subject. (Default is 100000.)

By default, Waveset evaluates the User Members Rule associated with a specified organization and creates a user members list cache containing a dynamic list of users. However, you can also create a user members list cache containing a specified user's set of dynamic member organizations for a given subject. The key to this cache is the object type concatenated with the object ID. For example, the User object type concatenated with the User#ID#Configurator object ID. The value for each key is a list of object groups for which this object is dynamically a member.

To determine whether the object being evaluated is a dynamic member, the cache evaluates the same User Members Rule per organization as that used by the list cache. If the object is a dynamic member, Waveset adds that object to the list and then caches the list. Waveset caches both empty and non-empty lists to ensure the highest cache hit rate.

For this cache, you can use these properties in `Waveset.properties` to control memory requirements affecting performance:

- Use `ruledrivenmembersobjectcache.size = value` to specify the number of member object group lists to be cached per subject. The default value is 100.
- Use `ruledrivenmembersobjectcache.rowlimit = value` to specify the maximum total number of member object groups to cache per subject. The default value is 100000.

Tracing Oracle Waveset Service Provider Delegated Administration

Enabling Waveset tracing at Method/Class Level 2 for the following classes allows you to trace authorization flows when listing or accessing Oracle Waveset Service Provider (Service Provider) users and when AdminRoles are dynamically assigned when Service Provider users login.

- Trace `com.sun.idm.idmx.view.IDMXBrowseViewer` when searching for Service Provider users.
- Trace `com.sun.idm.idmx.view.IDMXUserViewer` when creating, editing, or deleting Service Provider users.
- Trace `com.sun.idm.idmx.api.IDMXServerUtils` when using both of the preceding classes.
- Trace `com.waveset.security.authn.WSSPELoginModule` when you are logging in as a Service Provider user.

Note – You configure Service Provider tracing from the Waveset debug pages. If necessary, review [“Tracing the Waveset Server” on page 133](#) for instructions.

Tracing Reconciliation

If you are having problems with a reconciliation task, you can use the standard tracing facility on `com.waveset.task.Reconciler` to trace the Reconciler.

Use either of the following methods to enable tracing:

- Open the Waveset Properties page (`debug/Show_WSProp.jsp`) in the Waveset Administrator interface. Locate the `exception.trace` key and change the Value to **true**.
- Open the `config/Waveset.properties` file in a text editor, and change the `exception.trace` key value to **true**.

You can also enable tracing from the System Settings page and trace the following reconciliation methods, at the Method/Class trace Levels noted, to view useful debugging information.

TABLE 5-4 Reconciliation Methods/Classes to Trace

| Trace this <code>com.waveset.recon.Method/Class</code> | At this Trace Level | To See |
|---|---------------------|--|
| <code>ReconTask\$WorkerThread#reconcileAccount</code> | 2 | The individual account being reconciled |
| <code>ReconTask\$WorkerThread#performResponse</code> | 2 | The individual account and user during response |
| <code>ReconUtil#deleteAccountIndex</code> | 2 | User information to be deleted from the account index |
| <code>UserContext#aquireRepoLock</code> | 2 | The user who is being locked for update |
| <code>UserContext#releaseRepoLock</code> | 2 | The user who is being unlocked in the repository |
| <code>ReconUtil#deleteAccountIndex</code> | 2 | User information to be deleted from the account index |
| <code>UserContext#aquireRepoLock</code> | 2 | The user who is being locked for update |
| <code>ReconTask\$WorkerThread#failUserExamination</code> | 2 | All user examination requests that failed with the error |
| <code>ReconTask\$WorkerThread#failUserResponses</code> | 2 | All user response requests that failed with the error |
| <code>UserContext#releaseRepoLock</code> | 2 | The user who is being unlocked in the repository |
| <code>ReconTask\$ResourceThread#examineResource</code> | 3 | How many accounts read from the resource |
| <code>ReconTask\$ResourceThread#queueAccountReconciles</code> | 3 | Information about each account read from the resource such as <code>accountId</code> , <code>accountGUID</code> , <code>accountDisabled</code> |
| <code>ReconTask\$ResourceThread#examineLighthouse</code> | 3 | How many Waveset users claim to own an account on the reconciled resource queued |
| <code>ReconTask\$WorkerThread#findClaimants</code> | 3 | All Waveset users who claim to have an account on the resource |
| <code>ReconTask\$WorkerThread#confirmPossibleOwners</code> | 3 | A list of all confirmed owners of resource accounts |

TABLE 5-4 Reconciliation Methods/Classes to Trace (Continued)

| Trace this <code>com.waveset.recon.Method/Class</code> | At this Trace Level | To See |
|---|---------------------|---|
| <code>ReconTask\$WorkerThread#applyResponse</code> | 3 | The response list that is being applied |
| <code>AccountContext#processAttributeWorkflow</code> | 3 | The attribute changes and the formatted changes during the launch of the attribute change workflow |
| <code>ReconUtil#getRuleState</code> | 3 | Full User view with the user's attribute during rule processing |
| <code>ReconUtil#evaluateCorrelationRule</code> | 3 | The value of the correlation rule state and result for examination |
| <code>ReconUtil#confirmPotentialOwners</code> | 3 | A list of users who have been confirmed using the confirmation rule |
| <code>ReconUtil#getIfExistsAccountIndexEntry</code> | 3 | To output info related to examination of the account index for a specified entry |
| <code>ReconUtil#launchWorkflow</code> | 3 | The task instance and task definition information when launched |
| <code>ReconUtil#indexFoundAccount</code> | 3 | The account and situation recorded during a create or update of the index for an account known to exist |
| <code>ReconUtil#indexMissingAccount</code> | 3 | The account and situation recorded during a create or update of the index for an account NOT known to exist |
| <code>ReconUtil#listAccountsIndexSaysExist</code> | 3 | Account information that the index says exists |
| <code>ReconTask\$ResourceThreadwaitForLighthouseWorkItems#</code> | 4 | How many users queued and processed during the Waveset user examination process |
| <code>ReconTask\$ResourceThread#waitForReconcileWorkItems</code> | 4 | How many reconciles and responses queued and processed for the specified resource |
| <code>ReconTask\$WorkerThread#correlateUsers</code> | 4 | A list of correlated/confirmed users |
| <code>ReconTask\$WorkerThread#respondOrRequeue</code> | 4 | The error message if there is a problem performing the previous response method for an account |
| <code>Response#perform</code> | 4 | The user, response, and the user's resource info XML before and after the modifications took place |
| <code>Response#perform</code> | 4 | A summary of the response action on the user |
| <code>Response#createNewUserFromAccount</code> | 4 | Full details of the response action on the user |
| <code>ReconUtil#evaluateConfirmationRule</code> | 4 | The value of the confirmation rule state and result for examination |
| <code>ReconUtil#getCorrelatedUsers</code> | 4 | A list of correlated users matching the specified rule result |

Note – Remember, the higher the tracing level, the larger the trace file. Also tracing all of these methods at the same time will create a *very* large trace file.

When you are finished troubleshooting, remember to disable tracing by setting the `exception.trace` key value back to **false**.

Tracing the `setRepo` Command

If you see errors while you are using the `setRepo` command to configure the Waveset repository, use the following flags to isolate and debug the problem:

```
-Dtrace.enabled=true
-Dtrace.level.com.waveset.repository.AbstractDataStore=2
-Dtrace.level.com.waveset.repository.DefaultTypeHandler=4
// Use one of the following based on your repository type
-Dtrace.level.com.waveset.repository.OracleDataStore=4
-Dtrace.level.com.waveset.repository.SqlServerDataStore=4
-Dtrace.level.com.waveset.repository.MySQLDataStore=4
-Dtrace.level.com.waveset.repository.DB2DataStore=4
```

Waveset sends output from the `setRepo` command to the default `W$SHOME/config/W$Trace.log` file.

Tracing SPML

This section describes methods for enabling trace for SPML Version 1.0 and SPML Version 2.0.

To Enable Tracing for SPML 1.0

SPML 1.0 contains the following options for turning on trace output so you can log Waveset's SPML traffic and diagnose problems.

Method 1: Enable the `setTrace` Method

You can use the `setTrace` method, provided by the `Spm1Client` and `LighthouseClient` classes, to enable tracing for SPML 1.0.

When you enable this `setTrace` method, the XML for the request sent by the client and the XML for the response received from the server are printed to the *client* console as they are sent and received.

The `setTrace` method takes a Boolean argument. For example:

```
Spm1Client client = new Spm1Client();
client.setURL("http://localhost:8080/idm/spml");
client.setTrace(true);
```

Method 2: Initializing the `org.openspml.server.SOAPRouter Servlet`

You can enable tracing when initializing the `org.openspml.server.SOAPRouter` servlet, which is a third-party, open source class from the OpenSPML organization. This servlet controls the output of RPC traffic information for the servlet handling SPML requests.

To enable this tracing method, add the following to the `WEB-INF/web.xml` file:

```
<servlet>
  <servlet-name>rpcrouter2</servlet-name>
  <display-name>OpenSPML SOAP Router</display-name>
  <description>no description</description>
  <servlet-class>
    org.openspml.server.SOAPRouter
  </servlet-class>
  <init-param>
    <param-name>trace</param-name>
    <param-value>true</param-value>
  </init-param>
  ...
</servlet>
```

The following is sample output for an SPML 1.0 trace:

```
SpmlClient: sending to http://example.com:8080/idm/servlet/rpcrouter2
<spml:addRequest xmlns:spml='urn:oasis:names:tc:SPML:1:0'
xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core'>
  <spml:operationalAttributes>
    <dsml:attr name='session'>

<dsml:value>session token</dsml:value>

    </dsml:attr>
  </spml:operationalAttributes>
  <spml:identifier type='urn:oasis:names:tc:SPML:1:0#GUID'>
    <spml:id>suetonius</spml:id>
  </spml:identifier>
  <spml:attributes>
    <dsml:attr name='objectclass'>
      <dsml:value>person</dsml:value>
    </dsml:attr>
    <dsml:attr name='password'>
      <dsml:value>password</dsml:value>
    </dsml:attr>
    <dsml:attr name='gn'>
      <dsml:value>Suetonius</dsml:value>
    </dsml:attr>
    <dsml:attr name='sn'>
      <dsml:value>Tranquillus</dsml:value>
    </dsml:attr>
    <dsml:attr name='email'>
      <dsml:value>twelve@example.com</dsml:value>
    </dsml:attr>
  </spml:attributes>
</spml:addRequest>
```

```
SpmlClient: received
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
<SOAP-ENV:Body>
<spml:addResponse xmlns:spml='urn:oasis:names:tc:SPML:1:0'
  xmlns:dsm1='urn:oasis:names:tc:DSML:2:0:core' result='urn:oasis:names:tc:
  SPML:1:0#success'>
  <spml:operationalAttributes>
    <dsm1:attr name='session'>

<dsm1:value>session token</dsm1:value>
    </dsm1:attr>
  </spml:operationalAttributes>
  <spml:identifier type='urn:oasis:names:tc:SPML:1:0#GUID'>
    <spml:id>suetonius</spml:id>
  </spml:identifier>
</spml:addResponse>
/SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Note – For more information about the SOAP rpcrouter servlet, refer to your OpenSPML Toolkit documentation.

Method 3: Pass the `trace` Operational Attribute

You can enable tracing for an individual SPML RPC request by passing a `trace` operational attribute to the RPC request on the server side.

Tracing occurs during servlet initialization, and it controls how information is output for the RPC traffic of a servlet handling SPML Version 1.0 requests. For example, the trace prints the raw XML that is sent back and forth on whatever the `System.out` is for that servlet (which is a function of the Application container). For example:

```
AddRequest ar = new AddRequest();
ar.setOperationalAttribute("trace", "true");
```

When you use the `trace` attribute, how the attribute affects server operation is vendor-specific. Currently, Waveset prints the raw request and response data to the *server* console, which is useful if the client application is not associated with a console window.

For more information consult your OpenSPML Toolkit product documentation.

To Enable Tracing for SPML 2.0

SPML 2.0 contains the following options for turning on trace output so you can log Waveset's SPML traffic and diagnose problems.

Method 1: Using the `org.openspml.v2.transport.RPCRouterServlet` Servlet

As with SPML 1.0, you can enable tracing for SPML 2.0 when initializing the `org.openspml.v2.transport.RPCRouterServlet` class, which controls the output of RPC traffic information for the servlet handling SPML 2.0 requests.

To enable this tracing method, add the following to the `WEB-INF/web.xml` file:

```
<servlet>
  <servlet-name>openspmlRouter</servlet-name>
  <display-name>OpenSPML SOAP Router</display-name>
  <description>A router of RPC traffic - nominally SPML 2.0 over SOAP</description>
  <servlet-class>
    org.openspml.v2.transport.RPCRouterServlet
  </servlet-class>
  <init-param>
    <param-name>trace</param-name>
    <param-value>true</param-value>
  </init-param>
  ...
</servlet>
```

The following example illustrates output from an `rpcrouter` servlet trace:

```
RPCRouterServlet:
<?xml version='1.0' encoding='UTF-8'?><SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Header/>
<SOAP-ENV:Body><lookupRequest
xmlns='urn:oasis:names:tc:SPML:2:0' requestID='random name' executionMode='synchronous'
returnData='everything'>
<openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml' name='
session' value=session token' />
<psoID ID='random name' targetID='spml2-DSML-Target' />
</lookupRequest>
</SOAP-ENV:Body></SOAP-ENV:Envelope>

RPCRouterServlet: response:
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope

  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
<SOAP-ENV:Body>
<lookupResponse xmlns='urn:oasis:names:tc:SPML:2:0' status='failure' requestID='random
name' error='noSuchIdentifier'>
<openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
name='session'
value=session token' />
<errorMessage>Item User:random name was not found in the repository, it may have been
deleted in another session.</errorMessage>
</lookupResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Note – For more information see [“Method 2: Initializing the `org.openspml.server.SOAPRouterServlet`” on page 153](#), in the [“To Enable Tracing for SPML 1.0” on page 152](#) section.

Method 2: Using the SPML Access Log

SPML 2.0 has a simple text access log that can be a useful troubleshooting tool. This log is always available and it enables you to view information, such as what kind of requests have been received, how long it took to process those requests, and whether the requests were successful without having to enable tracing.

Instructions for configuring this SPML text access log are provided in [“Configuring SPML Tracing” in *Oracle Waveset 8.1.1 Web Services*](#)

Tracing the Task Scheduler

If a Scheduler task is having problems, you can use the standard tracing facility on `com.waveset.task.Scheduler` to trace the task scheduler. The output shows detailed information about task scheduling.

▼ To Enable Tracing:

- 1 Open a browser and log in to the Administrator interface.
- 2 Select **Configure** → **Servers**.
- 3 When the **Configure Servers** page displays, click the server name in the **Server** column to edit the settings.
- 4 On the **Edit Server Settings** page, click the **Scheduler** tab.
- 5 Select the **Tracing Enabled** box to activate Scheduler debug tracing and write the results to `stdout`.
- 6 Save your changes.

Note –

- In a clustered environment, tracing occurs on each instance.
 - See “[Tracing the Waveset Server](#)” on page 133 for more information about defining and editing trace configuration objects, and about viewing trace files.
-

Tracing Workflows

Enabling workflow tracing can help you resolve problems with workflow activities and understand the workflow process.

Note –

- In a clustered environment, tracing occurs on each instance.
 - To debug workflows in a multiple server deployment environment, consider shutting down all but one server. If all servers are running, you cannot determine which server in the environment is executing the workflow, which causes troubleshooting problems.
-

To Enable Tracing for Workflows

Use the following steps to enable workflow tracing:

▼ To Enable Workflow Tracing:

- 1 Open a browser and log in to the Waveset Administrator interface.
- 2 From the System Setting page, choose Configuration from the List Objects Type menu.
- 3 Click the List Objects button.
- 4 When the List Objects of type: Configuration page displays, scroll down the list of objects to locate the System Configuration object and click the edit link.
- 5 When the Checkout Object: Configuration, #ID#CONFIGURATION: SYSTEMCONFIGURATION page displays, you can edit any of the following workflow options in the SystemConfiguration object:

Note – Typically, you enable only one option, but it is possible to enable more than one option at a time.

These attributes are not dependent on each other. You can turn on one type of trace while the other types are turned off.

- Specify `workflow.consoleTrace=true` to redirect workflow trace messages to the application server console, which can be useful when workflows are terminating due to a fatal exception because this attribute prints more trace output than `workflow.fileTrace`. (Default value is `false`.)
- Specify `workflow.fileTrace=PathToFile` to redirect workflow trace messages to a file that is easy to read. This attribute does not have a value by default.

```
<Attribute name='fileTrace' />
```

Add a value tag to the `workflow.fileTrace` attribute and, using Unix-style forward slashes, enter the path to a log file. Waveset stores relative pathnames relative to the application server's installation directory. For example:

- **On Windows.** `<Attribute name='fileTrace' value='C:\mydir\workflow.txt' />`
- **On Solaris/UNIX.** `<Attribute name='fileTrace' value='/mydir/workflow.txt' />`

- Specify `workflow.traceLevel=tracingLevel` to specify the level of workflow tracing you want to see.

Specify `workflow.Trace=true` to trace workflow processing. You must restart the application server to start tracing with this option. Waveset stores the trace results in the task's `WavesetResult` object. Use this tracing option when file system access is unavailable. (Default value is `false`.)

- Trace messages in the task's `WavesetResult`. (Default value is 1.)

Specifying `workflow.Trace=true` appends trace messages into one long, unformatted string that is difficult to read. Use this option only when you do not have access to the file system.

Note – With the first two options, you might lose some of the workflow trace if a fatal exception occurs.

6 Save the `SystemConfiguration` object.

7 Restart your application server, or reload the `SystemConfiguration` object from the Waveset debug area.

To Enable Tracing for a Designated Workflow

Use one of the following methods to enable tracing for a designated workflow:

- **Editing the WFProcess definition.** To enable trace unconditionally for a particular process, edit the XML of the TaskDefinition object by adding the `trace='console'` attribute to the `<WFProcess>` element.
- **Editing the workflow variable.** Gives you more control over the timing of tracing by using a workflow variable. The workflow engine will look for a top-level workflow variable named `trace`.

The following example shows how to trace a workflow variable.

EXAMPLE 5-6 Tracing a Workflow Variable

```
<Variable name='trace'>
  <cond><eq><ref>accountId</ref><s>jfaux</s></eq>
  <s>workflowTrace.txt</s>
</cond>
</Variable>
```

The `trace` variable turns tracing on only if the workflow is operating on a user named `jfaux`. You could also specify `trace` in a form field to control tracing interactively.

In this example, the trace output is written to the `workflowTrace.txt` file.

Locating Version Information

You can use one of the following methods to get information about the Waveset version you are currently using:

- Hover your cursor above the Help button in the upper right corner of the Waveset application window. A pop-up is displayed that contains the version number.
- Open the `Waveset.properties` file, and check the information provided at the top of the file.

Use the Waveset System Properties page (`/debug/SysInfo.jsp`) to get information about which JVM versions, and other system software versions, you are currently using.

Tracing the Waveset Gateway Objects and Activities

This section describes how to trace objects and activities in Oracle Waveset Gateway, the information is organized as follows:

- [“How to Configure Gateway Tracing” on page 160](#)
- [“How to Capture Encrypted Information” on page 163](#)
- [“How to Configure Tracing for the PowerShellExecutor.dll Add-On” on page 164](#)
- [“How to Capture Dr. Watson Logs” on page 166](#)

Note –

- When viewing or editing a Gateway trace file, use Notepad to avoid file restrictions.
 - When you start the Gateway, the program appends new trace entries to the trace file instead of deleting entries. To locate the point at which the Gateway trace entries begin, look for a Gateway version string.
 - The Gateway version is output in the trace automatically when you start the Gateway. You can also type `gateway -v` from the command line to get the version.
-

How to Configure Gateway Tracing

If you cannot access the Gateway, you can enable tracing to debug problems with Windows accounts on Waveset. You can enable or disable tracing from the Gateway Debug page (`Gateway.jsp`) or, if you want a wider range of options, you can enable tracing from the command line

- [“To Enable Tracing from the Gateway Debug Page” on page 160](#)
- [“To Enable Tracing from the Command Line” on page 161](#)
- [“To Disable Tracing from the Debug Page” on page 163](#)
- [“To Disable Tracing from the Command Line” on page 163](#)

▼ To Enable Tracing from the Gateway Debug Page

Use the following instructions to enable and configure tracing from the Gateway Debug page (`Gateway.jsp`).

- 1 **Log in to the Waveset Administrator interface.**
- 2 **Type the following URL in to your browser to open the Gateway Debug page:**
`http://host:port/idm/debug/Gateway.jsp`
- 3 **Choose a resource to trace from the Gateway Resource list.**
- 4 **If necessary, modify existing settings.**

Click the following buttons to modify the settings:

- **Get Version.** Returns the Gateway version and the operating system of the machine on which you are running the Gateway.
- **Get Trace File.** Returns the contents of the trace file.
- **Get Trace Parameters.** Returns the path of the trace file, the trace level, and the maximum size of the trace file.
- **Set Trace Parameters.** See [“To Create a New Trace Configuration Object” on page 131](#) for information about these options.
- **Get Loaded Modules.** Returns the load addresses of modules (DLLs) being used by the Gateway.

The Get Loaded Modules list consists of load addresses, followed by module names and only includes loaded modules. The list does not include delay-loaded modules that have not been called.

The Get Loaded Modules option only supports Active Directory and Domino.

▼ To Enable Tracing from the Command Line

1 Open a command window.

2 Start the Gateway.

- To start the gateway as a service, type the following command:
#gateway -s
- To start the gateway as a regular application rather than a service, type the following command:
#gateway -d

Note –

- Do not use the -d and -s options at the same time on the same command line.
 - **-p.** Use this argument to specify a TCP/IP port number. (The default port is 9278.) For example,
#gateway -d -p 9278
-

3 Specify the necessary trace command arguments.

The following table describes the Gateway tracing command line arguments.

| Argument | Description |
|----------|--|
| -f | Specify the path to the trace file |
| -l | Specify the level of tracing: <ul style="list-style-type: none"> ■ Level 0. Disables tracing. (Default) ■ Level 1. Traces the flow of control between components and generally defines a <i>low-detail</i> trace point that includes entry and exit from high-level functional methods. ■ Level 2. Generally defines a <i>medium-detail</i> trace point that includes entry and exit from every method, and information and data trace points for high-level functional methods. Level 2 adds the flow of control within each component, major decision points, and items of information. ■ Level 3. Generally defines a <i>high-detail</i> trace point that includes entry and exit from every method, information and data trace points for high-level functional methods, and significant subroutines. Level 3 adds lower-level decision points and items of information. ■ Level 4. Generally defines a <i>hyper-detail</i> trace point that includes everything traced in the other trace levels. Level 4 traces at a very low level and can provide a level of detail that is seldom needed but might be useful in characterizing complex behaviors of some components. NOTE: Not all components support level 4. Trivial methods, such as getters and setters, generally do not have entry or exit trace points because they add overhead. |
| -m | Specify the maximum trace file size in kilobytes <p>When the trace file reaches -m Kbytes, Waveset closes the current trace file, deletes any existing back-up files, renames the current trace file to the name specified by the -f argument with .bk appended, and opens a new trace file with the -f argument name.</p> <p>For example, if you specified -f beebble.trc on the command line, the following two files result after -m Kbytes are recorded:</p> <pre>beebble.trc.bk beebble.trc</pre> <p>Where beebble.trc contains the most recent traces.</p> |

Usage: #gateway -l *level* -m *size* -f *name* -s

For example:

```
cd %WSHOME%\bin\winnt
#gateway -p 11319 -l 2 -m 500 -f %CD%\gateway.trc -s
```

The preceding invocation starts the Gateway with the following characteristics:

- -p 11319 – Use port 11319

You must configure this port for Gateway resources from the Waveset resource configuration. For example, for an Active Directory resource

- -l 2 – Trace the gateway at output level 2.
- -m 500 – Limit the size of the trace log file to 500 Kilobytes.
- -f C:\gateway.trc – Write the trace output to a file called gateway.trc in the C:\ directory.
- -s – Start the gateway service.

Note – If specified, Waveset saves -p, -l, -m, and -f values in the registry, so the next time you run Gateway from the command line or as a service, the same values are used.

If you started the gateway using the -d option, Waveset sends the Gateway trace output to the console *and* to a trace file.

▼ To Disable Tracing from the Debug Page

- 1 Log in to the Waveset Administrator interface.
- 2 Type the following URL in to your browser to open the Gateway Debug page:
`http://host:port/idm/debug/Gateway.jsp`
- 3 Deselect the Show Trace option or delete the file name from the Get Trace File text box.

▼ To Disable Tracing from the Command Line

- 1 If necessary, open a command window.
- 2 Stop the Gateway service.
 - If you started the Gateway service using the -s option, stop the service by typing the following command:
`#gateway -k`
 - If you used the -d option to start the gateway as a regular application, stop the gateway by typing **Control-C** in the same command window you used to start the gateway.
- 3 Start the gateway with trace level 0 using the -l option. (You do not have to specify the -m or -f option.) For example
`#gateway -d -l 0`

How to Capture Encrypted Information

Waveset encrypts all messages between the gateway and the Waveset server.

You can configure Waveset to log encryption and decryption message details while tracing at level 4 in the gateway, but this capability is disabled by default. Tracing encryption information at level 4 can cause a large amount of overhead because Waveset logs each message multiple times, once on entry to and once on exit from the encryption routines. In addition, Waveset limits the size of logged informational messages to 4000 characters, which can truncate large messages.

To enable message tracing, you must add the `traceEncryptedInfo` registry entry to the gateway registry key. This entry triggers a full log of each message before and after encryption and decryption. You must still use trace level 4 to trace this information.

You can use the following values for the `traceEncryptedInfo` registry entry.

- **1.** Trace input and output messages for all encryption and decryption routines.
- **Any other value.** Disables tracing.
- **No entry value** (*Default*). Disables tracing.

Note – Waveset preserves password encryption in the logs. Tracing encrypted information does not expose passwords in clear text.

How to Configure Tracing for the PowerShellExecutor.dll Add-On

The `PowerShellExecutor.dll` is an add-on that implements communication between the gateway and Microsoft PowerShell. You can use PowerShell to manage Exchange Server 2007 accounts. This add-on cannot share tracing facilities with the rest of the gateway and provides a similar stand-alone tracing facility as the rest of the gateway.

The trace configuration for the `PowerShellExecutor` is stored in the same registry key as the other gateway registry keys:

```
HKEY_LOCAL_MACHINE\Software\Waveset\Lighthouse\Gateway
```

You create this base key when you configure tracing through the Waveset debug pages or when you start the gateway with trace command arguments.

On shut down, the gateway writes the current `PowerShellExecutor` settings for the tracing to the registry. These settings include:

- `traceFileName`
Content. File name for the trace output (registry type REG_SZ)

Default. " "

Name of the trace file to generate for the `PowerShellExecutor` tracing. Where the name:

- Can be a fully qualified path, including the filename
- Cannot end in a slash (\)

The full path, except the file, provided in the `traceFileName` must exist.

If configured, log rotation adds a timestamp to the configured filename after rotation, when the file is no longer active. This timestamp displays in the following format:

```
yyyyMMddHHmss
```

- `traceLevel`

Content. Trace level (registry type REG_DWORD)

Default. 0 (no tracing)

Allowed. 0–4

This key is shared with the rest of the gateway. The whole gateway always provides tracing at the same level.

- `traceMaxSize`

Content. Maximum file size in bytes (registry type REG_DWORD or REG_QWORD)

Default. 100000 bytes

Minimum. 100000 bytes

Tracing text is written as UTF-8 encoded text with a byte order mark to make it portable to other systems.

- `traceMaxFiles`

Content. Number of trace files (registry type REG_DWORD)

Default. 2

Minimum. 1

This setting controls the number of trace files to keep on the system. Setting the maximum number of files to keep to **1**, causes the file to be overwritten when the maximum size is reached. The oldest file, based on last write time, is removed when the maximum number of files is reached.

- `traceConfigInterval`

Content. Time out in milliseconds (registry type REG_DWORD)

Default. 300000 ms (5 minutes)

Minimum. 60000 ms (1 minute)

All trace settings are reread from the registry based on this timeout value. In a production environment, consider setting this value to a large value, such as 24 hours, to minimize overhead.

How to Capture Dr. Watson Logs

If the Gateway encounters a serious problem and exits abnormally, you can send the resulting Dr. Watson logs to Oracle Support for analysis.

Note – You must have administrator privileges on the system to view these logs.

▼ To Capture a Dr. Watson Log:

- 1 Open the Windows Event Viewer.
- 2 Open the application log.
- 3 Look for an event with `DrWatson` source.
- 4 Open the event to view detailed information.
- 5 Ensure that the Bytes option is selected for Data.
- 6 Right-click in the display dialog and choose **Select all** from the menu.
- 7 Type `Ctrl-C` to copy the information.
- 8 Paste the information into Notepad and save the file.
- 9 Send the file in an email to Oracle Support with a detailed description of the problem. Be sure to indicate which version of the Waveset Gateway you are running.

Troubleshooting and Fixing Common Problems

Use the information provided in the following sections to help diagnose and fix problems you might encounter as you work with Waveset:

- “Working with Debugging Tools” on page 167
- “Debugging Errors Displayed in the Browser” on page 172
- “Troubleshooting Adapters” on page 173
- “Troubleshooting Auditor” on page 179
- “Troubleshooting `ClassNotFoundException` Exceptions” on page 180
- “Troubleshooting Form Problems” on page 180
- “Troubleshooting the Gateway” on page 181
- “Troubleshooting Java Code Problems” on page 182
- “Troubleshooting Low Memory Conditions” on page 183

- “Troubleshooting PasswordSync Problems” on page 186
- “Troubleshooting Reconciliation Problems” on page 188
- “Troubleshooting Repository Connection Problems” on page 189
- “Troubleshooting Server-Related Problems” on page 191
- “Troubleshooting Service Provider Problems” on page 192
- “Troubleshooting an SPML Configuration” on page 192
- “Troubleshooting Upgrades” on page 193

Note – For additional troubleshooting information, including the Waveset FAQ, go to the following URL:

https://sharespace.sun.com/gm/document-1.26.2296/IdM_FAQ.html?

You must sign up for a Share Space ID to access information provided on this site.

Working with Debugging Tools

You can use several different debugging tools to help identify and fix problems in your Waveset deployment. These tools include:

- “Using the Waveset Debug Pages” on page 167
- “Using Waveset IDE” on page 170
- “Using Waveset System Monitoring” on page 171
- “Working With Adapter Logs” on page 171
- “Debugging with DTrace” on page 171
- “Debugging with JConsole” on page 172

Using the Waveset Debug Pages

You can use Waveset Debug pages to help identify and fix problems in your deployment. For example, you can enable or disable tracing for various activities and objects, collect statistical information, verify that processes are running, or investigate bottlenecks and memory problems.

The following table describes the most commonly used Debug pages and their actual .jsp file names.

Note – For a comprehensive list of all Waveset Debug pages, open a command window and list the contents of the `idm/debugdirectory`.

TABLE 5-5 Waveset Debug Pages

| Page Name | Use This Page to |
|--|---|
| Control Timings (<code>callTimer.jsp</code>) | <p>Collect and view call timer statistics for different methods. You can use this information to track bottlenecks to specific methods and invoked APIs. You can also use options on the Call Timings page to import or export call timer metrics, such as.</p> <ul style="list-style-type: none"> ■ How long to fetch initial User view (with no resources) during a scan ■ How long to refresh initial User view (including resources) during a scan ■ How long to evaluate policy on the User view ■ How long each user scan takes (including User view fetch, policy evaluation, and so forth) ■ How long to fetch a list of users for an access scan ■ How long to evaluate the attestation rule in access review <p>Note – Call timing statistics are only collected while trace is enabled.</p> |
| Edit Trace Configuration (<code>Show_Trace.jsp</code>) | <p>Enable and configure trace settings for the Java classes provided with your Waveset installation. You can specify</p> <ul style="list-style-type: none"> ■ Which methods, classes, or packages to trace and the level of trace. ■ Whether to send trace information to a file or to standard output and how dates and times are formatted in the trace output file. ■ Maximum number of trace files to store and the maximum size of each file. ■ Specify the maximum number of methods to be cached. ■ Indicate how to write data to the trace file and whether to write data to the trace file as data is generated or to queue and then write the data to a file. |
| Host Connection Pool (<code>Show_ConnectionPools.jsp</code>) | <p>View connection pool statistics (if you are not using a data source), including pool version, how many connections were created, how many are active, how many connections are in the pool, how many requests were serviced from the pool, and how many connections were corrupted.</p> <p>You can also use the Host Connection Pool page to view a summary of the connection pools used to manage connections to the Gateway. You can use this information to investigate low-memory conditions.</p> |

TABLE 5-5 Waveset Debug Pages (Continued)

| Page Name | Use This Page to |
|--|---|
| List Cache Cleared (Clear_XMLParser_Cache.jsp) | Clear recently used XML parsers from the cache and investigate low memory conditions. |
| Method Timings (Show_Timings.jsp) | Detect and assess hotspots at a method level. Use this page to gather information from Waveset methods, including: <ul style="list-style-type: none"> ■ Method names ■ How many times the methods were called ■ How many times the methods exited with an error status ■ Average time consumed by the methods ■ Minimum and maximum times consumed by invocations of each method |
| Object Size Summary (Show_Sizes.jsp) | Detect problematically large objects that can affect your system. This page shows the size of objects (in characters) stored in the repository, including the objects' total combined size, average size, maximum size, and minimum size. Click entries in the Type column to view the ID, name, and size of the largest configuration objects in the repository. |
| Provisioning Threads for Administrator Configurator (Show_Provisioning.jsp) | View a summary of provisioning threads in use by the system (a subset of the information available in Show_Threads.jsp). |
| System Cache Summary (Show_CacheSummary.jsp) | View the following information to investigate low-memory conditions: <ul style="list-style-type: none"> ■ Administrator-associated object caches ■ System object cache ■ User login sessions ■ XML parser cache |
| System Memory Summary (Show_Memory.jsp) | View how much total and free memory is available (in MB) when you are using memory-intensive functionality, such as reconciliation, to help determine whether there is sufficient memory allocated to the JVM. You can also use this page to launch garbage collection or to clear unused memory in the JVM for investigating heap usage. |
| System Properties (SysInfo.jsp) | View information about your environment. |
| System Threads (Show_Threads.jsp) | View which processes are running to verify that automated processes are running. Includes information about the process type, process name, priority, if the process is a daemon, and if the process is still alive (running). |
| User Session Pool Cleared (Clear_User_Cache.jsp) | Use the Session Pool Clearer page to investigate low memory conditions. |

TABLE 5-5 Waveset Debug Pages (Continued)

| Page Name | Use This Page to |
|---|---|
| Waveset Properties (Show_WSPprop.jsp) | View and <i>temporarily</i> edit properties in the Waveset.properties file. Edited property settings remain in effect only until the next server restart. |
| XML Resource Adapter Caches Flushed and Cleared (Clear_XMLResourceAdapter_Cache.jsp) | Clear test XML resource adapters from the cache and use to investigate low memory conditions. |

Note – See “Working With Waveset Debug Pages” on page 84 for more information about these Debug pages.

▼ To Access Individual Waveset Debug Pages

Before You Begin

You must have the *Debug* capability to access and execute operations from the Waveset Debug pages. If you do not have the Debug capability, an error message results. Administrators and the Configurator are assigned the Debug capability by default.

- 1 Open a browser and log into the Administrator interface.
- 2 Type the following URL to open the System Settings page:

http://host:port/idm/debug

where:

- *host* is the local server on which you are running Waveset.
- *port* is the number of the TCP port on which the server is listening.

From this page, you can enable or disable tracing for various Waveset activities and objects and use the information displayed on these pages to troubleshoot problems in your deployment.

Some Debug pages are not linked to the System Settings page, and you must type the page's .jsp file name to open the page. For example:

http:// host:port/idm/debug/pageName.jsp

Where *pageName.jsp* is the particular Debug page you want to open.

Using Waveset IDE

The Oracle Waveset Integrated Development Environment (Waveset IDE) is Java application that enables you to view, customize, and debug Oracle Waveset (Waveset) objects in your deployment.

Specifically, the Waveset IDE provides a graphical Debugger that you can use to debug &Waveset forms, rules, and workflows. You can use this Debugger to set breakpoints and watches, step through code, examine and modify variables, examine classes and the callstack, follow threads, and run multiple sessions.

Instructions for installing and configuring the Oracle Waveset Integrated Development Environment (Waveset IDE) are now available from the following URL:<https://identitymanageride.dev.java.net>.

Using Waveset System Monitoring

You can configure Waveset system monitoring to track system events. System monitoring collects and aggregates statistics at various levels to present a real-time view of system events, based on your specifications.

Viewing this information in dashboard graphs enables you to quickly assess system resources, view abnormalities, understand historical performance trends, and interactively isolate problems before looking at audit logs. Although dashboards do not provide as much detail as audit logs, dashboards can indicate where to look for problems in the logs.

For more information about dashboards and system monitoring, see [Chapter 8, “Reporting,” in *Oracle Waveset 8.1.1 Business Administrator’s Guide*](#).

Working With Adapter Logs

Adapter logs capture information about the adapter that is currently processing. You can use this information to monitor the adapter’s progress and to diagnose and debug adapter problems.

Note – You must enable tracing and identify the methods for which tracing is requested before any logging can occur. Also, your customized adapter must include calls that create log entries for new methods.

Nearly every adapter has its own log file, path, and log level. You can specify the level of detail captured by the adapter log, along with these other values in the Logging section of the Synchronization Policy for the appropriate Waveset or Service Provider user type.

For more information about using adapter log files as a debugging tool, see [“Troubleshooting Adapters” on page 173](#).

Debugging with DTrace

DTrace is a comprehensive, dynamic tracing framework for the Solaris operating environment. DTrace provides more than 30,000 probes into your production system and integrates user- and kernel-level tracing. You can use DTrace to monitor JVM activity. This facility also allows you to use the D language (similar to C or awk) to trace arbitrary data and expressions.

Debugging with JConsole

The Java Monitoring and Management Console (*JConsole*) is a Java Management Extension (*JMX*) technology-compliant graphical management tool bundled with JDK 5 (and later). JConsole connects to a running JVM and gathers information from the JVM MBeans in the connected JMX agent.

For example, you can use JConsole to

- Detect low memory
- Enable or disable garbage collection
- Enable or disable verbose tracing
- Detect deadlocks
- Control Waveset log levels
- Access information about operating systems resources (Oracle's platform extension)
- Monitor and manage MBeans
- View information about the JVM and monitored values, threads running on the application, and class loading

Note – For more information about JConsole, see the article titled, *Using JConsole to Monitor Applications*. You can view this article from the following URL:

<http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html>

Debugging Errors Displayed in the Browser

If a red error message displays in the Waveset interface after you have performed an action, you might be able to view more complete information and further analyze the error by viewing and saving the page source.

To view the page source

- If you are using Internet Explorer, select View → Source from the menu bar.
- If you are using Netscape, select View → Page Source from the menu bar.

If you still need help resolving the problem,

1. View the page source, and then select File → Save to save the file to your system.
2. Locate the error in your saved file.
3. Send the error information, the URL from the page where the problem occurred, and a description of the problem in an email to Oracle Support for resolution assistance.

Troubleshooting Adapters

To troubleshoot an adapter, review the adapter's log file. Almost all adapters write their resource settings to a log file, and you can use this information to confirm that the adapter started and that all setting changes have been saved.

Note – You must enable tracing and identify the methods for which tracing is requested before any logging can occur. Also, your custom adapter must include calls that create log entries for new methods.

If necessary, review [“Tracing the Waveset Server” on page 133](#) for instructions about how to enable tracing.

Most adapter log files are located in the `$WSHOME/config` directory and they are named `WSTrace1.log`.

Active Sync-enabled adapters that make log calls to the `ActiveSyncUtil` instance create a log file or set of log files in a directory specified by the `Log File Path` resource attribute. Be sure to check these log files for additional Active Sync-related log entries.

The information in this section is organized as follows:

- [“To Debug an Adapter” on page 173](#)
- [“To Debug LoginConfig Changes” on page 175](#)
- [“To Debug Adapter Connection Problems” on page 176](#)

To Debug an Adapter

Follow these general steps to debug your custom adapter.

1. Create a test program for your adapter, and be sure this Java file performs the following basic functions:
 - a. Create a new resource
 - b. Create a user
 - c. Get a user
 - d. Update a user
 - e. Delete a user
 - f. Perform create, get, update and delete operations on multiple users

Note – A sample test file (`SkeletonResourceTests.java`) is provided in the `/REF` directory on your installation CD.

2. Set an appropriate logging level for debugging.

For example, for the first debugging pass, increase the logging level to 4 (maximum debugging output), set the log file path, and specify a maximum file size.

When you start the adapter, all of the resource settings are written to the log file. You can use this information to validate that the adapter started and that all setting changes were saved.

3. Compile and test your adapter.

- To compile the test program, open a command window and enter the `javac -d . test/filename.java` command. This command creates the class file in the appropriate `com/waveset/adapter/test` directory.
- To test your new adapter using this class file, be sure that your compiled adapter is in the `com/waveset/adapter` directory and use the following command to run the adapter:

```
java -D waveset.home=path com.waveset.adapter.test.  
MyResourceAdapter
```

4. Create an HTML help file for your resource.

Note –

- Example help files are supplied in the `idm.jar` file located in the `com/waveset/msgcat/help/resources` directory.
 - See [Oracle Waveset 8.1.1 Deployment Reference](#) for information about how to include online help with the application.
-

5. (For Active Sync-enabled adapters only) To reset synchronization on the last resource, delete the `XmlData SYNC_resourceName` object.

6. Read the error log and modify the adapter.

7. Reset the logging level.

For example, specifying Level 2 debugging yields information about the adapter settings and any errors, but limits the amount of log detail to a manageable level.

8. Before starting Waveset, you must identify the new adapter in the `$WSHOME/config/Waveset.properties` file by placing the adapter name under the `resource.adapters` entry or Waveset cannot recognize the adapter.

9. Install your adapter and its associated help file into Waveset.

Note – Before Waveset can recognize an instance of a new adapter in the display, you must create a new resource of that type from the List Resource page.

From this page, select **New** → *new adapter* and use the Resource Wizard to create the new adapter.

10. Use Waveset to create a resource and a user on that resource.

Tip – When troubleshooting an Active Sync-enabled adapter, if you edit the `XmlData SYNC_resourceName` object to remove the `MapEntry` for the Active Sync synchronization process from the Debug page, the adapter starts over from the first detected change.

If you used the IAPI event, you must set the `Property()` method to store synchronization state for the resource, such as a last change processed value. Setting this method is very useful for troubleshooting adapters. You can set the adapter to run and ignore past changes. Subsequently, you can modify the adapter and see your change results in the adapter log file.

If your resource is an Active Sync resource, you might see additional information if you enable logging on the resource edit page. Set the logging level (0-4) and the file path where the log file will be written (as `resource_name.log`).

11. (For Active Sync-enabled adapters only) Restart synchronization for the last resource.

To Debug LoginConfig Changes

To debug LoginConfig-related changes to your adapter, you must

1. Enable trace for the selected files and trace the following classes at Method/Class Level 1 trace:
 - `com.waveset.security.authn.WSResourceLoginModule`
 - `com.waveset.session.LocalSession`
 - `com.waveset.session.SessionFactory`
 - `com.waveset.ui.LoginHelper`
 - `com.waveset.ui.web.common.ContinueLoginForm`
 - `com.waveset.ui.web.common.LoginForm`
2. Test Single Sign-On (SSO) pass-through authentication login through Telnet as follows:
 - a. After correctly configuring the SSO login module, telnet directly to the HTTP port and send an HTTP request to `login.jsp`.
 - b. Paste the following request, which contains an SSO login module that looks for the `sm_user` HTTP header, into your telnet session:

```
HEAD /idm/login.jsp HTTP/1.0
Accept: text/plain,text/html,*/.*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: LOCALHOST
sm_user: Configurator
```

A trace displays to indicate that your user has logged in correctly. For example:

```
2003.07.08 14:14:16.837 Thread-7
WSResourceLoginModule#checkForAuthenticatedResourceInfo()
Found authenticated resource accountId, 'Configurator@Netegrity
```

```

SiteMinder'on Identity Manager user 'Configurator'.
null null 2003.07.08 14:14:16.837
Thread-7 WSResourceLoginModule#checkForAuthenticatedResourceInfo()
Exit null null 2003.07.08 14:14:16.837 Thread-7 WSResourceLoginModule#login()
Exit, return code = true null null 2003.07.08 14:14:16.847
Thread-7 LocalSession#login() Login succeeded via Netegrity SiteMinder
null null 2003.07.08 14:14:16.847 Thread-7 LocalSession#login()
Overall authentication succeeded null null 2003.07.08 14:14:16.897 Thread-7
LocalSession#checkIfUserDisabled()
Entry null null 2003.07.08 14:14:16.897 Thread-7
LocalSession#checkIfUserDisabled() Exit
null null 2003.07.08 14:14:16.927 Thread-7 LocalSession#login()
Exit null null
    
```

To Debug Adapter Connection Problems

This section describes methods for debugging some common adapter connection problems.

The topics in this section are organized as follows:

- “Adapter Authentication Problems” on page 176
- “Active Sync Adapter Problems” on page 177
- “Domino Gateway Adapter Problems” on page 177
- “Mainframe Host Adapter Problems” on page 177
- “PeopleSoft Adapter Problems” on page 178
- “SAP Adapter Problems” on page 179
- “UNIX Adapter Problems” on page 179

Note – Generally, you can identify adapter connection issues by tracing the adapter class `com.waveset.adapter.adapter_classname`. For example:

```
com.waveset.adapter.ADSIResourceAdapter
```

If necessary, review instructions for enabling trace in [“Tracing the Waveset Server” on page 133](#).

Adapter Authentication Problems

Some common authentication problems include

- Missing authentication properties

You must include a property name for the specified DataSource type in the set of names output in the trace.
- An Waveset user is missing a matching resource account

If authentication succeeds for the resource adapter, but an exception occurs indicating that no Waveset user could be found with a matching resource account ID, be sure that the resource accountId associated with the user is the same as the accountId returned by your resource adapter’s authenticate method.

To verify the Waveset user's resource `accountId`, review Waveset's Edit Trace Configuration page (`debug/Show_Trace.jsp`). If a mismatch exists, change the content of the name being returned by your `authenticate` method or change your resource's ID template. The template must generate a resource `accountId` that matches the `accountId` being returned by the `authenticate` method.

Active Sync Adapter Problems

The most common problems with custom Active Sync adapters are form-related. These errors generally occur because you have not provided necessary information, such as password or email information, in a required field.

Waveset prints form validation errors to the adapter log after the final XML of the view. For example:

```
20030414 17:23:57.469: result from submit (blank means no errors):  
20030414 17:23:57.509: Validation error: missing required field password
```

Waveset also prints all messages to the adapter log. These messages include account creation and update times, adapter errors, and a summary of the schema map data.

Active Sync resource adapters store information about the last change processed in the `SYNC.resourceName XMLData` object.

Domino Gateway Adapter Problems

Following are some common Domino gateway and adapter configuration errors and instructions for fixing these problems:

- If an error message states the 'New Domino Gateway' resource is not accessible and the connection is refused, try stopping and restarting the Oracle Waveset Gateway.
- If an error message states no ID file name is specified and the path to the userID file is set incorrectly, specify a target location for the userID file and edit the resource adapter to set this attribute to a correct path. Typically, the target location for the userID file is the directory into which you installed the Gateway.
- If an error message states you are not authorized to use the server, you have not set the correct access permissions for the ID file. Specify the correct permissions for this file and retry.

Mainframe Host Adapter Problems

When RACE, ACF2, or TopSecret host adapters fail to reuse or cache connections, users are forced to log in frequently, which negatively impacts performance. Generally, the cache timeout setting causes this problem.

To check the cache timeout setting, trace Waveset's adapter connection pool as follows:

1. From Waveset's Edit Configuration Object page, trace the `com.waveset.adapter.HostConnPool#reapConnections` method at level 4.
If necessary, review instructions for enabling trace in [“Tracing the Waveset Server” on page 133](#).
2. Capture trace for a sufficiently long period of time (*at least 30-60 minutes*), while the adapter performs operations.
3. Review the trace output in the application server `stdout` or trace file and look for `Info reaping connection` entries.

If this entry occurs more than once every 30 minutes, you have a good indication that connections are being timed out unnecessarily.

To resolve this problem, increase the `Idle Timeout` resource attribute value to prevent connections from being reaped too frequently. The `Idle Timeout` attribute controls how long a connection remains idle before the connection is logged out. The default value is 90 seconds, which causes new logins to occur frequently.

Ideally, specify a value that is greater than the average idle time for your deployment environment. For example, adjust the `Idle Timeout` attribute to 30 minutes (1800000 milliseconds) or more.

PeopleSoft Adapter Problems

This section describes methods for troubleshooting the following PeopleSoft adapter problems:

- Executing a “Test Connection” from the PeopleSoft resource adapter causes a failure with a generic exception.
 1. Open the `Waveset.properties` file and set `exception.trace=true`.
 2. Retry the test connection, and if you see the following results:

```

FormState: expansion complete
java.lang.NullPointerException: PSProperties not loaded from file
WavesetException:
WavesetException:
==> com.waveset.util.WavesetException:
FormState: derivation
    
```

Log in to the PeopleSoft web interface to verify that you are using the correct UID and password.

- The PeopleSoft application server logs are not showing login attempts.

This problem generally occurs because you did not use the `psj oa.jar` file supplied with the PeopleSoft installation to which you are connecting. (See the [Oracle Waveset 8.1.1 Resources Reference](#) for more information about the PeopleSoft resource adapter).

SAP Adapter Problems

If an error results when you try to test the connection from an SAP or SAP HR Active Sync adapter to the SAP system, open a command window and run this command from your installation directory WEB-INF/lib:

```
java -jar sapjco.jar
```

The `sapjco.jar` command shows which version of the SAP Java Connector (JCO) is installed and whether the adapter is installed correctly. The command also returns the Java Native Interface (JNI) platform-dependent and the RFC libraries that communicate with the SAP system.

If these platform-dependent libraries are not found, consult the SAP documentation to find out how to correctly install the SAP Java Connector.

UNIX Adapter Problems

This section contains information about debugging some common problems with UNIX adapters.

- If you see timeout errors when provisioning to a UNIX resource adapter, you can determine where the provisioning process is failing by tracing the `com.waveset.adapter.ScriptedConnection` method at Method/Class level 4, which gives you the maximum logging output.
- When becoming root to perform administrative commands, if the resource adapter executes a `su root` command instead of a `su - root` command, the environment does not inherit any of the custom environment variables defined for root; including any custom prompts (environment variable of PS1).

When configuring a UNIX adapter, you can determine which prompt to enter into the Root Shell Prompt field as follows:

1. Telnet or ssh to the system as the user you specified in the Login User field.
2. After typing the password and logging in, type **su root** without a dash and press return.
3. Type the root password.
4. The next prompt displayed is the prompt you must enter into the Root Shell Prompt field.

Troubleshooting Auditor

You can trace the following methods to troubleshoot issues with Identity Auditor:

- `com.sun.idm.auditor.policy` to trace issues with Audit Scans.
- `com.sun.idm.auditor.accessreview` to trace issues with Access Reviews.

- `com.sun.idm.auditor.report` to trace issues with Audit Reports.
- `com.sun.idm.auditor.view` to trace issues with Auditor Views.

In addition, you can set the following hidden flags by modifying Forms or TaskDefinitions:

- **Audit Policy Scan Task - maxThreads (int)**. Number of concurrent threads used. (Defaults to 5).
- **Audit Policy Scan Task - userLock (int)**. Time (in milliseconds) to wait to acquire a lock on the User object. (Defaults to 5000)
- **Audit Policy Scan Task - scanDelay (int)**. Time (in milliseconds) to delay between launching a new scan thread. (Defaults to 0, no delay)
- **Audit Policy Scan Task - clearuserLocks (boolean)**. If `true`, the scanner attempts to clear the user lock before scanning.

In addition, the Show Timings page (`/debug/callTimer.jsp`) provides the following information:

- How long it takes to fetch the initial User view, with no resources, during the scan
- How long it takes to refresh the User view, including resources, during the scan
- How long it takes to evaluate the policy on the User view
- How long each user scan takes, including User view fetch, policy evaluation, and so forth
- How long it takes to fetch a list of users for access review
- How long it takes to evaluate the attestation rule in access review

Troubleshooting ClassNotFoundException Exceptions

In the event of a `ClassNotFoundException` exception error, verify that the missing class is included in the server's classpath. If the classpath is configured properly, try configuring your application server such that the application class loader loads before the parent class loader. Sometimes loading the application classpath before the server classpath can resolve this issue. Consult your application server documentation for instructions.

Troubleshooting Form Problems

This section describes some common form problems and how to fix these problems.

- You added a `<script>` tag to your user form, but see the following exception when trying to access the user form:

```
java.lang.NoClassDefFoundError: org.mozilla/javascript/NativeScript
```

You must put the `WEB-INF/lib/javascript.jar` file into the application server's classpath. For example:

- If you installed Tomcat as a service (Windows only), edit the following registry key:
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tomcat\Parameters\JVMOption Number 0`
 Append the path to your jar file to the end of the JVM Option Number 0 key string value, wherever it is located. For example:
`;C:\tomcat\lib\javascript.jar`
- If you just started Tomcat from the command line, move the jar file into the `tomcat/lib` directory.

- You customized a Tabbed User Form, but when you try accessing the form through User Creation, you see the following exception:

```
com.waveset.util.WavesetException: Maximum form stack depth exceeded
```

You are limited to pushing 1000 hard-coded elements on the stack and you exceeded this limit.

If you added a check to detect Field containers that include a FieldRef or Field that matches the name of the Field container, you might have inadvertently created a circular reference on the form. To fix this problem, change the FieldRef or Field name.

- You used a MultiSelect field in a form for LDAP users, assigned groups to a user in this field, edited the user again, and then see the same group on both sides of the MultiSelect. For example:

- **Left side value.** `cn=Group1,dc=test,dc=Com`

- **Right side value.** `cn=Group1,dc=test,dc=com`

In this example, the LDAP baseContext resource field is set to `dc=test,dc=com` and the LDAP groups are listed as `dc=test,dc=Com`, which causes a problem because LDAP is not case sensitive, but the MultiSelect widget is case-sensitive.

To alleviate this problem, change the baseContext on the LDAP resource to match the case in your LDAP resource `dc=test,dc=Com` or use the `<uppercase>` XPRESS function to uppercase both the left and right side displays.

Troubleshooting the Gateway

When troubleshooting the Oracle Waveset Gateway, it is often useful to run the Gateway from the command line. Using command line options allows you to input a wider range of start-up options, which includes starting the Gateway as a normal application instead of a service and running the Gateway on a different port.

Note – You must kill the Waveset Gateway as a service before running it from the command line. For example, type

```
gateway.exe -k
```

The following table describes the Gateway command line arguments.

| Argument | Description |
|----------|---|
| -i | Install this program as an NT service, with specified startup |
| -r | Remove this program from the Service Manager |
| -s | Start the service |
| -k | Kill the service |
| -t | Set start-up for an existing service |
| -d | Debug, and run as a regular application |
| -p | Specify a TCP/IP port number (Default is 9278) |
| -f | Specify the path to the trace file |
| -l | Specify the level of tracing (Default is 0, no information) |
| -m | Specify the maximum trace file size in kilobytes |
| -v | Display the version |

Usage: gateway -i n -r -s -k -t n -d -p n -f name -l n -m n -v.

Note –

- The -d and -s options are mutually exclusive.
- See [“To Enable Tracing from the Command Line” on page 161](#) for more information about the Gateway tracing levels.

You can also use the Waveset Gateway Debug page ([debug/Gateway.jsp](#)) to troubleshoot the Gateway. See [“How to Configure Gateway Tracing” on page 160](#) for more information.

Troubleshooting Java Code Problems

If you have the basic Java programming skills required to work with Waveset, you should be able to diagnose and resolve most Java code problems.

However, a fairly common problem occurs where someone opens a connection to the database but does not close the connection properly. If you do not close the connection properly, performance issues result.

Troubleshooting Low Memory Conditions

This section describes several methods you can use to investigate low memory conditions, including:

- [“Monitoring the Admin Cache” on page 183](#)
- [“Monitoring the Waveset Debug Pages” on page 185](#)
- [“Using JConsole” on page 186](#)

Monitoring the Admin Cache

If you start experiencing slow response times when accessing users or when logging into accounts, you might have exceeded the admin cache size, which results in the admin cache being disabled.

The *admin cache* is a cache maintained on each server that contains all `WSUser` objects that are considered to be administrators. Waveset loads this cache when the server starts up, and automatically maintains this cache by watching for `WSUser` object changes.

Users are generally considered administrators when they have at least one capability and control an object group. If the number of administrator users exceeds the number specified for the cache, Waveset disables the cache, which causes poor system performance.

The default size of the admin cache is 10K. If you have more than 10K administrators, Waveset does not initialize the admin cache. You can adjust this limit by changing the `admincache.rowlimit` property in `Waveset.properties`. The cost of increasing admin cache size is the additional use of heap memory. The `WSUser` objects in the cache consume memory for the entire lifecycle of the server.

Waveset reads the `admincache.rowlimit` property at server startup, so any change to this property requires the server to be restarted.

Note – You must set this property on each server because `Waveset.properties` is not shared across servers.

Waveset no longer requires approvers to be administrators, as in earlier versions of the product, which potentially reduces the number of administrators in the cache and improves system performance.

Troubleshooting

Out-of-memory conditions can be difficult to troubleshoot, but you can start by ensuring that sufficient memory was initialized at start up and by checking that Waveset is tracing memory usage levels and alerts.

If you are debugging problems with the admin cache, tracing `com.waveset.server.InitAdminCacheThread` at level 4 can show the following information:

- What were the start-up timings
- How long did it take to load the admin and auth caches
- What data was loaded
- What happened when you added or edited a user from the Waveset Administrator interface

Tracing Methodology

You can use `com.waveset.server.InitAdminCacheThread` to trace start-up timings. For example,

EXAMPLE 5-7 Sample Trace Output

```
20091124 12:02:56.203 InitAdminCacheThread(0x010dae16)
InitAdminCacheThread#initAdminCache() Info Getting 9 admins from admin cache
took 0m0.266s 20091124 12:02:56.437 InitAdminCacheThread(0x010dae16)
InitAdminCacheThread#initAdminCache() Info Loading auth cache for
9 admins took 0m0.234s
```

Solution

Changing `admincache.rowlimit` in `Waveset.properties` only takes affect after you restart the server. Increasing the cache size to accommodate all of the administrators in the repository may increase server performance if there is enough memory to hold the cache without resulting in excessive JVM garbage collection.

Note – Be aware that this solution does not scale.

Available JVM heap memory is a limitation to consider when using the `admincache.rowlimit` property to configure cache size, particularly when implementing and tuning an x64 system.

Testing

To test your configuration,

1. Set `admincache.rowlimit=xxxxx` on VM (where `xxxxx` denotes the cache size setting).
2. Trace `com.waveset.server.InitAdminCacheThread` at level 4.
3. Review the trace file output.

- Verify that the cache was loaded.
- Confirm that you can disable the `admincache.rowlimit` setting.
- Compare the machine's memory usage before and after you disable the `admincache.rowlimit` setting.
Ensure that the operating system has sufficient memory available to run effectively. For example, check for excess paging, CPU activity, and so forth.
- Check the JVM memory usage.
- Verify how long it takes the cache to load and compare this time with that of a smaller cache, such as 1000.

Monitoring the Waveset Debug Pages

Note – You must have the *Debug* capability to access and execute operations from the Waveset Debug pages. Administrators and the Configurator are assigned this capability by default.

If you do not have the Debug capability, an error message results.

You can open the following Waveset Debug pages from the Administrator interface to monitor how much memory is being used by your system:

- **Host Connection Pool** page (`debug/Show_ConnectionPools.jsp`). View a summary of connection pool statistics (if you are not using a data source), including the pool version, how many connections were created, how many are active, how many connections are in the pool, how many requests were serviced from the pool, and how many connections were destroyed.
You can also use the Host Connection Pool page to view a summary of the connection pools used to manage connections to the Gateway. You can use this information to investigate low-memory conditions.
- **List Cache Cleared** (`debug/Clear_XMLParser_Cache.jsp`). Clear the cache of recently used XML parsers.
- **Private collection pool** (`debug/Show_JDBC.jsp`). View a summary of the cache of Java DataBase Connectivity (JDBC) connections being used by the repository and some resource adapters.
- **System Memory Summary** page (`debug/Show_Memory.jsp`). View the used and total memory in the system. You must click the Garbage Collect button to get the most current used memory value.
- **System Memory Summary** page (`debug/Show_Memory2.jsp`). View an updated `Show_Memory.jsp` page that allow you to clear all unused memory in the JVM so you can investigate heap usage.

- **User Session Pool Cleared** (`Clear_User_Cache.jsp`). Clear the cached sessions for recently logged in users.
- **XML Resource Adapter Caches Flushed and Cleared** (`Clear_XMLResourceAdapter_Cache.jsp`). Clear the cache of the test XML resource adapter.

Using JConsole

Use the Java Monitoring and Management Console (*JConsole*) to detect low memory and deadlocks. JConsole is a Java Management Extension (JMX) technology-compliant graphical management tool that is co-packaged with JDK 5 (and later).

JConsole accesses the memory system, memory pools, and MBeans garbage collector to provide information about memory use such as memory consumption, memory pools, and garbage collection statistics. In addition, You can use JConsole to monitor MBeans for information about current heap memory use and non-heap memory use.

Note – For information about using JConsole to monitor applications that run on the Java platform, see *Using JConsole to Monitor Applications*. This document is available from the following URL:

<http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html>

Troubleshooting PasswordSync Problems

When you are trying to troubleshoot problems with PasswordSync, review the following logs for information:

- **PasswordSync Error Logs.** PasswordSync writes all failures to the Windows Event Viewer. (For more information about Event Viewer, see Windows' Help.) The source name for error log entries is *PasswordSync*.
- **PasswordSync Trace Logs.** PasswordSync writes all trace logs to the file location specified when you configured tracing. See “Using the PasswordSync Configuration Tool” on page 145.

Some common PasswordSync problems and solutions include

- PasswordSync is not propagating password changes from the Windows server to Waveset. PasswordSync relies on the registry settings when creating a connection from Active Directory to the Waveset Server. PasswordSync reads the registry and processes the settings, but PasswordSync does not perform any checks to see if it can create a connection.

The following example shows a registry entry for a PasswordSync server. This example includes the default registry setting values, but does not show all of the settings used by PasswordSync.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Sun Microsystems\Sun Identity Manager\PasswordSync]
"reinitIntervalMilli"=dword:1B77400
"securityIgnoreCertRevoke"=dword:00000000
"securityAllowInvalidCert"=dword:00000000
"directMode"=dword:00000001
"lhuser"="config"
"lhcred"="rsVtQZpa5Ys="
"endpointURL"="http://10.10.10.10:8080/idm/servlet/PasswordSync"
"installdir"="C:\Program Files\Sun Microsystems\Sun Identity Manager
PasswordSync"
"tracelevel"=dword:00000000
"tracemaxKB"=dword:00002710
"tracefile"="C:\Program Files\Sun Microsystems\Sun Identity Manager
PasswordSync\trace.log"
```

If you have not enabled tracing at an appropriate level, PasswordSync does not log connection failures in much detail. To see more detailed trace information, edit the PasswordSync registry settings as described on [“Editing the Registry Keys” on page 146](#). Specify tracelevel 4 to output the maximum trace information, and change the tracefile value to point to a writable file. For example:

```
"tracelevel"=dword:00000004
"tracefile"="C:\Program Files\Sun\IdentityManager\PasswordSync\pwicsvc.log"
```

The registry settings will be reread based on the `<i>reinitIntervalMilli</i>` setting in the registry. After rereading the registry settings, PasswordSync automatically starts or stops tracing, depending on the trace parameters set in the registry. For each intercepted password change, PasswordSync logs the actions taken to push the password to Waveset.

- If a connection fails during creation, you might encounter the following situations:

Note – Each of these situations has its own error code and set of log entries. Waveset removes the date, time stamp, and process number from these entries to keep them short.

- An incorrect or unreachable URL error that occurs when the server cannot be reached, is not running, or does not reply with a correct response.
 - Check that PasswordSync can access the server and page.
 - Be sure the server is running and that you have configured your firewalls and routers correctly.
 - Check the application server to be sure it is running, and that PasswordSync can connect to the endpointURL without the application path. If PasswordSync can does not return a page or an error, the application server is not running.
 - Check the servlet response by opening the endpointURL in a standard browser. If you do not see an error that starts with: `com.waveset.util.WavesetException` see if the servlet is compiling and available..

- ❑ JMS usage for PasswordSync relies on the `jms.jar` being available in the classpath. The following exception message displays if you access the endpointURL without the correct file in place:

```
com.waveset.util.WavesetException: A JMS request arrived, but
JMS PasswordSync is unavailable. Is JMS jar file available?
```

- An incorrect user name error generally occurs when the `lhuser` entry is incorrect. Use the `Configure.exe` utility to replace the user or replace the `lhuser` registry key value with a valid userID.
- An incorrect password error generally occurs when the password stored in the `lhcred` entry is not correct when used in combination with the userID stored in `lhuser`. Use the `Configure.exe` utility to replace the password, but do not manually edit the `lhcred` registry key.
- A garbage in the password entry error generally occurs when the registry key is corrupted and or when the registry key is manually edited, which causes garbage in the password entry.
- This situation causes the process to hang in `RAEncryptor::Decrypt3DES` and PasswordSync cannot decrypt the entry. Use the `Configure.exe` utility to replace the password.

Troubleshooting Reconciliation Problems

When you are trying to troubleshoot problems with a reconciliation task, review the Reconciliation Status Debug page (`debug/Show_Reconciler.jsp`) to see what the resource threads are working on.

Some common reconciliation problems include

- Reconciliation does not start.
 1. Open the System Threads Debug page (`debug/Show_Threads.jsp`) to see if the `Reconcilerserver_name` task is running.
 2. If the task is not running, open the System Settings page and click Trace Scheduler. Running the Scheduler restarts the Reconciler `server_name` task.
 3. Check the System Threads page again, and if the Reconciler `server_name` task has not restarted, the Scheduler might be hung.
 4. Restart your application server.
- Reconciliation fails for a certain user object.
- If the object exists in Waveset, try editing the object directly. If the Waveset account does not exist, load the one account from the resource.
- Verify the reconciliation user has the proper access rights.
- Try extracting the object to a file using the same code path as reconciliation.

- Open the System Memory Summary Debug page (`debug/Show_Memory.jsp` or `debug/Show_Memory2.jsp`) and verify you have enough free memory.

Troubleshooting Repository Connection Problems

Waveset's `lh` commands are very useful when you are troubleshooting connection problems. These commands use Waveset's web application installation, but remove the application server from the equation.

This section describes the following

- [“Using `lh` Commands to Debug Problems” on page 189](#)
- [“Testing DataSource Connections” on page 190](#)

Using `lh` Commands to Debug Problems

This section describes how to use the `lh` commands; starting with using the more basic commands and progressing to using commands that exercise most of Waveset.

- [“Using `lh setRepo -c -n`” on page 189](#)
- [“Using `lh setRepo -c -v`” on page 189](#)
- [“Using `setRepo`” on page 190](#)
- [“Using `lh console`” on page 190](#)

After becoming familiar with these debugging tools, you can develop your own variations for using these `lh` commands.

Using `lh setRepo -c -n`

Use the `lh setRepo -c -n` command to perform the most basic connection test, which allows you to examine the current repository location *without connecting*. You can use this command to verify that parameters, such as URL and JDBC driver, are correct.

- If the connection is successful, you can read the `ServerRepository.xml` bootstrap file.
- If the connection fails, try to solve this failure first. A decryption error is the most common cause of this failure. For example, you might have a J2EE mismatch or classpath conflict.

Using `lh setRepo -c -v`

Use the `lh setRepo -c -v` command to *connect to* and examine the current repository location. (The `-v` provides verbose output.) You can use this command to exercise almost all of the Repository code without requiring the Waveset server.

- If the connection is successful, then you are successfully connected to the current repository location.
- If the connection fails, try to solve this problem first. Solving your connection problem can be very helpful in resolving DNS, firewall, or remote connection privilege problems.

Note – For more information, see [“Testing DataSource Connections” on page 190](#).

Using setRepo

Use the `setRepo` command throughout the debugging process, to specify a new repository location or to set the repository to the same location.

You can use this command to confirm that all of the necessary components, such as the JAR files, are in place. The `setRepo` command also lets you vary connection information, such as `userId` and `password`, to debug table ownership or privilege problems.

Using lh console

Use this command to actually start an Waveset Server using the JAR files in the `WEB-INF/lib` and the classes in `WEB-INF/classes` under `WSHOME`. The `lh console` command uses your Waveset installation environment and actually starts the Waveset application, but *removes* the application server from the equation.

- If the connection is successful, the problem is specific to the application server environment or configuration.
- If the connection fails, review the failure messages.
- If the connection failure is the same as the application server failure, you have reproduced this failure with significantly fewer variables.
- If the failure appears to be different from the application server failure, try fixing the Waveset connection problem first because there are fewer variables and more of these variables are under Waveset control.

Testing DataSource Connections

If you are testing a `DataSource` connection, the `lh setRepo -c` command might fail.

This failure is especially likely if you configured Waveset to use the application server’s database connectivity service or the application server’s directory service. These services often work only in the environment that a running application server provides to a web application.

Initially, approach the `DataSource` configuration you want in a step-by-step manner. Once you are comfortable with these steps, you can adapt your approach to suit your needs.

1. Try using a direct JDBC `DriverManager` connection, such as a non-`DataSource` connection, that bypasses the application server’s database connectivity service.
2. Use a `DataSource`, but store the `DataSource` object in a directory service *other than* application server’s directory service.

Note – If you have no other directory service available, you can download a free directory service, including the reference implementation of JNDI that uses only the local file system.

If these steps work, you have localized the problem to the application server.

Then, if useful, you can add the application server's database connectivity service or the application server's directory service, whichever service works outside of the environment that the application server provides to web applications.

Troubleshooting Server-Related Problems

This section describes how to diagnose and fix some common server-related problems.

Analyzing the Server Logs

You can analyze your application server logs for fatal errors and other server-related problems.

To troubleshoot server problems, use the application server's Administrative Console to increase the logging level for each module. For more information, see the product documentation supplied with your server.

Most application servers have a standard location for standard out files (`stdout`) and standard error files (`stderr`) for the JVM running the application server. To analyze your application server logs, locate the logs directory or the log files specified for your Waveset application server.

- Open the `stdout` file to view minor messages and other tracings.
- Open the `stderr` file to view fatal and critical exceptions.

Note – You will see Waveset start up and shut down the messages in this trace output.

Fixing Common Server-Related Problems

- Beginning with Sun Identity Manager Version 7.1, a sealing violation exception occurs in the application server log when you use Waveset with Oracle 10g on Sun Java System Application Server Enterprise Edition 8.2.

This exception generally occurs if you are using more than one Java Archive file (JAR file) containing Oracle JDBC drivers.

To prevent this problem, be sure the `CLASSPATH` contains only one JDBC driver JAR file, and that you use the `ojdbc14_g.jar` provided with Oracle 10g. In addition, you must use the `ojdbc14_g.jar` provided by the Oracle 10g installation to ensure correct operation.

- Waveset depends on, and includes, some jar files that may also be provided by the application server, which can cause a conflict.

For example, an error can result when you deploy:

- Sun Identity Manager 8.1 on jBoss
- Sun Identity Manager 7.0 on Apache Tomcat 4.1
Tomcat ships with a `commons-logging.jar` file in `$TOMCAT_HOME/server/lib/commons-logging-1.1.jar` and Identity Manager 7.0 ships with a `commons-logging` file in `$TOMCAT_HOME/webapps/idm/WEB-INF/lib/commons-logging.jar`.

The version of the jar file provided by the application server must be newer than the one provided by Waveset. To determine which jar file is newer, inspect the `Specification-Version` and `Implementation-Version` meta tags in the `META-INF/MANIFEST.MF` file.

If you deploy Waveset on certain application servers and errors result, you can remove the Waveset `commons-*.jar` files from the `Waveset WEB-INF/lib` directory to fix the problem.

Troubleshooting Service Provider Problems

If you are using the Oracle Waveset Service Provider End User Login page in WebSphere, and a `javax.servlet.UnavailableException` occurs with a 404 error displayed in your browser, you must reset some properties in the IBM 1.5 JDK.

Use the following steps:

1. In the `was-install/java/jre/lib` directory, rename the `jaxb.properties.sample` to `jax.properties` and uncomment these two lines:

```
javax.xml.parsers.SAXParserFactory=
    org.apache.xerces.jaxp.SAXParserFactoryImpl
javax.xml.parsers.DocumentBuilderFactory=
    org.apache.xerces.jaxp.DocumentBuilderFactoryImpl
```

2. Save the file and restart the application server.

Troubleshooting an SPML Configuration

To test an SPML configuration:

1. Open the Connect page and click Test.
A dialog indicating that the connection was successful pops up.
2. Open the Schema page and click Submit.

The system displays a hierarchical view of the schemas supported by the Waveset server.

If you cannot establish a successful connection

- Double-check the URL you entered.
- If the error you receive contains phrases such as no response or connection refused, then the problem is most likely the host or port used in the connection URL.
- If the error suggests that a connection was made, but the web application or servlet could not be located, the problem is most likely in the `WEB-INF/web.xml` file.

Troubleshooting Upgrades

If you encounter problems during the upgrade, check the upgrade log files located in the `$WSHOME/patches/logs` directory. The file names for the logs are based on a timestamp and the stage of the upgrade.

If, following an upgrade, Waveset fails to start with the following exception, your JDK/JRE may be the problem:

```
java.lang.IllegalStateException: Error attempting to decrypt:  
Given final block not properly padded
```

Verify that you are using a JDK/JRE supplied by the same vendor that you were using previously. For example, you cannot upgrade to a Sun JDK if previously you were using a JDK from IBM. To fix this problem, uninstall the JDK/JRE and install the JDK or JRE from your previous vendor.

Errors and Exceptions

This chapter describes the error and exception messages generated by Waveset.

The topics in this chapter include:

- “Before Working with Waveset Messages” on page 195
- “About Waveset Errors and Exceptions” on page 197
- “Viewing Errors in the System Log Report” on page 199
- “Customizing a Default Error Message” on page 202

Before Working with Waveset Messages

Review the following information before you start working with Waveset error and exception messages:

Important Notes About Waveset Messages

Be sure to read the following information before you start working with Waveset error and exception messages:

- Examples in this chapter use a locale (*xx_XX locale*) that was devised for example purposes only.
- When you interpret error messages, be aware of the following:
 - Some messages have different keys but display the same error message text.
 - Some messages are used by multiple components.
 - Exceptions are generally listed by exception type, component, or both.
 - Some exceptions are caused by internal programming errors that cannot be viewed by Waveset users.

- Some exceptions are simple wrappers and their parameters are the entire exception. For example, Waveset exceptions wrap resource messages, adapter code, and multipart exceptions, such as password policy violations.
- If you are using parameterized messages with single or double quotes in the message, you must use an additional single or double quote to escape the message. (You will see only one quote symbol when the message is output to the system.)
For example, the following message begins and ends with two *single* quote marks:
"'}"
- If you need help diagnosing problems, contact Oracle Technical Support:
<http://sunsolve.sun.com>

Related Documentation and Web Sites

In addition to the information provided in this chapter, consult the documents and web sites listed in this section for information related to tuning Waveset for your deployment.

Recommended Reading

See the following documents for information related to Waveset error and exception messages:

- For information about creating a customized message catalog, see [Chapter 9, “Customizing Message Catalogs,” in *Oracle Waveset 8.1.1 Deployment Guide*](#).
- For information about creating or editing System Log reports, see [“SystemLog Reports” in *Oracle Waveset 8.1.1 Business Administrator’s Guide*](#).

Useful Web Sites

The following table describes some web sites you might find useful.

TABLE 6–1 Useful Web Sites

| Web Site URL | Description |
|---|---|
| http://sharespace.sun.com/gm/document-1.26.2296 | Waveset FAQ on Oracle’s Share Space Note: This space is only available to Oracle employees and Oracle partners. In addition, these individuals must have a Share Space ID to access the information on this site. |

TABLE 6-1 Useful Web Sites (Continued)

| Web Site URL | Description |
|---|--|
| http://sunsolve.sun.com/ | <p>Web site containing diagnostic tools, forums, features and articles, security information, and patch contents</p> <p>Note: The information on this site is partitioned into three areas:</p> <ul style="list-style-type: none"> ▪ Internal. Oracle employees only ▪ Contract. Available only to customers with contract access ▪ Public. Available to everyone |

About Waveset Errors and Exceptions

This section describes the Waveset error and exception message system and the components that can generate errors and exceptions.

The information is organized into the following sections:

- “Where Messages Are Stored” on page 197
- “How Messages Are Displayed” on page 198
- “Error Severity Levels” on page 198

Where Messages Are Stored

Error messages are stored as follows:

- Waveset messages are stored in a `WPMessages.properties` file in the `idmcommon.jar` and in the `RAMessages.properties` file in the `idmadapter.jar`.
- The `WPMessages.properties` file is located in `project_directory/waveset/idm/common/src/com/waveset/msgcat`.
- The `RAMessages.properties` file is located in `project_directory/waveset/idm/adapter/src/com/waveset/adapter`.

Oracle Waveset Service Provider uses a standard Java message resource bundle for displaying strings in the user interface. This resource bundle is normally included in the `idmspe.jar` file and must be extracted if you plan to customize message strings.

```
$ cd $WSHOME/WEB-INF/classes
$ jar xvf ../lib/idmspe.jar com/sun/idm/idmx/msgcat/I
DMXMessages.properties
```

- Identity Auditor messages are stored in the `AUMessages.properties` file, which is located in `project_directory/waveset/idm/auditor/src/com/sun/idm/auditor/msgcat/AUMessages.properties`

- The Data Exporter's default implementation includes its own message bundle, called `WICMessages.properties`. The `WICMessages.properties` file is located in the `exporter.jar` file in

`com/sun/idm/warehouse/msgcat/WICMessages.properties`

Note – You can create a customized message catalog to add messages or to modify messages provided with the system.

See [Chapter 9, “Customizing Message Catalogs,”](#) in *Oracle Waveset 8.1.1 Deployment Guide* for instructions.

How Messages Are Displayed

For easy identification, Waveset displays page-level error and exception messages along the top of the page as boxed text with a unique error icon.

FIGURE 6-1 Example Login Authentication Error



Note – In the Waveset messaging system, items displayed as `{0}` or `{1}` represent parameters that are supplied by code. For example:

The file is larger than the maximum supported length of `{0}` bytes.

In this exception, the code replaces the `{0}` with the parameter value representing the maximum number of bytes.

Error Severity Levels

Within Waveset, error severities are defined as follows:

- **Fatal.** A severe error that causes your system to crash, resulting in the loss or corruption of unsaved data.
- **Error.** A severe error that might cause the loss or corruption of unsaved data. Immediate action must be taken to prevent losing data.

- **Warning.** Action must be taken at some stage to prevent a severe error from occurring in the future.
- **Info.** An informative message, usually describing server activity. No action is necessary.

Note – You can check the Waveset System Log for more information about an error or exception message. (See “[Viewing Errors in the System Log Report](#)” on page 199.)

Viewing Errors in the System Log Report

System Log reports can provide information about errors generated by Waveset. The System Log report consists of the error’s timestamp, severity, server name, component name, error code or ID, stack trace (structure of the execution stack at that point in the program’s life), and error text information.

You can use Waveset’s Administrator interface or command-line interface to run and view System Log reports.

Note – Instructions for creating and editing System Log reports are provided in *Oracle Waveset 8.1.1 Business Administrator’s Guide*.

▼ To Run a System Log Report From the Administrator Interface

Perform the following steps to run a System Log report from the Administrator interface:

- 1 **Log in to the Waveset Administrator interface.**
- 2 **Select Reports → Run Reports to open the Run Reports page.**
- 3 **Locate the appropriate System Log Report entry in the Report Type column, and then click the Run button in that same row.**

The Report Results page displays, listing the system messages that were reported during the specified interval. For example, [Figure 6–2](#) depicts information about two system messages.

FIGURE 6-2 Example Report Results Page

Report Results

System Messages from the previous week

Monday, February 25, 2008 3:00:20 PM CST

Lists all system messages reported during the past 7 days

Number of records reported: 2

| ▼ TimeStamp | Event | Severity | Server | Component | Error Code | Message |
|--|----------------|----------|----------|-----------|------------|---|
| Thursday, February 21, 2008 3:09:06 PM CST | N/A | Fatal | idmvm042 | Server | SECURITY | Security Violation: Incoming HttpServletRequest considered invalid by CSRFGUARD from address: 129.147.62.21 |
| Tuesday, February 19, 2008 7:21:15 PM CST | SV-0220-012115 | Error | idmvm042 | Server | ERROR | An error occurred starting the connection: io exception: The Network Adapter could not establish the connection ==> java.sql.SQLException: io exception: The Network Adapter could not establish the connection |

The Report Results table shows the following information:

- **Timestamp.** Shows the day, date, and time the error occurred.

Click the Timestamp links to view detailed information about that System Log record. For example, if you clicked the first Timestamp link shown in [Figure 6-2](#), the following information displays.

System Log Record Details

| | |
|-------------|---|
| Timestamp: | Tuesday, February 19, 2008 7:21:15 PM CST |
| Event: | EV-022D-012115 |
| Server: | itrvnm042 |
| Severity: | Error |
| Component: | Server |
| Error Code: | ERROR |
| Message: | <pre> An error occurred starting the connection: io.exception: The Network Adapter could not establish the connection==> java.sql.SQLException: io.exception: The Network Adapter could not establish the connection com.wavelet.util.WaveletException: An error occurred starting the connection: io.exception: The Network Adapter could not establish the connection: java.sql.SQLException: io.exception: The Network Adapter could not establish the connection => java.sql.SQLException: io.exception: The Network Adapter could not establish the connection at com.wavelet.adapter.OracleERPResourceAdapter.makeConnection(OracleERPResourceAdapter.java:787) at com.wavelet.adapter.OracleERPResourceAdapter.startConnection(OracleERPResourceAdapter.java:484) at com.wavelet.adapter.OracleERPResourceAdapter.getUser(OracleERPResourceAdapter.java:2736) at com.wavelet.adapter.ResourceAdapterProxy.getUser(ResourceAdapterProxy.java:902) at com.wavelet.provision.FetchContent.doFetchContent(fetchContent.java:368) at com.wavelet.provision.FetchContent.processOfFetchContent(fetchContent.java:236) at com.wavelet.provision.ThreadContent.processOfThreadContent(fetchContent.java:348) at com.wavelet.provision.ThreadContent.launchThread(ThreadContent.java:158) at com.wavelet.provision.Provisioner.fetchAccounts(USERProvisioner.java:2145) at com.wavelet.provision.Provisioner.fetchAccounts(Provisioner.java:2796) at com.wavelet.view.UserViewer.assembleView(UserViewer.java:598) at com.wavelet.view.UserViewer.checkoutView(UserViewer.java:754) at com.wavelet.object.ViewMaster.checkoutView(ViewMaster.java:632) at com.wavelet.session.LocalSession.checkoutView(LocalSession.java:681) at com.wavelet.util.DatabaseNewSource.getView(DatabaseNewSource.java:447) at org.apache.jsp.account.modify_jsp._jspService(modify_jsp.java:428) at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:94) at java.servlet.http.HttpServlet.service(HttpServlet.java:802) at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:224) at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:292) at org.apache.jasper.servlet.JspServlet.service(HttpServlet.java:238) at java.servlet.http.HttpServlet.service(HttpServlet.java:802) at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:237) at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:157) at com.sun.jmx.mbeanserver.RequestTracingFilter.doFilter(RequestTracingFilter.java:61) at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:186) at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:157) at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:214) at org.apache.catalina.core.StandardPipeline.invoke(StandardPipeline.java:154) at org.apache.catalina.core.StandardPipeline.invoke(StandardPipeline.java:526) </pre> |

- **Event.** Identifies the syslog ID of the target entry (when applicable).
- **Severity.** Shows the severity level of the error.

Severity levels include

- Fatal.** A severe error that causes the system to crash, resulting in the loss or corruption of unsaved data.
- Error.** A severe error that might cause the loss or corruption of unsaved data. Immediate action must be taken to prevent losing data.
- Warning.** Action must be taken at some stage to prevent a severe error from occurring in the future.
- Info.** An informative message, usually describing server activity. No action is necessary.

- **Server.** Identifies the server on which the error occurred.
- **Component.** Identifies the system component that generated the error.
- **Error Code.** Shows the error code associated with that error.
- **Message.** Shows the actual error message text.

▼ To Run a System Log Report From the Command-Line Interface

Note – These instructions assume you are familiar with the Waveset command-line interface and `lh` commands. For more information, read [Appendix A, “lh Reference,”](#) in *Oracle Waveset 8.1.1 Business Administrator’s Guide*.

Perform the following steps to run and view a System Log report from the command line:

- 1 **Open a command window.**
- 2 **Change directories to the default Waveset installation directory.**
- 3 **At the prompt, type the `lh syslog [options]` command.**

Use these *options* to include or exclude information:

- `-d: Number` – Show records for the previous number of days (Default is 1.)
- `-F` – Show only records with fatal severity level
- `-E` – Show only records with an error severity level or higher
- `-i logid` – Show only records with a specified Syslog ID
Syslog IDs are displayed on some error messages and reference a specific System Log entry.
- `-W` – Show only records with a warning severity level or higher (default)
- `-X` – Include reported cause of error, if available

Customizing a Default Error Message

Note – To add message catalog entries or to modify entries provided with the system, you must create a customized message catalog. See [Chapter 9, “Customizing Message Catalogs,”](#) in *Oracle Waveset 8.1.1 Deployment Guide* for instructions.

Waveset’s default error messages are stored in a `WPMessages.properties` file in the `idmcommon.jar` file and in an `RAMessages.properties` file in the `idmadapter.jar`.

- The `WPMessages.properties` file is located in
`project_directory/waveset/idm/common/src/com/waveset/msgcat`
- The `RAMessages.properties` file is located in
`project_directory/waveset/idm/adapter/src/com/waveset/adapter`

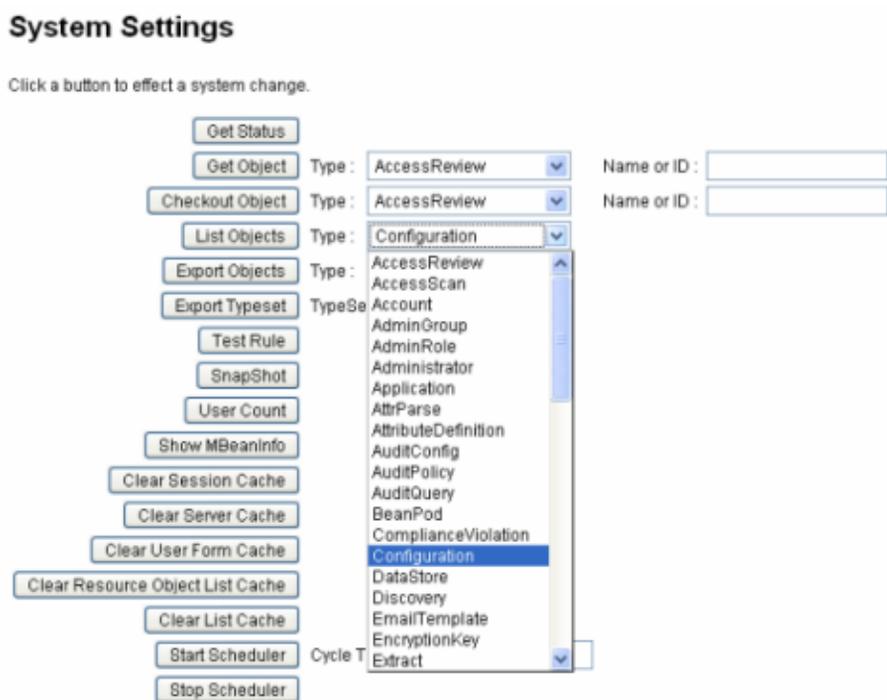
The Waveset Service Provider's default error messages are located in the `IDMXMLMessages.properties` file.

You can customize these default error messages by modifying attributes in the `ErrorUIConfig` object provided with that message.

▼ To Modify the `ErrorUIConfig` Object:

- 1 Log in to Waveset Administrator interface.
- 2 Open the System Settings page by typing `http://host:port/idm/debug` in to your browser.
- 3 Locate the Type menu, located next to the List Objects button. Choose Configuration from the menu.

FIGURE 6-3 List Objects Type Menu



- 4 Click the List Objects button.

5 On the List Objects of type: Configuration page, click the ErrorUIConfig edit link.

The following example shows the XML for an ErrorUIConfig object on Checkout Object: Configuration page:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Configuration PUBLIC 'waveset.dtd' 'waveset.dtd'>
<!-- MemberObjectGroups="#ID#Top" extensionClass="GenericObject"
id="#ID#9787BA467E01441B:178655:1156195C008:-7FFE"
lastModifier="com.waveset.object.ErrorUIConfig" name="ErrorUIConfig"-->
<Configuration id="#ID#9787BA467E01441B:178655:1156195C008:-7FFE" name='ErrorUIConfig'
lock='Configurator#1200600790328' creator='com.waveset.object.ErrorUIConfig'
createDate='1191343145328' lastModifier='com.waveset.object.ErrorUIConfig'
lastModDate='1191343145328'>
  <Extension>
    <Object>
      <Attribute name='Enabled'>
        <Boolean>true</Boolean>
      </Attribute>
      <Attribute name='ErrorMsgID' value='UI_ERROR_DEFAULT_FATAL_MESSAGE' />
    </Object>
  </Extension>
  <MemberObjectGroups>
    <ObjectRef type='ObjectGroup' id='#ID#Top' name='Top' />
  </MemberObjectGroups>
</Configuration>
```

6 Modify the ErrorUIConfig object attributes as needed.

You can modify any of the following attributes:

- **Enabled.** Controls whether the message is enabled (`true`) or disabled (`false`).

Note – The ability to disable this attribute is provided for backward-compatibility; however, disabling this attribute will result in cryptic messages and you will not see the typical extended messages in the System Log.

- **ErrorMsgID.** Identifies the error message to be displayed.

Change the ErrorMsgID attribute value to provide the message text you want displayed.

The current setting for this attribute is as follows, and it references the `UI_ERROR_DEFAULT_FATAL_MESSAGE` message in the message catalog:

```
<Attribute name='ErrorMsgID' value='UI_ERROR_DEFAULT_FATAL_MESSAGE' />
```

Note – If you are using parameterized messages with single or double quotes in the message, you must use an additional single or double quote to escape the message. (You will see only one quote symbol when the message is output to the system.)

For example, the following message begins and ends with two *single* quote marks:

```
"' {0}'"
```

7 When you are finished, click Save to save your changes.

Note – If you have difficulty creating a default error message, contact your Administrator or Administrator hotline.

Index

Numbers and Symbols

\$WSHOME/patches directory, 193

A

account data, 44-46

accountIds, verifying, 176-177

Active Sync adapters

logs, 171

performance tuning, 56

adapters

debugging, 110-114, 173-175

troubleshooting authentication, 176-177

tuning, 56

admin cache, tuning initialization, 56-57

administrator forms, optimizing, 59

allowInvalidCerts registry key, 146-147

application server, 41

application server, tuning, 41

application server DataSource, using

preferPreparedStatement, 50-51

application servers

analyzing logs, 191-192

determining URLs, 27-28

garbage collection, 38-39

increasing instances, 55

memory, 69

PeopleSoft, 178

redirecting workflow trace messages to

console, 157-158

testing DataSource connections, 190-191

application servers (*Continued*)

troubleshooting, 191-192

troubleshooting repository problems, 189-191

tuning, 38-39, 39-41, 69

tuning IBM WebSphere, 41

tuning Sun Java System Application Server, 38-39

viewing trace output files, 132-133, 177-178

attribute tables, 42-43

attributes

binary, 55

connectionPoolDisable, 47-48

modifying ErrorUIConfig object, 202-205

querying for predefined object, 42-43

using inline, 48

XML, 47-51

Audit Log Maintenance Task, 55

audit logs, preventing tampering, 29-31

authentication

optimizing pool size, 80

properties, missing, 176-177

testing pass-through, 175-176

troubleshooting adapter problems, 176-177

B

binary attributes, 55

bulk loading, tuning, 57-58

C

caching,, implicit statement, 50-51

- callTimer command, 85-86
- callTimer.jsp debug page, 85-86, 167-170, 179-180
- change tables, 42-43
- characters sets and encodings, 47
- cipher data, viewing in trace logs, 132
- Clear_List_Cache.jsp debug page, 68-69
- Clear_User_Cache.jsp debug page, 89
- Clear_XMLParser_Cache.jsp debug page, 86
- Clear_XMLResourceAdapter_Cache.jsp debug page, 89
- Compliance Violation data, 44-46
- concurrent
 - mode failures, 40-41
 - operations, limiting, 70-72
 - processes, 72-73
 - tasks, 77
 - user loads, 38-39
 - users, 68
- concurrent connections, limiting, 49-50
- configuration data, 44-46
- configuring
 - repository, 47-51
 - tracing, 127-132
 - Waveset.properties, 15-16
 - Waveset server settings, 16
- connection pool, using
 - preferPreparedStatement, 50-51
- connection pooling, disabling automatic, 47-48
- connectionPoolDisable attribute, 47-48
- Control Timings debug page, 85-86, 167-170
- customizing
 - default error messages, 202-205
 - Service Provider message strings, 197-198

D

- data classes, 43
- data mining, 38-39
- data Translation Look-aside Buffers (DTLBs), 40-41
- database repository, tuning, 41-54
- database statistics, tuning, 66
- database tunings, vendor-specific, 52-54
- DataSource connections, testing, 190-191

- DataSources
 - disabling automatic connection pooling, 47-48
 - using preferPreparedStatement, 50-51
- debug pages
 - accessing, 84-85
 - callTimer.jsp, 85-86
 - Clear_List_Cache.jsp, 68-69
 - Clear_User_Cache.jsp, 89
 - Clear_XMLParser_Cache.jsp, 86
 - Clear_XMLResourceAdapter_Cache.jsp, 89
 - Control Timings, 85-86, 167-170
 - debugging performance issues, 84-89
 - Show_CacheSummary.jsp, 88
 - Show_ConnectionPools.jsp, 86
 - Show_Memory.jsp, 88
 - Show_Provisioning.jsp, 88
 - Show_Sizes.jsp, 87-88
 - Show Timings, 179-180
 - Show_Timings.jsp, 86-87
 - Show_Trace.jsp, 86
 - Show_WSProp.jsp, 68-69, 89
 - SysInfo.jsp, 88
- debugging
 - adapters, 110-114, 173-175
 - errors displayed through browser, 172
 - LoginConfig changes, 175-176
 - PasswordSync, 186-188
 - performance issues, 84-89
- default error messages, customizing, 202-205
- default server settings, 25
- defvarmethod, 60
- deployment-specific settings for tuning, 36-37
- directories
 - \$WSHOME/patches, 193
 - logs, 193
 - patches, 193
- documentation, related
 - errors/exceptions, 196-197
 - performance tuning, 34-36
 - troubleshooting/tracing, 125
- domain.xml file, 38-39
- Dr. Watson logs, 166
- DTrace facility
 - commands, 108-109

DTrace facility (*Continued*)
 description/purpose, 108-110, 171
 enabling probes, 108-109
 writing scripts, 108-110

E

editing
 registry keys, 146-147
 RepositoryConfiguration object, 47-51
 RepostioryConfiguration object, 48
 Waveset schema for Users/Roles, 72-73

enabling
 methods, 152
 setTrace method, 152
 SOAP tracing, 153-154, 155-156
 trace, 175-176

encodings and character sets, 47

end-user forms, optimizing, 59-60

entitlement data, 44-46

environment, Java EE, 38-39

errors
 debugging, 172
 important notes, 195-196
 severity levels, 198-199
 viewing, 199-202

example locale, 195-196

exception logging
 description, 142-143
 disabling, 142-143
 enabling, 142-143

exceptions
 important notes, 195-196
 tracing, 142-143
 wrappers, 195-196

experience requirements
 for performance tuning, 9
 for troubleshooting, 117

export queue data, 44-46

F

File Cache Configuration page, 38-39

forms, optimizing
 administrator forms, 59
 end-user forms, 59-60
 form field expressions, 60
 new forms, 58-59

G

garbage collection, 38-39

garbage collector
 detecting low memory/deadlocks, 114-115
 tuning JVM performance, 38-39, 40-41

gateway tracing, 160-166

general XML, optimizing, 67-68

getResourceObjects, 75-76

global XPRESS tracing, 144-145

GUIDs, 46

H

heap size, 69

HTTP listeners, 38-39

HTTP requests, 27-28, 175-176

I

IBM WebSphere, 41

idmadapter.jar file, 202-205

idmcommon.jar file, 197-198, 202-205

idmspe.jar file, 197-198

IDMXMLMessages.properties file, 202-205

implicit statement caching, 50-51

important notes
 for errors/exceptions, 195-196
 for performance tuning, 34

inline attributes, 48

installdir, 146-147

instances, application server, 55

J

- Java
 - test programs, 173-175
 - tuning, 38
- Java EE environment, 38-39
- JConsole, configuring as a JMX client, 23-24
- JDBC connections, 50-51
- JMX
 - and server polling, 21
 - configuring a JMX client, 23-24
 - debugging resource adapter performance, 110-114
- JVM, tuning, 38-39

L

- Large Objects (LOBs), 42-43
- library cache contention, 46-47
- limiting concurrent connections, 49-50
- limiting concurrent operations, 70-72
- load balancers, configuring logs, 27-28
- LOB data types, 52-53
- locally managed tablespaces (LMTs), 52-53
- log data, 44-46
- logging SPML traffic, 152
- logs
 - analyzing application server, 191-192
 - configuring for load balancers, 27-28
 - Dr. Watson, 166
 - synchronization, 56
- logs, preventing tampering, 29-31
- logs directory, 193

M

- ManualActions, tuning, 62-64
- memory requirements, 69
- message catalogs
 - adding/editing entries, 202-205
 - creating customized, 196-197, 197-198
- messages
 - parameterized, 195-196, 203-205
 - where stored, 197-198
- methodology, tuning, 36-37

methods

- caching query results, 75-76
 - enabling setTrace, 152
 - improving form performance, 58-60
 - improving Gateway performance, 80-83
 - reconciliation, 150-152
 - tuning rules, 60
- multi-byte characters, using, 47

O

- object IDs, 46
- object tables, 42-43
- operating systems, tuning, 69-70
- operations, limiting concurrent, 70-72
- optimizing
 - administrator forms, 59
 - authentication pool size, 80
 - end-user forms, 59-60
 - form field expressions, 60
 - new forms, 58-59
- Oracle, Waveset IDE, 170-171
- Oracle database repository, 52-53
- Oracle Waveset Integrated Development Environment.
 - See Waveset IDE*, 170-171
- organization data, 44-46
- output, trace, 152

P

- parameterized messages, 195-196, 203-205
- pass-through authentication, testing, 175-176
- PasswordSync
 - debugging, 186-188
 - registry keys, 146-147
- patches directory, 193
- PeopleSoft application servers, troubleshooting, 178
- per-account workflows, 72-73
- performance
 - improving JVM, 38-39
 - profiler, 70
 - provisioner, 70-72

performance tuning
 experience requirements, 9
 important notes, 34
 Java EE environment, 38-39
 optimizing the database repository, 41-54
 optimizing Waveset, 54-83
 recommended reading, 34-35
 related documentation, 34-36
 XML, 67-68

pre-allocating, tables, 51-52

predefined object attributes, 42-43

preferPreparedStatement repository
 option, 50-51

prepared statements, 46-47

PreparedStatement, SQL statements, 50-51

PrintGCDetails script, 38-39

PrintGCStats script, 38-39

PrintGCTimeStamps script, 38-39

probes, enabling DTrace, 108-109

processes, concurrent, 72-73

profiler performance, 70

properties
 authentication, 176-177
 missing authentication, 176-177

provisioner performance, 70-72

putmap method, 60

R

RAMessages.properties file, 197-198, 202-205

recommended reading, 34-35

reconciler settings, 17-18

reconciliation
 methods, 150-152
 performance, 72-75
 tracing, 150-152
 troubleshooting, 188-189
 tuning, 72-75

registry keys, PasswordSync, 146-147

related documentation
 errors/exceptions, 196-197
 performance tuning, 34-36
 troubleshooting/tracing, 125

related web sites, 35-36, 125, 196-197

repository
 configuring, 47-51
 limiting concurrent connections, 49-50

repository databases
 tuning, 41-54
 tuning Oracle, 52-53
 tuning SQL Server, 54

repository tables
 attribute, 42-43
 change, 42-43
 object, 42-43

RepositoryConfiguration object, 47-51
 editing, 49-50

RepositoryConfiguration object, editing, 48

requests
 HTTP, 27-28, 175-176
 SPML RPC, 154

requirements, experience
 for performance tuning, 9
 for troubleshooting, 117

requirements, memory, 69

resource queries, tuning, 75-76

resource query, Waveset, 75-76

role data, 44-46

rules, tuning, 60

running System Log reports, 199-201

S

scheduler, tracing, 156-157

scripts
 DTrace, 108-110
 PrintGCStats, 38-39
 PrintGCTimeStamps, 38-39

sensitive data, viewing in trace logs, 132

server.xml file, 38-39

servers
 connection settings, 84-85, 128, 131, 170
 tracing, 131, 132, 133
 troubleshooting, 191-192
 tuning, 38-39, 39-41
 working with proxy, 27-28

Service Provider
 errors and exceptions, 197-198

- Service Provider (*Continued*)
 - troubleshooting, 192
 - tuning, 68
 - sessions, Waveset, 80
 - setlist method, 60
 - setTrace method, enabling, 152
 - setvar method, 60
 - severity levels, 198-199
 - Show_CacheSummary.jsp debug page, 88
 - Show_ConnectionPools.jsp debug page, 86
 - Show_Memory.jsp debug page, 88
 - Show_Provisioning.jsp debug page, 88
 - Show_Sizes.jsp debug page, 87-88
 - Show Timings debug page, 179-180
 - Show_Timings.jsp debug page, 86-87
 - Show_Trace.jsp debug page, 86
 - Show_WSProp.jsp debug page, 68-69, 89
 - Single Sign-On (SSO), 175-176
 - SOAP, tracing, 153-154, 155-156
 - special considerations
 - for errors/exceptions, 195-196
 - for performance tuning, 34
 - SPML
 - tracing messages, 152-156
 - troubleshooting, 192-193
 - SPML requests, RPC, 154
 - SQL Server databases, tuning, 54
 - SQL statements
 - PreparedStatement, 50-51
 - Statement, 50-51
 - Statement, SQL statements, 50-51
 - statistics, updating, 51-52
 - Sun Java System Application Server, 38-39
 - synchronization logs, 56
 - SysInfo.jsp debug page, 88
 - system log, trimming, 31-32
 - System Log Maintenance Task, 55
 - System Log reports, 199-202
- T**
- table names
 - attribute, 42-43
 - change, 42-43
 - table names (*Continued*)
 - object, 42-43
 - tables
 - attribute, 42-43
 - change, 42-43
 - object, 42-43
 - pre-allocating, 51-52
 - tamper-resistant logging, 30-31
 - tampering, preventing, 29-31
 - task bar, tuning, 83
 - task bar, Waveset, 83
 - task data, 44-46
 - task scheduler tracing, 156-157
 - tasks, specifying maximum concurrent, 77
 - testing adapters, 173-175
 - trace file, 132-133
 - trace output files, viewing, 132-133, 177-178
 - tracing
 - application server logs, 191-192
 - configuring, 127-132
 - debugging error display, 172
 - Dr. Watson logs, 166
 - enabling, 153-154, 155-156, 175-176
 - enabling for SPML, 152
 - exceptions, 142-143
 - gateway, 160-166
 - reconciliation, 150-152
 - related documentation, 125
 - SPML messages, 152-156
 - task scheduler, 156-157
 - viewing sensitive data, 132
 - Waveset server, 133
 - workflows, 157-159
 - XPRESS, 144-145
 - tracing method, 127-159
 - troubleshooting
 - application servers, 191-192
 - experience requirements, 117
 - PeopleSoft application servers, 178
 - reconciliation, 188-189
 - related documentation, 125
 - tuning
 - admin cache initialization, 56-57
 - application servers, 38-39

tuning (*Continued*)

- database statistics, 66
- deployment-specific settings, 36-37
- general performance tunings, 55
- IBM WebSphere, 41
- Java, 38
- ManualActions, 62-64
- methodology, 36-37
- profiler, 70
- Provisioner, 70-72
- reconciliation, 72-75
- repository database, 51-52
- resource queries, 75-76
- roadmap, 36-37
- rules, 60
- Service Provider, 68
- session performance, 80
- Sun Java System Application Server, 38-39, 40-41
- task bar, 83
- vendor-specific tunings, 52-54
- workflows, 60-62
- WorkItems, 62-64

U

- upgrade log files, 193
- URLs, how Waveset uses, 27-28
- user data, 43
- user loads, concurrent, 38-39
- users, concurrent, 68

V

- vendor-specific database tunings, 52-54
- viewing
 - errors, 199-202
 - trace output files, 132-133, 177-178

W

- Waveset
 - errors and exceptions, 195-205

Waveset (*Continued*)

- forms performance, optimizing, 58-60
- gateway tracing, 160-166
- general performance, optimizing, 55
- resource query performance, optimizing, 75-76
- server settings, 16
- session performance, optimizing, 80
- task bar performance, optimizing, 83
- workflows, optimizing, 60-62
- Waveset IDE, 170-171
- Waveset.properties, 27-28
 - configuring Waveset, 15-16
- Waveset schema, editing, 72-73
- web sites, useful, 35-36, 125, 196-197
- workflow trace messages, 157-158
- workflows
 - per-account, 72-73
 - tracing, 157-159
 - tuning, 60-62
- WorkItems, tuning, 62-64
- WPMessages.properties file, 197-198, 202-205
- wrappers, exception, 195-196
- writing DTrace scripts, 108-110
- WSHOME, 173-179

X

- XML, optimizing, 67-68
- XML columns, 43
- XPRESS tracing, 144-145

