

Oracle® Retail Integration Bus

Operations Guide

Release 13.2

January 2011

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Primary Author: Susan McKibbin

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 software or related documentation 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 USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software 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 in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software 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.

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

The following restrictions and provisions only apply to the programs referred to in this section and licensed to you. You acknowledge that the programs may contain third party software (VAR applications) licensed to Oracle. Depending upon your product and its version number, the VAR applications may include:

(i) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.

(ii) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Mobile Store Inventory Management.

(iii) the software component known as **Access Via**[™] licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.

(iv) the software component known as **Adobe Flex**[™] licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.

You acknowledge and confirm that Oracle grants you use of only the object code of the VAR Applications. Oracle will not deliver source code to the VAR Applications to you. Notwithstanding any other term or condition of the agreement and this ordering document, you shall not cause or permit alteration of any VAR Applications. For purposes of this section, "alteration" refers to all alterations, translations, upgrades, enhancements, customizations or modifications of all or any portion of the VAR Applications including all reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

The VAR Applications contain trade secrets of Oracle and Oracle's licensors and Customer shall not attempt, cause, or permit the alteration, decompilation, reverse engineering, disassembly or other reduction of the VAR Applications to a human perceivable form. Oracle reserves the right to replace, with functional equivalent software, any of the VAR Applications in future releases of the applicable program.

Contents

Preface	xiii
Audience	xiii
Related Documents	xiii
Customer Support	xiv
Review Patch Documentation	xiv
Oracle Retail Documentation on the Oracle Technology Network	xiv
Conventions	xiv
1 Introduction	
Oracle WebLogic Application Server	1-1
Oracle Retail RIB Supplied Components	1-1
2 Application Builder	
RIB Application Builder Directory Structure	2-1
Directory Structure and Key Files	2-2
RIB App Builder Tools	2-3
Logging	2-3
Backup and Archive of Key Files	2-3
rib-app-compiler	2-3
rib-app-deployer	2-4
Check-version-and-unpack	2-5
check-version-and-apply-defect-fix	2-5
inventory-management	2-6
setup-security-credential	2-6
Hot Fix Installation Reports	2-7
rib-adapter-controller	2-8
Start Flow	2-8
Stop Flow	2-8
List Flow	2-9
Start Adapters By Type	2-9
Stop Adapters by Type	2-9
Start Adapter	2-10

Stop Adapter	2-10
Test Durable Subscriber for Adapter	2-10
Test Durable Subscriber for RIB App	2-11
List RIB App Adapters	2-11
RIB Deployment Configuration File Editor	2-11
Important Installation Warning	2-12
Key Rule	2-12
Editor Usage.....	2-12

3 Backend System Administration and Logging

rib-<app>-adapters.xml	3-1
<subscribers> element.....	3-2
<publisher> element.....	3-2
<timer-driven>	3-2
<request-driven>	3-2
<hospital> element	3-2
rib-<app>-adapters-resource.properties	3-3
rib-<app>-plsql-api.xml	3-3
rib-<app>.properties	3-3
rib-system.properties	3-3
rib-integration-flows.xml	3-5
rib-deployment-env-info.xml	3-5
app-in-scope-for-integration	3-5
rib-jms-server	3-6
rib-application-server	3-6
rib-javaee-containers	3-6
rib-applications	3-6
commons-logging.properties	3-7
log4j.xml	3-7
rib-app-builder-paths.properties	3-7
rib-application-assembly-info.xml	3-7
retail_service_config_info_ribserver.xml	3-7
remote_service_locator_info_ribserver.xml	3-8
RIB Logging	3-8
Log Level Recommendations	3-8
Changing Logging Levels	3-8
RIB Administration GUI	3-8
log4j.xml Configuration File.....	3-8
Adapter Logging (RIBLOGS)	3-8
RIB Timing Logs.....	3-9
RIB Audit Logs	3-10
Other RIB Management Logs	3-11
deploy.rib.log.....	3-11
management.rib.log.....	3-11
global.rib.log	3-12

4	Integration Gateway Services (IGS) Testing	
	Using the soapUI Tool and the IGS Test Suite	4-1
	Download soapUI	4-1
	Import IGS.....	4-2
	Execute IGS Test Cases.....	4-3
5	RIB and JMX	
	RIB JMX Client	5-1
	User Interface.....	5-1
	Third Party JMX Client Example.....	5-3
6	Admin GUI	
	RIB Admin URLs.....	6-1
	RIB Admin GUI.....	6-1
	RIB Functional Artifacts.....	6-2
	RIB Message Flows	6-2
	RIB Payloads (xsds).....	6-2
	RIB Admin GUI Home.....	6-2
	Adapter Manager	6-3
	Adapter Manager Screen	6-3
	Log Viewer	6-4
	Log Manager.....	6-5
	RIB Logs.....	6-6
7	JMS Provider Management	
	The RIB on AQ JMS.....	7-1
	Queue Monitor Process Setup	7-2
	Optimizing Enqueue/Dequeue Performance.....	7-2
	Sizing Considerations.....	7-3
	The RIB on AQ JMS - Server Side Processes.....	7-3
	Types of Oracle Database Side Processes	7-3
	The RIB and Application Server and JDBC Connections.....	7-4
	RIB 13.2 Connections - Summary	7-4
	rib-rms Connections.....	7-4
	rib-rwms Connections	7-5
	rib-sim Connections.....	7-5
	rib-tafr Connections	7-5
	rib-rpm Connections.....	7-5
	Configuration Recommendations.....	7-6

Support for Multiple JMS Servers Within a Single Deployment	7-6
Design	7-6
rib-app-builder Validation Checks.....	7-6
How to Set Up Multiple JMS Servers	7-7
Process Overview	7-7
General Recommendations.....	7-7
AQ Recommendation	7-7
Sample Configuration.....	7-7
rib-integration-flows.xml	7-7
rib-deployment-env-info.xml.....	7-8
RIB-RMS Application Configuration	7-9
rib-rms-adapters.xml changes.....	7-9
rib-rms-adapters-resources.properties	7-9
RIB-TAFR Application Configuration	7-10
rib-tafr-adapters.xml.....	7-10
rib-tafr-adapters-resources.properties	7-10
RIB-SIM Application Configuration.....	7-11
rib-sim-adapters.xml	7-11
rib-sim-adapters-resources.properties	7-11
RIB-RWMS Application Configuration	7-12
rib-rwms-adapters.xml.....	7-12
rib-rwms-adapters-resources.properties	7-12
Compile and Deploy.....	7-12
RIB-ADMIN-GUI	7-13

8 Message Transform, Filtering and Routing (TAFR)

TAFR Adapter Process	8-2
Configuration	8-2
Transformation	8-2
Filtering Configuration.....	8-3
Routing	8-3
Configuration Example - Facility ID	8-3
Single RWMS Configuration	8-3
Configuration Process	8-4
Two RWMS Configuration	8-5
Description.....	8-5
Configuration Process	8-6

9 Diagnostic and Monitoring Tools

Functionality	9-1
RDMT and User Roles and Responsibilities.....	9-2
Local or Remote Installations and Capabilities	9-2
RDMT Support jars	9-2
Sample XML Messages	9-2

Tools Overview	9-2
RDMT as an Application.....	9-3
SCRIPTDIR	9-3
Setup	9-3
Current Configuration	9-3
RDMTLOGS.....	9-3
RDMT RAC Support.....	9-3
RDMT Main Menu	9-4
WLS/JMX Utilities	9-5
JMS Tools	9-7
PUB/SUB Msg Tools	9-8
RIB Health Tools	9-9
Hospital Scan Tools	9-10
RIB Admin Tools	9-10
RIB App Builder Tools	9-11
Scan RIB Logs / Scan RIB Logs (Delta)	9-12
RIB Health	9-12
RIB Configuration Report	9-13
RIB Timings Utility	9-13
JMS Publish Utility	9-14
EJB Publish Utility	9-14
TAFR Msg Utility	9-15
EJB Ping (RIB)	9-17
EJB Ping (APP)	9-18
Tool Usage Examples	9-18
"How do I know if the RIB installation is correct?"	9-18
"How do I know if the local WLS is running or not?"	9-18
"How do I know where my issue is occurring?"	9-19
"How do I know if the adapter's status is up or down?"	9-19
"How can I config/switch for a new WLS instance?"	9-19
"How do I know the subscriber for a particular JMS topic?"	9-19

10 The RIB in Operation

Operational Considerations	10-1
Alerts and Notifications	10-1
RIB Log File Monitoring.....	10-1
Log File Archive and Purge.....	10-1
Hospital Size and Growth.....	10-2
RMS MFQ and RWMS UPLOAD Tables Sizes	10-2
Remote RWMS.....	10-2
RIB Components Start and Stop	10-2
RIB Operation Support Staff Requirements	10-2
RIB Components - Source Code Control	10-3
RIB HA Requirements	10-3
RIB Disaster Recovery	10-3
RIB Admin Roles and Security.....	10-3

RIB Operation Support Staff Requirements	10-4
RIB System Administrator	10-4
Technology Background	10-4
Experience or Training	10-4
Areas of Responsibility	10-4
RIB Application Administrator.....	10-5
Technology Background	10-5
Experience or Training on	10-5
Areas of Responsibility:	10-5
Hospital Monitoring and Maintenance	10-5

11 Testing the RIB

RIB Test Harness	11-2
Master Checklist	11-2
PL/SQL Application API Stubs	11-3
Architecture and Design	11-3
The Common Subsystem	11-3
The Thin API layer.....	11-4
The Stub Admin and Setup Functions	11-4
Configuration Files	11-5
Installation and Setup.....	11-5
Prerequisite Tasks	11-5
Installation	11-6
Configure_api	11-7
Prerequisites	11-7
Java EE Application API Stubs	11-8
Architecture and Design	11-8
Installation and Setup.....	11-9
Prerequisite Tasks	11-9
Installation	11-9
Configuration of the rib-<app> to use Injection Stubs.....	11-11

12 Performance Considerations

Performance Factors	12-1
Performance Requirements	12-2
Multi-Channel	12-3
End-to-End Timing	12-4
How to Calculate Average Message Size	12-4
Purchase Order Example	12-5
Understand the Message Family	12-7
RIB Timing Log Analysis	12-8
Purchase Order Example	12-9
Key Interfaces to Consider	12-10
ASN (Inbound/Outbound)	12-10
Receipts.....	12-10
Stock Order (Allocations & Transfers).....	12-11
How to Approach a RIB Performance Test	12-12

Multi-Channel Adapters	12-14
Adding Multi-Channels to a Message Family	12-14
Logical Channels and threadValue	12-15
Algorithm Used to Calculate Channel	12-16
How to Configure a Multi-Channel Flow	12-16
Generalized Process	12-16
Example	12-17
RIB-RMS	12-17
RIB-TAFR	12-18
RIB-SIM.....	12-19
RIB-RWMS	12-19
Edit the RIB_SETTINGS table	12-20
Compile and Deploy.....	12-20
Message Aggregation	12-20
How to Configure Message Aggregate.....	12-21
Aggregation Example.....	12-21
Multiple Hospital Retry	12-23
Family Specific Hospital Retry Adapters	12-23
How Family Specific Hospital Retry Works	12-23
How to Configure a Family Specific Retry Adapter.....	12-24

Preface

Oracle Retail Operations Guides are designed so that you can view and understand the application's behind-the-scenes processing, including such information as the following:

- Key system administration configuration settings
- Technical architecture
- Functional integration dataflow across the enterprise
- Batch processing

Audience

Anyone who has an interest in better understanding the inner workings of the Oracle Retail Integration Bus (RIB) system can find valuable information in this guide. There are three audiences in general for whom this guide is written:

- Systems analysts and system operations personnel who need information about Oracle Retail Integration Bus processes.
- Integrators and implementers who are responsible for implementing RIB.
- Business analysts who need information about Oracle Retail Integration Bus processes and interfaces.

Related Documents

For more information, see the following documents in the Oracle Retail Integration Bus 13.2 documentation set:

- *Oracle Retail Integration Bus Data Model*
- *Oracle Retail Integration Bus Implementation Guide*
- *Oracle Retail Integration Bus Installation Guide*
- *Oracle Retail Integration Bus Release Notes*
- *Oracle Retail Integration Bus Hospital Administration Guide*
- *Oracle Retail Functional Artifacts Guide*
- *Oracle Retail Functional Artifact Generator Guide*
- *Oracle Retail Service-Oriented Architecture Enabler Tool Guide*

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

<https://support.oracle.com>

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to re-create
- Exact error message received
- Screen shots of each step you take

Review Patch Documentation

When you install the application for the first time, you install either a base release (for example, 13.2) or a later patch release (for example, 13.2.1). If you are installing the base release, additional patch, and bundled hot fix releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch and bundled hot fix releases can contain critical information related to the base release, as well as information about code changes since the base release.

Oracle Retail Documentation on the Oracle Technology Network

Documentation is packaged with each Oracle Retail product release. Oracle Retail product documentation is also available on the following Web site:

http://www.oracle.com/technology/documentation/oracle_retail.html

(Data Model documents are not available through Oracle Technology Network. These documents are packaged with released code, or you can obtain them through My Oracle Support.)

Documentation should be available on this Web site within a month after a product release.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Introduction

This chapter describes the components that make up the Oracle Retail Integration Bus (RIB). These components are distributed within the Oracle Fusion Middleware platform. The final deployed system may be distributed across multiple computing systems.

Oracle WebLogic Application Server

The RIB is configured and deployed to the Oracle WebLogic Application Server. Installation and configuration of the application server is not in the scope of the document, but a thorough understanding is strongly recommended.

Note: See Oracle® WebLogic Server Administrator's Guide 11g Release (10.3.3)

Oracle Retail RIB Supplied Components

This section contains a brief description of the components that Oracle Retail has built upon the Oracle Fusion Middleware platform to create the Oracle Retail Integration Bus.

- Publishing adapters create messages from the information captured by the applications. These publishing adapters are designed to publish events from a single "Message Family" and are specific to an Oracle Retail application, such as Oracle Retail Merchandising System (RMS).
- Subscribing adapters are used to consume messages. These are specific to Oracle Retail and are designed to consume all messages from a specific message family.
- Transformation Address Filters/Router (TAFR) adapters transform message data and route messages. Multiple, message family specific TAFRs have been implemented. A different TAFR adapter may be active on different message families or on the same message family depending on the needs of an application. Not all message families require TAFRs. The TAFR acronym is a generic term.
- RIB Database Objects are Oracle objects and tables to support the PL/SQL Message Family API stored procedures that are called by the Publishing and Subscribing Adapters. They are part of a specific PL/SQL Oracle Retail application, such as RMS and Oracle Retail Warehouse Management System (RWMS).

- RIB Hospital database tables are used as a basis for storing and re-trying problematic messages. Each application, both PL/SQL and Java EE, have a dedicated Hospital.
- RIHA is the RIB Hospital administration tool.
- The Integration Gateway Service (IGS) provides an integration infrastructure for external (third party) system connectivity.

Application Builder

The RIB Application Builder and its directories and content are not a temporary staging structure. The directory structure and the tools must be in a permanent location and treated as a core application home. The location of the rib-app-builder is a key implementation decision.

Note: See "Pre-Implementation Considerations" in the *Oracle RIB Integration Bus Implementation Guide*.

The RIB installation process builds and executes out of rib-home. The RIB installer gathers all of the information that these tools require, constructs the key XML file (rib-deployment-env-info.xml), and then performs the installation, assembly, configuration, and deployment by invoking, as appropriate, a given task. Therefore, for most RIB software life cycle activities, the RIB installer should be used instead of the command line tools.

RIB Application Builder Directory Structure

The rib-<app> application configuration and installation process follows the RIB lifecycle phases. Each of the lifecycle phases can be managed by a certain role. To support the separation of roles and responsibilities and to clearly define these phases the RIB has adopted a specific directory structure. The tools required for each of these roles are provided within this directory structure.

This directory structure supports access permissions to different tools that are managed according to the site-specific business requirements. For example; a sysadmin can be given access permissions to all the tools while a ribadmin or appadmin can be provided access to only certain operation tools.

The RIB App Builder directory structure is fixed and is created by the RIB kernel tar file; RibKernel<release>ForAll<release>Apps_eng_ga.tar.

The rib-home is a controlled structure and there are very specific rules for using the tools and the key files within it. A key rule is that the tools scan and check versions of all files within rib-home (except for tools-home). The processes do not allow files to have the same name with only an additional extension.

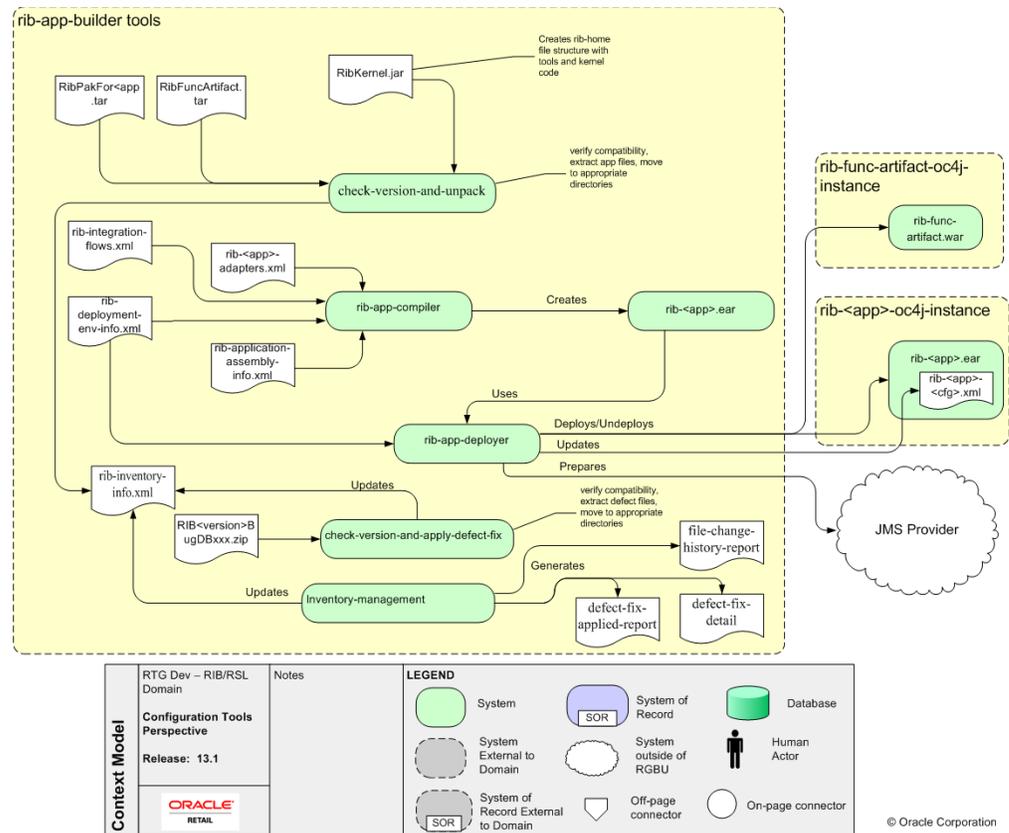
The following is not allowed: rib-rms.properties.bak

Directory Structure and Key Files

```
rib-home
  rib-installer.sh -- this is the RIB GUI Installer
  .retail-installer -- this directory contains the RIB GUI installer file
  application-assembly-home
    bin
      rib-app-compiler.sh
    conf
    log
    rib-aip
    rib-func-artifacts
      rib-func-artifact.war
      rib-private-tafr-business-impl.jar
      rib-public-payload-database-object-types.zip
      rib-public-payload-database-xml-library.zip
      rib-public-payload-java-beans.jar
      rib-public-payload-xml-samples.zip
    rib-rms
      rib-<app>-adapters-resources.properties
      rib-<app>-adapters.xml
  rib-<app>-plsql-api.xml
    rib-<app>.properties
  rib-rpm
  rib-rwms
  rib-sim
  rib-tafr
  deployment-home
    bin
      rib-app-deployer.sh
    conf
      rib-deployment-env-info.xml
    log
  download-home
    all-rib-apps
    all-rib-defect-fixes
    bin
  check-version-and-unpack.sh
    log
    rib-func-artifacts
  integration-lib
    internal-build
    third-party
  maintenance-home
    bin
      check-version-and-apply-defect-fix.sh
      inventory-management.sh
      setup-security-credential.sh
    history-repository
      rib-inventory-info.xml
    log
  operation-home
    bin
      rib-adapter-controller.sh
    log
  tools-home
    javaee-api-stubs
    plsql-api-stubs
    rdmt
    rib-func-artifact-gen
    riha
```

RIB App Builder Tools

All RIB Application Builder tools use the `rib-deployment-env-info.xml` as the source of all values.



Logging

Logging is done for each tool with a log directory, where the execution log is maintained (for example, `rib-app-builder.compiler.log`). These logs are maintained by `log4j` and the `log4j.xml` in `rib-home`. Do not edit this `log4j.xml`. It is set for `DEBUG` when the tools are executed by command line. When the RIB installer is used, it displays the logging at the console level as `INFO`, but the tools themselves write the logs at `DEBUG`.

Backup and Archive of Key Files

The `rib-app-builder` tools automatically generate a backup when a patch is installed. It is recommended that each site develop a backup plan to include a regular backup at the file system level of the `rib-app-builder` directory structure.

rib-app-compiler

The `rib-compiler` is a tool that drives the `rib-<app>.ear` creation process. It performs validation of the input XML files. The following XML files are used to build the `rib-<app>.ear`.

- `rib-<app>-adapters.xml`
- `rib-integration-flows.xml`

- rib-application-assembly-info.xml
- rib-deployment-env-info.xml.

The compiler tool generates the rib-<app> specific application level configuration files, collects the generated files, and packages them to create a deployable rib-<app>.ear file.

This tool works with all applications in scope in the rib-deployment-env-info.xml file.

Command Line Option	Description
-setup-security-credential	This argument must be used when running the rib-app-compiler for the first time. It prompts the user to enter user names and passwords required to install RIB components. It stores the details as credentials in a wallet file inside the rib-home/deployment-home/conf/security/ directory. The credentials are retrieved and used by the deployer when installing RIB components.

rib-app-deployer

This tool performs operations related to deploying the RIB components. It takes a command line set of arguments and values for each function. All functions are driven by the contents of the rib-deployment-env-info.xml.

Command Line Option	Description
-prepare-jms	Prepares the JMS server with RIB JMS topics using the information in rib-deployment-env-info.xml. The JMS server must be running. See Chapter 7, " JMS Provider Management ."
-deploy-rib-func-artifact-war	Deploys the rib-func-artifact.war to the Java EE application server defined in rib-deployment-env-info.xml. The Java EE server must be running.
-deploy-rib-app-ear rib-<app>	Deploys the rib-<app>.ear to the Java EE application server defined in rib-deployment-env-info.xml. The Java EE server must be running.
-update-remote-rib-app-config-files rib-<app>	Updates the rib-<app> application level configuration files in the remote server where rib-<app>.ear is or will be deployed. The remote server information is defined in rib-deployment-env-info.xml. The Java EE server must be running.
-undeploy-rib-func-artifact-war	Undeploys the rib-func-artifact.war from the Java EE application server defined in rib-deployment-env-info.xml. The Java EE server must be running.
-undeploy-rib-app-ear rib-<app>	Undeploys the rib-<app> from the Java EE application server defined in rib-deployment-env-info.xml. The Java EE server must be running.

Check-version-and-unpack

This tool verifies the version compatibility between the RIB paks and extract the files. The extracted files are moved to the appropriate directories under the rib-home.

The version compatibility between RibKernel, RibFuncArtifact and RIBPaks is determined based on the naming conventions used in the tar files and the information in the MANIFEST.mf file inside the kernel tar file.

The RIB infrastructure kernel, RIB functional Pak, and RIB functional artifacts version naming convention should be the same. All should have the same number of major and minor versions.

For verification, the tool does the following:

1. Gets the version of the Rib kernel from the MANIFEST.MF file of the RIB kernel tar file. This is the RibKernel<RIB_MAJOR_VERSION>ForAll<RETAIL_APP_VERSION>Apps_eng_ga.tar.
2. Reads the functional artifact file from rib-home/download-home/rib-func-artifacts.
3. Reads the list of all the RIB application packs from the -home/download-home/all-rib-apps directory.
4. Uses of the naming convention to check if the kernel version is the same as the functional artifact version. If the version is compatible, the tar file is un-tar'd into the rib-home/application-assembly/ rib-func-artifacts directory.
5. Uses the naming convention to check if the kernel version is the same as the application packs. If the version is compatible, the tar file is un-tar'd into the rib-home/application-assembly/rib-<app> directory.

check-version-and-apply-defect-fix

The RIB has been designed to centrally manage and track the application of defects. The check-version-and-apply-defect-fix tool is responsible for that activity.

All RIB defects come in the form of a zip file (for example, RIB13_HPQC1789.zip). The zip file always contains a README.txt file in the format below.

```

-----
Product      : Oracle Retail Integration Bus
Version #    : 13.2.x
Defect #     : 1789
Date        : 02/27/2010
-----

Defects Fixed by this patch:
-----
Resolution:
-----
Files included:
-----
Defect Fix Install Instructions:
-----

```

The README.txt file contains the specific instructions on the application of the defect. It is always applied to the rib-home and deployed from there. Depending on the type of defect it may be necessary to migrate a jar to one of the Oracle Retail applications into the appropriate directories.

All defects are applied to rib-home in the same manner.

1. Drop the Defect.zip into /rib-home/download-home/all-rib-defect-fixes directory.
2. Run the check-version-and-apply-defect-fix.sh from the /rib-home/maintenance-home/bin directory.
3. Run the rib-home/application-assembly-home/bin/rib-app-compiler.sh script from the rib-home/application-assembly-home/bin directory.
4. Run the rib-home/deployment-home/bin/rib-app-deployer.sh script from rib-home/deployment-home/bin directory to the appropriate rib-<app>s.

The tool check-version-and-apply-defect-fix.sh will perform version compatibility checks and will update the RIB inventory XML file.

inventory-management

The RIB jars and XML files in rib-home are tracked through an XML file called rib-inventory-info.xml located in the rib-home/maintenance-home/history-repository/ directory. This file is initially created when the RIB installer, or user, executes the check-version-and-unpack tool the first time to extract the RIB application packs and the functional artifacts. Thereafter this file is updated and tracks the file change history of the jars and xml files in the rib-home system.

Command Line Option	Description
-update-current-inventory	Scans the rib-home file system and updates the inventory database.
-generate-file-change-history-report	Generates a report of how the files in the rib-home file system have changed over time.
-generate-defect-fix-applied-report	Generates a report of what defect fixes have been applied to rib-home on this system.
-generate-defect-fix-detail <defect-fix-id>	Displays the long defect resolution description for a given defect fix id.

setup-security-credential

The user names and passwords required to install RIB components are stored as security credentials in a wallet file located in the rib-home/deployment-home/conf/security/ directory. The file is initially created when the RIB installer, or user, executes the rib-app-compiler tool with the setup-security-credential argument the first time and enters all the user names and passwords required for installing RIB components. Thereafter, this file can be modified using the setup-security-credential script located in rib-home/maintenance-home/bin directory. After updating existing credentials, the user must run the rib-app-compiler tool again and redeploy the rib apps to use the new credentials.

Command Line Option	Description
-setup-aq-credential <aq-id>	Updates the security credential for the AQ JMS Server ID specified in rib-deployment-env-info.xml
-setup-weblogic-credential<wls-id>	Updates security credential for the specified WebLogic instance.

Command Line Option	Description
-setup-admin-gui-credential rib-<app>	Updates security credential for the RIB Admin GUI user for the specified RIB App.
-setup-error-hospital-credential rib-<app>	Updates security credential for the error hospital database user for the specified RIB App.
-setup-app-database-credential rib-<app>	Updates security credential for the application database for the specified RIB App.
-setup-jndi-credential rib-<app>	Updates security credential for the remote JNDI for the specified RIB App.

Hot Fix Installation Reports

The following HTML reports can be used to verify the successful installation of RIB hot fixes:

- defect-fix-applied-report.html
- file-change-history-report.html
- defect-fix-detail-<defect-fix-id>.html

These reports are available at `rib-home/maintenance-home/history-repository/HTML-Report`.

Sample: file-change-history-report

File Change History Report

File ..\..\rib\home\integration\lib\internal\build\rib\admin.gui.war						
content-creation-date	creation-date-on-local-file-system	md5	size	defect-fix-ref	defect-tracking-identifier	
Tue Jul 08 11 56 22 IST 2008	Wed Jul 16 12 29 47 IST 2008	a7735470cfeeedbc0965c5234b9b19e	76601	NO_ASSOCIATED_DEFECT_FIX_REF	-NA-	
Back to top						
File ..\..\rib\home\integration\lib\internal\build\rib\app.builder.jar						
content-creation-date	creation-date-on-local-file-system	md5	size	defect-fix-ref	defect-tracking-identifier	
Wed Jul 16 12 29 42 IST 2008	Wed Jul 16 12 29 47 IST 2008	cc0899755a5e8bdec94c5861b3ed5e6	727768	NO_ASSOCIATED_DEFECT_FIX_REF	-NA-	
Back to top						
File ..\..\rib\home\integration\lib\internal\build\rib\config.agent.war						
content-creation-date	creation-date-on-local-file-system	md5	size	defect-fix-ref	defect-tracking-identifier	
Wed Jul 16 12 29 42 IST 2008	Wed Jul 16 12 29 48 IST 2008	e9852a3611451402420292e5dd3513	2310	NO_ASSOCIATED_DEFECT_FIX_REF	-NA-	
Back to top						
File ..\..\rib\home\integration\lib\internal\build\rib\private.app.plugin.jar						
content-creation-date	creation-date-on-local-file-system	md5	size	defect-fix-ref	defect-tracking-identifier	
Fri Jun 06 14 07 36 IST 2008	Wed Jul 16 12 29 47 IST 2008	5b437e72874a22eb302853f71a9ab7f	5197	NO_ASSOCIATED_DEFECT_FIX_REF	-NA-	
Back to top						

Sample: defect-fix-detail-<defect-fix-id>

Defect Fix Applied Report

defect.fix	defect.tracking.identifier	short.defect.description
19	HPQCDdefect_469	ASNOutToASNInLoc TAFR shuts itself down after the Transfer is dispatched
20	BugDB#7209670	ERROR HOSPITAL RETRY PROBLEM FOR ENTRIES WITH JMS REASON CODE
21	BugDB#7209670	ERROR HOSPITAL RETRY PROBLEM FOR ENTRIES WITH JMS REASON CODE
22	BugDB#7209670	ERROR HOSPITAL RETRY PROBLEM FOR ENTRIES WITH JMS REASON CODE
23	BugDB#7209670	ERROR HOSPITAL RETRY PROBLEM FOR ENTRIES WITH JMS REASON CODE

rib-adapter-controller

The rib-adapter-controller a set of tools that perform RIB adapter control functions such as start/stop and subscriber check, The command line options and usage are summarized here. See "[RIB Components Start and Stop](#)."

Start Flow

Starts all adapters in a message flow for a given family or family list (comma separated list without any space)

```
start integration-message-flows <family-name-list>[no-subscriber-check]
```

Description:

1. For a given family, it identifies all message flow IDs in which this family directly or indirectly participates.
2. Using the message flow IDs defined in the rib-integration-flows.xml, it connects to all application servers where the respective rib-apps are deployed.
3. It starts the adapters in the order as defined in the message flows.
4. It checks if durable subscribers exist before starting an adapter.
5. It ignores all rib apps that are not in scope.

Examples:

```
rib-adapter-controller.sh start integration-message-flows Alloc
rib-adapter-controller.sh start integration-message-flows Alloc,Order
```

Stop Flow

Stops all adapters in a message flow for a given family or family list (comma separated list without any space).

```
stop integration-message-flows <family-name-list>
```

Description:

1. For a given family it identifies all message flow IDs in which this family directly or indirectly participates.
2. Using the message flow IDs in the rib-integration-flows.xml, it connects to all application servers where the respective rib-apps are deployed.
3. It stops the adapters in the order as defined in the message flows.
4. It ignores all rib apps that are not in scope.

Examples:

```
rib-adapter-controller.sh stop integation-message-flows Alloc
rib-adapter-controller.sh stop integation-message-flows Alloc,Order
```

List Flow

Stops all adapters in a message flow for a given family or family list (comma separated list without any space).

```
list integation-message-flows <family-name-list>
```

Description:

1. It will display all message node ids for all message flows that are part of the given family.
2. It lists the adapters in the order as defined in the message flows.
3. It ignores all rib apps that are not in scope.

Examples:

```
rib-adapter-controller.sh list integation-message-flows Alloc
rib-adapter-controller.sh list integation-message-flows Alloc,Order
```

Start Adapters By Type

Starts all adapters by type given a rib-app or rib-app-list (comma separated list without any space).

```
start rib-app-adapters-by-type <sub,tafr,pub,hosp_retry,all><rib-app-list>
[no-subscriber-check]
```

Description:

1. For every adapter type specified in the input it collects the adapter instances from the given rib-app-list.
2. It reorders the input adapter types to start in the correct order.
3. It connects to the respective applications servers where rib-apps are deployed.
4. It starts the sub adapters first in all rib-apps, and then it moves on to start all the tafr adapters in all rib-apps and so on.
5. It checks if durable subscribers exist before starting an adapter.
6. It ignores all rib apps that are not in scope.

Examples:

```
rib-adapter-controller.sh start rib-app-adapters-by-type sub,tafr rib-rms
rib-adapter-controller.sh start rib-app-adapters-by-type pub,sub rib-rms,rib-sim
rib-adapter-controller.sh start rib-app-adapters-by-type all rib-rms,rib-sim
```

Stop Adapters by Type

Stops all adapters by type given a rib-app or rib-app-list (comma separated list without any space).

```
stop rib-app-adapters-by-type <sub,tafr,pub,hosp_retry,all><rib-app-list>
```

Description:

1. For every adapter type specified in the input it collects the adapter instances from the given rib-app-list.
2. It connects to the respective applications servers where rib-apps are deployed.
3. It stops the first adapter type first in all rib-apps, and then it moves on to stop the second adapter types in all rib-apps and so on.
4. It ignores all rib apps that are not in scope.

Examples:

```
rib-adapter-controller.sh stop rib-app-adapters-by-type sub,tafr rib-rms,rib-sim
rib-adapter-controller.sh stop rib-app-adapters-by-type pub,sub
rib-adapter-controller.sh stop rib-app-adapters-by-type all rib-rms,rib-sim
```

Start Adapter

Starts individual adapter instances. Adapter instance must be fully qualified as rib-<app>.<Family>_<type>_<n>. A comma separated list of adapter instances names can also be provided.

```
start rib-app-adapter-instance <rib-app.Family_type_1-list>[no-subscriber-check]
```

Description:

1. Checks if durable subscribers exist before starting an adapter.
2. Starts the adapter instance.

Examples:

```
rib-adapter-controller.sh start rib-app-adapter-instance rib-rms.Alloc_pub_1
rib-adapter-controller.sh start rib-app-adapter-instance rib-rms.Alloc_pub_
1,rib-sim.ASNIn_sub_1
```

Stop Adapter

Stops individual adapter instances. Adapter instances must be fully qualified as rib-<app>.<Family>_<type>_<n>. A comma separated list of adapter instances names can also be provided.

```
stop rib-app-adapter-instance <rib-app.Family_type_1-list>
```

Description:

- Stops the adapter instance.

Examples:

```
rib-adapter-controller.sh stop rib-app-adapter-instance rib-rms.Alloc_pub_1
rib-adapter-controller.sh stop rib-app-adapter-instance rib-rms.Alloc_pub_
1,rib-sim.ASNIn_sub_1
```

Test Durable Subscriber for Adapter

Tests if durable subscriber exist for topics associated with a given adapter class def. Adapter class def must be fully qualified as rib-<app>.<Family>_<type>. A comma separated list of adapter class def names can also be provided.

```
test durable-subscriber-exist-for-adapter-class-def <rib-app.Family_type-list>
```

Description:

1. Finds out the topic names the input rib app adapter class def publishes to.
2. For each topic it publishes to, it checks to see if there is a durable subscriber registered.

Examples:

```
rib-adapter-controller.sh test durable-subscriber-exist-for-adapter-class-def
rib-rms.Alloc_pub
rib-adapter-controller.sh test durable-subscriber-exist-for-adapter-class-def
rib-rms.Alloc_pub,rib-tafr.ASNOutToASNOutAT_tafir
```

Test Durable Subscriber for RIB App

Tests if durable subscriber exist for all publishing topics associated with a given rib-app or rib-app-list (comma separated list without any spaces).

```
test durable-subscriber-exist-for-rib-app <rib-app-list>
```

Description:

1. Finds out all adapter instances that publish to a topic name for the given rib-app-list.
2. For each topic it publishes to, it checks to see if there is a durable subscriber registered.

Examples:

```
rib-adapter-controller.sh test durable-subscriber-exist-for-rib-app rib-rms
rib-adapter-controller.sh test durable-subscriber-exist-for-rib-app
rib-rms,rib-sim
```

List RIB App Adapters

Lists all adapter instance for a given rib-app or rib-app-list (comma separated list without any spaces).

```
list rib-app-adapters <rib-app-list>
```

Description:

Lists all adapters that are part of the rib-app.

Examples:

```
rib-adapter-controller.sh list rib-app-adapters rib-rms
rib-adapter-controller.sh list rib-app-adapters rib-rms,rib-sim
```

RIB Deployment Configuration File Editor

The RIB Deployment Configuration File Editor is an application used to configure the rib-deployment-env-info.xml file, following installation. The editor tool simplifies user interaction with the XML file by hiding the raw text form of XML. It provides a user interface for adding, removing, and rearranging the elements of the RIB configuration.

Note: See the "[RIB App Builder Tools](#)" in this chapter and "[rib-deployment-env-info.xml](#)" in Chapter 3, "Backend System Administration and Logging."

The tool is located in the RDMT package and installed with RDMT in the <rib-home>/tools-home/RDMT directory. It is available as a menu selection from the ribadmin sub menu. See Chapter 9, "[Diagnostic and Monitoring Tools](#)."

Note: The editor is a GUI application. To execute it on a host other than the one on which RDMT is installed, use an X server, such as Xceed, and set the DISPLAY environment.

Important Installation Warning

All rib-app-builder tools use the rib-deployment-env-info.xml as the single source of truth about the deployment configuration. See "[RIB Deployment Configuration File Editor](#)" in Chapter 3, "Backend System Administration and Logging."

All tools use the values in this file. Editing the file directly affects the compilation, configuration, and deployment of the rib-apps. Use extreme caution and understand the ramification of the values being manipulated.

Note: See the *Oracle RIB Integration Bus Implementation Guide*.

Before editing the source file in rib-home, make a backup of the file and place it securely outside of rib-home. Do not create a backup in the rib-home.

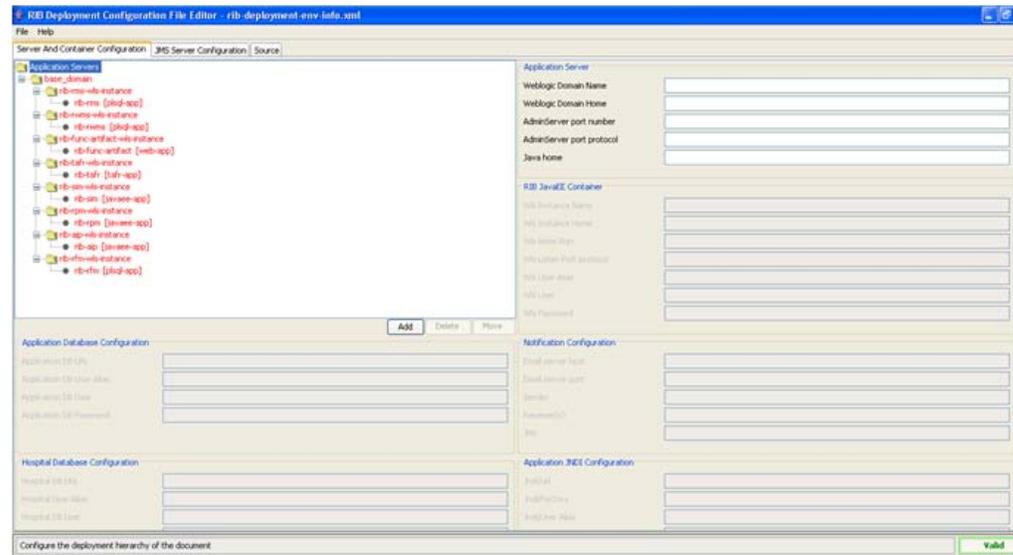
Key Rule

The rib-app-builder tools scan and check versions of all files within rib-home (except for tools-home). The processes do not allow files to have the same name with only an additional extension.

Editor Usage

The following is an illustration of the editor interface.

Note: See the online help provided in the tool for additional details.



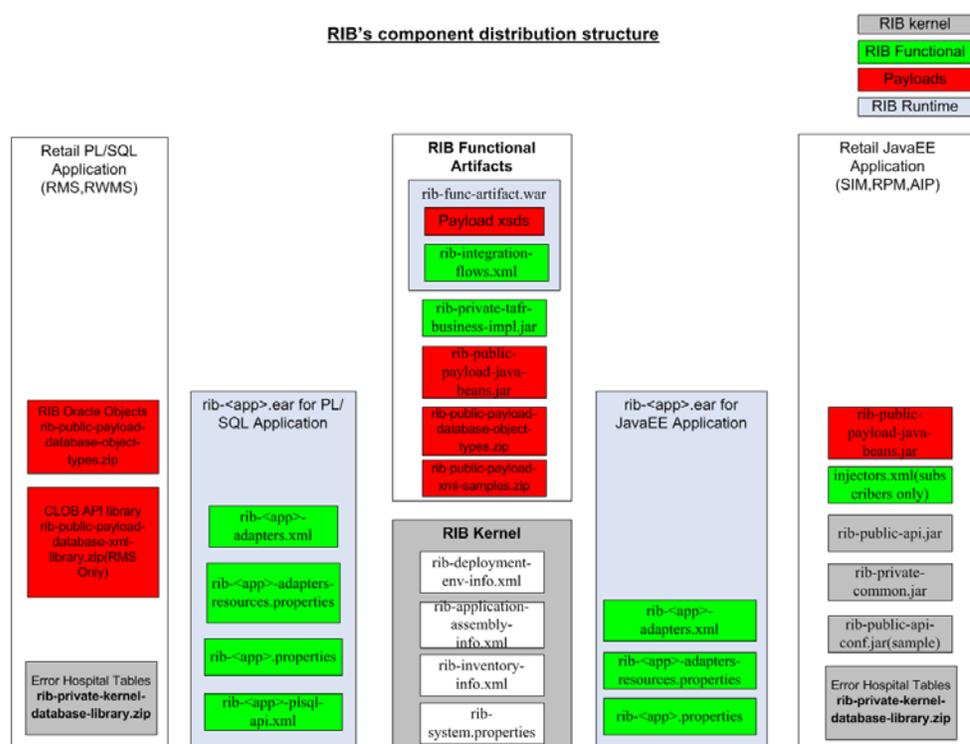
The RIB Deployment Configuration File Editor allows users to do the following:

- Add, delete and move applications.
- Add, delete and move WebLogic instances.
- Add and delete Application Server instances.
- Configure JMS servers.

To edit files using the editor, do the following:

1. Select File from the menu bar. Click **Open**.
2. Navigate to the directory containing rib-deployment-env-info.xml.
3. Select the file and open it.
4. Complete the required task (for example, add, delete, or move).
5. Save the file using the File menu.

Backend System Administration and Logging



This illustration includes the names of actual RIB files and shows where they are located in the deployment picture.

rib-<app>-adapters.xml

This file specifies all the adapter instances needed by RIB to interact with an application. Each rib-<app_name> has its own rib-<app-name>_adapter.xml.

The file is located in the rib-home/application-assembly/rib-<app> directory. After deployment, it is found in the path \$application_instance_home/\$application_name, where \$application_instance_home is the WebLogic instance path where the application is deployed.

These are the standard RIB defined adapter types.

<subscribers> element

<subscribers> elements consist of multiple occurrences of <message-driven> elements, that define all the subscribers available for a particular application. Each <message-driven> element consists of ID (the ID for the adaptor) and initialState (the initial state of the adaptor) attributes. The initialState attribute for <message-driven> adaptors accepts two values: running and stopped.

```
<subscribers>
  <message-driven id="ASNIn_sub_1" initialState="running" />
  <message-driven id="ASNOut_sub_1" initialState="running" />
```

Note: The only valid states are running and stopped, and they are case sensitive.

<publisher> element

<publisher> elements consist of multiple occurrences of <timer-driven> or <request-driven> elements, used to define all the publishers available for a particular application.

<timer-driven>

<timer-driven> is used to define publishers for PL/SQL (RMS and RWMS) applications. Each <timer-driven> element consists of an id (specifies id for adaptor), initialState (specifies the initial state of the adaptor) and timeDelay (delay after which the GETNXT needs to call each time) attributes. The initialState attribute for <timer-driven> adaptors accepts two values; running and stopped. This consists of an element called <timer-task> which specifies the implementation details of the adaptor. The <timer-task> element specifies the GETNXT implementation through the <class> element.

```
<publishers>
  <timer-driven id="Alloc_pub_1" initialState="running" timeDelay="10">
    <timer-task>
      <class name="com.retek.rib.app.getnext.impl.GetNextTimerTaskImpl" />
      <property name="maxChannelNumber" value="1" />
    </timer-task>
  </timer-driven>
```

<request-driven> <request-driven> is used to define publishers for Java EE (Oracle Retail Price Management (RPM), Oracle Retail Store Inventory Management (SIM), and Oracle Retail Advance Inventory Planning (AIP) applications. Each <request-driven> element consists of id (specifies id for adaptor) and initialState (specifies the initial state of the adaptor) attributes. The initialState attribute has a value of notConfigurable.

```
<publishers>
  <request-driven id="ASNOut_pub_1" initialState="notConfigurable" />
  <request-driven id="DSDReceipt_pub_1" initialState="notConfigurable" />
```

<hospital> element

<hospital> element specifies hospital related adaptor information. The structure is very similar to the <publisher> element except that the name and value attribute in the property element defines the different hospital adaptor types.

```
<hospitals>
  <timer-driven id="sub_hosp_0" initialState="running" timeDelay="10">
```

```

        <timer-task>
            <class name="com.retek.rib.j2ee.ErrorHospitalRetryTimerTask" />
            <property name="reasonCode" value="SUB" />
        </timer-task>
    </timer-driven>

```

rib-<app>-adapters-resource.properties

These are internationalizable strings for internal rib adapter key names.

Example:

```

sub_all.name=Subscribers
sub_all.desc=Manages all subscribers at the same time.

```

```

ASNIn_sub_1.name=ASNIn Subscriber, channel 1
ASNIn_sub_1.desc=Subscriber for the ASNIn family through channel 1.

```

```

ASNOut_sub_1.name=ASNOut Subscriber, channel 1
ASNOut_sub_1.desc=Subscriber for the ASNOut family through channel 1.

```

rib-<app>-plsql-api.xml

This configuration file is specific to RMS and RWMS. The RIB interfaces with RMS and RWMS through two database procedures: GETNXT and CONSUME. This file contains the calling signatures for these procedures, the parameters to be configured before calling these procedures, and the implementation class for handling the objects returned from these procedures.

rib-<app>.properties

These are internationalizable strings for internal rib adapter key names.

Property Name and Default Value	Description
facility_id defaultValue = "facility_id";	This property is used to refer to the warehouse routing configuration. The value of this property is used to construct the facility type
dc_dest_id defaultValue = "*";	This property is used to refer to the warehouse distribution center. Destination ID
facility_type_default defaultValue = "PROD";	Specifies the default facility type to be used by RWMS publishing adapters for calls to RWMS.

rib-system.properties

All properties for RIB have been classified into kernel properties and application properties. This file contains kernel properties that are used specifically for the functioning of the RIB kernel. They are mostly related to hospital retry configuration, payload locations, or alerting.

Property Name and Default Value	Description
hospital_attempt_max defaultValue = "5";	This property refers to the maximum number of attempts to try to push this record through the RIB automatically. Once this retry count is exceeded, the message remains in the RIB Hospital DB but is no longer retried automatically
hospital_attempt_delay defaultValue = "10";	This property refers to the value (in seconds) used to calculate the next attempt time.
hospital_attempt_delayIncrement defaultValue = "10";	This property refers to the value (in seconds) used to calculate the next attempt time. The next attempt time is calculated as: hospitalAttemptDelay + (hospitalAttemptDelyIncrement * attempt count). This is done so that the delay between each attempt is longer than the previous delay.
numOfRecordsToRetry defaultValue = "20";	This property refers to the maximum number of RIB Hospital records to be retried in one RIB Hospital retry attempt.
xml_schema_base_url defaultValue = "http://localhost:8888/rib-func-artifact";	This property refers to the location of web application (rib-func-artifact) which has RIB related XML Schema (XSD) files.
log.default.file_path defaultValue=\$DOMAIN_HOME/servers/\$\$SERVERNAME/logs/\$APP_NAME	This property refers to the location where log files are created by the RIB application. By default this location is in the logs/app_name directory inside the WebLogic instance home where the app has been installed.
mail_smtp_host defaultValue = "mail.smtp.host";	This property is used to identify the smtp host from which to send out emails.
mail_smtp_port defaultValue = "25";	This property is used to identify the smtp port from which to send out emails.
mail_smtp_from defaultValue = "admin@company.com";	This property refers to the email id that the RIB platform needs to use to send the emails for administrative purposes.
war_http_port defaultValue = "9080";	This property refers to the port number used by the web based Hospital Retry Admin Tool.
wls.wallet.file.location defaultValue=\$DOMAIN_HOME/serves/\$SERVERNAME	This property refers to the wallet file that contains the user name/password details for connecting to the WebLogic instance. The user should not change this value.
wls.wallet.map.name defaultValue=rib-rms-wls	This property refers to the map name that is stored in the wallet file for connecting to the WebLobic instance. The user should not change this value.
wls.wallet.user.alias defaultValue=rib-rms-wls-user-alias	This property refers to the alias stored in the wallet file for connecting to the WebLogic instance. The user should not change this value.

rib-integration-flows.xml

This file is the single source of all values used by the RIB Application Builder tools to define and configure the JMS topics as well as perform start and stop activities, including subscriber checks. For RIB deployments this file should not be edited.

This file is packaged and deployed as part of the rib-func-artifacts war file.

Example:

```
<message-flow id="1">
  <node id="rib-rms.Alloc_pub" app-name="rib-rms"
    adapter-class-def="Alloc_pub" type="DbToJms">
    <in-db>default</in-db>
    <out-topic>etAllocFromRMS</out-topic>
  </node>
  <node id="rib-tafr.Alloc_tafr" app-name="rib-tafr"
    adapter-class-def="Alloc_tafr" type="JmsToJms">
    <in-topic>etAllocFromRMS</in-topic>
    <out-topic name="topic-name-key-iso">etStockOrdersISO</out-topic>
    <out-topic
name="topic-name-key-wh">etStkOrdersFromRIBToWH{*}</out-topic>
  </node>
  <node id="rib-sim.StockOrder_sub" app-name="rib-sim"
    adapter-class-def="StockOrder_sub" type="JmsToDb">
    <in-topic>etStockOrdersISO</in-topic>
    <out-db>default</out-db>
  </node>
  <node id="rib-rwms.StockOrder_sub" app-name="rib-rwms"
    adapter-class-def="StockOrder_sub" type="JmsToDb">
    <in-topic>etStkOrdersFromRIBToWH1</in-topic>
    <out-db>default</out-db>
  </node>
</message-flow>
```

rib-deployment-env-info.xml

This file is the single source of all values used in the RIB App Builder tools and is the only (or should be the only) file that requires editing for using them. The RIB Installer gathers the appropriate values from the user, constructs the file, and invokes the appropriate tools.

For example, when the RIB Application Builder is used to extract error hospital tables from an application schema, this file supports those tables.

The RIB Application Builder tools can be executed independent of the RIB installer tool. In some cases the file must be edited manually.

app-in-scope-for-integration

This section defines what applications are in scope for this environment.

Example:

```
<app id="rms" type="plsql-app"/>
<app id="tafr" type="tafr-app"/>
<app id="sim" type="javaee-app"/>
<app id="rwms" type="plsql-app"/>
<app id="rpm" type="javaee-app"/>
<app id="rfm" type="plsql-app"/>
```

rib-jms-server

This section defines the JMS server information.

Example:

```
<jms-server-home>linux1@linux1:/home/oracle/oracle/product/10.2.0/db_
1</jms-server-home>
<jms-url>jdbc:oracle:thin:@linux1:1521:ora10g</jms-url>
<jms-port>1521</jms-port>
<jms-user>riabaq</jms-user>
<jms-password>riabaq</jms-password>
```

rib-application-server

This section defines the WebLogic Server information.

Example:

```
<weblogic-domain-name>base_domain</ weblogic-domain-name >
<weblogic-domain-home>soa1@linux1:/home/soa1/Oracle/Middleware/user_
projects/domains/base_domain</weblogic-domain-home>
<weblogic-admin-server-port>7001</ weblogic-admin-server-port >
<java-home>/usr/java/jdk1.5.0_01</java-home>
```

rib-javaee-containers

This section defines the WebLogic instances for each of your rib-<app> applications that are in-scope.

Example:

```
<wls id="rib-rms-wls-instance">
<wls-instance-name>rib-rms-wls-instance</wls-instance-name>
<wls-instance-home>
soa1@linux1:/home/soa1/Oracle/Middleware/user_projects/domains/base_
domain/servers/rib-rms-wls-instance</wls-instance-home>
<wls-listen-port protocol="http">7003</wls-listen-port>
<wls-user-alias>rib-rms-wls-user-alias</wls-user-alias>
</wls>
```

rib-applications

This section defines the rib-<app> specific information for each applicable rib-<app>.

Example 1: RIB-RMS (for app-type=PL/SQL)

```
<rib-app id="rib-rms" type="plsql-app">
<deploy-in refid="rib-rms-wls1" />
<rib-admin-gui>
<web-app-url>URL to the rib admin gui web app.</web-app-url>
<web-app-user-alias>rib-rms_rib-admin-gui_
web-app-user-alias</web-app-user-alias>
</rib-admin-gui>

<notifications>
<email>
<email-server-host>mail.oracle.com</email-server-host>
<email-server-port>25</email-server-port>
<from-address>rib@oracle.com</from-address>
<to-address-list>rib@oracle.com</to-address-list>
```

```

    </email>
    <jmx/>
</notifications>

```

Example 2: Application Database (for app-type=PL/SQL)

```

<app-database>
<app-url> DB host URL for pl/sql application</app-url>
<app-db-user-alias>rib-rms_app-database_user-name-alias</app-db-user-alias>
</app-database>

```

Example 3: Java EE application with JNDI information defined

```

<jndi>
<url>t3://mspdv170:18022/sim-app</url>
<factory>weblogic.jndi.WLInitialContextFactory</factory>
<user-alias>sim_jndi_user-name-alias</user-alias>
</jndi>

```

Example 4: Error Hospital Database (for app-type=JavaEE/TAFR)

```

<error-hospital-database>
<hosp-url>jdbc:oracle:thin:@mspdv170:1522:rrts170dv64</hosp-url>
<hosp-user-alias>rib-rms_error-hospital-database_user-name-alias</hosp-user-alias>
</error-hospital-database>

```

commons-logging.properties

The RIB uses the Apache Commons Logging subsystem as the logging interface. For RIB deployments this file should not be edited.

log4j.xml

The log4j Open Source software is used to control all RIB logging. This software requires the log4j.xml file to configure the file name, logging level, and type of file used.

rib-app-builder-paths.properties

For RIB deployments this file should not be edited

rib-application-assembly-info.xml

This is a non editable file that describes the structure of the rib-<app>.ear and the resources it uses.

retail_service_config_info_ribserver.xml

This is a non editable file that describes the service related configuration used by rib-<app> to identify the relevant service implementations.

remote_service_locator_info_ribserver.xml

This is a non editable file that describes the J NDI related configuration used by rib-<app> to invoke remote EJB 's hosted on Java retail apps (for example, RPM and SIM). This file is built runtime, based on the information provided in rib-deployment-env-info.xml.

RIB Logging

All logging in the RIB is through log4j, the Apache Software Foundation's Open Source software. For details about log4j visit the Apache Software Foundation's log4j home page.

Log Level Recommendations

The logging level must be adjusted for the phase of the deployment. What is appropriate in development and test (DEBUG) is not appropriate in production (INFO).

There are some logs such as audit and timing that may be used differently at certain phases as well. Audit is either on (DEBUG) or off (INFO), the same is true with timings.

Note: See "[RIB Timings Utility](#)."

As a rule the appropriate level will always be INFO.

Changing Logging Levels

RIB use of log4j allows the control of logging levels to suit the deployment and situation. There are two methods of setting the logging levels; directly manipulating the log4.xml file using a text editor, and the RIB Administration GUI.

RIB Administration GUI

The RIB Administration GUI allows control of the logging levels for each adapter individually. It permits the change to affect only the runtime logging and is dynamic. It also provides the ability to persist the change so that the adapters retains that level when restarted. This is the recommended approach.

log4j.xml Configuration File

The RIBLOGS log4j.xml file can be directly edited. This requires that the adapter(s) be bounced for the change to take effect. See the sections below on what to edit related to the type of log (RIBLOG, Timing Log, and so on).

Adapter Logging (RIBLOGS)

The RIB adapter code contains logging logic that writes all of it runtime logs to the RIBLOG log files. The logs are written to the path <rib-application_instance_home>/logs/<rib-app>.

Example:

```
/u01/rrtswls/Oracle/Middleware/user_projects/domains/base_
domain/servers/rib-rms-wls-instance/logs/rib-rms
```

The RIBLOG file names are in this format: <adapter-instance-name>.rib.log.

Example:

```
Alloc_pub_1.rib.log
ASNIn_sub_1.rib.log
ASNOut_sub_1.rib.log
```

To enable this function, parameters must be set per adapter.

Be careful because there are multiple entries for each adapter instance in the log4j.xml file. Search for the section of the log4j.xml file:

```
<!--RIB Appender for adapterInstance: Alloc_pub_1-->
```

RIB Timing Logs

The RIB messaging components code is instrumental to log timing entries on the internal activities whenever they create, transform, route, filter, or subscribe to messages on the RIB. These timings logs are written using the log4j logging mechanism.

The timings log files follow the name convention <adaptor-instance-name>.timings.log and are found in the same locations as the RIBLOGS.

Typically, one timings log file is created per component (EJB or other) which holds the entries for that component. These files are cumulative, meaning that they do not get overwritten with every initialization of the component, but they append new entries to the current information already recorded. The files do roll over after they reach a certain configurable size and backup files are created to preserve previous entries.

Each entry in the timings log represents a timestamp of a particular event in the RIB component, listing the date and time information, name of the component, thread ID and a distinct message for each event. The list of time stamped events includes such items as the start time and/or end time of the following actions:

- Overall publication, subscription, routing, or transformation process
- Calls to store procedures (getnxt and consume)
- Actual publication and subscription of messages to and from the JMS server
- Calls to the RIB Hospital to check for dependencies and insert messages
- Calls to other applications to process messages after subscription (injectors)

The log4j.xml file must have the "level value" property set to DEBUG. This tag is not normally present in the standard log4j.xml file, it must be added. The example below shows how and where.

Be careful, because there are multiple entries for each adapter instance in the log4j.xml file. Search for the section of the log4j.xml file:

```
<!--Timings Logger for adapterInstance: -->".
```

Before:

```
<logger additivity="false" name="rib.pub.timings.Order_pub_1">
  <!-- Possible levels are TRACE, DEBUG, INFO, WARN, ERROR and FATAL -->
  <level value="INFO"/>
  <appender-ref ref="appender.rib.pub.timings.Order_pub_1"/>
</logger>
```

After:

```
<logger additivity="false" name="rib.pub.timings.Order_pub_1">
```

```

        <!-- Possible levels are TRACE, DEBUG, INFO, WARN, ERROR and FATAL -->
        <level value="DEBUG"/>
        <appender-ref ref="appender.rib.pub.timings.Order_pub_1"/>
    </logger>

```

RIB Audit Logs

The RIB has an auditing feature that logs a message as it passes through the RIB infrastructure. Each messaging component can be set to write the message, and only the message, to a separate log file. This allows the tracing of message content from publication to subscription, and all steps, such as a TAFR, in between.

There are two benefits to this mechanism: the ability to audit each step, and the ability to create a recovery plan. The messages can be played back, without effort being spent to extract them from inside other more systemic log files.

The log4j.xml can be edited to remove the <audit-entry> tag from the output and to have only the message in the file.

```

<!--Audit Appender for adapterInstance: ASNIn_sub_1-->
    <appender class="org.apache.log4j.FileAppender"
        name="appender.rib.sub.audit.ASNIn_sub_1">
        <param name="File" value= "/u00/webadmin/product/10.3.3/WLS/user_
projects/domains/rib_domain/servers/rib-rms-server/logs/rib-rms/ASNIn_sub_
1.audit.log"/>
        <!--param name="MaxFileSize" value="2048KB"/-->
        <!--param name="MaxBackupIndex" value="1"/-->
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="&lt;audit-entry
audit-time=&quot;%d{yyyy.MM.dd
HH.mm.ss,SSS}&quot;&gt;&gt;%n%m%n&lt;/audit-entry&gt;%n"/>
        </layout>
    </appender>

```

Remove the "value=" in the ConversionPattern with %m%n

The RIB can also log a set of audit logs used to audit all the events processed by RIB. To enable this function, parameters must be set per adapter.

Proceed cautiously because there are multiple entries for each adapter instance in the log4j.xml file. Search for the section of the log4j.xml file:

```

<!--Audit Logger for adapterInstance: ItemLoc_pub_1-->.

```

Before:

```

<!--Audit Logger for adapterInstance: ItemLoc_pub_1-->
    <logger additivity="false" name="rib.pub.audit.ItemLoc_pub_1">
        <!-- Possible levels are TRACE, DEBUG, INFO, WARN, ERROR and FATAL -->
        <level value="INFO"/>
        <appender-ref ref="appender.rib.pub.audit.ItemLoc_pub_1"/>
    </logger>

```

After:

```

    <!--Audit Logger for adapterInstance: ItemLoc_pub_1-->
    <logger additivity="false" name="rib.pub.audit.ItemLoc_pub_1">
        <!-- Possible levels are TRACE, DEBUG, INFO, WARN, ERROR and FATAL -->
        <level value="DEBUG"/>
        <appender-ref ref="appender.rib.pub.audit.ItemLoc_pub_1"/>

```

Sample Log Entry:

```

<audit-entry audit-time="2008.01.28 11:37:57,642">
<?xml version="1.0" encoding="UTF-8"?>
<RibMessages
xmlns="http://www.oracle.com/retail/integration/rib/RibMessages"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oracle.com/retail/integration/rib/RibMessages
http://mspdev85:7777/rib-func-artifact/integration/xsd/RibMessages.xsd" >
<ribMessage><family>Banner</family><type>BannerCre</type> <id>1</id>
<ribmessageID>Banner_pub_1|2008.01.28 11:37:57.500|6936</ribmessageID>
<publishTime>2008-01-28 11:37:57.500 CST</publishTime>
<messageData>&lt;BannerDesc
xmlns="http://www.oracle.com/retail/integration/payload/BannerDesc"
xmlns:ribdate="http://www.oracle.com/retail/integration/payload/RIBDate"
;
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oracle.com/retail/integration/payload/BannerDe
sc http://mspdev81:7777/rib-func-artifact/payload/xsd/BannerDesc.xsd
http://www.oracle.com/retail/integration/payload/RIBDate
http://mspdev81:7777/rib-func-artifact/payload/xsd/RIBDate.xsd"
&lt;banner_id&gt;1&lt;/banner_id&gt; &lt;banner_name&gt;B&M&lt;/banner_
name&gt;&lt;/BannerDesc&gt;
</messageData>
<customData></customData><customFlag>F</customFlag>
</ribMessage>
</RibMessages>

```

Other RIB Management Logs**deploy.rib.log**

This log will track the source rib-app-builder home that pushed the changes to this WebLogic instance. For example:

```

Uploading configuration file from machine(mspdev81)
dir (/stage/Rib1300-ms7.1/Rib1300ForAll13xxApps/rib-home/deployment-home/bin/../../
../rib-home) at (Mon Jan 28 11:15:57 PST 2008).

```

management.rib.log

The RIB maintains a management log which is used to keep track of the WebLogic instance on the whole.

This is usually written during the startup of an WebLogic instance. The recommendation is that each rib-app be deployed in a separate WebLogic instance, so management logs are specific to a rib-app.

The management log writes RIB information common to all the components like loading property files and creating logging files.

Example:

```

2008-02-01 14:33:23,928 [AJPRequestHandler-RMICallHandler-6] DEBUG
com.retek.rib.management.adapters.client.action.StopAdapterAction - Invoking
operation to stop the adapters
2008-02-01 14:33:23,928 [AJPRequestHandler-RMICallHandler-6] DEBUG
com.retek.rib.monitor.engine.MBeanAbstractFactory - Invoking MBean operation
domain(rib-rms) objectNameProperty(level=adapters,type=sub,name=Receiving_sub_1)
methodName(stop) parameter([Ljava.lang.Object;@1452a1)
signature([Ljava.lang.String;@3d06a4).

```

global.rib.log**Example:**

```

2008-02-06 10:14:26,688 [AJPRequestHandler-RMISCallHandler-7] DEBUG
retex.com.retek.rib.ui.view.tags.IteratePropertyTag.com.retek.rib.management.adapt
ers.model.AdapterTypes - Invoking Operation returnStatusForAll of MBean.
2008-02-06 10:14:26,777 [AJPRequestHandler-RMISCallHandler-7] DEBUG
retex.com.retek.rib.ui.view.tags.IteratePropertyTag.com.retek.rib.monitor.engine.M
BeanAbstractFactory - Invoking MBean operation domain(rib-rms)
objectNameProperty(level=types,type=pub,name=pub_all)
methodName(returnStatusForAll) parameter(null) signature(null).
2008-02-06 10:14:26,780 [AJPRequestHandler-RMISCallHandler-7] DEBUG
retex.com.retek.rib.ui.view.tags.IteratePropertyTag.com.retek.rib.management.adapt
ers.model.AdapterTypes - Operation returnStatusForAll for type pub invoked
successfully :<type name="pub"><adapter id="Alloc_pub_1" name="Alloc Publisher,
channel 1" state="running" /><adapter id="SeedData_pub_1" name="SeedData
Publisher, channel 1" state="running" /><adapter id="SeedObj_pub_1" name="SeedObj
Publisher, channel 1" state="running" /><adapter id="WOOOut_pub_1" name="WOOOut
Publisher, channel 1" state="running" /><adapter id="Banner_pub_1" name="Banner
Publisher, channel 1" state="running" /><adapter id="Transfers_pub_1"
name="Transfers Publisher, channel 1" state="running" /><adapter
id="RcvUnitAdj_pub_1" name="RcvUnitAdj Publisher, channel 1" state="running"
/><adapter
id="Vendor_pub_1" name="Vendor Publisher, channel 1" state="running" /><adapter
id="WH_pub_1" name="WH Publisher, channel 1" state="running" /><adapter
id="RTVReq_pub_1" name="RTVReq Publisher, channel 1" state="running" /><adapter
id="MerchHier_pub_1" name="MerchHier Publisher, channel 1" state="running"
/><adapter id="UDAs_pub_1" name="UDAs Publisher, channel 1" state="running"
/><adapter id="Order_pub_1" name="Order Publisher, channel 1" state="running"
/><adapter id="Items_pub_1" name="Items Publisher, channel 1" state="running"
/><adapter id="DiffGrp_pub_1" name="DiffGrp Publisher, channel 1" state="running"
/><adapter id="Item
Loc_pub_1" name="ItemLoc Publisher, channel 1" state="running" /><adapter
id="Partner_pub_1" name="Partner Publisher, channel 1" state="running" /><adapter
id="Diffs_pub_1" name="Diffs Publisher, channel 1" state="running" /><adapter
id="WOIn_pub_1" name="WOIn Publisher, channel 1" state="running" /><adapter
id="Stores_pub_1" name="Stores Publisher, channel 1" state="running" /></type>

```

Integration Gateway Services (IGS) Testing

The RIB Integration Gateway Services (IGS) component is a set of standard Web services that provide access to the RIB infrastructure. Once the IGS is installed, it should be tested. The *Oracle Retail Integration Bus Installation Guide* describes a basic test completed through the Oracle WebLogic Administration Console.

Note: See "Integration Gateway Services Installation Tasks" in the *Oracle Retail Integration Bus Installation Guide*.

As part of this release, IGS includes a test suite that can be used by soapUI, a product of Eviware that is used to test Web services. An open source version of soapUI is available through the Eviware internet site, shown below.

Using the soapUI Tool and the IGS Test Suite

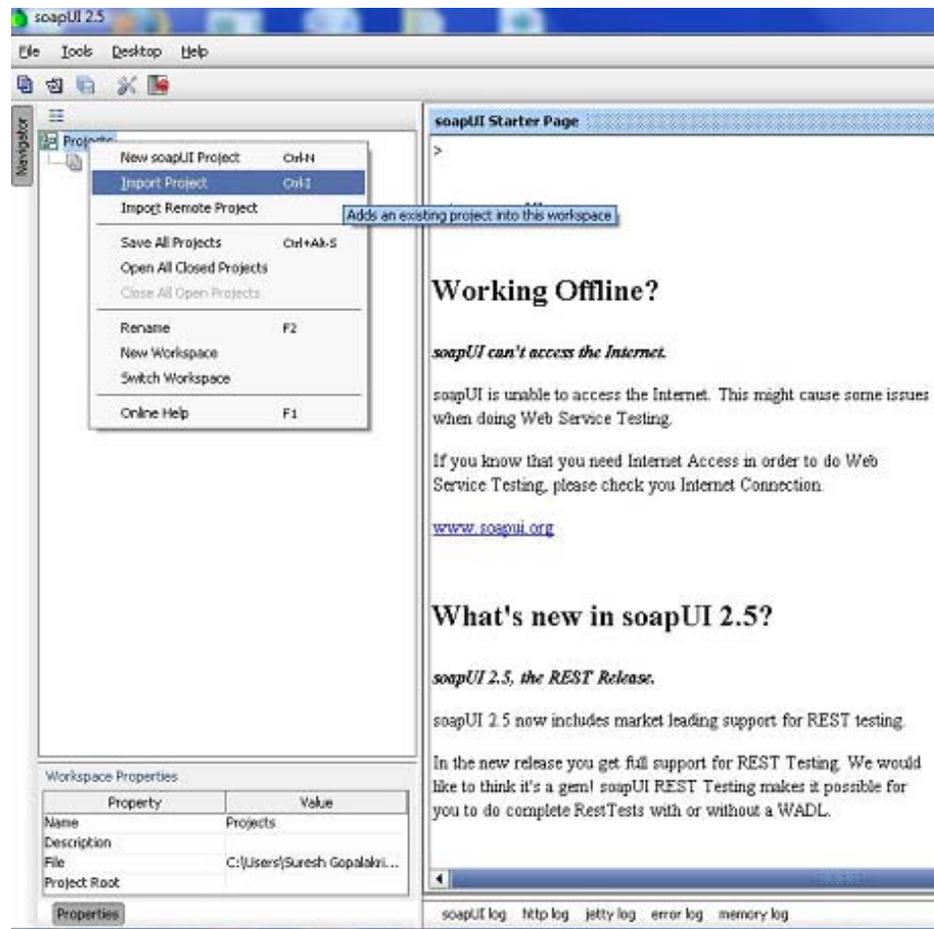
To complete IGS testing, follow these steps:

Download soapUI

1. Download the latest release of the open source version of soapUI from the Eviware internet site: <http://www.soapui.org/>
2. Using the documentation found at Eviware's product home page, follow the instructions for installing and configuring soapUI.

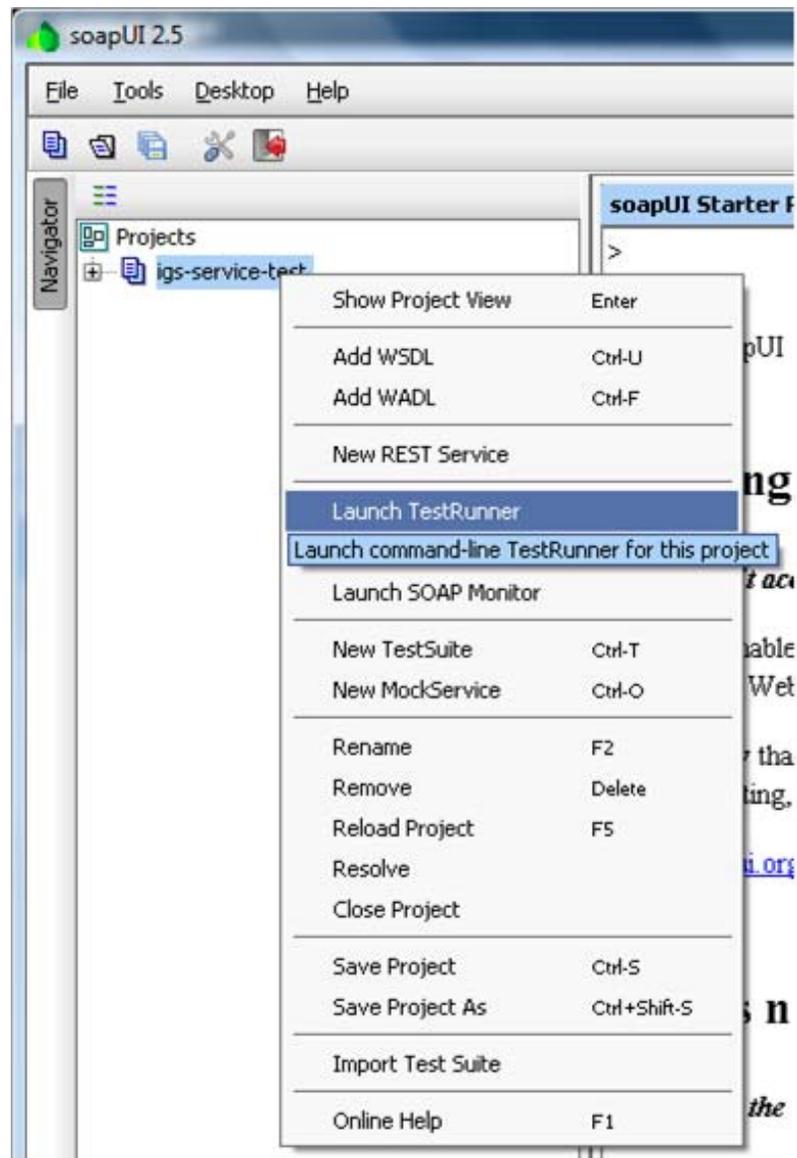
Import IGS

1. Open the soapUI tool. Right click **Projects**.
2. Click **Import Project**.
3. Select `igs-service-test-soapui-project.xml`, as shown in the following illustration.

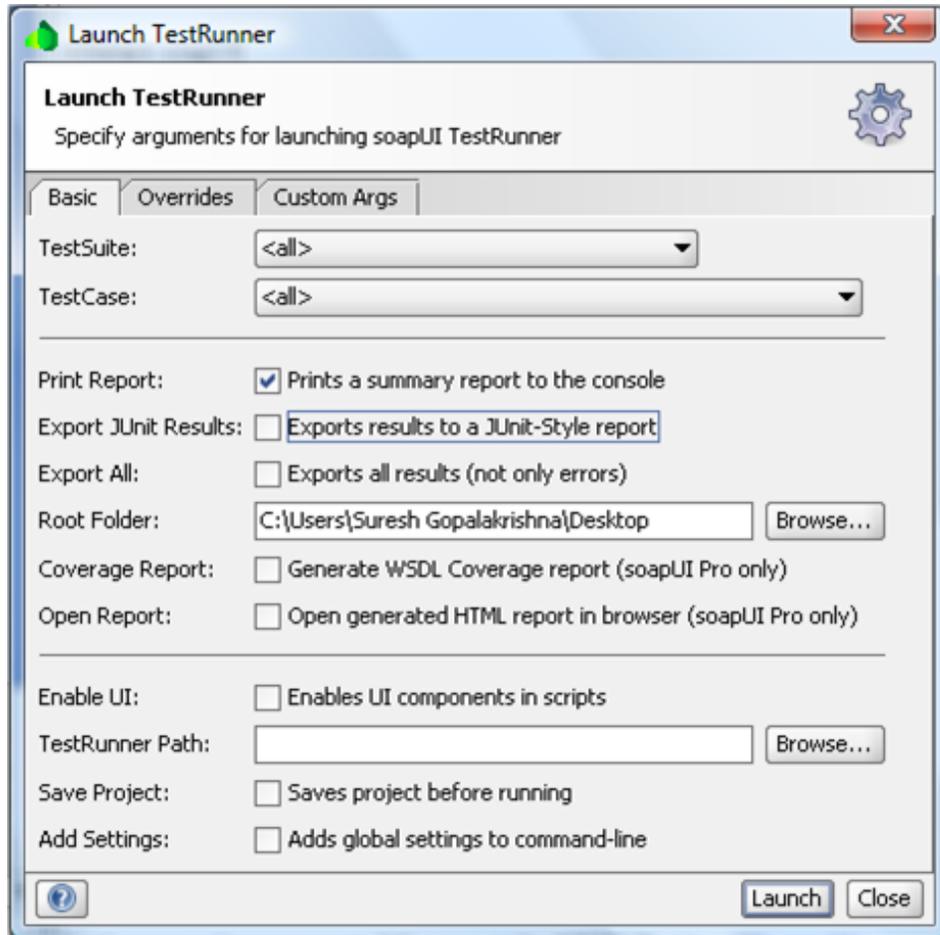


Execute IGS Test Cases

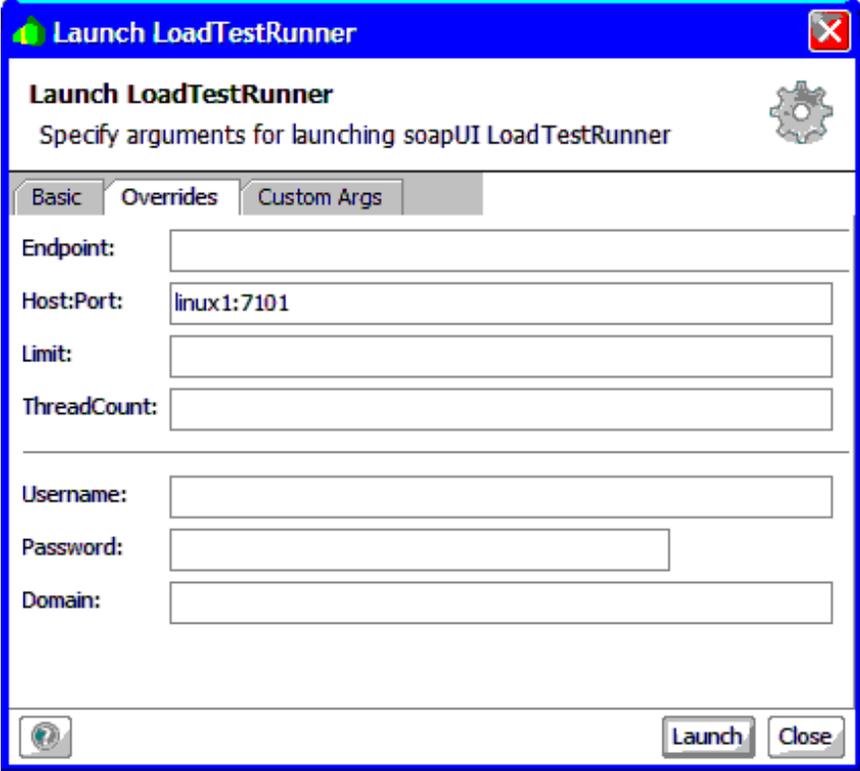
1. Right click the project to select run time, Launch TestRunner, as shown in the following illustration.



2. In the Launch TestRunner screen (below), for TestSuite, select <all> for all Web services; or select the Web service to test.
3. In the Launch TestRunner screen (below) For TestCase, select <all> for all operations; or select the operation to test.



- Click the Overrides tab to enter the environment host and port. (This is the host and port of the Oracle WebLogic server on which the igs-service.ear is deployed. For example, linux1:7101.)



Launch LoadTestRunner

Specify arguments for launching soapUI LoadTestRunner

Basic Overrides Custom Args

Endpoint:

Host:Port:

Limit:

ThreadCount:

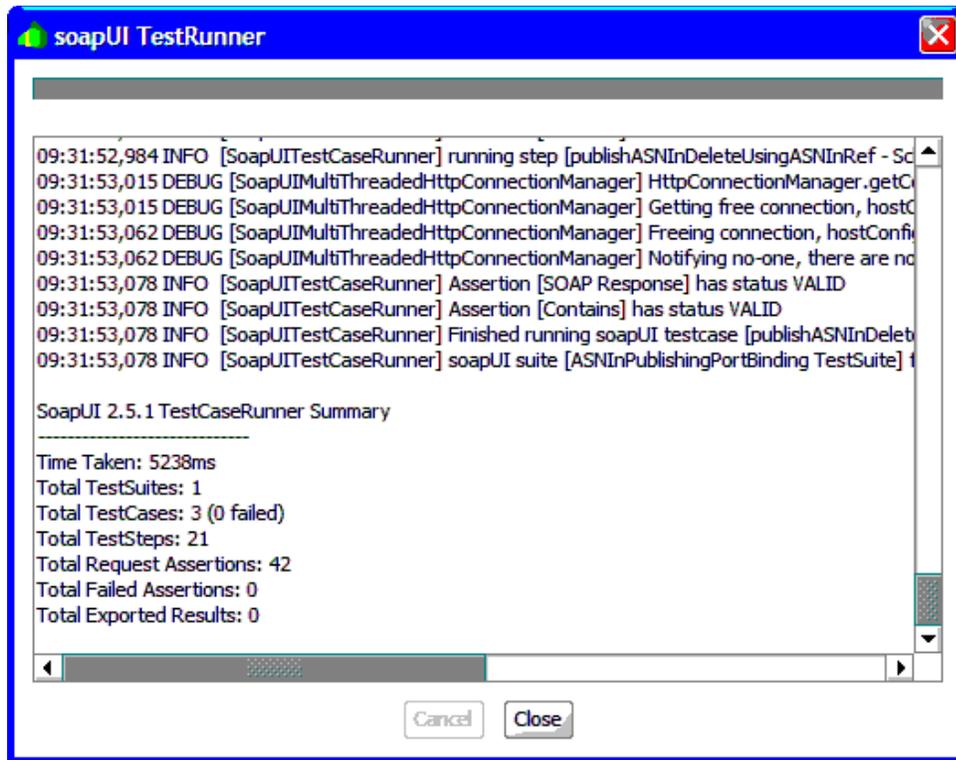
Username:

Password:

Domain:

- Click Launch.

- Results similar to those shown below are displayed on the console.



Note: For detailed instruction on test execution and to gain a better understanding of testing output, see soapUI tool documentation at the Eviware internet site: <http://www.soapui.org/>

RIB and JMX

This section describes the RIB JMX infrastructure. JMX is a specification that provides capability for runtime management of java components. Each of the RIB's software components (PublisherEjb, SubscriberEjb, TafrEjb, HospitalRetryEjb, and so on) provides its own management facility by implementing management beans.

The RIB's MBean components use uniform registration, deployment, and communication mechanisms provided by the RIB JMX infrastructure.

RIB uses log4j to log business and system events in the RIB runtime system. The definitions of the loggers are statically defined and come from a configuration file (log4j.xml). As logging is an expensive process we need to provide capability to manage log levels dynamically. The RIB Administration UI Log Manager MBean registers itself through the standard RIB JMX registration process at application startup. It provides an API to access current RIB loggers and change the log levels.

The AlertPublisherFactory is a factory that allows the user to select what alerting mechanism they want. A new JMX alerting mechanism will be added to the system. The JmxAlertPublisher class extends NotificationBroadcasterSupport and provides JMX notification capability. The JMX alerting capability is only available when running inside a container. A message type attribute will be added to the Alert class to provide the message filtering capability.

Any third party JMX console compatible with the Java EE container can be used to manage RIB components. RDMT uses the JMX command line interface provided by this design.

Note: See "Java Management Extensions (JMX)" in the *Oracle RIB Integration Bus Implementation Guide*.

RIB JMX Client

The RIB provides a command line interface to the RIB JMX system. The client is shipped as java classes in the jmx-cmd-line-ui.jar. The entry point is JmxClientMain and allows the user to either execute a single JMX command or run JMX commands in an interactive shell. There is a menu selection that invokes the interactive shell feature. See Chapter 9, "[Diagnostic and Monitoring Tools](#)," for examples of how to interface to this utility.

User Interface

```
JMXClient> help
Executing command : JmxCommand(help).
exit
```

```

Example: exit

info objectName
Example ==> OAS: info oc4j:j2eeType=JVM,name=single,J2EEServer=standalone

setattribs ObjectName attributeName1 attributeValue1 [attributeName2
attributeValue2]...
    setattribs will only work with attribute types that have
    a constructor with java.lang.String argument.
Example ==> OAS: setattribs
oc4j:j2eeType=JTAResource,name="oc4j-tm",J2EEServer=standalone transactionTimeout
31

getattribs ObjectName [attribute1] [attribute2]..
Example ==> OAS: getattribs oc4j:j2eeType=JVM,name=single,J2EEServer=standalone
javaVersion freeMemory
Example ==> WLS: getattribs rib-rms:appName=rib-rms,name=ribLogManager
CurrentLoggerNames

help [command]

invoke ObjectName methodName paramType1 paramValue1 [paramType2 paramValue2]...

Example ==> OAS: invoke
oc4j:j2eeType=JNDINamespace,name=JNDINamespace,J2EEServer=standalone
getAllBindingsAsXMLString
Example ==> OAS: invoke oc4j:j2eeType=JVM,name=single,J2EEServer=standalone
getProperty java.lang.String java.library.path
Example ==> WLS: invoke rib-rms:appName=rib-rms,level=application startAll
list objectName
Example: list *.*

close
Example: close

connect jmxServiceUrl jmxUser jmxPassword jmxConnectionProtocolProviderPackage
Example ==> OAS: connect service:jmx:rmi://localhost:23791/oc4j oc4jadmin
oc4jadmin oracle.oc4j.admin.jmx.remote
Example ==> WLS: connect
service:jmx:t3://localhost:8001/jndi/weblogic.management.mbeanservers.runtime
weblogic weblogic1 weblogic.management.remote

Previous command successful: JmxCommand(help).
JMXClient>

```

For clients like RDMT that might issue "single" JMX commands from shell scripts, the `JmxClientMain` class provides a user interface that is similar to the JMX commands above, with additional connection options for each command.

Third Party JMX Client Example

This example is for Sun JConsole tool.

See:

<http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html>

<http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.htm>

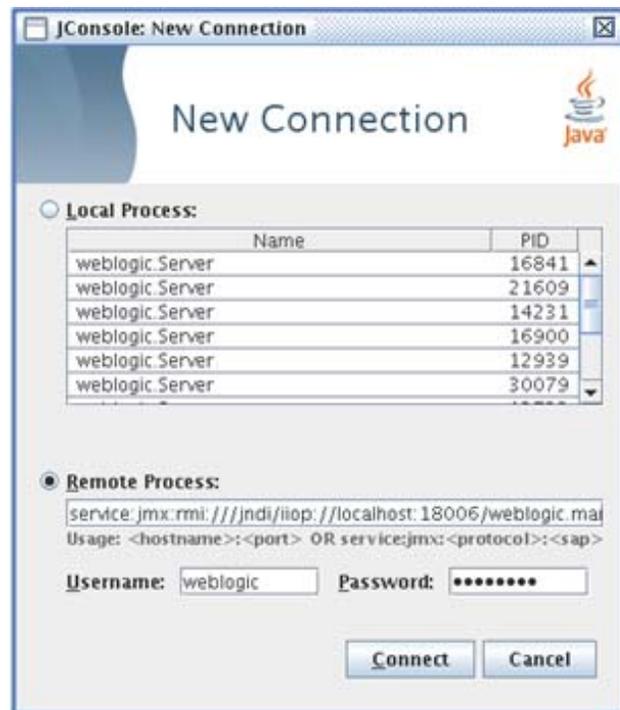
1. Copy the following files to the host where jconsole will run.

wljsxclient.jar

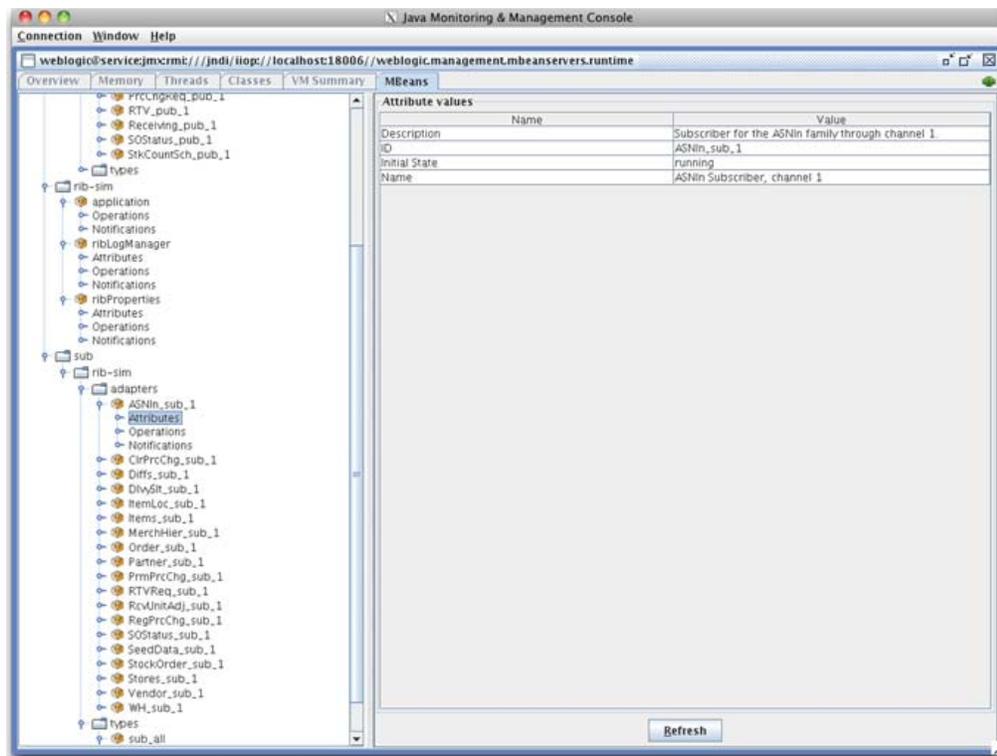
2. Create a startup file that sets the properties and classpath:

```
jconsole
-J
-Djava.class.path=$JAVA_HOME/lib/jconsole.jar:$JAVA_HOME/lib/tools.jar:$WL_
HOME/server/lib/wljsxclient.jar
-J-Djmx.remote.protocol.provider.pkgs=weblogic.management.remote -debug
```

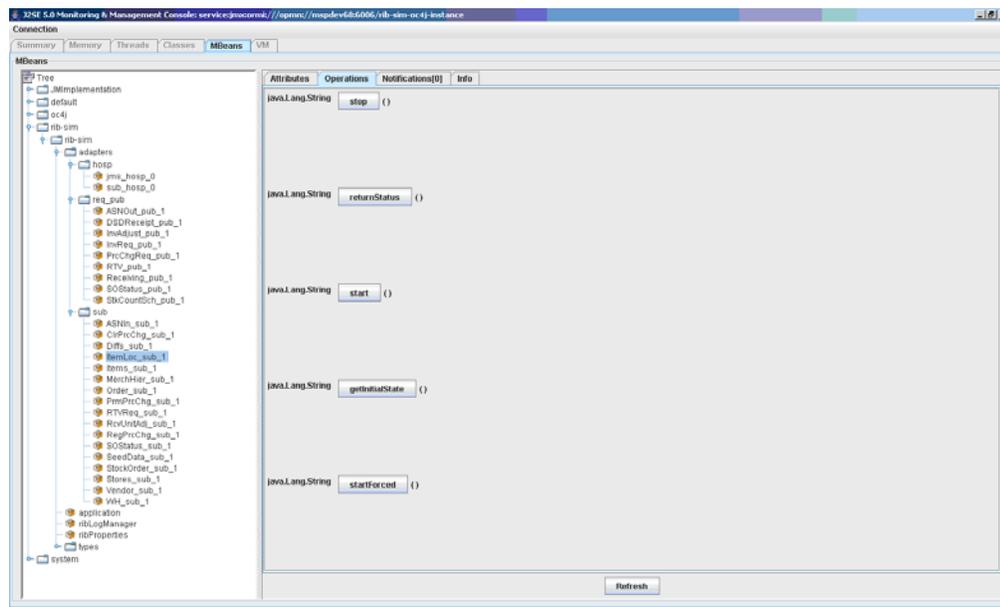
3. Start the JConsole and log in to MBean server using a connect URL (for example, service:jmx:rmi:///jndi/iiop://localhost:18006/weblogic.management.mbeanservers.runtime).



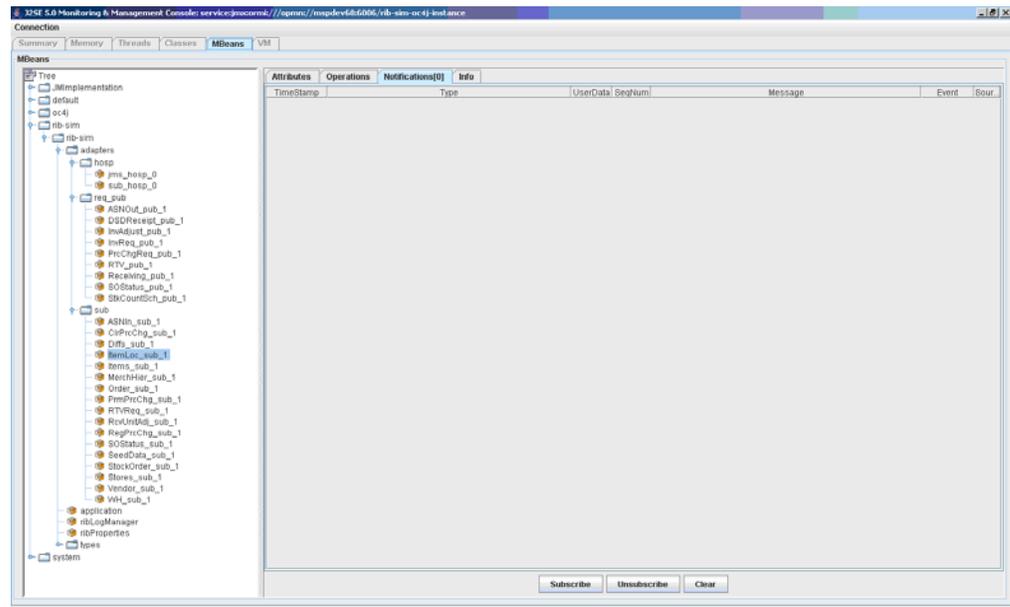
4. Select and open any one of the MBean. It opens a window with four tabs: Attributes, Operations, Notifications, and Info.
 - a. The Attributes tab provides information about the attributes of the MBean.



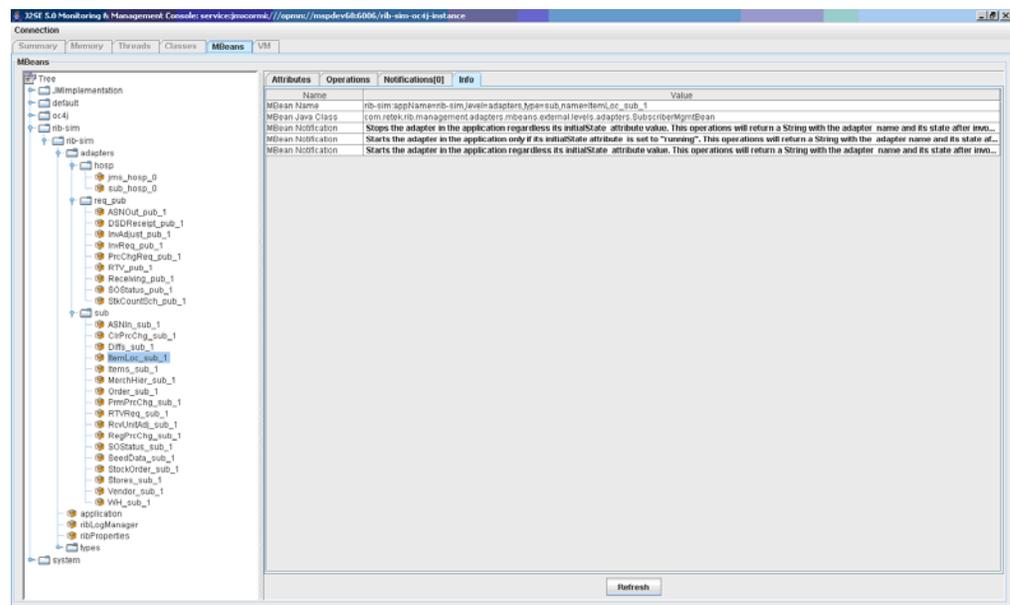
- b. The Operations tab gives the list of operations supported by that MBean.



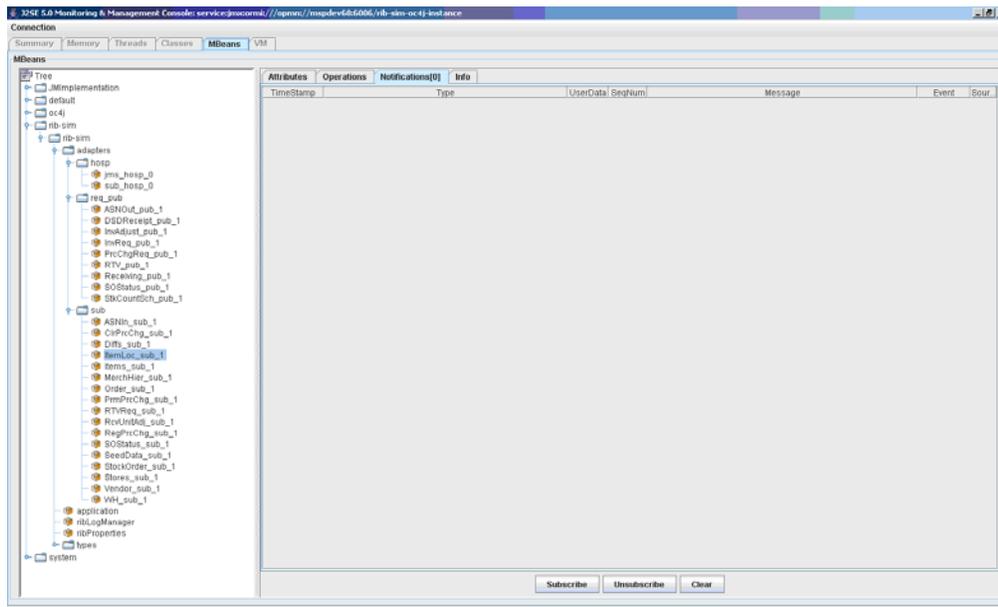
- c. The Notifications tab gives the list of notifications captured on that MBean. (You must subscribe for capturing the notifications.) Subscribing, unsubscribing, and clearing of notifications can be done from this tab.



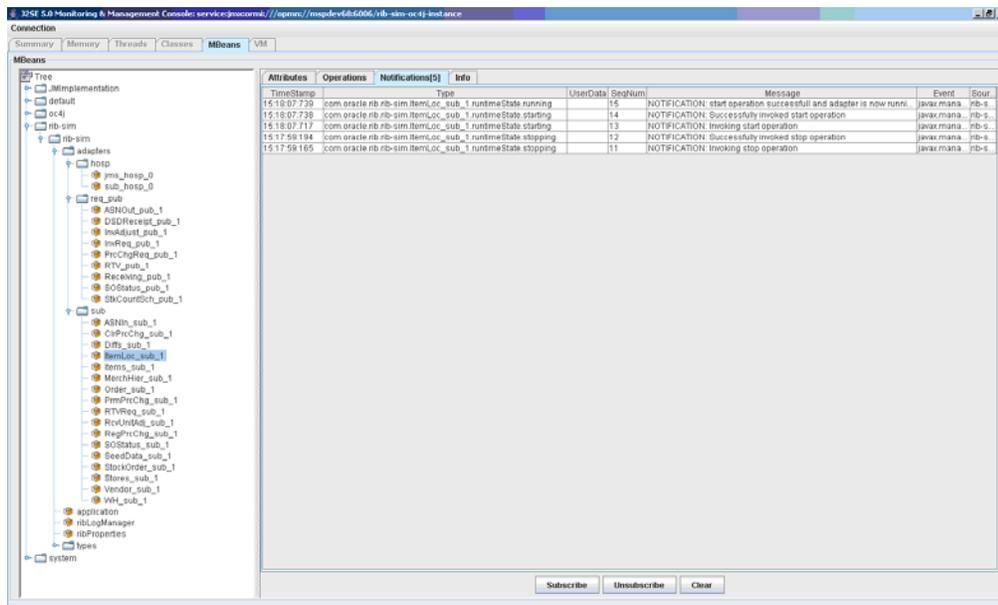
- d. The Info tab provides details about the MBean.



- e. Once an MBean is subscribed for notifications, you can see the list of notifications that occurred for that MBean. The default is zero.



- f. When some operations of the subscribed MBean are executed/invoked, notifications are captured under the Notifications tab.



The RIB provides four types of adapters that Oracle Retail applications can exploit to integrate with one another. These adapter types are publisher, subscriber, TAFR, and hospital retry adapters. They have been built using different technologies based on their particular needs.

Subscriber and TAFR adapters use Message Driven Bean (MDB) technology to register with JMS topics and receive messages for further processing.

Publisher and hospital retry adapters make use of the Java SE (Standard Edition) timer facility to schedule repetitive events. These events trigger calls to Enterprise Java Beans (EJB) to query application tables for messages to publish to the JMS server.

A fifth type of adapter exists for publishing messages in a pushing fashion, which the Retail javaEE applications, such as SIM and RPM, invoke at will for publishing messages. These are not controlled via this framework, they are always on.

Due to the variety of technologies used by the adapters, the goal of RIB Admin UI is to isolate users from these differences and provide a common management interface that can be used to control the state of the adapters and logging.

RIB Admin URLs

The RIB Admin tools are reached via URL's within each of the deployed rib-<apps>.

RIB Admin GUI

`http://<server>.us.oracle.com:<http-port>/rib-<app>-admin-gui/`

Replace <server> with the name or IP address of the server in the environment in which the rib-<app> is deployed.

Replace <http-port> with the port number that the Oracle Application Server is listening on (for example, 7777).

Replace <app> with one of the following:

- rms
- tafr
- rwms
- sim

RIB Functional Artifacts

`http://<server>.us.oracle.com:7777/rib-func-artifact/`

Replace <server> with the name or IP address of the server in the environment that has the rib-<app>'s deployed.

RIB Message Flows

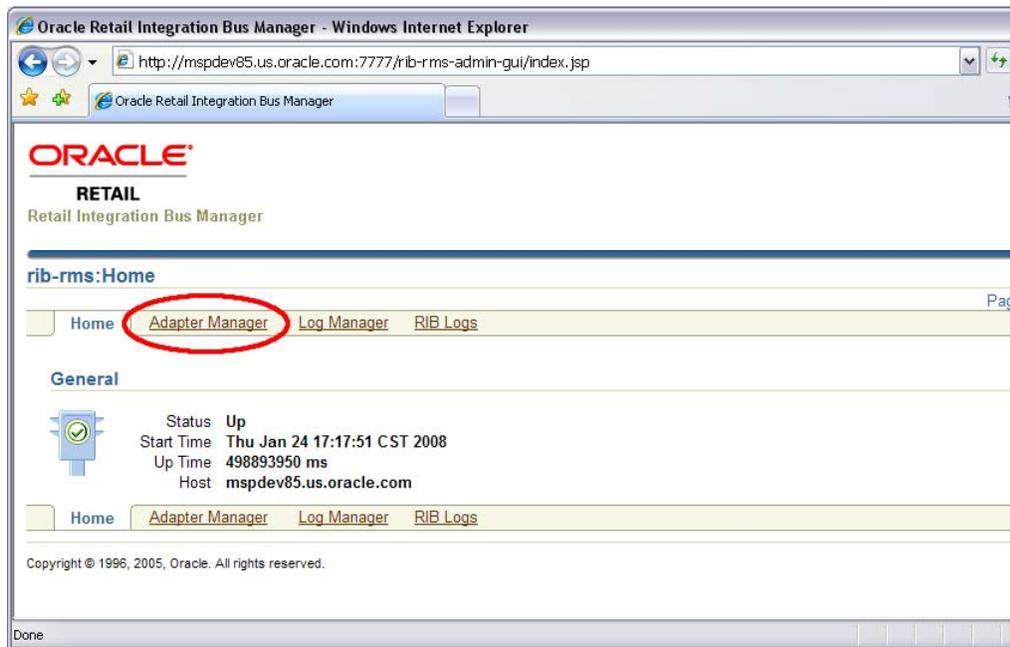
`http://<server>.us.oracle.com:7777/rib-func-artifact/rib-integration-flows.xml`

RIB Payloads (xsds)

`http://<server>.us.oracle.com:7777/rib-func-artifact/payloads/xsd`

RIB Admin GUI Home

On the Home screen, click **Adapter Manager** to view all adapters for the given application.



Adapter Manager

All message functions in the RIB are performed by adapters. There are four categories of adapters: publishers, subscribers, TAFRs (transform, address, filtering and routing), and RIB hospital retry. The adapter manager console is used to start and stop adapters, configure settings, and view adapter log files.

Adapter Manager Screen

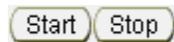
This screen shows the current status of all adapters for the specified application. The following signifies an adapter is up and running:



The following signifies that the adapter is offline or has shut itself down:

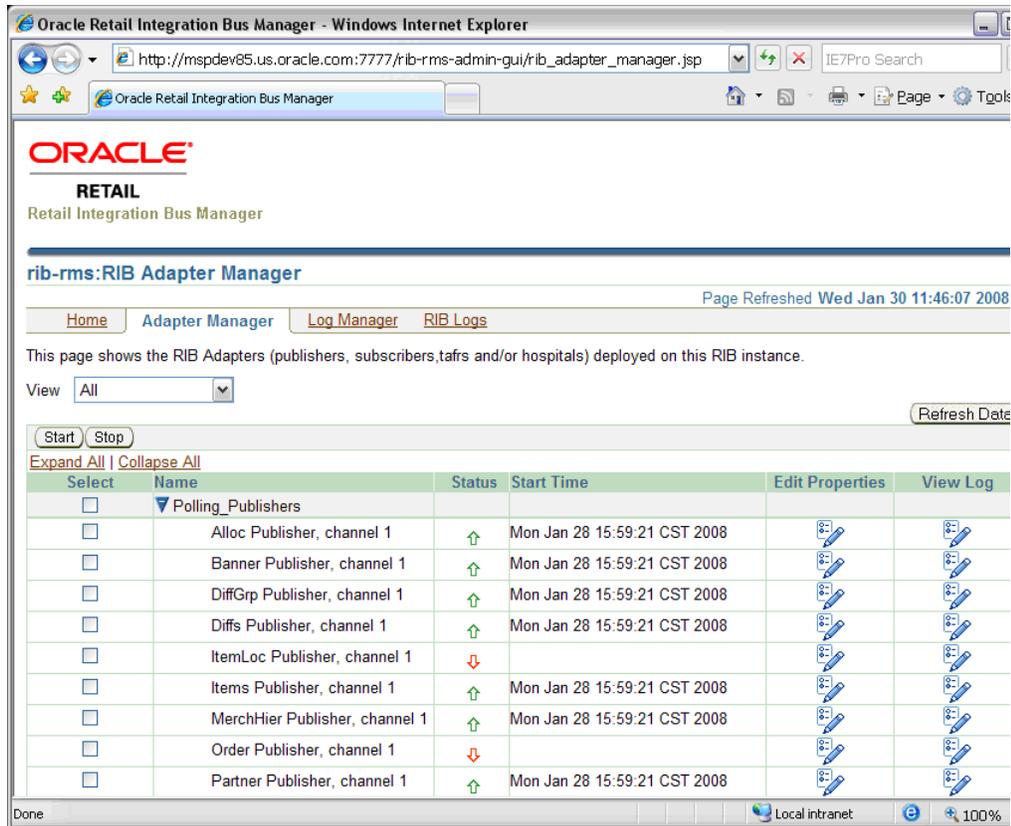


From this screen any listed adapters can be started and stopped by selecting the check box related to the adapter and then using the following buttons:



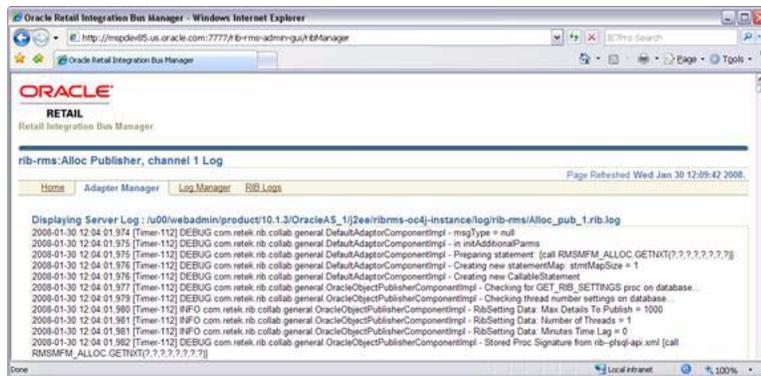
Click the following symbol in the "View Log" column to return to the log file viewer for the specified adapter.





Log Viewer

Depending on what level the logging is set to, the log for the adapter can contain very little to extreme amounts of data, errors, and message failures.



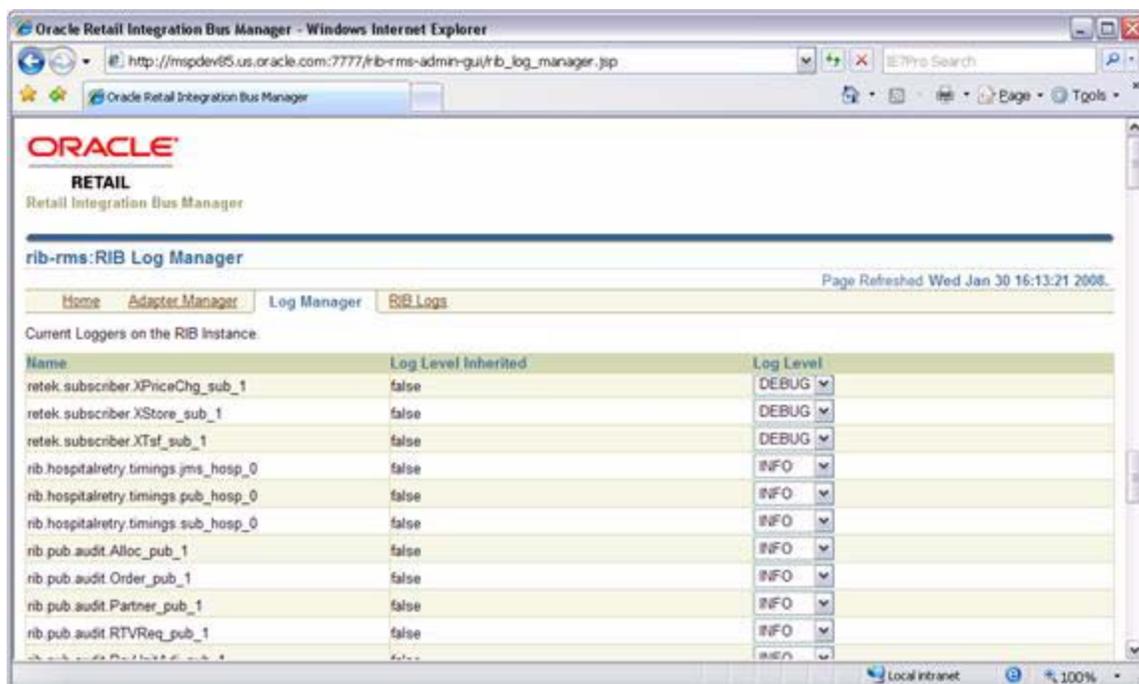
Log Manager

The Log Manager screen allows the user to change the logging level of the adapters. It also allows the user to enable audit and timings logging.

The UI displays each logger and the current log level. If the log level is inherited, it displays a * along with the log level. The user is able to select a logger and modify the log level by clicking "Set Option". The changed settings are an in memory representation of the logger. If the user wants to persist the log level between an application server bounce, they have the option to click the "Save" button. The save operation updates the log4j.xml file in the file system in the location where it was loaded from.

When Audit logging is turned on, each message that is processed by the adapter, the XML payload is persisted to an audit log. Audit logging only works when the audit log level is set to DEBUG for the specified adapter.

The Timings logging captures adapter processing performance data to another separate log. As with the audit log, this only works with the logging level set to DEBUG. The RDMT command line tool can be used to process and view the results of the timings logging output.



RIB Logs

The RIB Logs screen can be used to view the regular adapter log file as well as the Timings and Audit logs for each adapter, if they have been activated. (See instructions in the "Log Manager Screen.")

The screen also is accessed by clicking the following symbol in the "View Log" column on the Adapter Manager screen:



Oracle Retail Integration Bus Manager - Windows Internet Explorer

http://mepdev05.us.oracle.com:7777/rib-rms-admin-gui/logs.jsp

Oracle Retail Integration Bus Manager

ORACLE
RETAIL
Retail Integration Bus Manager

rib-rms:RIB Adapter Manager

Page Refreshed Wed Jan 30 16:11:18 2008.

Home Adapter Manager Log Manager **RIB Logs**

Files from /u00/webadmin/product/10.1.3/OracleAS_11j2ee/ribrms-oc4j-instance/log/rib-rms

File Name	Size	Last Modified
deploy.rib.log	189	Mon Jan 28 16:00:38 CST 2008
ASIn_sub_1.timings.log	0	Wed Jan 23 15:34:11 CST 2008
ASIn_sub_1.audit.log	0	Wed Jan 23 15:34:11 CST 2008
ASIn_sub_1.rib.log	0	Wed Jan 23 15:34:12 CST 2008
ASNOut_sub_1.timings.log	0	Wed Jan 23 15:34:12 CST 2008
ASNOut_sub_1.audit.log	0	Wed Jan 23 15:34:12 CST 2008
ASNOut_sub_1.rib.log	0	Wed Jan 23 15:34:12 CST 2008
COCogs_sub_1.timings.log	0	Wed Jan 23 15:34:12 CST 2008
COCogs_sub_1.audit.log	0	Wed Jan 23 15:34:12 CST 2008
COCogs_sub_1.rib.log	0	Wed Jan 23 15:34:12 CST 2008

Done Local intranet 100%

JMS Provider Management

The Oracle Enterprise Messaging Service (OEMS) provides a robust architecture for integrating business-critical applications. Built on Java 2 Enterprise Edition (J2EE) standards such as the Java Message Service (JMS) and the J2EE ConnectorArchitecture (JCA), OEMS reduces the time, cost, and effort required to build integrated and distributed applications. Through a common interface, JMS, OEMS offers developers a quality of service (QoS) choice for persisting messages.

The RIB is designed to be JMS provider agnostic and will be certified with several, starting with the OEMS JMS Database persistence option; which is the JMS interface to the Oracle Database Streams Advanced Queuing (AQ) feature. Subsequent releases will add certification of the WLS JMS (for the file and memory-persistence version) that is bundled with the WebLogic Application Server, as well as other JMS standard providers.

For more details on OEMS, see the Oracle® Containers for J2EE Services Guide 11g - Using Oracle Enterprise Messaging Service.

The RIB on AQ JMS

The AQ JMS is a database and needs to be installed, configured, and tuned to support the anticipated transaction loads for a retailer's production message volumes.

There are some areas to be considered by the RIB team and the Database Administrators.

- It is strongly recommended that the Oracle Database Instance that is configured to be the AQ JMS provider is not shared with any other applications and is not on the same host (physical or logical) with any other applications.
- AQ, on the server side, is I/O intensive. Pay close attention to the disk layout.
- AQ JMS as used by the RIB will have high transaction rates. Consider this in configuration of the redo logs.

AQ JMS should be run in archive log mode. If the database crashes, it must be recoverable to a point-in-time or messages (business events) will be lost.

- The RIB is a client of the AQ database and uses JDBC connections via the aqapi client. The average message size for a given interface affects the network and overall performance behavior.
- AQ JMS sizing to avoid out-of-space situations is critical.

Queue Monitor Process Setup

The QMON processes are optional background processes for Oracle Streams Advanced Queuing (AQ) which monitor and maintain all the system and user owned AQ objects. They provide the mechanism for message expiration, retry, and delay, maintain queue statistics, remove processed messages from the queue table and maintain the dequeue IOT.

The number of queue monitor processes is controlled by the dynamic initialization parameter AQ_TM_PROCESSES. There can be a maximum of 10 QMON processes. The parameter AQ_TM_PROCESSES can be set in the PFILE or SPFILE:

- aq_tm_processes=4
- alter system set aq_tm_processes=4

Starting with Oracle RDBMS release 10.1, Oracle automatically manages the QMON monitor processes depending on the system load. It is no longer required to explicitly set AQ_TM_PROCESSES. However, it is recommended to monitor the workload and make any adjustments. If the QMON processes lag behind, there is a chance of expired messages remaining in the queue and the tablespace eventually running out of space.

If explicitly setting AQ_TM_PROCESSES, our recommended value is between 2 to 8. Do not set the value to the maximum allowed value of 10 in Oracle 11g. This is due to the fact that all explicitly started QMON processes work only with persistent messages. Oracle can automatically start processes to maintain buffered messages. Setting AQ_TM_PROCESSES to a maximum value of 8 still leaves 2 processes for Oracle that can be started to maintain buffered messages.

Optimizing Enqueue/Dequeue Performance

The AQ database performance needs to be tuned as per Oracle database tuning practices.

Tuning the SGA. Use tools such as Statspack, Oracle Enterprise Manager and SQL trace to find out bottlenecks. An inefficiently configured SGA will slow-down enqueue and dequeue transactions.

Tune the Server Resources: Check server CPU, memory, I/O and network utilization. Tools such as nmon, sar, iostat, vmstat, glance can be used to collect system statistics. Use shared memory and semaphore parameters that are recommended for the Oracle database on that type of server.

Tune Physical Schema setup: This will include creating right tablespaces, placements of datafiles, tables, and indexes.

Note: See also Oracle® Database Administrator's Guide 11g Release 2 (11.2), Oracle® Streams Advance Queuing User's Guide, and Reference 11g Release 2 (11.2)

Sizing Considerations

There are some important considerations required by the RIB team and Database Administrators for sizing for the deployment of the RIB on AQ JMS.

- The enqueueing/dequeueing rate for the messages per message family affects the requirement for the number of available database segments.

By default, all RIB topics are created in a single tablespace. AQ creates multiple tables for each topic within that tablespace. A topic (message family) that has a high transaction rate can quickly consume available segments. If the tablespace is not sized appropriately, then a single interface can negatively impact all interfaces.

The QMON background process that is responsible for space management will not keep up the transaction rates of some RIB interfaces. In this case, the transaction rate is defined as the rate of enqueueing vs. dequeueing. Messages that are subscribed (consumed) are not removed from the AQ tables immediately. It is the normal case that the enqueue rate will be faster than the dequeue rate. This time lag should be a sizing consideration.

- The total tablespace sizing must be calculated based on the business requirement for the number of messages that have to be retained per message family if a subscribing application is off-line.

It is very common for a subscribing application to go off-line. This means that messages have to be retained (persisted) on the JMS until the subscriber comes back on-line. The general sizing guideline for any RIB JMS sub-system is for the disk (mount points or database) to be able to handle 24 hours of maximum messages per topic as defined by the site's projected volume requirements. For example, OrdersFromRMS may be specified to retain 355,000 details (such as 1000 1M messages = 1GB). This calculation must be performed for each of the 90+ topics in the GA RIB system and based on the customer's estimated volume per interface.

Note: See "[How to Calculate Average Message Size.](#)"

The RIB on AQ JMS - Server Side Processes

A process is a "thread of control," or a mechanism in an operating system, that can run a series of steps. (Some operating systems use the terms "job" or "task.") A process normally has its own private memory area in which it runs.

When the RIB is configured to use the Oracle AQ JMS, there are considerations that affect RDBMS tuning and the configuration of database processes. This section is intended to outline these considerations.

Types of Oracle Database Side Processes

The processes in an Oracle database system are categorized into two major groups:

- User processes run the application or Oracle tool code.
- Oracle database processes run the Oracle database server code. They include server processes and background processes.

The RIB and Application Server and JDBC Connections

The number of RIB related server side processes can grow based on activity. It is related to the way the application server container manages jdbc connections. The following rules apply:

- Each subscriber uses one JDBC connection to AQ JMS.
- Each Publisher or Hospital Retry may use one or more connections, depending on volume and activity.
- When a RIB adapter (java code) asks for "a" connection, the application server may decide to get more than one connection and add it to its pool.

RIB 13.2 Connections - Summary

RIB Adapter Type	Total Adapters in RIB 13.2
rib-app Subscriber	69
TAFR Subscriber	22
rib-app Polling Publisher	35
rib-app Request-driven Publisher	14
TAFT Publishers	22
Hospital Retry - Polling Publisher	11
Total	173

At any time, depending on deployment options in a non-multiple channel deployment, the RIB can have at least 173 AQ connections. The application server may ask for more than 173 from the database.

These numbers will increase if there are multiple retry adapters configured and if message flows are configured for multiple channels. So the calculation includes the base numbers plus one for each additional retry--and one for each multiple channel publisher or subscriber. Always assume that the result is the lowest number of connections, because the container can ask for more.

rib-rms Connections

RIB Adapter Type	Total Adapters in RIB 13.2
Subscriber	35
Polling Publisher	22
Hospital Retry - Polling Publisher	3
Total	60

At any time, depending on deployment option, the rib-rms app can have at least 60 AQ connections. The application server may ask for more than 60 from the database.

rib-rwms Connections

RIB Adapter Type	Total Adapters in RIB 13.2
Subscriber	15
Polling Publisher	11
Hospital Retry - Polling Publisher	2
Total	28

At any time, depending on deployment option, the rib-rwms app can have at least 28 AQ connections. The application server may ask for more than 28 from the database.

rib-sim Connections

RIB Adapter Type	Total Adapters in RIB 13.2
Subscriber	19
Request Driven Publishers	9
Hospital Retry - Polling Publisher	2
Total	30

At any time, depending on deployment option, the rib-sim app can have at least 30 AQ connections. The application server may ask for more than 30 from the database.

rib-tafr Connections

RIB Adapter Type	Total Adapters in RIB 13.2
Subscriber	22
Publishers	22
Hospital Retry - Polling Publisher	2
Total	47

At any time, depending on deployment option, the rib-tafr app can have at least 47 AQ connections. The application server may ask for more than 47 from the database.

rib-rpm Connections

RIB Adapter Type	Total Adapters in RIB 13.2
Subscriber	0
Request Driven Publisher	3

RIB Adapter Type	Total Adapters in RIB 13.2
Hospital Retry - Polling Publisher	1
Total	4

At any time, depending on deployment option, the rib-rpm app can have at least four AQ connections. The application server may ask for more than four from the database.

Configuration Recommendations

It is strongly recommended that, for the production RIB deployment, the Oracle database instance configured as the AQ JMS be separate from all other uses. There are performance considerations as well as architectural reasons for maintaining this separation.

Note: See the *Oracle RIB Integration Bus Implementation Guide*.

For the testing and QA phases of the deployment life cycle, co-location is not recommended. Regardless of the life cycle phase, the AQ JMS should not be configured with any other applications, including the rib-app, Error Hospital.

If the option to co-locate is chosen, work with the database administrators to determine and set the appropriate maximum database sessions and processes, depending on the RIB environment setup (single channel or multiple channel, for example.) Note that the result may be more than 500 processes. The issues that may arise from having this many processes can be obscure, and it is difficult to isolate their root cause.

Support for Multiple JMS Servers Within a Single Deployment

Employing multiple JMS servers allows for the isolation of flows (for example, high volume versus low, custom versus base, and message families) for performance and operational QoS.

Design

To meet the JMS agnostic requirement for the RIB, a unique JMS server ID (jms-server-id) is assigned to each RIB adapter. Accordingly, each RIB adapter can identify the JMS server to which it is associated. As the default, "out-of-the-box" adapters are configured to be on jms-server, "jms1."

For each new jms-server-ID, a new resource adapter must be configured to point the application server to the JMS provider's resource. The adapter communicates with the JMS server and is deployed as part of the application. Where customization is required, the adapter can be configured to point to a different JMS server.

rib-app-builder Validation Checks

The rib-app-builder performs several validation checks, as listed below. To prevent the rib-app-builder compilation process from failing, the following criteria must be met:

- Each jms-server-id is unique where more than one JMS server is configured.
- Within a message flow, the jms-server-id is the same for all applications.

- A `jms-server-id` is present in the `rib-deployment-env-info.xml` and present in at least one of the `rib-<app>-adapters.xml` files.
- A `jms-server-id` is present in `rib-<app>-adapters.xml` and present in the `rib-deployment-env-info.xml` file.
- Multiple channels are configured for a give family are on the same JMS server.
- Proper hospitals are configured for all JMS servers. (Where additional JMS servers are configured, the `rib-app-builder` checks to see if hospital adapters are configured for all JMS servers.)

How to Set Up Multiple JMS Servers

Process Overview

1. Determine the family to be configured.
2. Examine the `rib-integration-flows.xml` to identify all the RIB applications in the full integration flow.
3. Add a new JMS server by updating `rib-deployment-env-info.xml`.
4. In the `rib-home`, modify the appropriate files for each of the `rib-<apps>` participating in the integration flow. Point the adapters to the correct JMS server:
 - a. `rib-<app>-adapters.xml`
 - b. `rib-<app>-adapter-resources.properties`
5. Compile all applicable `rib-<apps>`.
6. Run `prepare-jms` for the newly created JMS server.
7. Deploy.

General Recommendations

- The default ID for out-of-the-box JMS servers is `jms1`. It is recommended that the same naming convention is followed when additional JMS servers are configured (for example, `jms2`).
- If multiple JMS servers require configuration, it is recommended that the application (for example, `rib-rms`) be completely removed (or undeployed) before the new deployment begins.

AQ Recommendation

If multiple AQ JMS servers are configured, each must be on a different database server instance.

Sample Configuration

Following are portions of the Items message flow from `rib-integration-flows.xml`. The message originates from RMS flows through a a TAFR. The TAFR sends the message to two topics, and the message is subscribed by RWMS and SIM. The samples below assume that a new `jms-server-id` (`jms2`) is required for the message flow.

`rib-integration-flows.xml`

```
<message-flow id="6">
  <node id="rib-rms.Items_pub" app-name="rib-rms"
    adapter-class-def="Items_pub" type="DbToJms">
```

```

        <in-db>default</in-db>
        <out-topic>etItemsFromRMS</out-topic>
    </node>
    <node id="rib-tafr.ItemsToItemsTL_tufr" app-name="rib-tafr"
        adapter-class-def="ItemsToItemsTL_tufr" type="JmsToJms">
        <in-topic>etItemsFromRMS</in-topic>
        <out-topic>etItemsTLFromRIB</out-topic>
    </node>
    <node id="rib-tafr.ItemsToItemsISO_tufr" app-name="rib-tafr"
        adapter-class-def="ItemsToItemsISO_tufr" type="JmsToJms">
        <in-topic>etItemsFromRMS</in-topic>
        <out-topic>etItemsISO</out-topic>
    </node>
    <node id="rib-rwms.Items_sub" app-name="rib-rwms"
        adapter-class-def="Items_sub" type="JmsToDb">
        <in-topic>etItemsTLFromRIB</in-topic>
        <out-db>default</out-db>
    </node>
    <node id="rib-sim.Items_sub" app-name="rib-sim"
        adapter-class-def="Items_sub" type="JmsToDb">
        <in-topic>etItemsISO</in-topic>
        <out-db>default</out-db>
    </node>
</message-flow>

```

Note: The following are the configuration changes required for the message flow. The example assumes that all applications apply (RMS, TAFR, SIM, and RWMS).

rib-deployment-env-info.xml

A new JMS server with `jms-server-id="jms2"` is added in `rib-deployment-env-info.xml` file as follows:

```

<aq-jms-servers>
    <aq-jms-server jms-server-id="jms1">
        <jms-server-home>user@host:/u00/db</jms-server-home>
        <jms-url>jdbc:oracle:thin:@host:port:SID</jms-url>
        <jms-port>1521</jms-port>
        <jms-user>ribaq</jms-user>
        <jms-password>ribaq</jms-password>
    </aq-jms-server>
    <aq-jms-server jms-server-id="jms2">
        <jms-server-home>user@host:/u00/db</jms-server-home>
        <jms-url>jdbc:oracle:thin:@host:port:SID</jms-url>
        <jms-port>1521</jms-port>
        <jms-user>aq2</jms-user>
        <jms-password>aq2</jms-password>
    </aq-jms-server>
</aq-jms-servers>

```

RIB-RMS Application Configuration

To configure the RIB-RMS application, complete the following steps:

rib-rms-adapters.xml changes

1. Edit \$RIB_HOME/application-assembly-home/rib-rms/rib-rms-adapters.xml, where \$RIB_HOME is the rib-home directory.
2. Point the Items_pub_1 adapter to jms-server-id "jms2" as follows:

```
<timer-driven id="Items_pub_1" initialState="stopped" timeDelay="10"
jms-server-id="jms2">
    <timer-task>
        <class
name="com.retek.rib.app.getnext.impl.GetNextTimerTaskImpl"/>
        <property name="maxChannelNumber" value="1" />
    </timer-task>
</timer-driven>
```

3. Add hospital adapters for jms-server-id "jms2", as follows:

```
<!--Hospital adapter configuration starts here -->
<timer-driven id="sub_hosp_2" initialState="stopped" timeDelay="10"
jms-server-id="jms2">
    <timer-task>
        <class name="com.retek.rib.j2ee.ErrorHospitalRetryTimerTask"/>
        <property name="reasonCode" value="SUB"/>
    </timer-task>
</timer-driven>
<timer-driven id="pub_hosp_2" initialState="stopped" timeDelay="10"
jms-server-id="jms2">
    <timer-task>
        <class name="com.retek.rib.j2ee.ErrorHospitalRetryTimerTask"/>
        <property name="reasonCode" value="PUB"/>
    </timer-task>
</timer-driven>
<timer-driven id="jms_hosp_2" initialState="stopped" timeDelay="10"
jms-server-id="jms2">
    <timer-task>
        <class name="com.retek.rib.j2ee.ErrorHospitalRetryTimerTask"/>
        <property name="reasonCode" value="JMS"/>
    </timer-task>
</timer-driven>
```

rib-rms-adapters-resources.properties

Add the following properties to the resource file:

```
sub_hosp-2.name=SUB Hospital Retry jms2
```

```
sub_hosp-2.desc=Inject messages into from the Error Hospital.
```

```
pub_hosp-2.name=PUB Hospital Retry jms2
```

```
pub_hosp-2.desc=Re-publish messages from to JMS.
```

```
jms_hosp-2.name=JMS Hospital Retry jms2
```

```
jms_hosp-2.desc=Re-publish messages from to JMS after JMS is brought up again.
```

RIB-TAFR Application Configuration

To configure the RIB-TAFR application, complete the following steps:

rib-tafr-adapters.xml

1. Edit \$RIB_HOME/application-assembly-home/rib-rms/rib-rms-adapters.xml, where \$RIB_HOME is the rib-home directory.
2. Point the ItemsToItemsTL_tufr_1 adapter to jms-server-id "jms2", as shown below.
3. Point the ItemsToItemsISO_tufr_1 adapter to jms-server-id "jms2", as shown below:

```
<tafrs>
  <message-driven id="ItemsToItemsTL_tufr_1" initialState="stopped"
tafr-business-impl="com.retek.rib.domain.tafr.bo.impl.ItemsToItemsTLFromRibBOIm
pl" jms-server-id="jms2" />

  <message-driven id="ItemsToItemsISO_tufr_1" initialState="stopped"
tafr-business-impl="com.retek.rib.domain.tafr.bo.impl.ItemsToItemsISOFromRibBOI
mpl" jms-server-id="jms2" />
</tafrs>
```

4. Add hospital adapters for jms-server-id "jms2".

```
<!--Hospital adapter configuration starts here -->
<timer-driven id="sub_hosp_0" initialState="stopped" timeDelay="20"
jms-server-id="jms2">
  <timer-task>
    <class name="com.retek.rib.j2ee.ErrorHospitalRetryTimerTask"/>
    <property name="reasonCode" value="SUB"/>
  </timer-task>
</timer-driven>

<timer-driven id="jms_hosp_0" initialState="stopped" timeDelay="30"
jms-server-id="jms2">
  <timer-task>
    <class name="com.retek.rib.j2ee.ErrorHospitalRetryTimerTask"/>
    <property name="reasonCode" value="JMS"/>
  </timer-task>
</timer-driven>
```

rib-tafr-adapters-resources.properties

Add the following properties to the resource file:

sub_hosp-2.name=SUB Hospital Retry jms2

sub_hosp-2.desc=Inject messages into from the Error Hospital.

jms_hosp-2.name=JMS Hospital Retry jms2

jms_hosp-2.desc=Re-publish messages from to JMS after JMS is brought up again.

RIB-SIM Application Configuration

To configure the RIB-SIM application, complete the following steps:

rib-sim-adapters.xml

1. Edit \$RIB_HOME/application-assembly-home/rib-rms/rib-rms-adapters.xml, where \$RIB_HOME is the rib-home directory.

```
<subscribers>
  <message-driven id="Items_sub_1" initialState="running"
jms-server-id="jms2"/>
</subscribers>
```

2. Add hospital adapters for jms-server-id "jms2".

```
<!--Hospital adapter configuration starts here -->

<timer-driven id="sub_hosp_0" initialState="stopped" timeDelay="20"
jms-server-id="jms2">
  <timer-task>
    <class name="com.retek.rib.j2ee.ErrorHospitalRetryTimerTask"/>
    <property name="reasonCode" value="SUB"/>
  </timer-task>
</timer-driven>

<timer-driven id="jms_hosp_0" initialState="stopped" timeDelay="30"
jms-server-id="jms2">
  <timer-task>
    <class
name="com.retek.rib.j2ee.ErrorHospitalRetryTimerTask"/>
    <property name="reasonCode" value="JMS"/>
  </timer-task>
</timer-driven>
```

rib-sim-adapters-resources.properties

Add the following properties to the resources file:

```
sub_hosp-2.name=SUB Hospital Retry jms2
```

```
sub_hosp-2.desc=Inject messages into from the Error Hospital.
```

```
jms_hosp-2.name=JMS Hospital Retry jms2
```

```
jms_hosp-2.desc=Re-publish messages from to JMS after JMS is brought up again.
```

RIB-RWMS Application Configuration

To configure the RIB-SIM application, complete the following steps.

rib-rwms-adapters.xml

1. Edit \$RIB_HOME/application-assembly-home/rib-tafr/rib-tafr-adapters.xml
2. Point the Items_sub_1 adapter to jms-server-id "jms2."

```
<subscribers>
  <message-driven id="Items_sub_1" initialState="running"
jms-server-id="jms2"/>
</subscribers>
```

3. Add hospital adapters for jms-server-id "jms2".

```
<!--Hospital adapter configuration starts here -->

<timer-driven id="sub_hosp_0" initialState="stopped" timeDelay="20"
jms-server-id="jms2">
  <timer-task>
    <class name="com.retek.rib.j2ee.ErrorHospitalRetryTimerTask"/>
    <property name="reasonCode" value="SUB"/>
  </timer-task>
</timer-driven>
<timer-driven id="jms_hosp_0" initialState="stopped" timeDelay="30"
jms-server-id="jms2">
  <timer-task>
    <class name="com.retek.rib.j2ee.ErrorHospitalRetryTimerTask"/>
    <property name="reasonCode" value="JMS"/>
  </timer-task>
</timer-driven>
```

rib-rwms-adapters-resources.properties

sub_hosp-2.name=SUB Hospital Retry jms2

sub_hosp-2.desc=Inject messages into from the Error Hospital.

jms_hosp-2.name=JMS Hospital Retry jms2

jms_hosp-2.desc=Re-publish messages from to JMS after JMS is brought up again.

Compile and Deploy

Using the RIB Installer or the RIB App Builder command line tools compile, and deploy the new rib-<app>.ears.

RIB-ADMIN-GUI

After deployment check if the adapters configured point to the correct JMS server.

The screenshot shows the Oracle Retail Integration Bus Manager GUI. The page title is "rib-rms:RIB Adapter Manager". Below the title, there are navigation tabs: "Home", "Adapter Manager", "Log Manager", and "RIB Logs". The "Adapter Manager" tab is selected. The page content shows a table of RIB Adapters. The table has columns for "Name", "Status", "Start Time", "JMS Server ID", "Edit Properties", and "View Log". The "Status" column shows a red downward arrow for all entries. The "JMS Server ID" column shows "jms1" or "jms2".

Select	Name	Status	Start Time	JMS Server ID	Edit Properties	View Log
<input type="checkbox"/>	▼ Polling_Publishers					
<input type="checkbox"/>	ASVOut Publisher, channel 1	↓		jms2		
<input type="checkbox"/>	Alloc Publisher, channel 1	↓		jms1		
<input type="checkbox"/>	Banner Publisher, channel 1	↓		jms1		
<input type="checkbox"/>	DMOp Publisher, channel 1	↓		jms1		
<input type="checkbox"/>	Diffs Publisher, channel 1	↓		jms1		
<input type="checkbox"/>	ItemLoc Publisher, channel 1	↓		jms1		
<input type="checkbox"/>	Items Publisher, channel 1	↓		jms2		
<input type="checkbox"/>	Items Publisher, channel 10	↓		jms2		
<input type="checkbox"/>	Items Publisher, channel 2	↓		jms2		
<input type="checkbox"/>	Items Publisher, channel 3	↓		jms2		
<input type="checkbox"/>	Items Publisher, channel 4	↓		jms2		
<input type="checkbox"/>	Items Publisher, channel 5	↓		jms2		
<input type="checkbox"/>	Items Publisher, channel 6	↓		jms2		
<input type="checkbox"/>	Items Publisher, channel 7	↓		jms2		
<input type="checkbox"/>	Items Publisher, channel 8	↓		jms2		
<input type="checkbox"/>	Items Publisher, channel 9	↓		jms2		
<input type="checkbox"/>	Merchant Publisher, channel 1	↓		jms1		
<input type="checkbox"/>	Order Publisher, channel 1	↓		jms1		
<input type="checkbox"/>	Order Publisher, channel 10	↓		jms1		
<input type="checkbox"/>	Order Publisher, channel 2	↓		jms1		
<input type="checkbox"/>	Order Publisher, channel 3	↓		jms1		
<input type="checkbox"/>	Order Publisher, channel 4	↓		jms1		
<input type="checkbox"/>	Order Publisher, channel 5	↓		jms1		
<input type="checkbox"/>	Order Publisher, channel 6	↓		jms1		

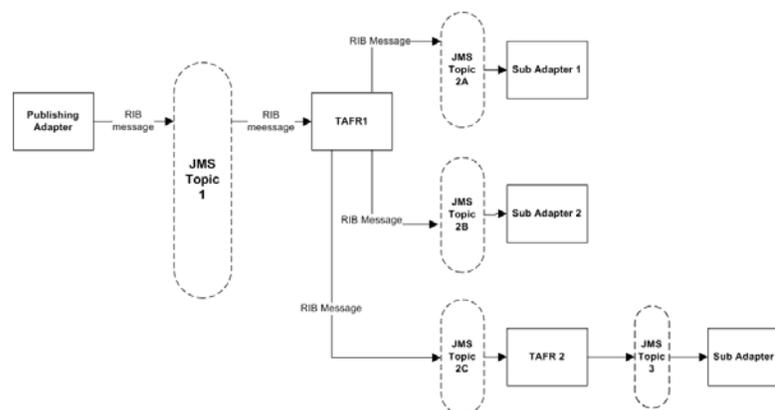
Message Transform, Filtering and Routing (TAFR)

After initial publication, a message may require a series of transformation, filtering, or routing operations. The RIB component that implements these operations is known as a Transformation and Address Filter/Router (TAFR) component.

- A transformation operation changes the message data or contents.
- A filter operation examines the message contents and makes a determination as to whether the message is appropriate to the specific subscriber.
- A router operation examines the message contents and forwards the message to a subset of its subscribers. A filter operation can be considered a special case of a routing operation. Although logically separate operations, for performance reasons, TAFR components usually combine as many as is appropriate.

A TAFR's operation is specific to a message family and the set of subscribers to it. Multiple TAFRs may process a single message for a specific subscriber and different specific TAFRs may be present for different subscribers. Different sets of TAFRs are necessary for different message families.

Multiple TAFRs may be needed, depending on the types of subscribers. The following diagram shows the message flow with TAFR.



TAFR Adapter Process

A Transformation Address Filter/Router (TAFR) adapter is used to perform operations on all messages from a single message family. The specific activities performed are dependent on the needs of its subscribers.

- TAFRs in a message flow are an exception rather than a norm. (For example, a TAFR that does message transformation for only a single application is not recommended.) The subscribing application is responsible for filtering and transformation of the payload data.
- Payload content based routing is not recommended as it degrades performance.
- TAFR adapters take advantage of the RIB hospital.
- Error messages are automatically retried by the hospital retry adapter.
- The TAFR configuration makes most of the routing decision dynamic without requiring any configuration.
- TAFRs are standard Java EE Message Driven Beans(MDB).
- Custom TAFR business implementation can be easily plugged in by editing rib-tafr-adapters.xml.

Configuration

Deployment configuration of the TAFR in the javaEE container is handled by the rib-app-builder application. Refer the documentation for the rib-app-builder on how to deploy a TAFR application. The following is an example configuration in rib-tafr-adapters.xml.

```
<tafrs>
  <message-driven id="Alloc_tafr_1" initialState="running"
  tafr-business-impl="com.retek.rib.domain.tafr.bo.impl.AllocToStockOrderFromRibBOImp
  l" />
</tafrs>
```

- message-driven - Indicates that the TAFR is deployed as an MDB.
- id - The ID for this particular adaptor.
- InitialState - The state of the adaptor.
- Tafr-business-impl - The implementation class for this TAFR. This class contains the implementation for transformation, filtering, and routing of RIBMessage.

Transformation

Message transformation is the process of converting one message family payload to another message family payload.

Filtering Configuration

Filtering configuration involves updating the rib-tafr.properties file with the appropriate information.

The property follows the usual properties naming convention (name=value).

The property that is used for filtering is for.<tafr name>_tafr.drop-messages-of-types.

Example:

```
for.ItemsToItemsISO_tufr.drop-messages-of-types=
ISCDimCre,ISCDimMod,ISCDimDel,ItemImageCre,ItemImageMod,ItemImageDel,
ItemUdaDateCre,ItemUdaDateMod,ItemUdaDateDel,ItemUdaFfCre,ItemUdaFfMod,ItemUdaFfDe
l,
ItemUdaLovCre,ItemUdaLovMod,ItemUdaLovDel
```

This property should be read as "for ItemsToItemsISO tufr" drop these message types. A comma delimits the message types.

If customization is required then the rib-tafr.properties file needs to be updated for filtering to take place.

Routing

Routing is enabled by default for TAFR's, the RIB infrastructure handles this routing. If a TAFR requires routing based on message content, then implementation classes override the following method.

```
public void routeRibMessage(RibMessage newMsg,MessageRouterIface router) throws
TafrException {
    router.addMessageForTopic(eventType, newMsg);
}
```

Configuration Example - Facility ID

One of the common configurations requirements is to set up the flow of transfers and orders to RWMS. This is based on Facility ID.

These examples and step-by-step instructions illustrate how to configure a TAFR for one and two RWMS deployments.

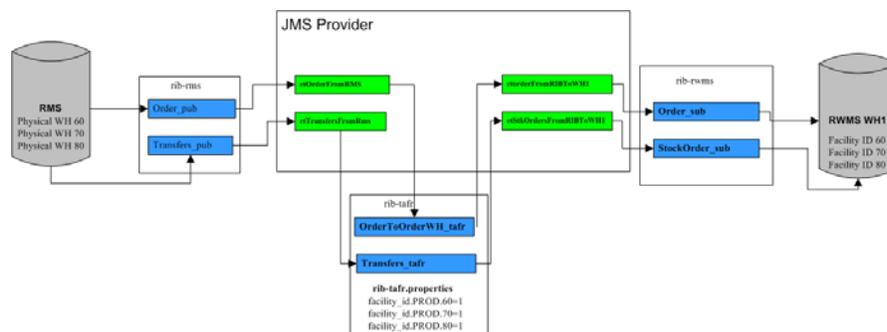
Single RWMS Configuration

RIB allows stock based transactions to be routed between different RWMS instances. An RWMS instance is assigned to a physical distribution center which may have one or more facilities assigned to it. A company may have one or more distribution centers.

By default the standard RIB configuration is set for a single RWMS instance. This means that all physical warehouses in RMS route directly to a single RWMS instance (in this case denoted as WH1) with each RMS physical warehouse directly correlating to a facility ID in RWMS.

Configuration Process

1. Modify the TAFR routing settings:
 - For each physical warehouse set up in RMS there should be a matching entry in the rib-tafr.properties file. This file resides in the \$RIB_HOME/application-assembly-home/rib-tafr directory and is used by the TAFR adapters, amongst other things, to route messages by facility ID to the correct RWMS instances.
 - The file by default contains the following mappings:
 - facility_id.PROD.1=1
 - facility_id.PROD.2=1
 - facility_id.PROD.3=1
 - The routing properties are structured in the following way: facility_id.<FACILITY_TYPE>.<RMS_PHYSICAL_WH_ID>=<RWMS_INSTANCE_NAME>
 - <FACILITY_TYPE> - This should match the facility_type.default value in the rib-tafr.properties file. In most cases it is defaulted to PROD.
 - <RMS_PHYSICAL_WH_ID> - The physical warehouse ID from RMS.
 - <RWMS_INSTANCE_NAME> - The RWMS installation topic name identifier to which the warehouses messages is routed.
 - These mappings must be edited so that each physical warehouse in RMS has its own entry. The physical warehouses can be found by running the following query in the RMS schema:
 - SELECT wh FROM wh
WHERE wh.wh = wh.physical_wh;
 - For the example in the diagram above, the query returns physical warehouse IDs 60, 70, and, 80 .
 - There is only one RWMS instance (WH1) in this example, and the RWMS installation topic name identifier is 1. This corresponds to the name of the topics that the RIB routes the messages to. It also is the default name suffix of the RWMS topics in the rib-integration-flows.xml file.



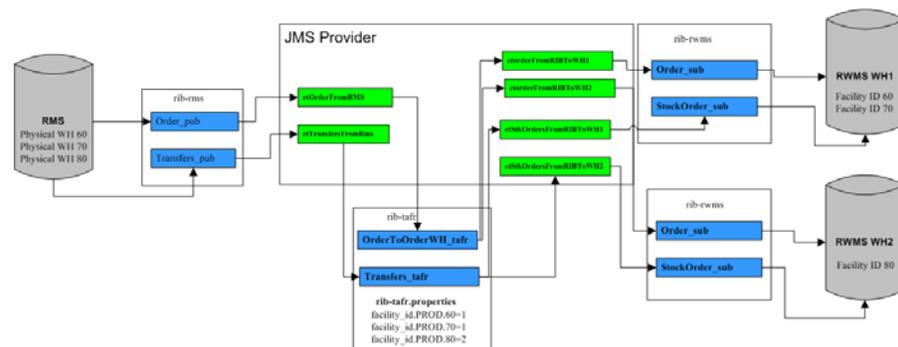
- Therefore, mapping in the rib-tafr.properties file should read as follows:
 - facility_id.PROD.60=1
 - facility_id.PROD.70=1
 - facility_id.PROD.80=1
- 2. Deploy the settings to the rib-tafr instance:

The new TAFR routing settings must be migrated to the rib-tafr instance. Run the following script found in the \$RIB_HOME/deployment-home/bin directory.

```
rib-app-deployer.sh -deploy-rib-app-ear rib-tafr
```
- 3. Configuration should now be complete.

Note: For every new physical warehouse added to RMS, the rib-tafr.properties file requires a new entry. The new settings must be deployed to the instance.

Two RWMS Configuration



Description

RIB can be configured to route stock based transactions between multiple distribution centers, each with their own RWMS instance. The purpose of this is to only send stock transactions that are shipped to or from a certain warehouse to the distribution center that contains that warehouse (facility).

From RMS the user only has visibility to the warehouse that they are performing a stock shipment to or from, the RIB TAFR's then route the messages to the separate RWMS instances based on the configuration stated in the rib-tafr.properties file. In the above example, RMS physical warehouses 60 and 70 are assigned to the RWMS instance called WH1 while RMS physical warehouse 80 is assigned to another RWMS instance called WH2.

Configuration Process

1. Modify the TAFR routing settings:
 - For each physical warehouse set up in RMS there should be a matching entry in the rib-tafr.properties file. This file resides in the \$RIB_HOME/application-assembly-home/rib-tafr directory and is used by the TAFR adapters, amongst other things, to route messages by facility ID to the correct RWMS instances.
 - The file by default contains the following mappings:
 - facility_id.PROD.1=1
 - facility_id.PROD.2=1
 - facility_id.PROD.3=1
 - The routing properties are structured in the following way: facility_id.<FACILITY_TYPE>.<RMS_PHYSICAL_WH_ID>=<RWMS_INSTANCE_NAME>
 - <FACILITY_TYPE> - This should match the facility_type.default value in the rib-tafr.properties file (in most cases, PROD).
 - <RMS_PHYSICAL_WH_ID> - The physical warehouse ID from RMS.
 - <RWMS_INSTANCE_NAME> - The RWMS installation topic name identifier to which the warehouses messages are routed.
 - These mappings must be edited so that each physical warehouse in RMS has its own entry. The physical warehouses can be found by running the following query in the RMS schema:
 - ```
SELECT wh FROM wh
 WHERE wh.wh = wh.physical_wh;
```
  - Before editing the file for multiple RWMS instance routing, the user should know which RMS physical warehouses are to be routed to the particular RWMS instances and the RWMS installation topic name identifiers.
  - For the example, in the diagram above, physical warehouse IDs 60 and 70 are routed to RWMS instance WH1, where the RWMS installation topic name identifier is 1 and RMS physical warehouse ID 80 are routed to RWMS instance WH2, where the RWMS installation topic name identifier is 2. To support this, the mapping in the rib-tafr.properties file should read:
    - facility\_id.PROD.60=1
    - facility\_id.PROD.70=1
    - facility\_id.PROD.80=2
2. Modify the rib-integration-flows.xml file:
  - The RIB requires information on how to route the messages between the two RWMS instances. This is done by adding new entries to the rib-integration-flows.xml file.
  - By default the file contains entries for the RWMS instance "rib-rwms" and all appropriate warehouse based adaptor mappings point to the et<TOPIC\_NAME>WH1 topics. When adding multiple RWMS instances all the entries for RWMS need to be duplicated for the second instance "rib-rwms2" and all adaptor mappings for the new instance need to point to et<TOPIC\_NAME>WH1 topics.

- The entire "RWMS PUBLISHERS" section in the integration-flows.xml file needs to be duplicated and all new entries need to be changed to the second RWMS instance name of "rib-rwms2" for example:
    - ```
<node id="rib-rwms2.ASNIn_pub" app-name="rib-rwms2"
  adapter-class-def="ASNIn_pub"
  type="DbToJms"><in-db>default</in-db><out-topic>etASNIn</out-top
ic></node>
```
 - Each RWMS adapter mapping in the file that follows the et<TOPIC_NAME>WH1 format needs to be duplicated as well but needs to point to et<TOPIC_NAME>WH2. With the original adapter mapping and the new adapter mapping to route to the second RWMS instance, for the Stock Order adapter, the entry should be similar to the following example:
 - ```
<node id="rib-rwms.StockOrder_sub" app-name="rib-rwms"
 adapter-class-def="StockOrder_sub"
 type="JmsToDb"><in-topic>etStkOrdersFromRIBToWH1</in-topic><out-
db>default</out-db></node>
```
    - ```
<node id="rib-rwms2.StockOrder_sub" app-name="rib-rwms2"
  adapter-class-def="StockOrder_sub"
  type="JmsToDb"><in-topic>etStkOrdersFromRIBToWH2</in-topic><out-
db>default</out-db></node>
```
 - The rib-integration-flows.xml file can be edited and then deployed in the following way:
 - cd \$RIB_HOME/application-assembly-home/rib-func-artifacts
 - jar -xvf rib-func-artifact.war
 - cd integration
 - vi rib-integration-flows.xml
 - Make the changes specified above.
 - jar -uvf rib-func-artifact.war integration/rib-integration-flows.xml
3. Deploy the settings to the rib-tafr instance:
- The new TAFR routing settings need to be migrated to the rib-tafr instance, to do this run the following script found in the \$RIB_HOME/deployment-home/bin directory.
- ```
rib-app-deployer.sh -deploy-rib-app-ear rib-tafr
```
4. Deploy the settings to the functional artifact:
- The new integration flow settings need to be migrated to the rib-func-artifact instance, to do this run the following script found in the \$RIB\_HOME/deployment-home/bin directory.
- ```
rib-app-deployer.sh -deploy-rib-func-artifact-war
```
5. Configuration should now be complete.

Note: For every new physical warehouse added to RMS the rib-tafr.properties will require a new entry and the new settings will need to be deployed to the instance.

Note: Multiple RWMS instances can be added as per the instructions above.

Changes to this configuration affect the following TAFRS.

- AllocToStockOrder
- ASNOutToASNInLoc
- CustOrderToStockOrder
- ItemLocToItemLocLoc
- OrderToOrderWH
- PendReturnToPendReturnWH
- RTVReqToRTVReqLoc
- TransfersToStockOrder
- WOInToWOInWH
- WOOOutToWOOOutWH

Diagnostic and Monitoring Tools

The RIB Diagnostic and Monitoring Tool Kit (RDMT) is a collection of command line tools, written in UNIX shell script along with supporting Java classes packaged in jar files. There are various tools to address these areas:

- Installation Verification (reports)
- Operations (scanning and monitoring)
- Production (scanning and quick triage)
- Test and Support (scanning and fine grain control)
- AQ JMS support and tools

The RIB is a complex collection of distributed components, and there are a variety of GUI tools. These tools augment those tools and provide command line control and access to all levels of the RIB functions. The tools are written to be stand-alone and to provide examples and capabilities for integration into enterprise level OSS and management frameworks, such as Oracle Enterprise Manager, Tivoli or HP OpenView.

Functionality

- Support for Oracle WLS RIB Version.
- Support for local/remote installation.
- Support for Oracle Streams AQ JMS as the JMS Provider.
- Support for RIB Hospital databases.
- Support for RAC Configured Databases.
- Support for JMX control of all RIB Components
- Support for message Pub/Sub.

All of the scripts are written to be examples of specific functionality, but have been integrated into a simple tool kit that is configuration driven and has a very simple character-based menu system provided to allow a single point of integration.

RDMT and User Roles and Responsibilities

The tools are written to provide capabilities and examples of functions for users with various roles and responsibilities.

The primary target role is the RIB administrator as someone who is responsible for the installation, configuration, and deployment of the RIB components. The ribadmin is also usually then tasked with ongoing RIB Software Life Cycle management as well as production operation support. This person has full permissions on all of the application server directories and has full read and execute permissions on the Oracle Application Server tools such as opmnctl and the WLS instance subdirectories.

Local or Remote Installations and Capabilities

RDMT can be installed by a user on the system that may or may not have the RIB/WLS environment. RDMT tools support local and remote WLS functions through JMX.

In remote installations, some scripts in the toolkit expect the installing user to have read permissions of the WebLogic home RIB WLS sub-directories or require execute permission of opmnctl. Therefore, these will return file or permissions errors.

Once the roles and responsibilities of the user have been understood and established, follow the installation instructions. See the *Oracle RIB Integration Bus Installation Guide*.

RDMT Support jars

.jar File	Description
rib-jms-api.jar	Support classes for jms.
rib-jms-admin-aq-impl.jar	Specific impl for AQ
rib-jms-admin.jar	Support classes for AQ admin
jmx-cmd-line-ui.jar	JMX client
rdmt.jar	Support tools

Sample XML Messages

The RDMT release packages a zip file of example xml messages for each message family and message type payload. The zip file is located in the RDMT subdirectory testmsgs.

Tools Overview

RDMT has been designed as a set of command line tools that can serve generally needed functions with examples for retailer specific uses, and to provide a ready to use, low impact application. In many situations, it is a requirement to have tools that consume low bandwidth to manage and triage the RIB. These tools provide alternatives to the GUI based tools. The other common requirement is for control and monitoring command line scripts that can be incorporated into enterprise operations scheduling frameworks, such as Autosys or Appworkx.

RDMT has been organized around a very simple character-based menu system that can be modified to suit the deployment roles and responsibilities and to provide some structure by functional area.

RDMT as an Application

SCRIPTDIR

All of the tools have been organized into a simple application and accessible via the character-based menu system. All of the tools have been designed to execute relative to a based directory (readmit). Within that base directory all tools expect to find all of the support libraries and other scripts. To execute any tool, all that is needed is to set the base directory as an environmental variable, SCRIPTDIR.

Setup

RDMT can be installed either inside/outside rib-home or in a remote server. To install inside/outside rib-home, the installation script (configbuilder.sh) automatically fetches all necessary configuration parameters from rib-deployment-env-info.xml from inside the specified rib-home/deployment-home/conf directory.

However, if RDMT is installed in a remote server, the installation script prompts for the RIB deployment environment specific values. All of the scripts have been designed to be configuration driven by property files. The setup process updates these files.

Current Configuration

Because there are multiple configurations possible with the fully distributed RIB, all of the tools are designed to work against a set of property files that provide the values need to execute. Collectively, these are called "current." In the menu system there are functions that allow configuration of n-number of configurations. For example there can be n-number of rib-<app>'s configured. Other functions set runtime configuration files to these "current" configurations. All tools then read these "current" values and perform tasks against them.

RDMTLOGS

All of the tools are designed to produce logs and to use temporary files. The location of these logs is a configuration parameter and defaults to RDMTLOGS within the rdmt base directory.

RDMT RAC Support

RDMT supports RAC configured databases. The user needs to provide the entire database connection URL when prompted during the setup process.

The user needs to provide the same JDBC connection URL for AQ JMS or Hospital Databases as supplied in rib-deployment-env-info.xml during the RIB installation. The user can provide either thin JDBC connection URL or long JDBC connection URL format (in case of RAC configured database) depending on the user's environment.

The user needs to provide only the database user name, password, and connection URL to configure for any AQ JMS/ Hospital database. The example below shows the configuration for an hospital database during the RDMT setup process.

WLS/JMX Utilities

Script Used:

rdmt_jmx_submenu

Description:

This menu option exposes the various tools that use JMX to interact with the WLS instance and to control or status the current rib-<app> and its components.

```

wlsadmin@wls001:~$ cd $ORACLE_HOME/bin; ./rdmt_jmx_submenu
WLS Diagnostic & Monitoring Tools
WLS/JMX Submenu

This Host : kirw001r10.oracle.com
STARTLOGS : /u00/wlsadmin/RIB_INSTALL_MBT/RIK1370ForAllJ2EEApps/rih-home/tools-home/rdmt/STARTLOGS
RIB DB : rih-rms_error-hospital-database_user-name=Allw001@dc.oracle.com:1522:rtta170dy64
TAFB DB : needs_value@needs_value
RMSB DB : needs_value@needs_value
RFB DB : needs_value@needs_value
RFP DB : needs_value@needs_value
RTH DB : needs_value@needs_value
JMS Type : rjms
JMS Connection URL : jdbc:oracle:thin:@wls001r10.us.oracle.com:1522:rtta170dy64
JMS CFG : jms3.conf
WLS : wls001r10.us.oracle.com:18003
Instance : rih-rms-wls-instance
RIB APP : rih-rms
Type : euk
Adapter : default

Selections:

1 - JMX cmdline Utility          5 - Get WLS Instance State      9 - Start ALL Adapters
2 - WLS/JMX Config Utility      6 - Get RIB App State          10 - Stop ALL Adapters
3 - Switch WLS/JMX Config      7 -                             11 - Status ALL Adapters
4 - Status ALL config'd WLS    8 -                             12 - Remove ALL Adapters

13 - Get Logger Name(s)        17 - Status ALL RIB           21 - Get Adapter RIB
14 - Get Logging Levels        18 - Status ALL RFB           22 - Get Adapter RFB
15 - Get Logging Levels(s)     19 - Status ALL RFP           23 - Get Adapter RFP
16 - Get ALL Levels (xml)      20 - Status ALL TAFB         24 - Get Adapter TAFB

25 - Status Based Adapter      29 - Start ALL RIB           33 - Stop ALL RIB
26 - Remove Based Adapter      30 - Start ALL RFB           34 - Stop ALL RFB
27 - Start Based Adapter (force) 31 - Start ALL RFP           35 - Stop ALL RFP
28 - Stop Based Adapter        32 - Start ALL TAFB         36 - Stop ALL TAFB

99 - Main Menu

```

Script	Description
jmx_app_state.sh	status of the currently active rib application
jmxcmdline_interactive.sh	A wrapper to the jmx client support classes. This script directly invoke the interactive functions.
jmxcmdline.sh	General wrapper for other tools to invoke specific jmx functions.
jmxconfig.sh	There are multiple configurations possible with the fully distributed J2EE RIB. This utility is used to manage the configuration files that allow the rdmt tools to access them. This option can also be used to switch/re-configure the previously configured WLS/JMX configuration.
jmx_get_logger_names.sh	RIB app logging tool
jmx_get_logging_levels_all.sh	RIB app logging tool
jmx_get_logging_levels.sh	RIB app logging tool
jmx_set_logging_levels.sh	RIB app logging tool
jmx_managed_adapters.sh	Common script used by all jmx tools to interact with the jmx client jar. Many of the menu selections merely set the calling parameters to this tool.
jmx_OC4Jribstatus.sh	Get the run state of the rib-app WLS instance and application for all configured.

Script	Description
jmx_OC4Jrib_scan.sh	For all configured rib-app scans the state of the instance, app and adapters.
jmx_oc4j_state.sh	Status of the currently active WLS instance.
jmx_switch_config.sh	This utility is used to switch the active configuration file that the rdmt tools use.
jmx_tester.sh	Test script for testing arbitrary jmx commands within the RDMT framework. This is not a menu selection since it requires user editing.

Utility	Description
start all adapters (jmx)	This utility option starts all adapters of a rib-<app>, where app refers to rms, rwms, sim, rpm, or tafr. The adapters start only when the initial state is specified as running for the adapters in the rib-<app>-adapters.xml. If the initial state=stopped, an error is thrown: "Cannot start; initial state is set to stopped". If an adapter already is running, executing this option keeps the adapter in the previous state.
start adapter (jmx)	This utility option starts a single adapter of a rib-<app>, where app refers to rms, rwms, sim, rpm, or tafr. It starts the adapter only when the initial state is specified as running for the adapter in the rib-<app>-adapters.xml. If the initial state=stopped, an error is thrown: "Cannot start; initial state is set to stopped". If the adapter already is running, executing this option keeps the adapter in the previous state itself.
startForced adapters (jmx)	This utility starts all adapters of a rib-<app>, where app refers to rms, rwms, sim, rpm, or tafr. It starts all adapters irrespective of their initial state in rib-<app>-adapters.xml.
startForced adapter (jmx)	This utility option starts a single adapter of a rib-<app>, where app refers to rms, rwms, sim, rpm, or tafr. It starts the adapter irrespective of its initial state in rib-<app>-adapters.xml.

Hospital Scan Tools

Script Used:

rdmt_hosp_submenu

Description:

This option leads the user to the RIB hospital for various applications submenu through which the user can get the current RIB hospital status.



Script	Description
hstest.sh	This script calls a java class that uses JDBC to access the database(s) containing the Hospital tables. It scans the Hospital RIB_MESSAGES table and report thing such as: <ul style="list-style-type: none"> how many messages (row count), how many have exceed the retry count, how many messages of a topic as present.
hstest_failures.sh	This script calls a java class to scan the RIB Hospital Message Failure table using a JDBC connection.

RIB Admin Tools

Script Used:

rdmt_ribadmin_submenu

Description:

The ribadmin script was stand-alone in previous RIB releases. Those functions have since been moved into this menu item. The ribadmin.sh script is sourced to make the existing functions available to the menu items and the variables that the scripts expected have been mapped to rdmt configuration files.

Since many of the functions expect execute permissions on opmnctl as well as read/write permissions on the WLS directory tree, this menu and the tools are designed for the ribadmin role.


```

This Host: hlcw008 ide-oracle.com
RDMTLOGS: /u01/ohadmin/RIB_INSTALL_007/03132007/ora1113/ribapps/rib-home/tools-home/rdmt/RDMTLOGS
RIB ID: rib-rms_server-hospital-database_user-name-allartjdb:oracle:thin:scspdv170.us.oracle.com:1527:rtt4170v44
TMR ID: needs_valloc/needs_valloc
RMR ID: needs_valloc/needs_valloc
RIR ID: needs_valloc/needs_valloc
RIF ID: needs_valloc/needs_valloc
RIS ID: needs_valloc/needs_valloc
JMS Type: ojms
JMS Connection URL: jdbc:oracle:thin:scspdv170.us.oracle.com:1527:rtt4170v44
JMS URL: jmsl.com?
WLS: scspdv170.us.oracle.com:15063
Instance: rib-rms-wls-instance
RIB APP: rib-rms
Type: web
Adapter: default

Commands:
1 -
2 - remove all adapters (jms)
3 - remove named adapter (jms)
4 -
5 -
16 - View managed adapter xml
17 - View rib-app.properties
18 - View rib-system.properties
25 - Edit managed adapter xml
26 -
27 - rib-app-builder tools menu
28 - Main menu

6 - stop all adapters (jms)
7 - stop named adapter (jms)
8 -
9 -
19 -
20 -
22 - View adapter log
23 - Archive APP logs
24 - Switch WLS/JMS Config

10 - start all adapters (jms)
11 - start adapter (jms)
12 - startPeriod adapter (jms)
13 - startPeriod adapter (jms)
14 - startPeriod adapter (jms)
15 - startPeriod adapter (jms)

29 - Editor for rib-deployment-over-info.xml
30 -
31 -
32 -
33 -
34 -
35 -
36 -
37 -
38 -
39 -
40 -
41 -
42 -
43 -
44 -
45 -
46 -
47 -
48 -
49 -
50 -
51 -
52 -
53 -
54 -
55 -
56 -
57 -
58 -
59 -
60 -
61 -
62 -
63 -
64 -
65 -
66 -
67 -
68 -
69 -
70 -
71 -
72 -
73 -
74 -
75 -
76 -
77 -
78 -
79 -
80 -
81 -
82 -
83 -
84 -
85 -
86 -
87 -
88 -
89 -
90 -
91 -
92 -
93 -
94 -
95 -
96 -
97 -
98 -
99 -
100 -

```

Scan RIB Logs / Scan RIB Logs (Delta)

Scripts Used:

loglookoc4j.sh, scan_logs.sh, loglookoc4j_delta.sh, scan_logs_delta.sh

Description:

These scripts perform a log scan to look for a pattern ("Exception") in all of the log files in a directory of the currently active WLS instance. Since they perform file system scans, the RDMT tools must be installed on the host that contains these logs and must have read permissions on the directories and the files.

As the tool scans all of the logs it writes the matches to a single log file. This becomes the base file. A second script (delta) looks for the same pattern, but compares the matches against the base file, and outputs only new ones. The primary scripts are the scan_logs.sh and the scan_logs_delta.sh. The files created and used by these scripts are controlled by the rdmt.conf entries.

The location of these files should be sized to handle large text files, since it is possible for there to be many exceptions and these will contain the consolidated entries from potentially hundreds of logs. The location is the tmp files parameter set during RDMT installation and is defaulted to RDMTLOGS/tmp.

RIB Health

Script Used:

cron_ribhealth.sh

Description:

This utility is a general purpose script that invokes other tools and functions in the tool kit to take a snapshot in time of the run-time state of all of the configured rib-apps. Because this script uses specific jar files as well as other tools in the tool kit, it requires that SCRIPTDIR be set to the rdmt base directory.

It produces a rib_health report on the console as well as a time-stamped log written to the RDMTLOGS/tmp directory. Each execution of the script produces one of the logs, and then over-writes a log called lastrun as well. There is a menu selection that views the lastrun report.

RIB Configuration Report

Script Used:

OC4JConfigReport.sh

Description:

After the RIB has been installed and configured on WLS, the user can verify installations and configurations using RDMT. A script is linked in the RDMT menu that scans the installations and configurations of rib applications deployed using the configuration settings in the RDMT configuration files.

It is recommended that after the installation is complete, the user runs the RIB Config Report utility from the RIB Health Menu option. This outputs the results of the scan on the console as well as in an output file under the specified TEMP FILES DIRECTORY. Each run produces a time-stamped log and updates a log called lastrun-config that is viewable from a menu selection.

This script was written to take a snapshot of the RIB environment and test for basic configuration issues. This utility does the following:

- Displays all RIB apps and shows the status for each RIB WLS instance and application.
- Performs JMX related functions such as scanning configurations for each jmxX.conf file and displays the status of the adapters, exceptions from scanning the logs, and so on.
- Performs checks using JMS configuration.
- Performs checks using Hospital configurations.

RIB Timings Utility

Script Used:

timingsutil.sh

Description:

RIB can log a set of timing entries whenever it creates, transform, routes, filters, or subscribes to messages on the RIB. This utility on functions when RDMT is installed on the host system where the logs are generated and the RDMT user has permissions to read the log directories.

The timingsutil.sh script is a wrapper to the RIBTimings java class. This script runs the RibTimings post processor on an adapter's timing file. It prompts for the adapter name then it analyzes the timings logs for that particular adapter. The output is to the screen as well as a file of CSV format in the RDMT temp files directory; RDMTLOGS/tmp/<adapter>.csv. which contains the detailed analysis of timings logs.

This csv file can be directly viewed by Excel. To use this function, the adapter timing log parameters must be set to DEBUG.

JMS Publish Utility

Script Used:

pubmsgutil.sh

Class:

RibJmsPublisherTester

Description:

This utility was developed to publish a message to a JMS topic. The pubmsgutil.sh is a wrapper script to RibJmsPublisherTester. It calls the JMS API and publishes the message on to the topic.

Usage:

- " java com.retek.rib.jms.RibJmsPublisherTester
- -j <JMS provider>
- -x <JMS URL>
- -u <JMS username>
- -p <JMS password>
- -t <topicName>
- -n <xmlFileName>
- -wm <should messages be wrapped in RIBMessage envelop>
- -f <messageFamily>
- -m <messageType>
- -ri <routingInfo>
- -tv<threadValue>
- -nt <<number of times - optional (default value is 1)>>

EJB Publish Utility

Script Used:

ejbpub_util.sh

Class:

RibMessagePublisherClient

Description:

This utility was developed to wrapper the EJB Message Publish Java API. The ejbpub_util.sh is a wrapper script to RibMessagePublisherClient. It calls the specified EJB service and publishes the message on to the AQ JMS. It uses the platform service to publish the message. The user needs to specify the necessary parameters.

Usage:

- " java com.oracle.rib.rdmtd.util.RibMessagePublisherClient
- -host <<host | 1>>
- -port <<RMI port -- required>
- -app <<App name -- required>>

- -fa <<family -- required>>
- -ty <<type -- required>>
- -us <<user | optional>>
- -pw <<password | optional>>
- -fi <<file -- required>>

TAFR Msg Utility

Script Used:

tafrmsgutil.sh

Class:

TestAnyTafrBOImpl

Description:

This utility is developed to test any TAFR Business implementation. This helps to check the output of a particular TAFR by providing the necessary prompted arguments. It prompts the user for the TAFR name, the location of the input sample file, adapter id, and the output file name to direct the output (optional). If the output file name is not specified, the default output is routed to stdout. Once provided the valid arguments, the user can see the output of that particular TAFR.

Sample Output:

```
-----
          TAFR TEST UTILITY
-----

This utility was developed to test any TAFR Business implementation.
These are designed to help diagnose/validate the TAFR implementation.

Usage: java TestAnyTafrBOImpl
       <tafr name -- required>
       <file path -- required>
       <true|false print message data -- required>
       <adapter-id -- required>
       <file name -- optional | default output routed to stdout>

Do you wish to continue ?
Enter to Continue? y/n/q [y]:

Existing parameter values:
StoresToStoresPhys /u00/rib-home/tools-home/rdmt/testmsgs/PODesc.xml true
StoresToStoresPhys_tufr_1

Do you wish to execute using existing values?

Enter to Continue? y/n/q [y]: n

Enter TAFR Name [StoresToStoresPhys]:
You entered: StoresToStoresPhys
Enter to Continue? y/n/q [y]:

Enter the Input message file and path
/u00/rib-home/tools-home/rdmt/testmsgs/PODesc.xml]:
/u00/rib-home/tools-home/rdmt/testmsgs/storedel.xml
You entered: /u00/rib-home/tools-home/rdmt/testmsgs/storedel.xml
```

```

Enter to Continue? y/n/q [y]:

Print Message Data? [ true | false] [true]:
You entered: true
Enter to Continue? y/n/q [y]:

Enter Adapter ID [StoresToStoresPhys_tufr_1]:
You entered: StoresToStoresPhys_tufr_1
Enter to Continue? y/n/q [y]:
Enter the Output file name and path if required (default output routed to stdout)
[]:
You entered:
Enter to Continue? y/n/q [y]:
log4j:WARN No appenders could be found for logger
(com.retek.rib.domain.ribmessage.bo.RibMessagesFactory).
log4j:WARN Please initialize the log4j system properly.
Trying to load rib-system.properties from
class.path=lib/rib-private-tufr-business-impl.jar:../../application-assembly-home/
rib-tufr/
rib-system.properties loaded from
file:/u00/rib-home/application-assembly-home/rib-tufr/rib-system.properties
Trying to load rib-tufr.properties from
class.path=lib/rib-private-tufr-business-impl.jar:../../application-assembly-home/
rib-tufr/
rib-tufr.properties loaded from
file:/u00/rib-home/application-assembly-home/rib-tufr/rib-tufr.properties

*****RIBMESSAGES TRANSFORMED DATA*****
<?xml version="1.0" encoding="UTF-8"?>
<RibMessages xmlns="http://www.oracle.com/retail/integration/rib/RibMessages"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oracle.com/retail/integration/rib/RibMessages
file:///C:/rib-home/application-assembly-home/rib-tufr/integration/xsd/RibMessages
.xsd">
  <ribMessage>
    <family>Stores</family>
    <type>storedel</type>
    <ribmessageID>11.0.x|RIBMessagePublisherEjb|null|2006.01.23
11:45:46.052|510</ribmessageID>
    <routingInfo>
      <name>loc_type</name>
      <value>p</value>
    </routingInfo>
    <routingInfo>
      <name>dest_id</name>
      <value/>
      <detail>
        <dtl_name>loc_type</dtl_name>
        <dtl_value>p</dtl_value>
      </detail>
    </routingInfo>
    <publishTime>2006-01-23 11:45:46.052 CST</publishTime>
    <messageData>&lt;AllocDesc&gt;&lt;store&gt;5&lt;/store&gt;
&lt;/AllocDesc&gt;
    </messageData>
    <customData/>
    <customFlag>F</customFlag>
  </ribMessage>
</RibMessages>
*****PAYLOAD DATA*****

```

```

<?xml version="1.0" encoding="UTF-8"?>
<AllocDesc>
<store>5</store>
  </AllocDesc>
*****
***OUT MESSAGE WILL BE ROUTED TO THE FOLLOWING TOPIC(S)*****
*****
1) etStoresPhys
*****

```

EJB Ping (RIB)

Script Used:

ribejbping.sh

Class:

RibMessageInjectorClient

Description:

This utility is developed to ping any RIB-<app> EJB component. This helps to test the connectivity between the rib-<app> and the <app> where app refers to RMS, SIM, RWMS, RPM, and AIP. This uses the platform configuration to call the EJBs. The user needs to put the necessary config files namely services_*.xml, jndi_providers_*.xml and service_flavors_*.xml of the particular rib-<app> under \$HOME/rdmtXX/rib-app/extracted_conf_from_rib-<app>/retek folder before using this utility option. The user can also use the jndi_config_extractor.sh script for extracting the necessary config files from the system.

The jndi_config_extractor.sh script can be found under the RDMT home directory. The user needs to copy the jndi_config_extractor.sh script to the machine where the particular rib-<app> is deployed and then execute the script. The user can execute this script from any place in that machine. The user also needs to set JAVA_HOME before running this script. This script will extract the files namely services_*.xml, jndi_providers_*.xml and service_flavors_*.xml files of that particular rib-<app> instance by checking the entire file system. All the extracted xml files are finally zipped into xmlfiles.zip in the current directory from where this script was executed. The user then needs to transfer/ftp this xmlfiles.zip into the server where RDMT is installed and put these files under \$HOME/rdmtXX/rib-app/extracted_conf_from_rib-<app>/retek folder.

After performing the above steps, the user can execute the RIB EJB Ping menu option.

EJB Ping (APP)

Script Used:

appejbping.sh

Class:

RibMessagePublisherClient

Description:

This utility is developed to ping any <app> EJB component. This helps to test the connectivity between the rib-<app> and the <app> where app refers to only SIM, RPM, and AIP. This uses the platform configuration to call the EJBs. The user needs to put the necessary config files namely services_*.xml, jndi_providers_*.xml and service_flavors_*.xml of the particular <app> under \$HOME/rdmtXX/app/extracted_conf_from_<app>/retek folder before using this utility option. The user can also use the jndi_config_extractor.sh script for extracting the necessary config files from the system.

The jndi_config_extractor.sh script can be found under the RDMT home directory. The user needs to copy the jndi_config_extractor.sh script to the machine where the particular <app> is deployed and then execute the script. The user can execute this script from any place in that machine. The user also needs to set JAVA_HOME before running this script. This script will extract the files namely services_*.xml, jndi_providers_*.xml and service_flavors_*.xml files of that particular <app> by checking the entire file system. All the extracted xml files are finally zipped into xmlfiles.zip in the current directory from where this script was executed. The user then needs to transfer/ftp this xmlfiles.zip into the server where RDMT is installed and put these files under \$HOME/rdmtXX/app/extracted_conf_from_<app>/retek folder.

After performing the above steps, the user can execute the APP EJB Ping menu option.

Tool Usage Examples

"How do I know if the RIB installation is correct?"

1. Using the RDMT Menu system, select the RIB Health SubMenu.
2. Execute RIB Config Report option. This produces the basic report on installation.

This step scans the installations and configurations of rib applications deployed in wls. It finally produces a RIB WLS configuration report on the console as well as written into a file under the RDMT Temp directory, which contain the status of all the RIB configurations necessary to detect/diagnose any RIB related issues.

3. If you find any discrepancies, refer to the *Oracle Retail Integration Bus Installation Guide* and follow the steps mentioned there.

"How do I know if the local WLS is running or not?"

1. Using the RDMT Menu system, select the WLS/JMX Utilities SubMenu.
2. Execute Get WLS Instance State option. It displays the current WLS status.
3. If it is not running, start the local WLS instance. See the *Oracle Retail Integration Bus Installation Guide* for how to start it.

"How do I know where my issue is occurring?"

1. Select RDMT Main Menu.
2. Execute the Scan RIB Logs option. It performs a log scan to look for a /pattern/ ("Exception") in all of the log files in a directory of the currently active WLS instance.
3. Select JMS Topic Scan. Look for topics with messages stuck.

"How do I know if the adapter's status is up or down?"

1. Select WLS/JMX Utilities Menu.
2. Execute Status ALL Adapters option. It displays the status of all the adapters, namely the publishers, subscribers, hospital and TAFR for the currently active WLS instance.
3. If anything is down, use the Start ALL adapters option and start the same.

"How can I config/switch for a new WLS instance?"

1. Select WLS/JMX Utilities Menu.
2. Execute WLS/JMX Config Utility option.
3. Provide the desired parameters and configure an instance.
4. You can switch to the desired instance using the same option.

"How do I know the subscriber for a particular JMS topic?"

1. Select RDMT Main Menu.
2. Then select JMS Utilities Menu.
3. Execute the JMS Topics Subscribe List option.
4. Provide the topic name for which the subscriber name is needed. It provides the same.

The RIB in Operation

Operational Considerations

This section contains common issues that need to be thought about and addressed by a retailer as they progress towards a production environment involving the RIB. It is not a comprehensive list, nor does it seek to answer the questions, since they are very dependent on the retailer implementation. The intent of this section is to provide a starting point for a site-specific RIB Operations planning effort.

Alerts and Notifications

The RIB has built in alerts and notification through JMX. An external system can subscribe to all of the built-ins.

Note: See Chapter 5, "RIB and JMX."

RIB Log File Monitoring

Because the RIB is a subsystem that runs with no console, it is important to monitor the various log file that are created. Not only for the content (looking for exceptions), but also their size and growth.

RDMT includes several tools to assist in scanning and can provide examples on how to customize them to conform to particular site.

Note: See "Scan RIB Logs / Scan RIB Logs (Delta)" in this manual.

Log File Archive and Purge

The RIB use log4j for all of its logging control. It manages the logs size via its control file.

Note: See Apache Software Foundation <http://logging.apache.org/log4j/docs/documentation.html> for details.

In various phases of deployment and in triaging a problem it is often desirable or necessary to archive the logs so the logs are smaller and scanning by tools or people is easier. RDMT includes tools to assist and can provide examples on how to customize them to conform to particular site.

Note: See Chapter 9, "[Diagnostic and Monitoring Tools](#)."

Hospital Size and Growth

The Hospital tables, wherever they are, need to be monitored for size and growth. They have a huge effect on the performance of the entire RIB. As it gets larger, several interfaces dramatically slow down.

RDMT includes tools to assist and can provide examples on how to customize them to conform to particular site.

Note: See "[Hospital Scan Tools](#)."

RMS MFQ and RWMS UPLOAD Tables Sizes

The MFQ and Upload table size and growth need to be monitored. They can indicate a poorly performing (hung) adapter or forecast a slow interface because the Hospital tables are filling. In the case of some of the slower interfaces there will be slow down of dependency records being processed.

RDMT includes tools to assist and can provide examples on how to customize them to conform to particular site.

Note: See, "[Hospital Scan Tools](#)."

Remote RWMS

If the situation exists where a retailer is deploying instances of RWMS in different geographic locations connect by a WAN then there are several RIB deployment architectural alternatives that need to be considered and decided.

RIB Components Start and Stop

The RIB component must be started and stopped in particular order, and there are recommendations on when and how to do this and tools to assist in building out operational processes to suite a retailers site requirements.

It is always recommended that the order of startup be SUB, TAFR, PUB and the shutdown is in the reverse order. The RIB supplies tools to control the adapter start and stop process in the proper sequence in the rib-app-builder tool called rib-adapter-controller.

Note: See "[RIB App Builder Tools](#)."

RIB Operation Support Staff Requirements

The RIB application environment often presents a new dimension to a retailer's infrastructure, and there are training and support issues that do not fit the existing organization and current staff skill sets.

RIB Components - Source Code Control

The RIB contains code and configurations that are critical to the Enterprise. This version of the RIB is designed to be centrally managed and contains tools for tracking inventory and versions and configuration changes. A backup strategy also needs to be developed specific to the site.

Note: See Chapter 2, "[Application Builder](#)."

The RIB has an inventory tracking mechanism that is maintained by the tools in the RIB App Builder. These tools also manage the application of defects and tracking the defects applied in the inventory.

Note: See "[check-version-and-apply-defect-fix](#)."

RIB HA Requirements

The RIB is usually considered a HA requirement, so an architecture and operations plan to handle this needs to be developed.

Note: See the *Oracle Retail Integration Bus Installation Guide: The RIB and Oracle Database Cluster (RAC)*

Oracle® Application Server High Availability Guide 11g Release 1 (11.1.1)

Oracle® Database Administrator's Guide 11g Release 2

RIB Disaster Recovery

In addition to the HA requirements there is the issue of message retention, auditing and recovery. It is a common issue that an end-point application experiences an issue such as a crash, and requires recovery or rebuild. Syncing the data that the other applications have been publishing and subscribing to during the down time presents a major challenge.

It is important for a site to develop a plan and approach for this. In a large volume site, the JMS topics can build to huge numbers very quickly and over-run a system or the ability of the recovered system to catch-up in a time frame the business finds acceptable.

Note: See "[RIB Audit Logs](#)."

See Chapter 9, "[Diagnostic and Monitoring Tools](#)."

RIB Admin Roles and Security

The users and roles for the production environment need to be determined and put in place.

RIB Operation Support Staff Requirements

Regardless of the organization structure or where the staff reports to, there are two distinct sets of roles and capabilities needed; the RIB system administrator role and RIB application administrator role. The number of the persons filling those roles is dependent on the size of the deployment, breadth of the products being integrated, levels of customization and schedule compression.

Integration support is a team effort, with one or two strong RIB Admin people who can help work through difficult failure modes using the RIB logs to help isolate the issue and determine type. Users with knowledge of Oracle Retail application (such as RMS, RWMS, and SIM) must also have a good level of RIB understanding. As a team, they triage issues and then work them. By the Integration Test phase of an implementation, the types of RIB failure issues become more related to complicated data sets for business case tests. Gross level functionality issues are generally solved by then.

Production requirements are similar, but need to reflect the realities of pager rotation, 24x7 issues, as well as how many applications are deployed and over what geography.

RIB System Administrator

This section describes the RIB System Administrator role and responsibilities.

Technology Background

- UNIX (strong) - shell scripts and Unix tools
- Oracle Database and Stored Procedures
- Oracle Application Server (strong)
- Java EE (strong) - ability to read and understand exceptions and log files.
- Message Oriented Middleware (MOM) or communication technologies.

Experience or Training

- WebLogic Application Server
- RIB
- Java EE concepts
- JMS technology

Areas of Responsibility

- Installation or WLS and patches
- Configuration of WebLogic Application Server
- Installation and configuration of the RIB
- Support and configuration of Adapters and well as patches.
- Operational issues such as backup/restore, failure analysis using RIBLOGS and App Server logs as well as tools and various UNIX scripts and programs, and aid in the determination of error causes resulting in RIB Hospital entries.

RIB Application Administrator

This section describes the RIB Application Administrator role and responsibilities.

Technology Background

- UNIX - shell scripts, Unix tools
- Oracle Database and Stored Procedures
- Oracle Retail Applications - Strong (RMS, RWMS, RPM, and SIM)

Experience or Training on

- RIB
- Oracle Retail Applications
- JMS technology

Areas of Responsibility:

- Operational support and failure analysis using RIBLOGS and the RIB Hospital.

Hospital Monitoring and Maintenance

Under normal operations, messages go into the hospital, get retried and are automatically deleted from the hospital. But if there is a steady increase in hospitalized messages, the reasons should be immediately determined and worked.

Triage of messages placed in the RIB Hospital is a time consuming task. This is a tough task when only Oracle Retail applications are involved; adding other outside applications, as many retailers do, further complicates this process. Problems can be introduced at the application level, in the extract, or the transformation process.

Having the integration team take a first look at the messages is another common practice at Oracle Retail customer sites. This team's success at resolving and correcting data issues is dependant on their access to business analysts who understand the desired function.

The RIB Hospital tables need to be monitored for size and growth. The number of entries in the RIB Hospital has a large impact on the performance of the entire RIB. Each adapter checks the RIB Hospital for previous related failures for each message (to see if the message should be held until any previous errors have been resolved). As the RIB Hospital gets larger interfaces can dramatically slow down.

The RIB Hospital is a crucial component in the operation and performance of the RIB. Processes and procedures to handle it are very important, and should be decided on and practiced early. It is suggested that discussions and planning be started as soon as possible in the implementation phase to work through the possible scenarios and develop tools and procedures to handle them.

There are tools in RDMT that can be leveraged to not only build monitoring scripts but to aid in the initial triage of issues.

RIHA is the recommended tool for maintenance of the Hospital. It understands the Hospital table structure and how to appropriately correct, submit and, as needed, delete messages. The use of tools such as SQLDeveloper or TOAD is discouraged. Although they allow similar activities, they do NOT provide the safe guards of RIHA in maintaining the integrity of the tables and the JMS.

RIB Hospital tables are packaged with applications and therefore reside in the base schema of the applications. To reduce maintenance, upgrade and support concerns, users may choose to extract Hospital tables from application schemas.

Using the RIB Application Builder tool, error Hospital tables can be removed from the application space and placed under the control of the RIB kernel, where data sources meant for Hospital-related database operations are differentiated from application calls (such as GetNext and Consume). The data source, hosp-managed-datasource, supports the separation of the Hospital schema from the application schema.

To facilitate the externalization of the RIB Hospital tables from the application schema, two placeholders (one for PL/SQL applications and one for JavaEE applications) exist in the rib-deployment-env-info.xml file, as described in Chapter 3 of this guide.

Testing the RIB

The RIB is difficult to test as a stand-alone sub-system. It is part infrastructure and part application, and needs to have the integrating application end-points for even a simple installation.

To aid in the initial installation and evaluation of the RIB, a test harness has been developed and made available. The test harness is comprised of the these components:

- **plsql-api-stubs** - An API simulator of the PL/SQL API applications; RMS and RWMS.
- **javaee-api-stubs** - An API simulator of those applications exposing JavaEE APIs; SIM, RPM, and AIP.
- **RDMT** - The RIB Diagnostic and Monitoring Tool kit is a collection of command line tools, written in UNIX shell script along with supporting Java classes packaged in jar files.
- **Sample XML files** - These samples conform to the message payloads (XSDs).
- **Message auditing** - This is a feature of RIB 13 that allows the end-to-end auditing of a message as it passes through all of the RIB components.

Initially installing and deploying the RIB requires connecting to the Oracle Retail applications to verify that messages could flow end to end. The RIB installation requires that end points exist and respond. To test it the end points must be configured to publish or subscribe.

This test harness is completely independent of the applications, but uses the same RIB artifacts (payloads and Oracle Objects) as the actual applications. Additional tools and artifacts support the construction of test messages and the publication of these test messages.

Note: See "[RIB Test Harness](#)."

See Chapter 9, "[Diagnostic and Monitoring Tools](#)."

See "[RIB Logging](#)."

RIB Test Harness

The ability to initially install and deploy the RIB has always been difficult because of the need to connect to the Oracle Retail applications to verify that messages could flow end-to-end. The RIB installation requires that end-points exist and respond, and to test it requires that the end-points are configured to publish or subscribe.

The dependency on the application end-points can be not only a scheduling issue, but to produce messages for can require data seeding and coordination with the individual application teams.

RIB has several tools, including application API simulators that combine to provide a test harness that allows the installation, configuration, and testing of the RIB. These were developed to address the requirement for the full application to be present to validate a RIB installation as well as a providing a tool for integration and system tests.

This test harness is completely independent of the applications, but uses the same RIB artifacts (payloads and Oracle objects) as the actual applications. Additional tools and artifacts support the construction of test messages and the publication of these test messages.

Master Checklist

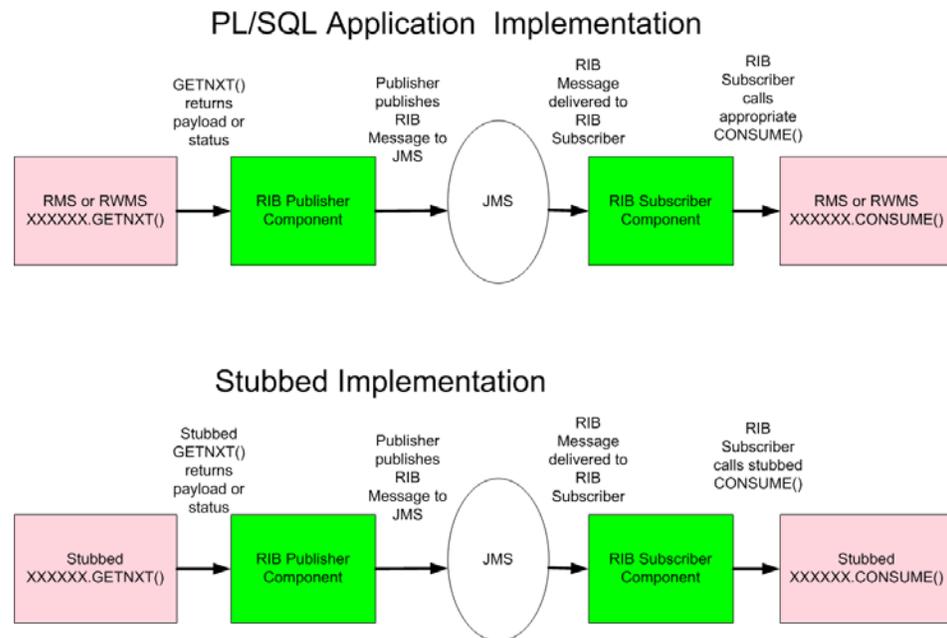
This check list covers all of the sequential steps required to create a stand-alone RIB Test Harness.

Task	Notes
Create the rib-home	Follow the guidelines in the <i>Oracle Retail Integration Bus Installation Guide</i> and the <i>Oracle Retail Integration Bus Implementation Guide</i> for prerequisites. Do not invoke the installer yet.
Install the javaee-api-stubs and plsql-api-stubs into the rib-home/tools-home.	Follow the instruction in the tools section.
Install the pl/sql api stubs	Follow the instruction in the tools section. The plsql-api-stubs can simulate both RMS and RWMS from the same user, but if it is desired to test full flow including hospital, then install to two users.
For the PL/SQL app subs install a set of Hospital Tables in the same user account.	See the <i>Oracle Retail Integration Bus Installation Guide</i> . See note about two stubs.
Deploy the javaee-api-stubs.	Follow the instruction in the tools section.
Install the RIB using the stubs as application end-points.	See the <i>Oracle Retail Integration Bus Installation Guide</i> .

PL/SQL Application API Stubs

The `plsql-api-stubs` is an API simulator designed to act in the same manner as when the RIB is connected to the actual application, but at the same time, have means to process specific status and other parameters from a "stubbed" application. This set of tools is designed to emulate those applications exposing PL/SQL API's to the RIB: RMS and RWMS.

Architecture and Design



The tool set contains three main subsystems

- A common set of PL/SQL packages, stored procedures and database tables. These are used by the other subsystems.
- A thin API-specific set of packages and stored procedures that the RIB directly interfaces with. These interfaces map calls to the common subsystem to output parameters or statuses.
- The Stub Admin and Setup Application. A set of simple application function and a character based menu that allow installation and set up of specific behaviors for a specific API.

The Common Subsystem

The purpose of the common subsystem is to provide a standard means of implementing specific behavior by an API. The stubbed APIs simulate a real application by using the common subsystem which will be loaded during the installation through JDBC calls to the database. It is comprised of a group of tables, sequences and other database objects created for each stubbed API.

There is a set of tables and sequences created for each GETNXT procedure. These tables are generated with the OUT and IN/OUT parameters of the GETNXT procedure as the fields. The user is prompted to enter data into these tables when he is trying to test for a particular API.

For example:

If there is a GETNXT procedure in a package called RMSMFM_ORDER then the common subsystem for this procedure would be a table RMSMFM_ORDER_GE_TBL and sequence called RMSMFM_ORDER_GE_SEQ created in the data base.

For each PUB_RETRY Procedure in the API a set of tables and sequences are created the same as GETNXT except that the names of tables and sequences have PU instead of GE

For a CONSUME API there is a table called RIB_CONSUME created with the O_STATUS_CODE, O_ERROR_MESSAGE and EXCEPTION_TO_THROW as the fields. If the user needs the CONSUME to throw a specific type of exception then the exception can be uploaded into the RIB_CONSUME table, so when the consume procedure is execute it will throw the specified exception type.

The Thin API layer

The API subsystem consists of packages and stored procedures that have the exact same signature as those found within the real application. This layer queries the appropriate common subsystem tables, sequences and other database objects to get the appropriate out parameters. These are then mapped to the API specific parameters of the stubbed application API.

The implementation of the stubbed API is written as java classes and loaded into the database during installation. The PL/SQL stubbed APIs are implemented in a way that these API internally call the java functions present in the classes then the PL/SQL OUT parameters are mapped with the java return types.

So when the RIB calls the GETNXT stubbed API as it normally calls the GETNXT API of a real application, the stubbed API internally calls the java class which uses the common subsystem tables to get messages as a CLOBs, it then converts the CLOB to an Oracle Object and then maps it with the PL/SQL OUT parameters and returns.

The Stub Admin and Setup Functions

These are a set of simple application functions written in java and wrapped by shell scripts and a character based menu that allow installation and set up of specific behaviors for a specific API.

Shell Script	Description
stubbymenu.sh	Simple character based menu that calls the wrapper scripts.
install.sh	Wrapper script that calls the java classes to install the RIB Objects and stubby java classes dynamically created from the metadata into the database (see stubby.properties).
configure_api.sh	Wrapper script that calls the java classes to set up the behavior and messages of a given consume or getnxt API.
read_metadata.sh	Wrapper script to call a java utility that will read a PL/SQL application (RMS, RWMS) schema and create a metadata file as input to create the stubbed API's.

Configuration Files

/conf directory files

Configuration File	Description
stubby.properties	Primary configuration file. Contains database url info and the metadata scripts to load.
commons-logging.properties	Apache logging conf
simplelog.properties	Apache logging conf
SqlToJavaMapper.java	generated from the storedproceduremetadat.xml specified in the Stubby.properties file.
	Note: Do not edit.
StoredProcedureMetaData_ RWMS.xml	Note: Do not edit.
StoredProcedureMetaData_ RMS.xml	Note: Do not edit.

Installation and Setup

Prerequisite Tasks

Task	Notes
Select a location for the plsql-api-stubs to reside.	Recommended location is in the rib-app-builder/rib-home tree structure: rib-app-builder/rib-home/tools-home
Get the latest version of the plsql-api-stubs.	The plsql-api-stubs is packaged as a stand-alone tar.
Get the latest version of the rib-public-payload-database-object-types.	rib-public-payload-database-object-types.zip is packaged with the RibFuncArtifacts and should be extracted from there. If this installation is in rib-home then the objects will be located in the rib-home/download-home/rib-func-artifacts
Create a database user that will own the plsql-api-stubs schema and the objects.	The user requires no special permissions. CREATE USER "PLSQLSTUB" PROFILE "DEFAULT" IDENTIFIED BY " PLSQLSTUB " DEFAULT TABLESPACE "USERS" TEMPORARY TABLESPACE "TEMP"; GRANT "CONNECT" TO " PLSQLSTUB "; GRANT "RESOURCE" TO " PLSQLSTUB ";
This version requires a path to jdk1.5 for compiling java stored procedures.	Be prepared to specify the path when prompted.

Installation

Task	Notes
Extract the tar file. cd rib-app-builder/rib-home/tools-home tar xvf PlsqlApiStubs13.0.0ForAll13.x.xApps_eng_ga.tar	This will create the file folders and place the executables and config files.
Place the database objects file in the scripts subdirectory	In rib-home/tool-home there is a directory already. It is a placeholder and this will over write it.
Extract the rib-public-payload-database-object-types.zip into the scripts directory. unzip rib-public-payload-database-object-types.zip	
Edit /conf/stubby.properties to point to the database url and user/password (see prerequisites). vi stubby.properties	# Database details hostname=linux1 port=1521 sid=ora10g username=plsqlstub password=plsqlstub
Base Script File names	This is where the selection of either RMS or RWMS objects is made. There can be only one per installation.
Execute the installation using menu item or install.sh in the stubby base directory cd rib-app-builder/rib-home/tools-home/ plsql-api-stubs ./install.sh Or ./stubbymenu.sh Then select the menu item to install.	The installation performs these actions: Runs a cleanup that will remove any existing RIB related tables, sequences, packages and types installed in the configured user schema. Runs all the scripts files in the udt sub-directory. Runs a drop java utility to remove any existing classes in the configured user schema. Note -- The warnings generated by the drop java can be ignored. Runs the load java utility to load Java classes as objects in the configured user schema. All the RMS and RWMS packages are created in the configured user schema.
Install Hospital tables	See the <i>Oracle Retail Integration Bus Installation Guide</i> .
Enter the complete path for jdk1.5:	This version of stubby and the RDBMS requires jdk1.5 for compiling java stored procedures.

The installation is now complete, and the tool is ready to be used.

Configure_api

The next step in using the tool set is to configure the desired behavior of the APIs under test. Use of the tool requires that the user understand the APIs involved at enough detail to understand and answer several prompts during the configuration process. See *the Oracle Retail Integration Bus Integration Guide* and the operations guides for the RMS and RWMS applications.

Prerequisites

Task	Notes
Create a sub-directory for the test messages to configure the API to use. These can be any location on the same host where the tool user has permissions to read.	The RIB ships with sample xml files for each message family. These are packaged with RDMT and are located under the testmsg subdirectory in the rdmt directory. rib-public-payload-xml-samples.zip. These should be used as a basis for testing and modified to suit the test cases.
Understand and know which API and its type to configure. For example: API Type: GETNXT API Package name: RMSMFM_ITEMS Message Type: ITEMCRE	API Types supported: GETNXT CONSUME PUB_RETRY

Execute the `configure_api.sh` script or select the menu item and respond to the prompts.

Prompts during configuration of a GETNXT and PUB_RETRY.

Prompt	Notes
Status Code the GETNXT API should return: S for Success, H for hospital, N for no message, and E for exception	(Case Sensitive)
Enter Error Message to be returned [to be entered only for H or E status codes:	
Enter data for O_MESSAGE (Give the complete file name):	The complete file path of the message to uploaded
Enter Business Object ID to be returned (Optional):	
Do you want to enter Routing Information for the message? [Y/N]:N	
Enter the Thread Value for the message:	
Enter the number of times the message must be replicated.	

Prompts during configuration of a CONSUME.

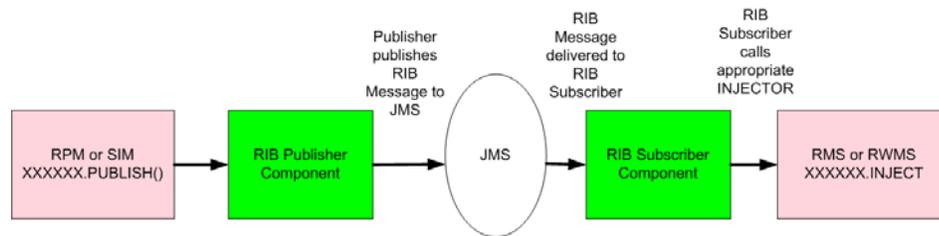
Prompt	Notes
Enter Status Code the Consume should return [S-Success]/[E-Error]	
Enter the Exception to be Thrown eg:nullpointerexception:	The Exception_To_Throw and Error Message with only be prompted if the status code is E.
Enter the Exception Message to be Thrown:	
Enter Message Type the Consume should return [CRE,MOD,DEI] eq:ITEMCRE:	

Java EE Application API Stubs

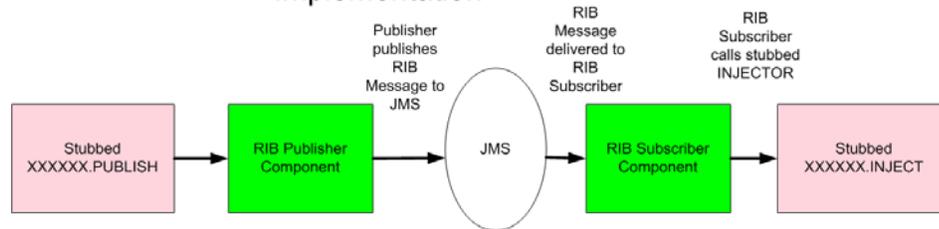
The javaee-api-stubs is an API simulator designed to acts in the same manner as when the RIB is connected to the actual application, but at the same time, have means to process specific status and other parameters from a stubbed application. This set of tools is designed to emulate those applications exposing Java EE APIs to the RIB: SIM, RPM, and AIP.

Architecture and Design

Java EE Application Implementation



Java EE Stubbed Implementation



Installation and Setup

Prerequisite Tasks

Task	Notes
Select a location for the javaee-api-stubs to reside.	Recommended location is in the rib-app-builder/rib-home tree structure: rib-app-builder/rib-home/tools-home/javaee-api-stubs
Get the latest version of the plsql-api-stubs.	The plsql-api-stubs is packaged as a stand-alone tar.
Create a database user that will own the javaee-api-stubs objects.	The user requires no special permissions. CREATE USER JAVAEEESTUB PROFILE DEFAULT IDENTIFIED BY STUBBY DEFAULT TABLESPACE USERS TEMPORARY TABLESPACE TEMP; GRANT CONNECT TO JAVAEEESTUB; GRANT RESOURCE TO JAVAEEESTUB;

Installation

Task	Notes
Determine the WebLogic instance to which to deploy the javaee-api-stubs.ear.	It is recommend but not required that an instance separate from the rib-<app> instance is used.
Using the WebLogic console, select the WebLogic instance and then deploy javaee-api-stubs-.ear.	See WebLogic deployment documentation for more details on how to deploy a Java EE application.

Task	Notes
<p>Using the WebLogic console, configure the database resources for the javaee-api-stubs JDBC resources.</p> <ul style="list-style-type: none">■ Log in to the WebLogic admin console■ Navigate to the Data Sources screen using Services > JDBC > Data Sources menu.■ Click New. Enter the following values in the respective fields. Name: javaee-api-stubs-non-xa-managed-datasource JNDI Name: jdbc/OracleRibDsNonXA Database Type: Oracle Database Driver: Oracle's Driver(Thin)■ Click Next. Uncheck Supports Global Transactions.■ Define connection properties for the database user in question.■ Verify the configuration by clicking Test Configuration.■ Do not proceed if the test fails. Ensure that the configuration is accurate.■ Select target as the server that would host javaee-stubby (for example, javaee-stubby-instance). Click Finish.	See WebLogic documentation for details.

Task	Notes
<p>Create one more data source named <code>javaee-api-stubs-xa-managed-datasource</code>. Navigate to the Data Sources screen using <code>Services > JDBC > Data Sources</code> menu.</p> <ul style="list-style-type: none"> Click New. Enter the following values in the respective fields. <p>Name: <code>javaee-api-stubs-non-xa-managed-datasource</code></p> <p>JNDI Name: <code>jdbc/OracleRibDs</code></p> <p>Database Type: Oracle</p> <p>Database Driver: Oracle's Driver(Thin)</p> Click Next for Transaction Properties. Define connection properties for the database user in question. Verify the configuration by clicking Test Configuration. Do not proceed if the test fails. Ensure that the configuration is accurate. Select target as the server that would host <code>javaee-stubby</code> (for example, <code>javaee-stubby-instance</code>). Click Finish. Verify that both data sources are listed on <code>Services > JDBC > Data Sources</code> screen. <p>Install Hospital tables</p>	<p>See the <i>Oracle Retail Integration Bus Installation Guide</i>.</p>

Configuration of the `rib-<app>` to use Injection Stubs

Task	Notes
<p>Decide which <code>rib-<app></code> to configure for.</p> <p>Using RIB App Builder or the RIB Installer configure and deploy the <code>rib-app</code> using the jndi information of the <code>javaee-api-stubs</code> in place of the app.</p>	<p>The stubbed implementation has been written to insert the payload to a database once inject has been called. <code>Injectors.xml</code> has been configured to include all the RPM,SIM subscribing families.</p> <pre><app id="sim" type="javaee-app"> <jndi> <url>t3://mspdv170.us.oracle.com:18022/javaee-api-stubs</url> <factory>weblogic.jndi.WLInitialContextFactory</factory> <user-alias>sim_jndi_ user-name-alias</user-alias> </jndi> </app></pre>

Performance Considerations

The chapter discusses the performance characteristics of the Oracle Retail RIB, the factors that affect it, and a process to test it.

Performance of the RIB within a customer site is critical to the performance of the business, and is determined by factors that are very specific to a given deployment. Because of this is, it is recommended by Oracle Retail that a Performance Test be part of every deployment plan. Even if formal testing is not planned, the use of the tools and processes discussed can measure the relative performance of the RIB sub-system and can be used to diagnose bottlenecks.

It is beyond the scope of this document to discuss all of the tools and techniques available at the host, network, database, and application server level.

See the *Oracle RIB Integration Bus Implementation Guide*.

Performance Factors

The performance of each of these components is influential in the overall performance of the system:

- The Application Server topology and configuration.
- The RIB deployment approach.
- The hardware sizing and configuration of the RIB hosts.
- The hardware sizing and configuration of the applications that are connected to the RIB.
- The hardware sizing and configuration of the JMS provider host.
- The hardware sizing and configuration of the RIB Hospitals hosts.

There are other factors that determine the performance of the overall system. Some of these factors in a RIB environment are:

- Number of channels configured
- Number of messages present in the topic
- Size of the message
- Database clustering
- Application Server topology
- Number of TAFRs in the processing of the message
- Message aggregation

Performance Requirements

Each of the RIB Message Family volume requirements is almost always described for the end-to-end message flow. This is defined as from the publication to the completion of subscription by all Oracle Retail applications. Retail businesses express volume requirements in terms of details per hour and per day.

Each Message Family has different volume requirements and any given family may have an intermediate component between the originating publication and the end subscriber. These components are called TAFRs (Message Transform, Filtering, and Routing). This is an important concept, since it means that in a given flow, a message published by the source system may be subscribed to and then re-published by a TAFR before it is subscribed to by the destination application. This is true in many of the Message Families.

Example Volume Requirements:

Family	Details Per Day	Details Per Hour
Purchase Orders	355,000	355,000
ASN Inbound	19,200,000	19,200,000
Appointments	240,000	30,000
PO Receipt	240,000	30,000
Store Receipt	4,000,000	2,000,000
Transfers	1,000,000	250,000
Stock Order Allocation	600,000	75,000
Stock Order Transfer	1,000,000	75,000
Stock Order Status	600,000	75,000
ASN Outbound (BOL)	285,000	285,000
Promotions	5,000,000	250,000
Item Locations	1,000,000	300,000
Items	100,000	20,000

Note: These examples are for illustration, but are representative of actual customer requirements.

Message construction is the same for all Oracle Retail Publishing applications and RIB adapters. There are configuration control points that allow flexibility in the size of the message. The application side has the ability to specify the number of details per message. The RIB has a setting that controls the aggregation of the messages (ribMessage) within the larger RibMessages envelope. There is a setting that controls the number of RibMessages published within a commit to the JMS.

See "[Message Aggregation](#)."

Multi-Channel

The RIB is designed to support parallel message handling to increase through-put via a mechanism call multi-channel. Multi-channel is a concept to logically partition the flow of messages within the JMS topic so that multiple publishers and subscribers can simultaneously use the same JMS topic without any contention or interference with each other and preserve publication message ordering within the logical channel.

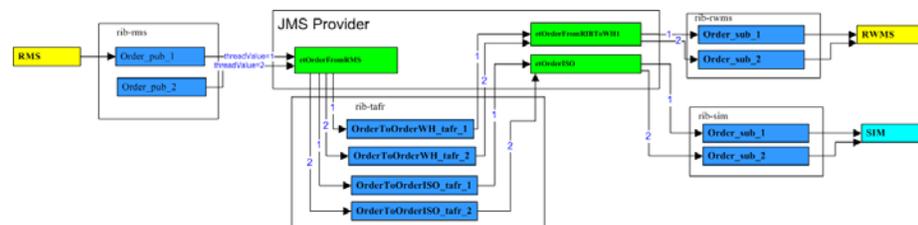
Every adapter instance of a publisher, subscriber, or TAFR, configured in the RIB is considered to belong to a logical channel for processing messages. By multi-channel adapters, we mean multiple adapter instances for the same message family, each processing messages asynchronously and in parallel. When multiple channels are used, they must be defined and configured across all publisher, subscriber, and TAFRs that participate in an end-to-end message flow to and from all Oracle Retail applications for that message family.

Each messaging RIB component involved in publishing or subscribing to a logical channel is distinctly identified by a JMS Message property known as "threadValue" with a specific value. This JMS message property and the value it contains define the logical channel.

JMS Message properties are user-defined additional properties that are included with the message. Message properties have types, and these types define application-specific information that message consumers can use to select the messages that interest them. Each RIB subscriber has the "threadValue" property and this value is part of its JMS Durable Subscriber selector and each RIB publisher sets the "threadValue" JMS message property to a specific value for each message it publishes.

Oracle Retail RIB components are capable of being multi-channelled by making configuration changes to the system. The base RIB configuration, as shipped GA, provides each Message Family with one channel where all components set or look for "threadValue" of 1 (one). The naming convention and the RIB kernel code identify the RIB adapters by adding the logical channel to the end of the adapter class name.

The diagram below demonstrates the multi-channeling of the Purchase Order flow to two channels.



End-to-End Timing

As mentioned, the RIB's performance is judged by the average time a Message Family detail takes to flow from a publisher to consumption by all active subscribers. This is not a straight forward measurement.

Message through-put is not a calculation of the sum of the individual message times. Although the average time per message will remain fixed, messages are processed in parallel. So the total time to process n messages on a single channel will not be the serial sum of the individual messages.

Additionally, it is possible to configure multiple logical channels to increase overall through-put.

How to Calculate Average Message Size

It is important to understand the average messages in an integration flow. In the RIB where interfaces are separated into messages families with differing payloads per type of message, these calculations can be difficult. This section outlines an approach for arriving at averages using the sample XML files that ship with the RIB.

The RIB delivers sample files generated for each Message Family that contains all elements for a message type and payload *with all possible characters for each element*.

Note: Several Families have variable types of "details" per header so a close investigation is required to understand what the relationship is and what a representative message can be.

In practice, of course, this size will vary depending on the number of characters that a description element may contain, but for performance testing calculations, this is a reasonable start for calculations.

Note: An alternative is to use the audit feature of the RIB. These messages can then be used to estimate the average sizes.

The RIB message envelope, called a RibMessages contains a variable number of ribMessage nodes. Within a ribMessage node is a Message Family payload. A minimum payload for this exercise is defined as one header and a variable number of details.

The general process to determine the size in bytes of a Message Family message per detail using the RIB sample xml messages and xsds is as follows:

- Determine the RIB envelope size (RibMessages elements + ribMessage elements).
- Determine the size of a single header.
- Determine the size of a detail.

The RIB has a standard message envelope (RIBMessages.xsd) that can be easily calculated exclusive of the message family payload.

ribMessage header elements no payload	823 Bytes
RibMessages header elements	324 Bytes
RibMessages with 1 rM no payload	1147 Bytes

Each message family is comprised of message type and an associate payload (e.g. POCre uses PODesc.xsd). These relationships are defined in the RIB Integration Guide.

The sample XML messages for each release are packaged in the functional artifacts war file and with RDMT in the rib-home/tools-home/rdmt/testmsgs directory.

Select the message payload file and look at the byte count. This will always be 1 header and 1 detail(s). Be aware that this relationship will vary by family and can be complex for some message types (e.g. ItemCre and ItemDesc.xsd) where optional details can be present.

Select the payload file and remove all detail nodes and look at the byte count. This will be the standard header. Use the same procedure for the detail(s). This will be the detail size.

Example Message - PODesc	
Header Size (PODesc no detail)	9413
1 Detail Size (PODt1)	1943

The next step is to determine the average number of details per message. This will vary based on the business needs and the RIB configuration that is selected.

See "[Message Aggregation](#)."

Using the desired number of details per message, this calculation is the result:

Total 1 RibMessages + 1 ribMessage + 1 Header + 1 Detail	Avg Message Size
--	------------------

For example:

Total 1 RibMessages + 1 ribMessage + 1 PODesc + 1 PODt1	12,053
---	--------

Purchase Order Example

ribMessage header elements no payload	823
RibMessages header elements	324
RibMessages with 1 rM no payload	1147
Message - PODesc	
Header Size (PODesc no detail)	9413
1 Detail Size (PODt1)	1943
Total 1 RibMessages + 1 ribMessage + 1 PODesc + 1 PODt1	12,053

The following is an example using the default settings.

The RIB messages created by the Order publishing adapter (details per message):

- Contains a maximum of 20 ribMessages per RibMessage.
- Has 20 details per PODesc payload in a ribMessage.

For a 400 Details PO Message the calculation is:

RibMessage = 1 RibMessages header + 20 ribMessage headers + 20 PODesc + 400 PODtls

# ribMessage nodes	20
# Details	400
RibMessages Header (1)	324
ribMessage Header (20)	16,460
PODesc (20)	188,260
PODtl (400)	777,200
Total Bytes/Msg	982,644

Using the example volume requirement for the Purchase Orders, and using the same RIB message configuration settings:

Details per hour requirement (Total Through-put)	355,000
Details per Message	400
Total messages per hour (355,000/400)	887
Message/sec required (982 KB each - 60*60/887)	4.058

So:

End-to-End — 1 message with 400 details can take a max of 4.058 seconds.

End-to-End — 982,644 Bytes can take a max of 4.058 seconds (which in this example is 400 details).

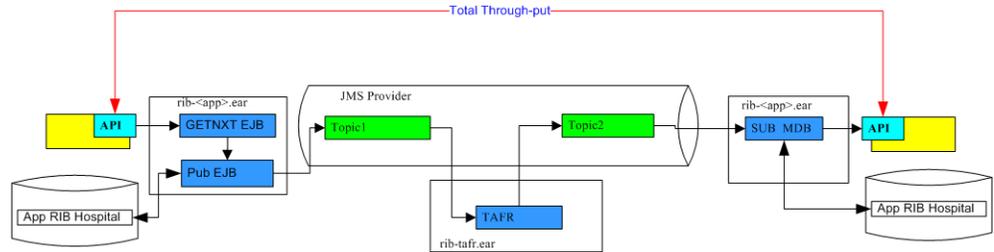
So:

982644 Bytes/4.058 sec = 242149.83 bytes/sec = 0.2421498 MB/sec Total end-to-end throughput to meet the Purchase Order example requirements.

Understand the Message Family

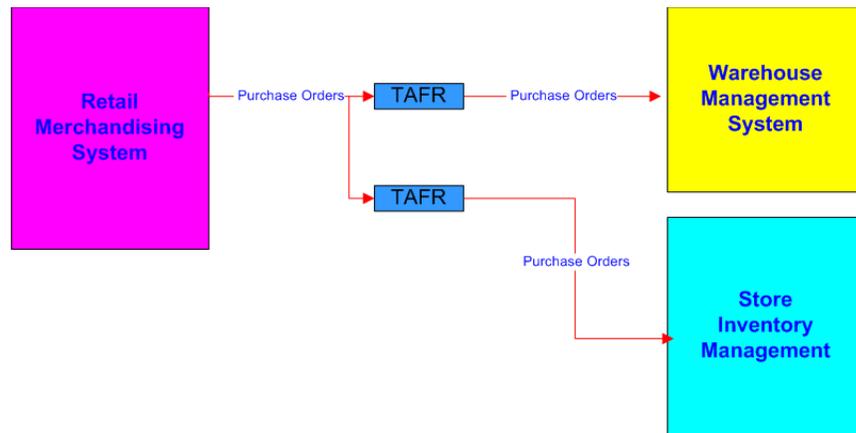
These are end-to-end processing time requirements across the entire message flow from Publisher to Subscription completion.

The following diagram is a generic message flow.

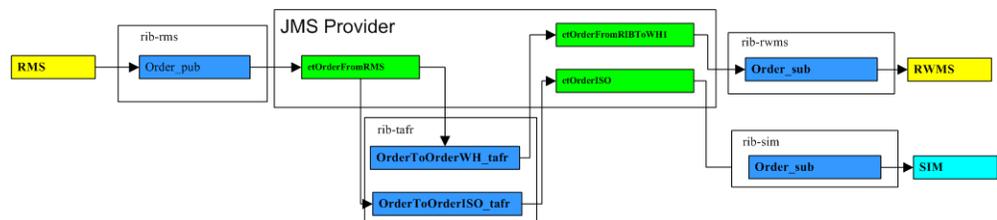


To continue the Purchase Order example, the requirements and timings have to be broken down further. The Purchase Order flow has a TAFR as well as multiple subscribers. For purposes of this example, consider the Subscribers Consume times as equivalent. As the diagram depicts, for a flow like the Purchase Orders, there are multiple components and for a single message to flow there will be, at a minimum, a message published twice and subscribed twice, as well as a marshalling and un-marshalling of the message twice (Family dependent). There will be at least one, and possibly two, Hospital Dependency checks as well.

The following diagram is a logical view of the Oracle Retail Purchase Order flow.



The following diagram is a functional, detailed view of the Oracle Retail Purchase Order flow.



RIB Timing Log Analysis

The performance of the RIB is a complicated subsystem to measure and involves not only host level performance, but database, network, and application server subsystems performance. To make measurement of the RIB components timing characteristics available for analysis, the RIB kernel code has been instrumented to log events as it processes events. The logging of these events is through log4j. The timings are logged per adapter. Once the timings are enabled the events are logging continuously to the file. The RIB RDMT supplies a post-processing tool to take the timing file and produce summary reports.

This table lists the currently pre-defined times that are tracked in the RIB Timings logs. The description is the definition of interval calculation.

	Timing Type	Description
T1	PUB_B4_GETNXT_CALL	Time interval between start of the publisher and the actual GETNXT call.
T2	PUB_TIME_IN_GETNXT_CALL	Time taken by the GETNXT call to the plsql app.
T3	PUB_TIME_IN_EJB_PUBLISH_CALL	Time taken for the publish call in the EJB, includes RIB overhead surrounding the actual publish to the JMS.
T4	PUB_TOTAL_PUBLISH_TIME	Time taken for the complete PUB process = GETNXT + hospital dependency + publish + commit.
T5	PUB_TIME_IN_REAL_JMS_PUBLISH	Time taken to publish a message to the AQ JMS.
T6	SUB_TIME_IN_CONSUME_CALL	Time taken by the CONSUME call to the plsql app.
T7	SUB_TOTAL_SUBSCRIBE_TIME	Time taken for the complete SUB process = CONSUME/INJECT + hospital dependency + subscribe + commit.
T8	SUB_TIME_IN_EJB_SUBSCRIBE_CALL	Time taken for the subscribe call in the EJB, includes RIB overhead surrounding the actual subscribe.
T9	SUB_TIME_IN_INJECT_CALL	Time taken by the INJECT call to the java app.
T10	TAFR_TOTAL_MSGPROCESS_TIME	Time taken in the complete message tafring Process = TAFRing + hospital dependency + publish + RIB overhead.
T11	TAFR_TIME_IN_EJB_CALL	Time taken for the TAFR call in the EJB, includes RIB overhead surrounding the actual TAFRing.
T12	TAFR_TIME_IN_REAL_JMS_PUBLISH_EJB	Time taken by the TAFR to publish a message to the AQ JMS.

Purchase Order Example

Note: The following are examples illustrate the process and concepts, but not test results.

Order_pub_1 (Publisher)

TIMING_TYPE	COUNT	AVERAGE	TIME_SUM	MIN_TIME	MAX_TIME
PUB_B4_GETNXT_CALL	100	0.03787	3.7904	0.036	0.07
PUB_TIME_IN_GETNXT_CALL	100	0.06546	6.5528	0.061	0.254
PUB_TIME_IN_EJB_PUBLISH_CALL	100	0.04192	4.1961	0.039	0.308
PUB_TOTAL_PUBLISH_TIME	100	0.19675	19.6947	0.186	2.738
PUB_TIME_IN_REAL_JMS_PUBLISH_EJB	100	0.02931	2.9341	0.027	0.292

OrderToOrderTafr_1 (TAFR)

TIMING_TYPE	COUNT	AVERAGE	TIME_SUM	MIN_TIME	MAX_TIME
TAFR_TOTAL_MSGPROCESS_TIME	100	1.58708	158.708	1.296	4.135
TAFR_TIME_IN_EJB_CALL	100	1.51371	151.371	1.23	3.24
TAFR_TIME_IN_REAL_JMS_PUBLISH_EJB	100	1.1802	118.02	0.914	2.414

Order_sub_1

TIMING_TYPE	COUNT	AVERAGE	TIME_SUM	MIN_TIME	MAX_TIME
SUB_TIME_IN_CONSUME_CALL	100	1.359	135.9	0.671	2.203
SUB_TOTAL_SUBSCRIBE_TIME	100	1.93943	193.943	0.718	5.593
SUB_TIME_IN_EJB_SUBSCRIBE_CALL	100	1.92386	192.386	0.687	5.593

In this example, to describe the serial processing through-put time to Publish 100 messages through the TAFR to Subscriber Consume:

Publisher (19.69 Sec) + TAFR (158.708 sec) + Subscriber (193.943 sec) = 372.341 seconds
= Average 3.72 msg/sec

It is important to understand that the actual message through-put is not a calculation of the sum of the individual message times. Although the average time per message will remain fixed, messages are processed in parallel. So the total time to process n messages on a single channel will not be the serial sum of the individual messages.

Note: This is an illustration. The number of message needed to arrive at a calculation of through-put require much higher counts, a broad spectrum of time, and system load. Other factors include average size of message.

Key Interfaces to Consider

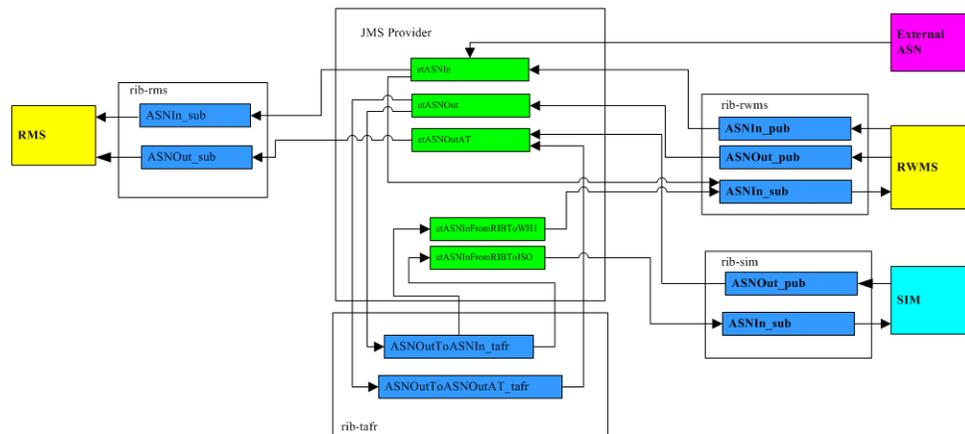
Every customer site has unique requirements and flows, so the ones to focus on will vary. However, there are ones that always make to the list.

- ASN
- Receipts (PO and Store)
- Promotions
- Stock Order (Allocation & Transfers)
- Item Locations
- Items

It is strongly recommended that during the deployment planning phase, the business requirements for these and others be gathered and analyzed. Some form of performance testing should be planned, even if only a characterization by measuring the actual flows during other test phases (for example, Integration Test).

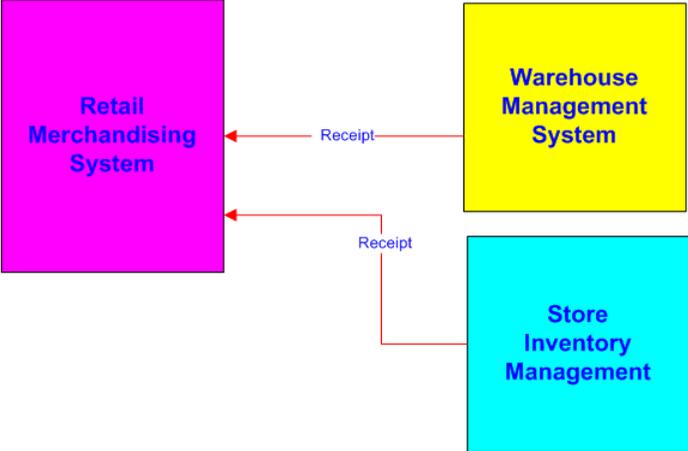
ASN (Inbound/Outbound)

The following diagram is a functional, detailed view of the Oracle Retail ASNIn/ASNOut Flows.

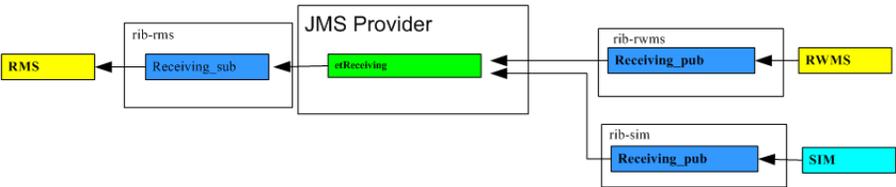


Receipts

The following diagram is logical view of the Oracle Retail Receipts Flow.



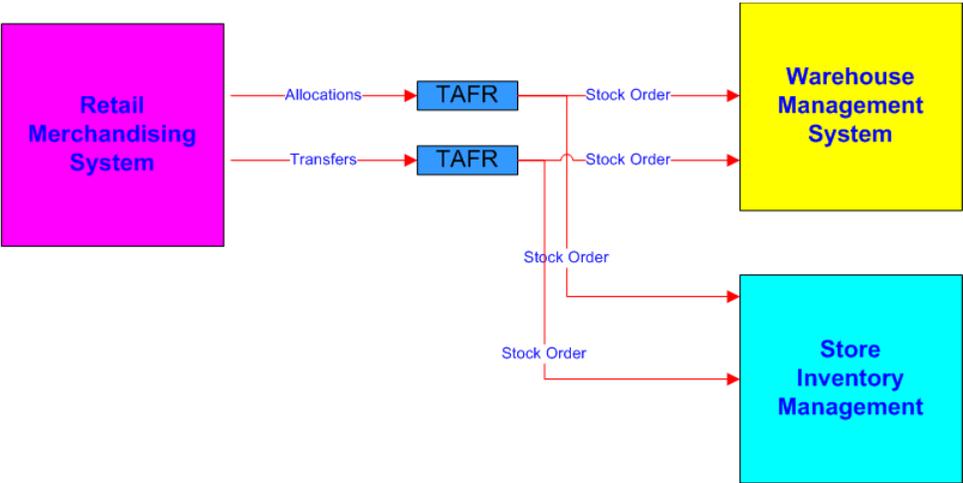
The following diagram is functional, detail view of the Oracle Retail Receipts Flow.



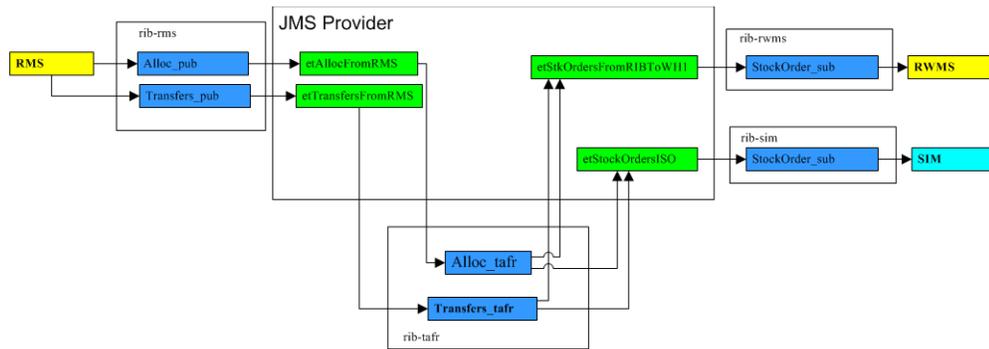
The Receipts Message Family is transactional data, and often a candidate for performance testing. Receiving consists of appointment and receipt messages that are published to the RIB for RMS providing open to buy visibility. An appointment is information about the arrival of merchandise at a location. A receipt message informs RMS when merchandise arrives in a warehouse or store system.

Stock Order (Allocations & Transfers)

The following diagram is a logical view of the Oracle Retail Stock Order Flow.



The following diagram is a functional, detail view of the Oracle Retail Stock Order Flow.



How to Approach a RIB Performance Test

There are two distinct approaches to measuring RIB performance; using actual application end-points or using the RIB API simulators. Both are useful at different phases of deployment.

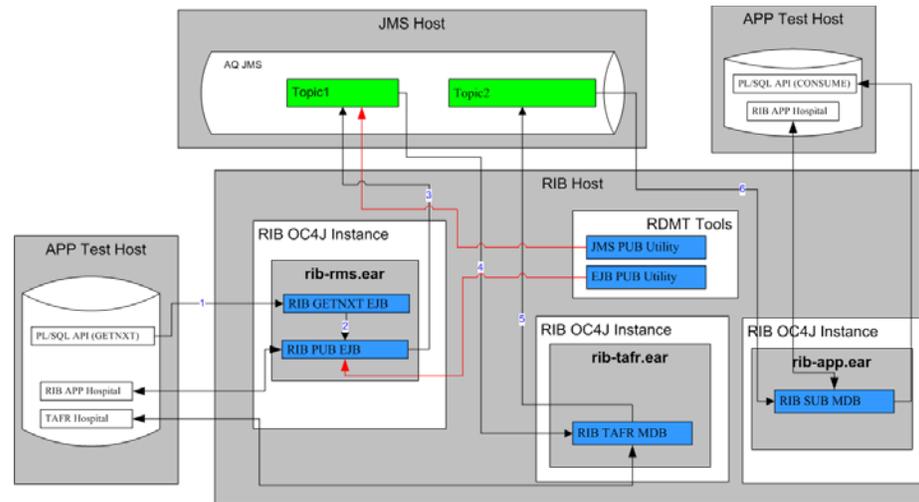
Keep in mind, that performance measuring is possible at any time in any phase, performance testing is more formal and requires planning, dedicated people and systems and test data. Building test data is difficult. Do not underestimate the complexity and this time consuming aspect of testing. To do testing with the applications involved, all of the data has to be consumable without errors.

There are tools available in RDMT to assist in this, as well as the audit feature of the RIB. By enabling audit on an interface all messages are saved to a file in a form that can be played back by RDMT utilities.

The API Simulators (PL/SQL and Java EE) allow the focus to be on the RIB infrastructure and is possible without resources outside of the RIB team. The value is limited to profiling the deployment architecture independent of the application API behavior and is much simpler in terms of data generation.

The performance measures of the end-to-end flow using the application's API is the only way to match performance against requirements since the majority of the time spent in the flow is in the application API. Customers do not distinguish a separation between the RIB components and the application APIs.

The following illustrates the RIB Performance Test Harness.



The RIB supplied tools to support both forms of tests are the RIB Test Harness, the API simulators, and the RDMT tools; timing utility, JMS Publish and EJB Publish.

This is a general process on how to measure the flow end-to-end.

1. Prepare for the run.
 - Use RIB Admin GUI to stop all adapters (PUB, SUB, TAFR).
 - Use RDMT to archive all logs so that the run has clean logs.
 - Use RIB Admin GUI to enable timings logs (DEBUG) on all adapters.
 - Use RIB Admin GUI to set all other adapter logs to INFO.
2. Determine how to generate the messages.
 - Using the Oracle Retail Application (e.g. RMS to generate some orders).
 - Using RDMT EJB Publish (will use a portion of the PUB Adapter).
 - Using RDMT JMS Publish (will not use the PUB Adapter).
3. Start the appropriate adapters depending on the above decision.
 - Use RIB Admin GUI to start adapters (PUB, SUB, TAFR).
4. Generate the test messages.
5. Stop the adapters.
6. Analyze the data.
 - Use RDMT to run the Timing Analysis Utility on each adapter timing log. This will create a .csv file.
 - Upload the .csv files for display and further analysis using a tool such as Excel.

Multi-Channel Adapters

A channel is a solution approach to maintaining the previous RIB release concept of a Logical Channel.

Multi-channel applies to the logical partitioning of the flow of messages within the JMS topic. Multiple publishers and subscribers can simultaneously use the same JMS topic without any contention or interference, thus preserving publication message ordering within the logical channel.

Every adapter instance of a publisher, subscriber, or TAFR configured in the RIB belongs to a logical channel for processing messages. Multi-channel adapters are multiple adapter instances for the same message family, each processing messages asynchronously and in parallel.

There are critical rules of behavior that must be observed and enforced to maintain the two primary RIB functional requirements of once-and-only-once successful delivery and guaranteed sequencing of messages within a message family.

To ensure that these rules are followed--and to simplify RIB configuration tasks that support a multi-channel message flow--the process has been integrated into the RIB App Builder tools.

Multiple channels must be defined and configured across all publisher, subscriber, and TAFRs that participate in an end-to-end message flow, to and from all Oracle Retail applications, for that message family. The RIB App Builder tools have checks and verification logic to prevent deployment of incomplete flows.

Use of multi-channels can increase performance, but it does not help in every situation. There is overhead and complexity associated with implementing multiple channels so they should not be considered unless a defined and performance problem exists.

Adding Multi-Channels to a Message Family

The process of adding multi-channels to a message family should be part of a performance test and tuning process. Multi-channeling capability for a message family is limited by the multi-channel support in the publishing performed by applications.

It is a known issue that currently none of the publishing APIs of RWMS do not support multi-channeling. Therefore, integration flows involving RWMS publishing cannot be multi-channelled.

For example, the Inventory Adjustment (InvAdjust) message family is published by RWMS and subscribed to by RMS. Because RWMS supports only single-channel publishing, RMS must be set up for single-channel processing for the InvAdjust message family. All RWMS subscription APIs support multi-channel processing.

The following RMS 13.2 publishing APIs support multi-channel processing:

- Allocations Publication API
- Item Location Publication API
- Item Publication API
- Merchandise Hierarchy Publishing API
- Order Publication API
- Receiver Unit Adjustment Publication API
- RTV Request Publication API

- Seed Object Publication API
- ASNOUT Publication API
- Transfers Publication API
- Work Orders in Publication API
- Work Orders out Publication API

The following RMS 13.2 publishing APIs do not support multi-channel processing:

- Banner Publication API
- Differentiator Groups Publication API
- Differentiator ID Publication API
- Partner Publication API
- Seed Data Publication API
- Store Publication API
- Vendor Publication API
- UDA Publication API
- Warehouse Publication API

Logical Channels and threadValue

Each messaging RIB component involved in publishing or subscribing to a logical channel is distinctly identified by a JMS Message property known as "threadValue" with a specific value. This JMS message property and the value it contains define the logical channel.

JMS Message properties are user-defined additional properties that are included with the message. Message properties have types, and these types define application-specific information that message consumers can use to select the messages that interest them.

So each RIB subscriber has the "threadValue" property and this value as part of its JMS Durable Subscriber selector and each RIB publisher sets the "threadValue" JMS message property to a specific value for each message it publishes.

Oracle Retail RIB components are capable of being multi-channeled by making configuration changes to the system. The base RIB configuration, as shipped GA, provides each Message Family with one channel where all components set or look for "threadValue" of 1 (one). The naming convention and the RIB kernel code identify the RIB adapters by adding the logical channel to the end of the adapter class name.

Algorithm Used to Calculate Channel

Channels are calculated based on Business object ID(BOID) found in the RibMessages <id> tag. The algorithm used to calculate is as follows.

```
MOD(MD5(family + ":" + businessObjectId)%maxChannelNumber) + 1
```

- First the algorithm calculates the message digest of the string family+":"+businessObjectId which produces a unique number.
- Then this number is divided by the maxChannelNumber, which is calculated by the number of configured channels for that message family.
- A 1 is added to the result so that the channel number is always greater than 0.

For example:

```
Family = Alloc  
BusinessObjectID (BOID) = 10202123  
MaxChannelNumber = 7 (Total number of channels configured for the Alloc family)  
Then the channel number for the BOID is calculated as  
sMOD(MD5(Alloc + ":" + 10202123)%7) + 1 = 4  
which means that all the messages that have BusinessObjectID of 10202123  
are ALWAYS sent through channel 4 (Alloc_pub_4).
```

Note: The channels have to be configured throughout the integration flow using the rib-app builder tool.

Example of a Message Family Flow with a TAFR:

```
Alloc_pub_1  
Alloc_tafr_1  
StockOrder_sub_1
```

How to Configure a Multi-Channel Flow

Generalized Process

1. Determine the Family to multi-channel
2. Examine the rib-integration-flows.xml to identify all participants in the full flow.
3. In the rib-home modify the appropriate configuration files for each of the rib-<apps>.
 - a. rib-<app>-adapters.xml
 - b. rib-<app>-adapter-resources.properties
4. For PL/SQL Application edit the RIB_SETTINGS table.
5. Compile and deploy.

Example

This example is to configure the Alloc message flow with five channels. Alloc is a complex flow in that it has multiple Oracle Retail application subscribers and a TAFR that transforms the messages from one family to another; Alloc to StockOrder.

Backup the following files.

- "rib-home/application-assembly-home/rib-rms/rib-rms-adapters.xml
- rib-home/application-assembly-home/rib-rms/rib-rms-resources.properties.

The following is the message flow for the Alloc Family from rib-integration-flows.xml that this example uses.

```
<message-flow id="1">
  <node id="rib-rms.Alloc_pub" app-name="rib-rms" adapter-class-def="Alloc_pub"
    type="DbToJms">
    <in-db>default</in-db>
    <out-topic>etAllocFromRMS</out-topic>
  </node>
  <node id="rib-tafr.Alloc_tafr" app-name="rib-tafr" adapter-class-def="Alloc_tafr"
    type="JmsToJms">
    <in-topic>etAllocFromRMS</in-topic>
    <out-topic name="topic-name-key-iso">etStockOrdersISO</out-topic>
    <out-topic name="topic-name-key-wh">etStkOrdersFromRIBToWH{*}</out-topic>
  </node>
  <node id="rib-sim.StockOrder_sub" app-name="rib-sim"
    adapter-class-def="StockOrder_sub" type="JmsToDb">
    <in-topic>etStockOrdersISO</in-topic>
    <out-db>default</out-db>
  </node>
  <node id="rib-rwms.StockOrder_sub" app-name="rib-rwms"
    adapter-class-def="StockOrder_sub" type="JmsToDb">
    <in-topic>etStkOrdersFromRIBToWH1</in-topic>
    <out-db>default</out-db>
  </node>
</message-flow>
```

RIB-RMS

1. Modify rib-rms-adapters.xml to add multiple channels.

Following is a portion of rib-rms-adapters.xml

```
<publishers>
  <timer-driven id="Alloc_pub_1" initialState="running"
    timeDelay="10">
    <timer-task>
      <class
        name="com.retek.rib.app.getnext.impl.GetNextTimerTaskImpl"/>
      <property name="maxChannelNumber" value="5" />
    </timer-task>
  </timer-driven>
  <timer-driven id="Alloc_pub_2" initialState="running"
    timeDelay="10">
    <timer-task>
      <class
        name="com.retek.rib.app.getnext.impl.GetNextTimerTaskImpl"/>
      <property name="maxChannelNumber" value="5" />
    </timer-task>
  </timer-driven>
  <timer-driven id="Alloc_pub_3" initialState="running"
```

```

timeDelay="10">
    <timer-task>
        <class
name="com.retek.rib.app.getnext.impl.GetNextTimerTaskImpl"/>
            <property name="maxChannelNumber" value="5" />
        </timer-task>
    </timer-driven>
    <timer-driven id="Alloc_pub_4" initialState="running"
timeDelay="10">
        <timer-task>
            <class
name="com.retek.rib.app.getnext.impl.GetNextTimerTaskImpl"/>
                <property name="maxChannelNumber" value="5" />
            </timer-task>
        </timer-driven>
        <timer-driven id="Alloc_pub_5" initialState="running"
timeDelay="10">
            <timer-task>
                <class
name="com.retek.rib.app.getnext.impl.GetNextTimerTaskImpl"/>
                    <property name="maxChannelNumber" value="5" />
                </timer-task>
            </timer-driven>

```

2. Modify rib-rms-adapter-resources.properties.

```

Alloc_pub_1.name=Alloc Publisher, channel 1
Alloc_pub_1.desc=Publisher for the Alloc family through channel 1.

Alloc_pub_2.name=Alloc Publisher, channel 2
Alloc_pub_2.desc=Publisher for the Alloc family through channel 2.

Alloc_pub_3.name=Alloc Publisher, channel 3
Alloc_pub_3.desc=Publisher for the Alloc family through channel 3.

Alloc_pub_4.name=Alloc Publisher, channel 4
Alloc_pub_4.desc=Publisher for the Alloc family through channel 4.

Alloc_pub_5.name=Alloc Publisher, channel 5
Alloc_pub_5.desc=Publisher for the Alloc family through channel 5.

```

RIB-TAFR

1. Modify rib-tafr--adapters.xml to add channels for a family.

```

<tafrs>
    <message-driven id="Alloc_tafr_1" initialState="running"
tafr-business-impl="com.retek.rib.domain.tafr.bo.impl.AllocToStockOrderFromRibB
OImpl" />
        <message-driven id="Alloc_tafr_2" initialState="running"
tafr-business-impl="com.retek.rib.domain.tafr.bo.impl.AllocToStockOrderFromRibB
OImpl" />
            <message-driven id="Alloc_tafr_3" initialState="running"
tafr-business-impl="com.retek.rib.domain.tafr.bo.impl.AllocToStockOrderFromRibB
OImpl" />
                <message-driven id="Alloc_tafr_4" initialState="running"
tafr-business-impl="com.retek.rib.domain.tafr.bo.impl.AllocToStockOrderFromRibB
OImpl" />
                    <message-driven id="Alloc_tafr_5" initialState="running"
tafr-business-impl="com.retek.rib.domain.tafr.bo.impl.AllocToStockOrderFromRibB
OImpl" />

```

2. Modify rib-tafr-adapters-resources.properties.

```
Alloc_tufr_1.name=AllocToStockOrder TAFR, channel 1
Alloc_tufr_1.desc=TAFR for converting Allocation messages to StockOrders and
routing them to the correct warehouse or store system
```

```
Alloc_tufr_2.name=AllocToStockOrder TAFR, channel 2
Alloc_tufr_2.desc=TAFR for converting Allocation messages to StockOrders and
routing them to the correct warehouse or store system
```

```
Alloc_tufr_3.name=AllocToStockOrder TAFR, channel 3
Alloc_tufr_3.desc=TAFR for converting Allocation messages to StockOrders and
routing them to the correct warehouse or store system
```

```
Alloc_tufr_4.name=AllocToStockOrder TAFR, channel 4
Alloc_tufr_4.desc=TAFR for converting Allocation messages to StockOrders and
routing them to the correct warehouse or store system
```

```
Alloc_tufr_5.name=AllocToStockOrder TAFR, channel 5
Alloc_tufr_5.desc=TAFR for converting Allocation messages to StockOrders and
routing them to the correct warehouse or store system
```

RIB-SIM

1. Modify rib-sim-adapters.xml to add channels for a family.

```
<subscribers>
  <message-driven id="StockOrder_sub_1" initialState="running"/>
  <message-driven id="StockOrder_sub_2" initialState="running"/>
  <message-driven id="StockOrder_sub_3" initialState="running"/>
  <message-driven id="StockOrder_sub_4" initialState="running"/>
  <message-driven id="StockOrder_sub_5" initialState="running"/>
</subscribers>
```

2. Modify rib-sim-adapters-properties.properties.

```
StockOrder_sub_1.name=StockOrder Subscriber, channel 1
StockOrder_sub_1.desc=Subscriber for the StockOrder family through channel 1.
```

```
StockOrder_sub_2.name=StockOrder Subscriber, channel 2
StockOrder_sub_2.desc=Subscriber for the StockOrder family through channel 2.
```

```
StockOrder_sub_3.name=StockOrder Subscriber, channel 3
StockOrder_sub_3.desc=Subscriber for the StockOrder family through channel 3.
```

```
StockOrder_sub_4.name=StockOrder Subscriber, channel 4
StockOrder_sub_4.desc=Subscriber for the StockOrder family through channel 4.
```

```
StockOrder_sub_5.name=StockOrder Subscriber, channel 5
StockOrder_sub_5.desc=Subscriber for the StockOrder family through channel 5.
```

RIB-RWMS

1. Modify rib-rwms-adapters.xml to add channels for a family.

```
<subscribers>
  <message-driven id="StockOrder_sub_1" initialState="running"/>
  <message-driven id="StockOrder_sub_2" initialState="running"/>
  <message-driven id="StockOrder_sub_3" initialState="running"/>
  <message-driven id="StockOrder_sub_4" initialState="running"/>
  <message-driven id="StockOrder_sub_5" initialState="running"/>
</subscribers>
```

2. Modify rib-rwms-adapters-properties.properties.

```

StockOrder_sub_1.name=StockOrder Subscriber, channel 1
StockOrder_sub_1.desc=Subscriber for the stockorder family through channel 1.

StockOrder_sub_2.name=StockOrder Subscriber, channel 2
StockOrder_sub_2.desc=Subscriber for the stockorder family through channel 2.

StockOrder_sub_3.name=StockOrder Subscriber, channel 3
StockOrder_sub_3.desc=Subscriber for the stockorder family through channel 3.

StockOrder_sub_4.name=StockOrder Subscriber, channel 4
StockOrder_sub_4.desc=Subscriber for the stockorder family through channel 4.

StockOrder_sub_5.name=StockOrder Subscriber, channel 5
StockOrder_sub_5.desc=Subscriber for the stockorder family through channel 5.

```

Edit the RIB_SETTINGS table

When a PL/SQL Publishing adapter is multi-channeled, the application code needs to designate the message to a specific thread. In order to do this, a change needs to be made in the RIB_SETTINGS table.

Find the Family of messages that is being multi-channeled, and adjust the column NUM_THREADS to the appropriate number. In this example, the number will be set to 4 for the Alloc Family.

Compile and Deploy

Using the RIB Installer or the RIB App Builder command line tools, compile and deploy the new rib-<app>.ears.

Message Aggregation

To improve message publication throughput within the integration system, the RIB provides multiple capabilities. The most efficient way to increase throughput of any system is to start working on the collection of data units instead of single data units. Using that philosophy, RIB provides capabilities to process the collection of multiple detail payloads in one transaction. To control the number of details (payload details) per payload header, the user must update the **RIB_SETTING.MAX_DETAILS_TO_PUBLISH** column in the PL/SQL retail applications database schema. This configuration allows users to control the size of the payload published within the RIB system.

Users also may aggregate messages in a transaction by bundling multiple payloads within a single message published to the JMS server, for example. Through message aggregation (<family>.maxNodesPerMessages), users can control the number of ribMessage nodes bundled into a single RibMessages message. Different families can have different nodes per message, so this property is qualified (prefixed) by the family name. This property allows control of the overall size of the RibMessages XML message.

RIB also allows users to optimize/minimize XA transaction overhead by allowing the system to commit multiple RibMessages to the JMS server in a single, two-phase XA commit. The number of messages committed to the JMS server in a single XA commit is controlled by the property named `<family>.messagePerCommit`. Different families may need different RibMessages per commit, so this property is qualified (prefixed) by the family name.

The configurable properties (`<family>.maxNodesPerMessages` and `<family>.messagePerCommit`) applies to each individual rib-`<app>`. To update the property and propagate the configuration to the app server, edit the corresponding rib-`<app>`.properties in rib-home and redeploy the updated rib-`<app>`.

Keep in mind the bigger the payload size, the bigger the memory requirement. A process (JVM) gets limited amounts of operating system memory. If the size is too large, memory will run out, resulting in `OutOfMemoryError`.

If numerous ribMessageNodes are bundled into the same RibMessages message, a single failure in one of the ribMessages will roll back the full transaction. which will result in the following: The error hospital table will fill up and throughput will decrease by many factors, because now it has to go through the retry process.

The general best practice is to not prematurely optimize. Test with business data and only if the default values are not meeting business needs. Think about optimization by updating these properties.

How to Configure Message Aggregate

1. Edit the following file in rib-home:
 - rib-home/application-assembly-home/rib-`<app>`/rib-`<app>`.properties
2. Add the following properties:
 - `<family>.maxNodesPerMessages=<your value>`
 - `<family>.MessagePerCommit=<your value>`
3. Using the app-builder tool compile/deploy the application.
 - `rib-app-compile.sh`
 - `rib-app-deployer.sh -deploy-rib-app-ear rib-<app>`

Aggregation Example

Suppose there are 1,300 payload details waiting to be published for a family. Suppose the following configuration in RIB:

```
MAX_DETAILS_TO_PUBLISH=100
maxNodesPerMessages=5
MessagePerCommit=2
```

The diagram below explains the message aggregation in play in the RIB system. All 1,300 payload details will be published in three RibMessages within only two XA transaction commits. Each of the first two RibMessages will have five ribMessage nodes, and each of the ribMessage nodes will have a payload with 100 payload details. The example shows 1,300 payload details; the third RibMessages XML will have only three ribMessage nodes, each with 100 payload details.

XA transaction 1 = (RibMessages1 + RibMessages2)
 RibMessages1 = ribMessage1 + ribMessage2 + ribMessage3 + ribMessage4 + ribMessage5.

ribMessage1 = PayloadHeader + 100 * PayloadDetail

ribMessage2 = PayloadHeader + 100 * PayloadDetail

.....

ribMessage5 = PayloadHeader + 100 * PayloadDetail

RibMessages2 equivalent to RibMessages1

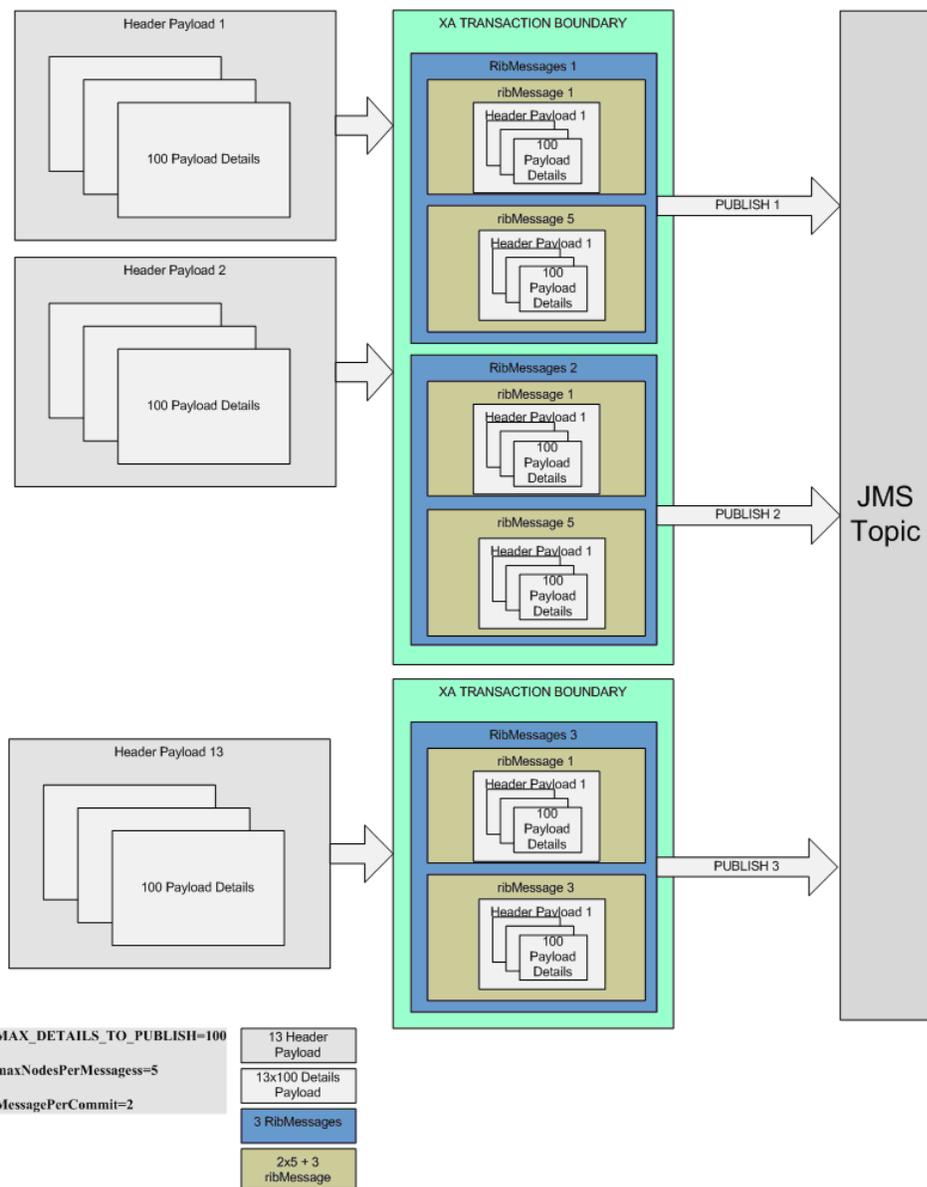
XA Transaction 2 = RibMessages3

RibMessages3 = ribMessage1 + ribMessage2 + ribMessage3

Total = (XA Transaction 1 + XA Transaction 2)

$100 \times 5 + 100 \times 5 + 100 \times 3 = 1300$

The following is an illustration of RIB Message Aggregation.



Multiple Hospital Retry

This section explains the multiple hospital retry process.

Family Specific Hospital Retry Adapters

The RIB supports configuration of hospital retry adapters specific to message families. The family based adapters are configured to address performance issues when the error hospital gets very large--and a single retry adapter cannot handle the load.

How Family Specific Hospital Retry Works

Errors during processing result in messages in the error hospital. Reasons for errors include the following:

- Incomplete or partial data from RMS: In this case, the messages are inserted into the error hospital with a reason code of PUB.
- JMS related publication error conditions: (For example, the JMS server is down or not available due to network failures.) In this case, the messages are inserted into the error hospital with a reason code of JMS.
- The subscriber application is not able to consume the message: In this case, the messages are inserted into the error hospital with a reason code of SUB.

By default, there are three kinds of hospital adapters, as listed below:

- Sub retry adapter
- JMS retry adapter
- Pub retry adapter (RMS is the only application for which the Pub retry adapter is required.)

The sub retry adapter retries only those messages with a reason code of SUB. Similarly, the JMS retry adapter and the Pub retry adapter retry messages with reason codes of JMS and PUB, respectively.

Note: For more information about the hospital retry mechanism, see "RIB Hospital Retry" in the *Oracle RIB Integration Bus Implementation Guide*.

Each message in the error hospital belongs to a particular message family. When the error hospital has a large number of messages from different families, the retry process becomes a performance bottleneck, as the default retry adapters retry the messages one by one (first in, first out), irrespective of message family.

To alleviate a bottleneck situation, retry adapters can be configured for a specific family and reason code. A family retry adapter can coexist with the default retry adapters. However, the default retry adapters will not retry those messages for which family retry adapters have been configured.

A family based retry adapter retries messages only for the family and reason code for which it is configured. For example, if a retry adapter is configured for the Order family and the SUB reason code, it retries only those messages from the Order family that failed with a reason code of SUB.

For each message family, a maximum of three family retry adapters can be configured--one for each reason code (PUB, SUB, and JMS).

How to Configure a Family Specific Retry Adapter

Process Overview:

1. Determine the rib-<apps> where the family specific hospital retry adapter is to be configured.
2. Determine the family for which the retry adapter should be configured.
3. Determine the reason code (for example, PUB, SUB, or JMS) for the family retry adapter.
4. In the rib-home, modify the appropriate configuration files for the rib-<apps>:
 - a. rib-<app>-adapters.xml
 - b. rib-<app>-adapter-resources.properties
5. Compile and deploy.

Example:

To configure a family specific adapter for the Order family, where reason code = SUB and application = rib-rms, complete the following steps:

1. Backup the following files:
 - rib-home/application-assembly-home/rib-rms/rib-rms-adapters.xml
 - rib-home/application-assembly-home/rib-rms-resources.properties
2. Modify rib-rms-adapters.xml to add the family specific hospital retry adapter. The following is a portion of rib-rms-adapters.xml:

```

<hospitals>
  <timer-driven          id="Order_familysubhosp_0"
  initialState="stopped" timeDelay="10" >
    <timer-task>
      <class
      name="com.retek.rib.j2ee.ErrorHospitalRegryTimerTask"/>
    </timer-task>
  </timer-driven/>
</hospitals>

```

3. Modify rib-rms-adapter-resources.properties as follows:
 - Order_familysubhosp_0.name=Order SUB Hospital Retry
 - Order_familysubhosp_0.desc=Inject messages into JMS from Error Hospital

Note: Only one instance of family retry adapter can be configured per family and per reason code.

4. Compile and deploy:

Using the RIB Installer or the RIB App Builder command line tools, compile and deploy the new rib-<app>.ears.