Oracle® Retail Integration Bus

Operations Guide

Release 16.0

E80332-01

December 2016

ORACLE®

Oracle Retail Integration Bus Operations Guide, Release 16.0

E80332-01

Primary Author: Sanal Parameshwaran

Contributing Author: Maria Andrew

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

3 Backend System Administration and Logging

4 RIB and JMX

5 RIB Administration GUI

6 JMS Provider Management

7 Message Transform, Filtering and Routing (TAFR)

8 RIB in Operation

9 Testing RIB

10 Performance Considerations

Send Us Your Comments

Oracle Retail Integration Bus Operations Guide, Release 16.0

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document.

Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

> **Note:** Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the new Applications Release Online Documentation CD available on My Oracle Support and www.oracle.com. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: retail-doc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

The *Oracle Retail Integration Bus Operations Guide* is 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.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

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

- *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 Integration Bus Security Guide*
- *Oracle Retail Integration Bus Support Tools Guide*
- *Oracle Retail Integration Bus Java Messaging Service (JMS) Console Guide*
- *Oracle Retail Enterprise Integration Guide*
- *Oracle Retail Integration Bus Integration Gateway Services Guide*
- *Oracle Retail Functional Artifacts Guide*
- *Oracle Retail Functional Artifact Generator Guide*
- *Oracle Retail Service-Oriented Architecture Enabler Tool Guide*
- *Oracle Retail Integration Bus Data Model*
- *Oracle Retail Payload Mapper 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, 16.0) or a later patch release (for example, 16.0.1). 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.

Improved Process for Oracle Retail Documentation Corrections

To more quickly address critical corrections to Oracle Retail documentation content, Oracle Retail documentation may be republished whenever a critical correction is needed. For critical corrections, the republication of an Oracle Retail document may at times not be attached to a numbered software release; instead, the Oracle Retail document will simply be replaced on the Oracle Technology Network Web site, or, in

the case of Data Models, to the applicable My Oracle Support Documentation container where they reside.

This process will prevent delays in making critical corrections available to customers. For the customer, it means that before you begin installation, you must verify that you have the most recent version of the Oracle Retail documentation set. Oracle Retail documentation is available on the Oracle Technology Network at the following URL:

http://www.oracle.com/technetwork/documentation/oracle-retail-100266.html

An updated version of the applicable Oracle Retail document is indicated by Oracle part number, as well as print date (month and year). An updated version uses the same part number, with a higher-numbered suffix. For example, part number E123456-02 is an updated version of a document with part number E123456-01.

If a more recent version of a document is available, that version supersedes all previous versions.

Oracle Retail Documentation on the Oracle Technology Network

Oracle Retail product documentation is available on the following web site:

http://www.oracle.com/technetwork/documentation/oracle-retail-100266.html
(Data Model documents are not available through Oracle Technology Network. You can obtain them through My Oracle Support.)

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.
italic	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.

1
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

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: For more information, see the *Oracle WebLogic Server Administrator's Guide*12c Release (12.2.1).

Oracle Retail Integration Bus 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 each subscribing adapter is 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. Different TAFR adapters 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 or 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, has a dedicated Hospital.
- RIHA is the Oracle Retail Integration Bus Hospital Administration tool.
- The Integration Gateway Services (IGS) provides an integration infrastructure for external (third party) system connectivity.

2

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 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, 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 systems administrator can be given access permissions to all the tools, while a RIB administrator or applications administrator can be provided access to only certain operation tools.

The RIB Application 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 with in 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

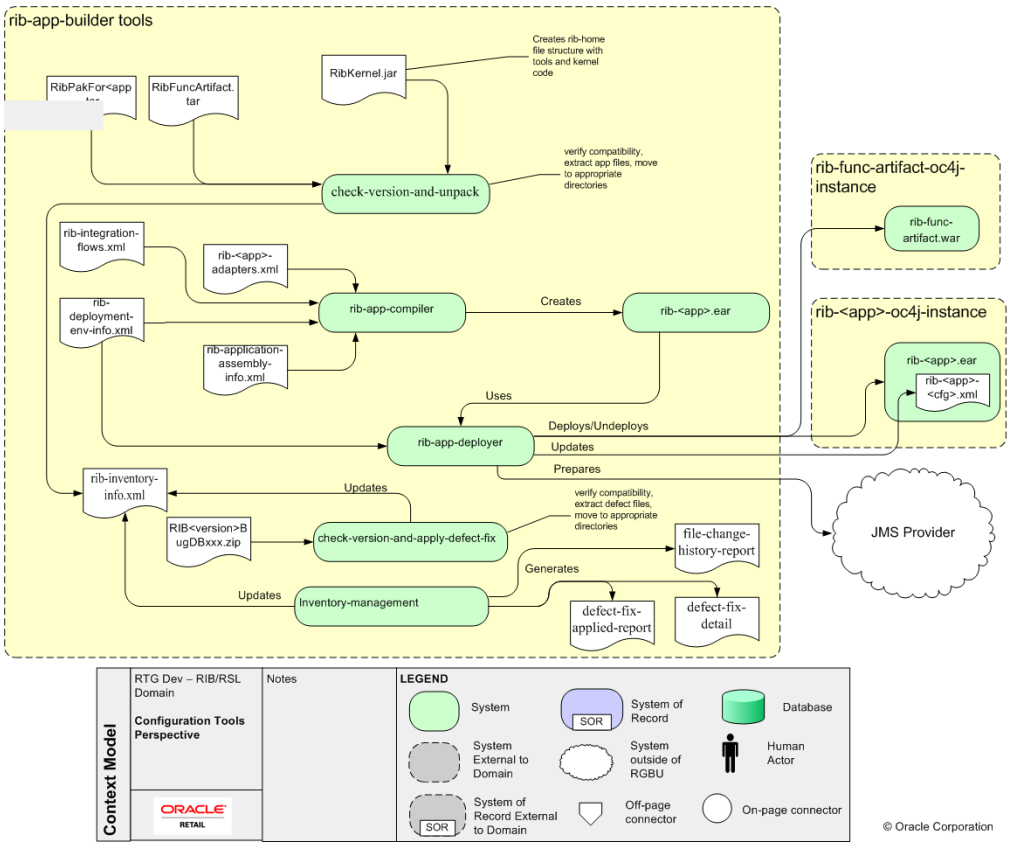
```
rib-home
```

```
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-<version>.war
            rib-private-tafr-business-impl-<version>.jar
            retail-public-payload-java-beans-base-<version>.jar
            retail-public-payload-database-object-types-<version>.jar
            retail-public-payload-database-xml-library-<version>.jar
            retail-public-payload-java-beans-<version>.jar
            retail-public-payload-xml-samples-<version>.jar
        rib-rms
            rib-<app>-adapters-resources.properties
            rib-<app>-adapters.xml
            rib-<app>-plsql-api.xml
            rib-<app>.properties
        rib-rpm
        rib-rfm
        rib-rwms
        rib-sim
        rib-tafr
        rib-oms
        rib-rxm
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
            inventory-reports.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
```

```
integration-bus-gateway-services
rdmt
rib-func-artifact-gen
riha
```

RIB Application 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 log4j2 and the log4j2.xml in rib-home. Do not edit this log4j2.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 the credentials for each user alias 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 RIB components. It takes a set of command line 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 6, "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.

Command Line Option	Description
-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 RIB paks and extracts 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 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 the naming convention to check if the kernel version is the same as the functional artifact version. If the versions are 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 versions are compatible, the tar file is un-tar'd into the rib-home/application-assembly/rib-<app> directory.

check-version-and-apply-defect-fix

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

All RIB defect fixes are in the form of a zip file (for example, RIB16_Bug1234.zip). The zip file always contains a README.txt file in the format below.

```
------------------------------------------------------------
    Product     : Oracle Retail Integration Bus
    Version #   : 16.0.0
    Defect #    : 1234
    Date        : MM/DD/YYYY
------------------------------------------------------------

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

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

Take the following steps to apply the defect fixes to the rib-home:

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 script 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 deploy 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

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 RIB applications 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

Command Line Option	Description
-setup-weblogic-credential< wls-id>	Updates security credential for the specified WebLogic instance.
-setup-admin-gui-credential rib-<app>	Updates security credential for the RIB Administration GUI user for the specified RIB application.
-setup-error-hospital-credential rib-<app>	Updates security credential for the error hospital database user for the specified RIB application.
-setup-app-database-credential rib-<app>	Updates security credential for the application database for the specified RIB application.
-setup-jndi-credential rib-<app>	Updates security credential for the remote JNDI for the specified RIB application.

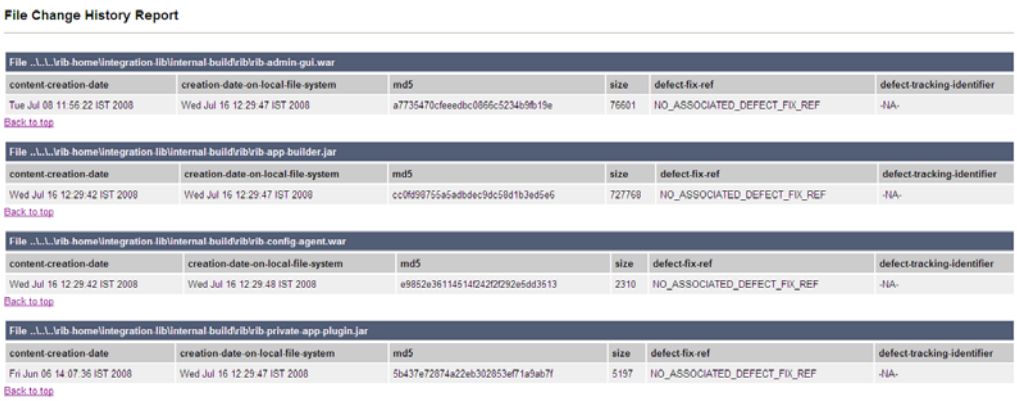
Hot Fix Installation Reports

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\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	a7735470cfeeedbc0866c5234b9b19e	76601	NO_ASSOCIATED_DEFECT_FIX_REF	-NA-

Back to top

File ..\..\..\rib-home\integration-lib\internal-build\rib\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	cc0fd98755a5adbdec9dc58d1b3ed5e6	727768	NO_ASSOCIATED_DEFECT_FIX_REF	-NA-

Back to top

File ..\..\..\rib-home\integration-lib\internal-build\rib\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	e9852e36114514f242f2f292e5dd3513	2310	NO_ASSOCIATED_DEFECT_FIX_REF	-NA-

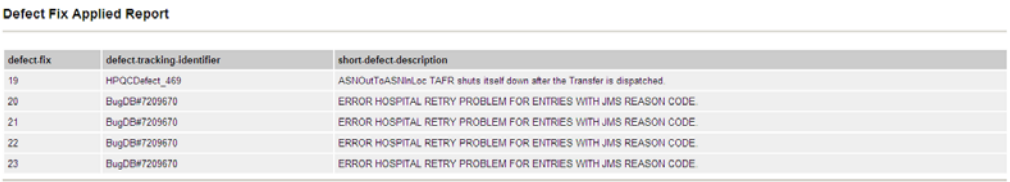
Back to top

File ..\..\..\rib-home\integration-lib\internal-build\rib\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	5b437e72874a22eb302853ef71a9ab7f	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	HPQCDefect_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 provides 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. For more information, see "RIB Components Start and Stop."

> **Note:** The rib-adapter-controller does not work if the adapters have not been started from the RIB Admin GUI at least once. If the rib-adapter-controller does not work, log in to the RIB Admin GUI and start all adapters. After this initial start, the adapters can be in any status and will respond to rib-adapter-controller commands.

Start Flow

This function 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]
```

The function does the following:

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 applications 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

This function 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>
```

The function does the following:

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 RIB applications that are not in scope.

Examples:

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

List Flow

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

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

The function does the following:

1. It displays all message node IDs for all message flows in which the given family participates.
2. It lists the adapters in the order as defined in the message flows.
3. It ignores RIB applications that are not in scope.

Examples:

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

Start Adapters By Type

This function 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]
```

The function does the following:

1. For every adapter type specified in the input, it collects the adapter instances from the given rib-app-list.
2. It re-sorts 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 moves on to start all 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 applications 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

This function 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>
```

The function does the following:

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 applications 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

This function starts individual adapter instances. The 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]
```

The function does the following:

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

This function 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>
```

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

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

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

The function does the following:

1. It finds the topic names to which the input RIB application adapter class definition publishes.
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_tafr
```

Test Durable Subscriber for RIB Application

This function tests if durable subscribers 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>
```

The function does the following:

1. Finds all adapter instances that publish 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 Application Adapters

The rib-adapter-controller lists all adapter instances for a given rib-app or rib-app-list (comma separated list without any spaces).

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

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 Application 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 RIB Admin sub menu.

> **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 Exceed, and set the DISPLAY environment variable.

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 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.

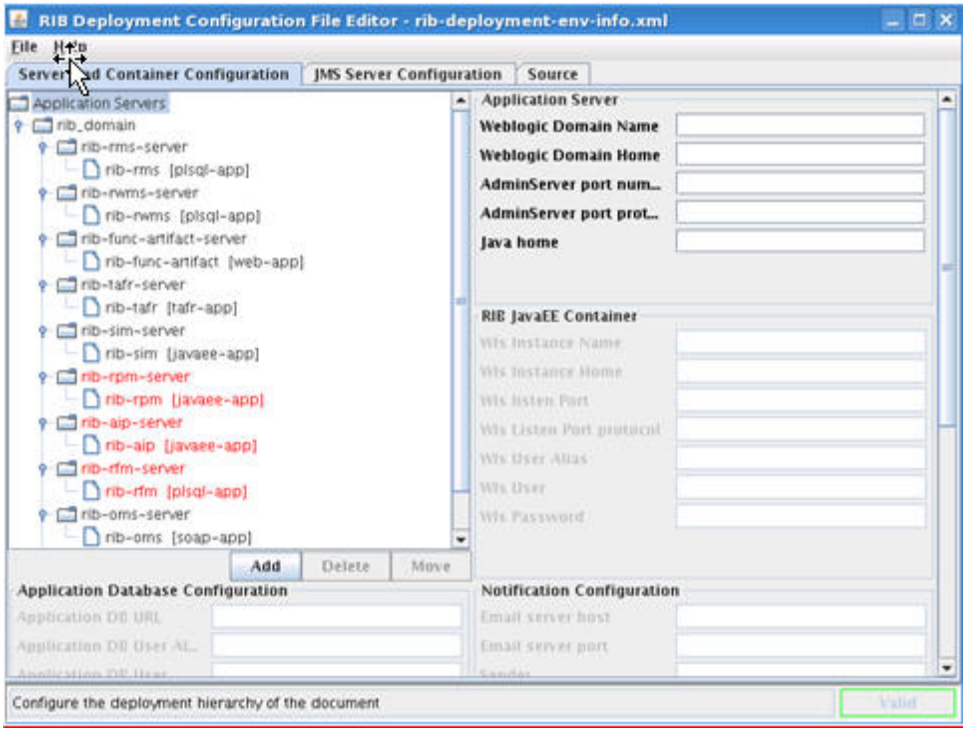
Key Rule

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:

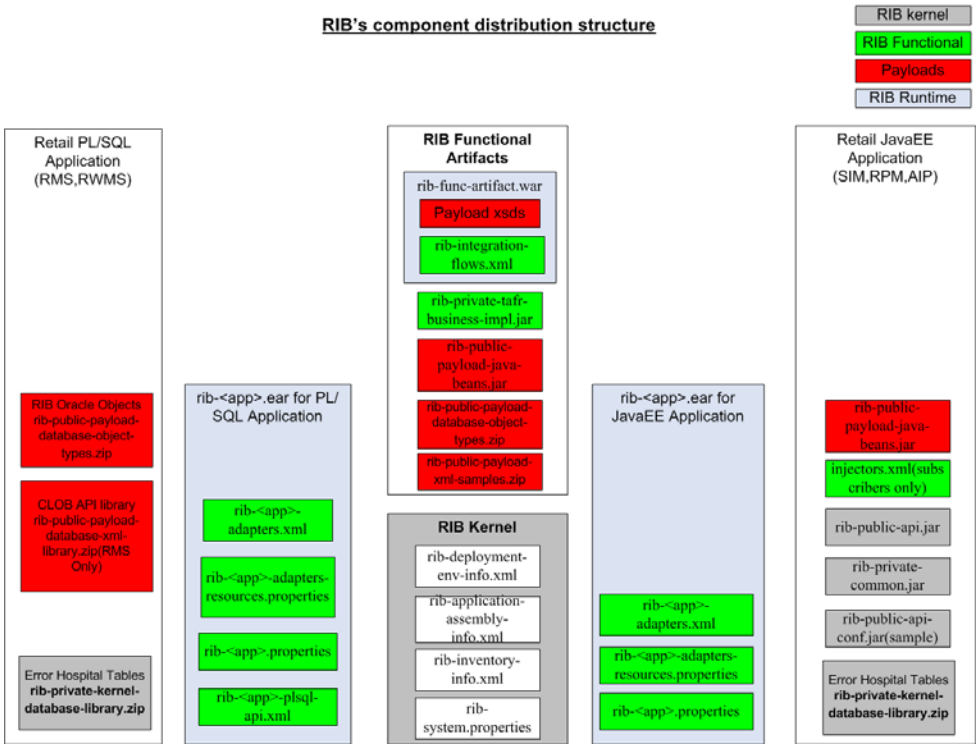
1. Select File from the menu bar. Click **Open**.
2. Navigate to the directory containing rib-deployment-env-info.xml.

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.

3

Backend System Administration and Logging

The following graphic illustrates the names of the actual RIB files and shows their location 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, where $application_instance_home is the WebLogic instance path where the application is deployed.

The following sections describe the standard RIB defined adapter types.

<subscribers> elements

The <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 adapter) and initialState (the initial state of the adapter) attributes. The initialState attribute for <message-driven> adapters 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> elements

The <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>

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

This element 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 adapter) and initialState (specifies the initial state of the adapter) 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

This element specifies hospital related adapter information. The structure is very similar to the <publisher> element except that the name and value attributes in the property element define the different hospital adapter 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 properties internationalize 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, RFM, and RWMS. RIB interfaces with RMS, RFM, 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 properties internationalize 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 defaultValues = 1,2 and 3;	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 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/$$SERVER NAME/logs/$APP_NAME	This property refers to the location where log files are created by the RIB application. By default this location is in the logs/app_name directory inside the WebLogic instance home where the app has been installed.
mail_smtp_host defaultValue = "mail.smtp.host";	This property is used to identify the smtp host from which to send out emails.
mail_smtp_port defaultValue = "25";	This property is used to identify the smtp port from which to send out emails.
mail_smtp_from defaultValue = "admin@company.com";	This property refers to the email id that the RIB platform needs to use to send the emails for administrative purposes.
war_http_port defaultValue = "9080";	This property refers to the port number used by the web based Hospital Retry Administration Tool.
wls.wallet.file.location defaultValue=$DOMAIN_ HOME/serves/$SERVER NAME	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 Application 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"/>
      <app id="oms" type="soap-app"/>
      <app id="rxm" type=" javaee -app"/>
```

rib-jms-server

This section defines the JMS server information.

Example:

```
<jms-server-home>linux1@linux1:/home/oracle/oracle/product/12.1.0.2/db_
1</jms-server-home>
<jms-url>jdbc:oracle:thin:@linux1:<port>:ora12c</jms-url>
<jms-port><port></jms-port>
<jms-user-alias>jms1_user-name-alias</jms-user-alias>
```

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
protocol="http">7001</weblogic-admin-server-port><java-home>/usr/java/jdk1.8</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.example.com</email-server-host>
       <email-server-port>25</email-server-port>
       <from-address>rib-email@example.com</from-address>
       <to-address-list>rib@example.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://simhost.example.com:<port>/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:@simdbhost.example.com:<port>:orcl</hosp-url>
<hosp-user-alias>rib-rms_error-hospital-database_user-name-alias</hosp-user-alias>
</error-hospital-database>
```

commons-logging.properties

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

log4j2.xml

The log4j2 Open Source software is used to control all RIB logging. This software requires the log4j2.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 JNDI related configuration used by rib-<app> to invoke remote EJBs hosted on Java retail apps (for example, RPM and SIM). This file is built at runtime, based on the information provided in rib-deployment-env-info.xml.

RIB Logging

All logging in RIB is through log4j2, the Apache Software Foundation's Open Source software. For details about log4j2 visit the Apache Software Foundation's log4j2 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). It is important to note that while the DEBUG logging level provides insight into the low-level processing occurring within the RIB, logging at such a low level comes at a cost. Specifically, such detailed logging is very CPU-intensive, and depending on the application server hardware configuration, the actual RIB message processing logic could be forced to wait for the CPU cycles being occupied by the detailed DEBUG logging, which will result in an increase in overall message processing time. In a production environment, the logging level must be set to the lowest level possible in order to ensure proper resource allocation to all RIB message processing logic.

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 as described for the logging level above. To summarize once again, the lowest level of audit and timing logging should be used in a production environment.

As a rule, the appropriate level is INFO.

Changing Logging Levels

RIB use of log4j2 allows the control of logging levels to suit the deployment and situation. There are two methods of setting the logging levels: directly manipulating the log4j2.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 retain that level when restarted. This is the recommended approach.

log4j2.xml Configuration File

The RIBLOGS log4j2.xml file can be directly edited. This requires that the adapters be bounced for the change to take effect. See the following sections for what to edit, as 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 RIB's runtime logs to the RIBLOG log files. The logs are written to the path <rib-application_instance_home>/logs/<rib-app>.

Example:

```
/u01/webadmin/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 log4j2.xml file. Search for the section of the log4j2.xml file:

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

RIB Timing Logs

The RIB messaging components code is instrumental in logging timing entries on the internal activities whenever they create, transform, route, filter, or subscribe to messages on RIB. These timings logs are written using the log4j2 logging mechanism.

The timings log files follow the name convention <adapter-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) that 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 stored 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 log4j2.xml file must have the "level value" property set to DEBUG. This tag is not normally present in the standard log4j2.xml file, it must be added. The following example shows how and where.

Note that there are multiple entries for each adapter instance in the log4j2.xml file. Search for the section of the log4j2.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

RIB has an auditing feature that logs a message as it passes though 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.

Typically, one audit log file is created per component (EJB or other) that holds the entries for that component. These files are non rolling, meaning that they do get overwritten after they reach a certain configurable size. Customer has to take care to manage audit log file size.Audit logs has some performance impact so should be enabled only when there is need to save messages.

Refer Chapter 6 “RIB Administrator GUI” for more information on setting the non-persistent log levels to persistent i.e. DEBUG and viewing audit logs in GUI.

The log4j2.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.log4j2.FileAppender"
 name="appender.rib.sub.audit.ASNIn_sub_1">
"/u00/webadmin/product/12.2.1/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.log4j2.PatternLayout">
            <param name="ConversionPattern" value="&lt;audit-entry
audit-time=&quot;%d{yyyy.MM.dd
 HH.mm.ss,SSS}&quot;&gt;%n%m%n&lt;/audit-entry&gt;%n"/>
        </layout>
    </appender>

Remove the “value=” in the ConversionPattern with %m%n
```

RIB also can 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 log4j2.xml file. Search for the section of the log4j2.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://ribhost.example.com: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=&quot;http://www.oracle.com/retail/integration/payload/BannerDesc&quot;
xmlns:ribdate=&quot;http://www.oracle.com/retail/integration/payload/RIBDate&quot
;
xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot;
xsi:schemaLocation=&quot;http://www.oracle.com/retail/integration/payload/BannerDe
sc http://ribhost.example.com:7777/rib-func-artifact/payload/xsd/BannerDesc.xsd
http://www.oracle.com/retail/integration/payload/RIBDate
http://ribhost.example.com:7777/rib-func-artifact/payload/xsd/RIBDate.xsd&quot;&gt
;  &lt;banner_id&gt;1&lt;/banner_id&gt;  &lt;banner_
name&gt;B&amp;amp;amp;M&lt;/banner_name&gt;&lt;/BannerDesc&gt;
</messageData>
<customData></customData><customFlag>F</customFlag>
</ribMessage>
</RibMessages>
```

Other RIB Management Logs

The following are examples of 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(ribhost.example.com)
dir(/stage/Rib1600-ms7.1/Rib1600ForAll16xxApps/rib-home/deployment-home/bin/../../
../rib-home) at(Mon Jan 28 11:15:57 PST 2016).
```

management.rib.log

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

This log usually is written during the startup of a 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-RMICallHandler-7] DEBUG
retek.com.retek.rib.ui.view.tags.IteratePropertyTag.com.retek.rib.management.adap
ters.model.AdapterTypes - Invoking Operation returnStatusForAll of MBean.
2008-02-06 10:14:26,777 [AJPRequestHandler-RMICallHandler-7] DEBUG
retek.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-RMICallHandler-7] DEBUG
retek.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="WOOut_pub_1" name="WOOut
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>
```

4

RIB and JMX

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

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

RIB uses log4j2 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 (log4j2.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: For more information, see the "Java Management Extensions (JMX)" section in the *Oracle Retail Integration Bus Implementation Guide*.

Third Party JMX Client Example

This example is for the Sun JConsole tool.

See:
http://docs.oracle.com/javase/6/docs/technotes/guides/management/jconsole.html

Complete the following steps:

1. Copy the following file to the host where the jconsole runs:

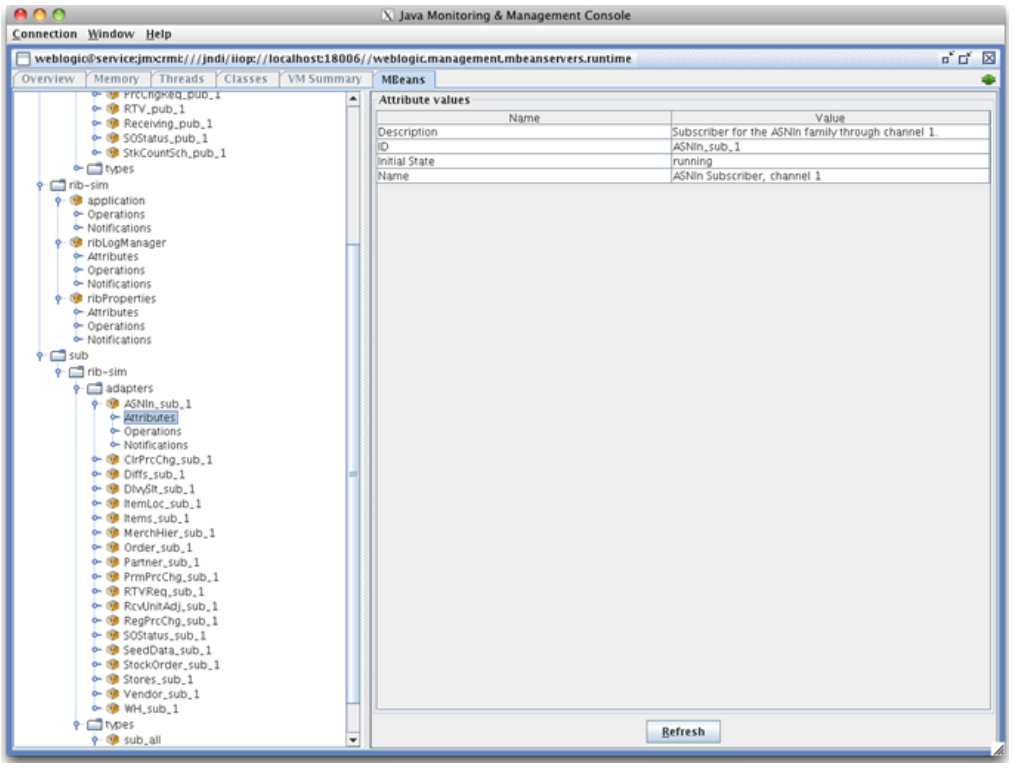
   wljmxclient.jar

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

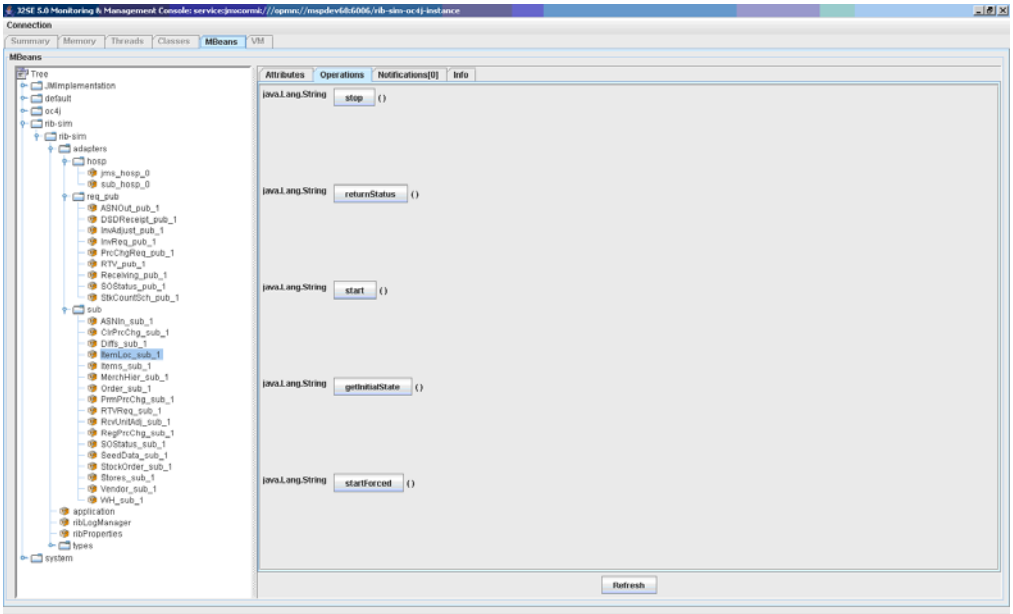
```
jconsole
-J
-Djava.class.path=$JAVA_HOME/lib/jconsole.jar:$JAVA_HOME/lib/tools.jar:$WL_
HOME/server/lib/wljmxclient.jar
-J-Djmx.remote.protocol.provider.pkgs=weblogic.management.remote -debug
```

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).

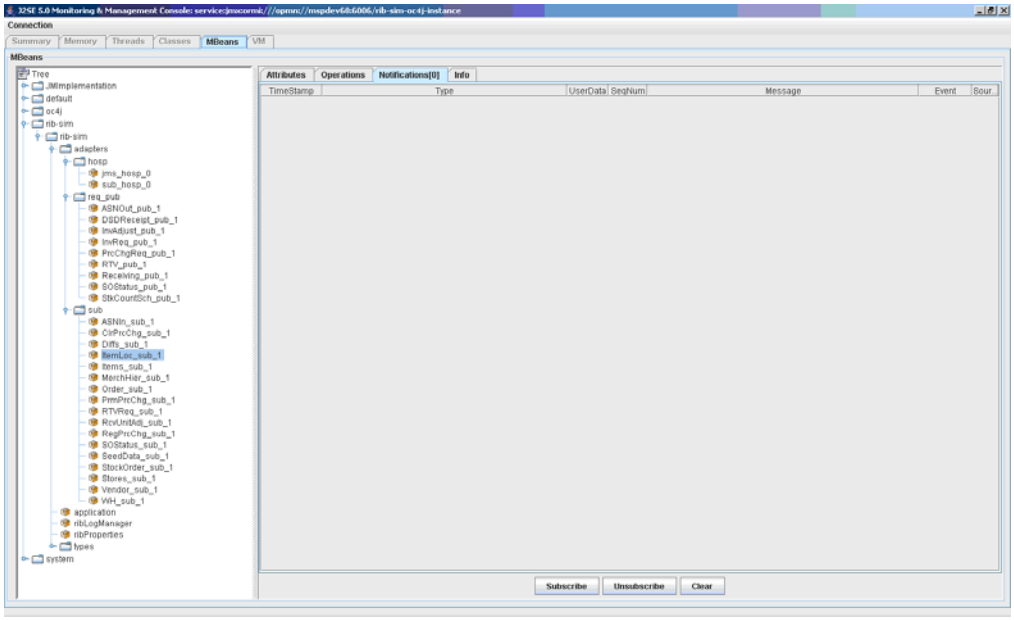
4. Select and open any one of the MBean. It opens a window with four tabs: Attributes, Operations, Notifications, and Info.
   a. The Attributes tab provides information about the attributes of the MBean.



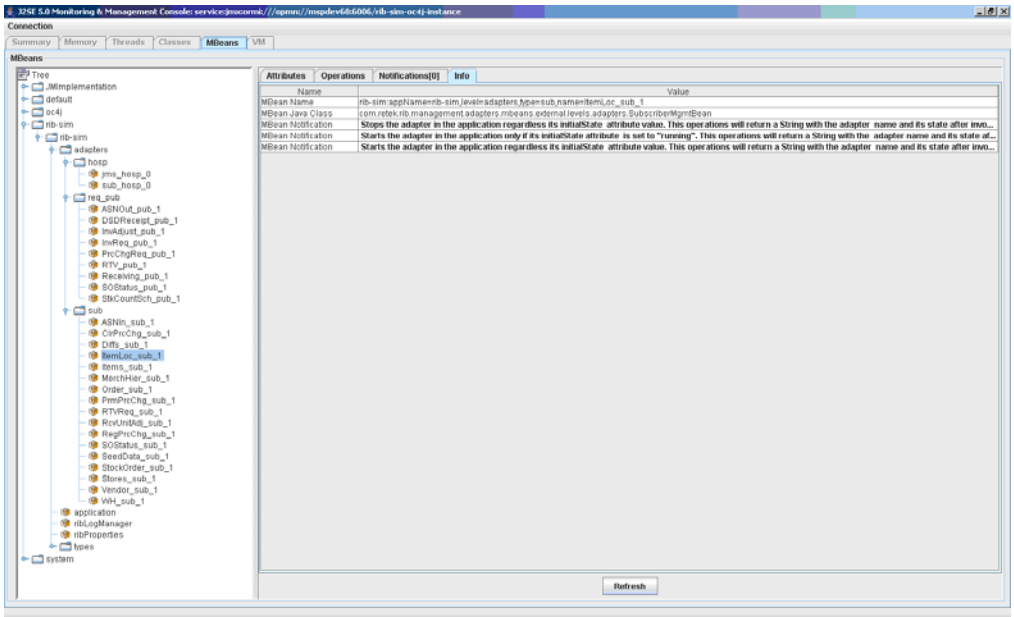
b. The Operations tab includes the list of operations supported by that MBean.



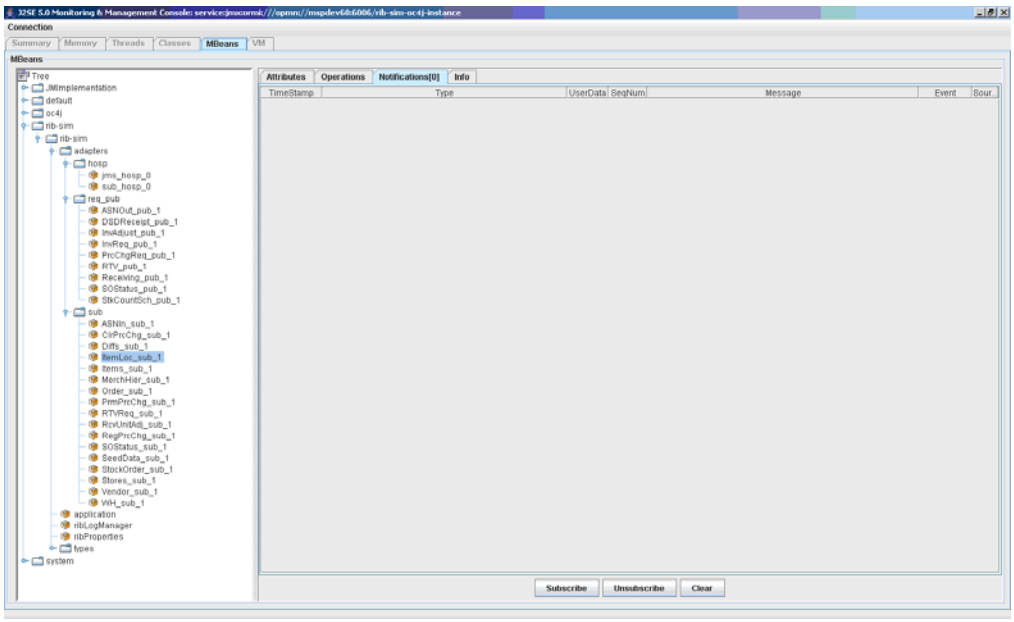
c. The Notifications tab includes the list of notifications captured on that MBean. (You must subscribe for capturing the notifications.) Subscribing, unsubscribing, and clearing notifications can be done from this tab.



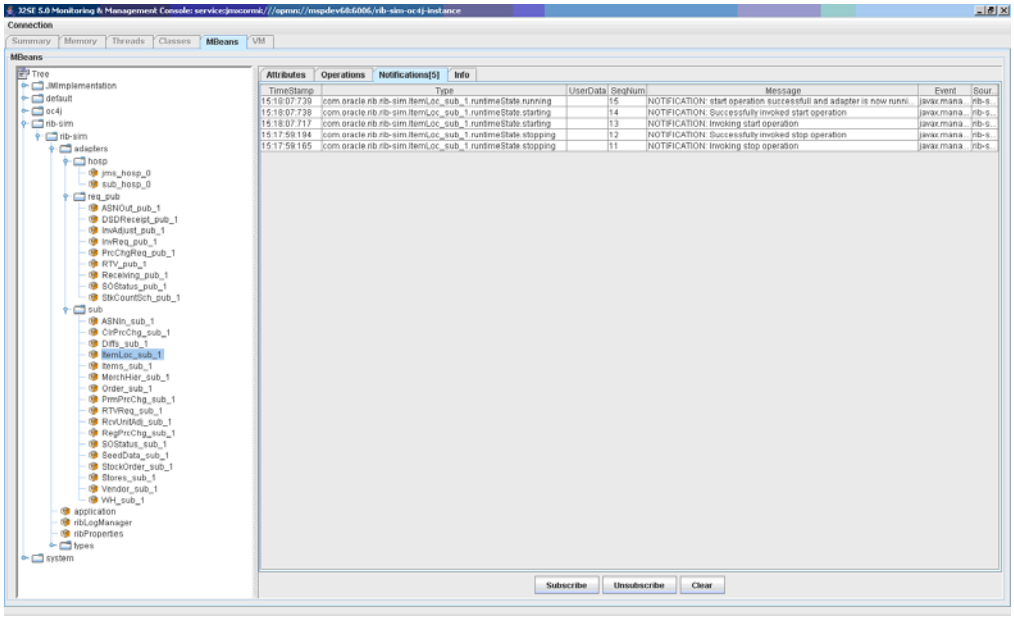
d. The Info tab provides details about the MBean.



e. When 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.



5

RIB Administration GUI

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 Java EE 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 the RIB Administration GUI 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 Administration URLs

RIB Administration tools are obtained through URLs within each of the deployed rib-<apps>.

RIB Administration GUI

```
http://<server>:<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 WebLogic Server is listening on (for example, 7777).

Replace <app> with one of the following:

- rms
- rfm
- rpm
- aip
- tafr

- rwms
- sim
- oms
- rxm

RIB Functional Artifacts

```
http://<server>:<port>/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

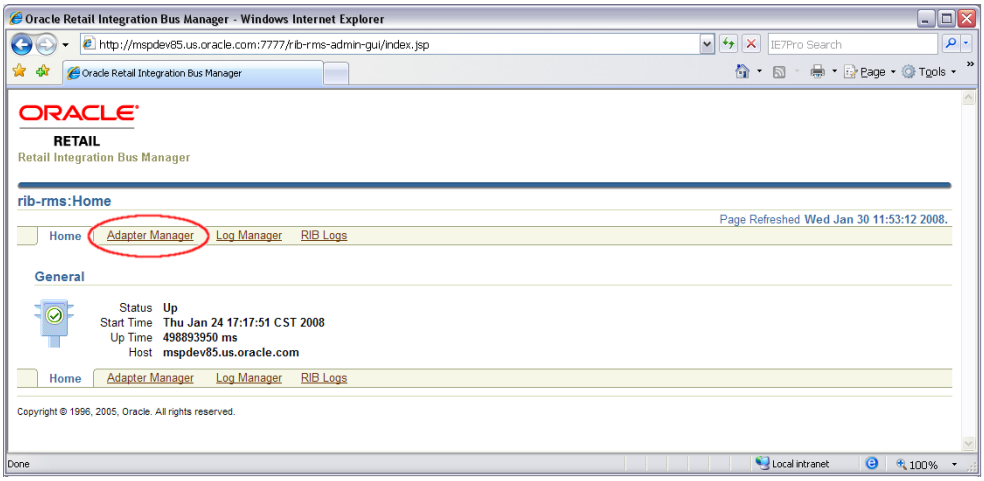
```
http://<server>:<port>/rib-func-artifact/rib-integration-flows.xml
```

RIB Payloads (xsds)

```
http://<server>:<port>/rib-func-artifact/payloads/xsd
```

RIB Administration GUI Home

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



Adapter Manager

All message functions in RIB are performed by adapters. The four categories of adapters are 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.

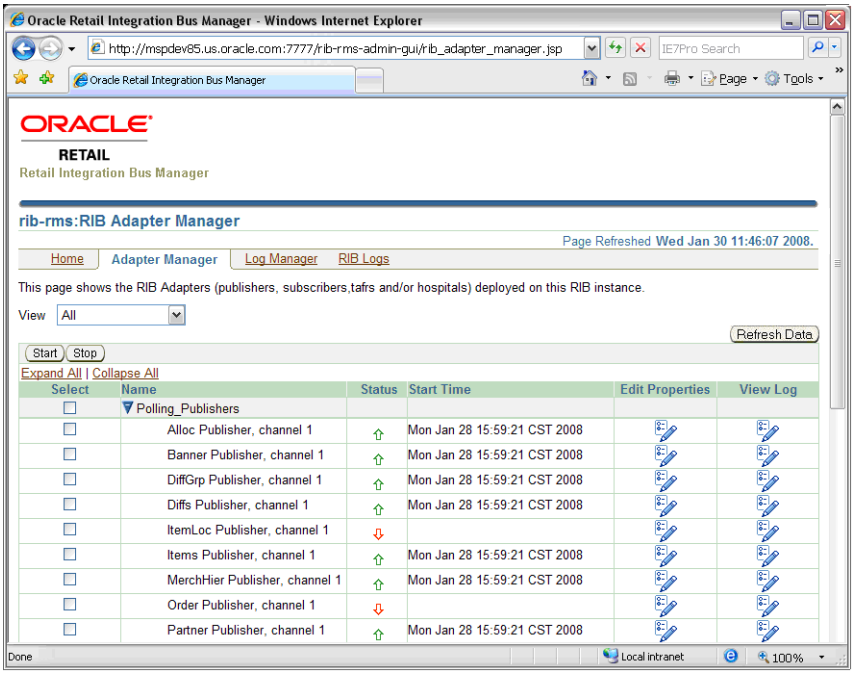
Adapter Manager Screen

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:

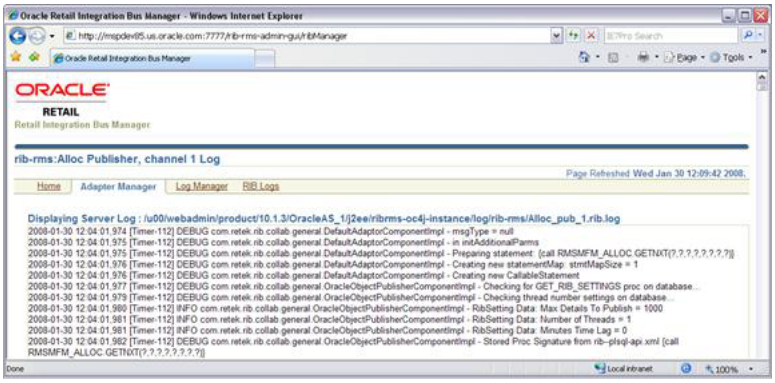
From this screen any listed adapters can be started and stopped by selecting the check box related to the adapter and then using the following buttons:

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.



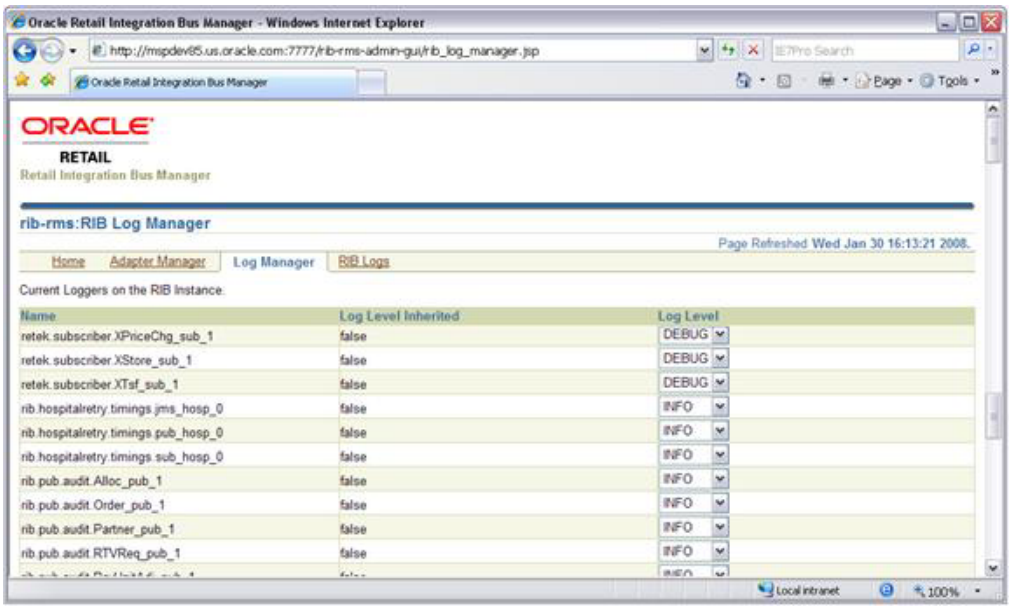
Log Manager

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.

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.

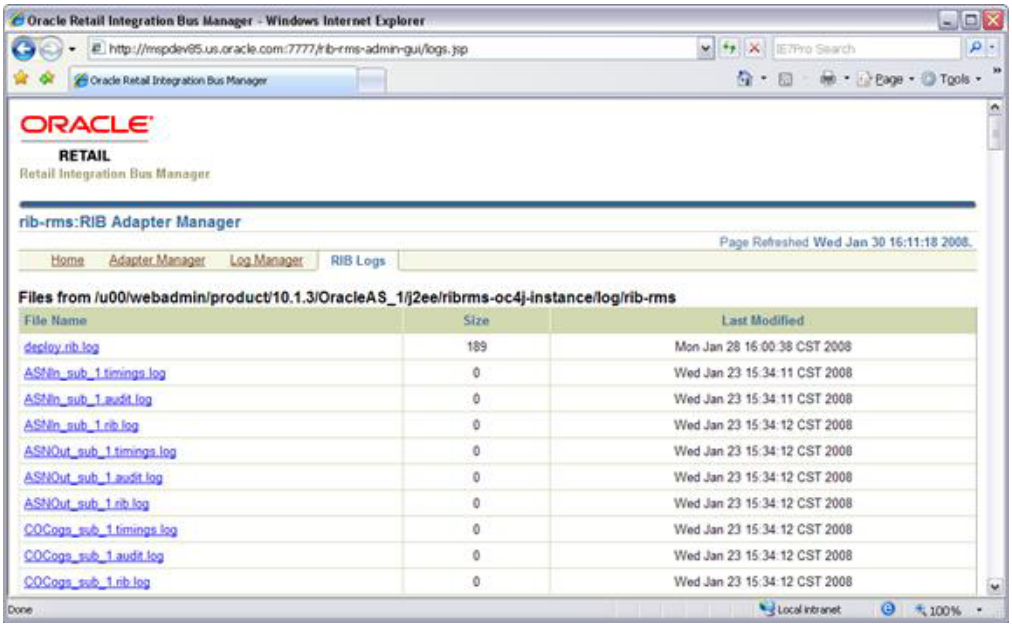
The Timings logging captures adapter processing performance data to another separate log. As with the audit log, this only works with the logging level set to DEBUG. The RDMT command line tool can be used to process and view the results of the timings logging output.



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 for the Log Manager screen.)

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



6

JMS Provider Management

The Oracle Enterprise Messaging Service (OEMS) provides a robust architecture for integrating business-critical applications. It is built on Java 2 Enterprise Edition (J2EE) standards such as the Java Message Service (JMS) and the J2EE ConnectorArchitecture (JCA). In addition, 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.

RIB will be certified with several JMS providers, 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 - Using Oracle Enterprise Messaging Service.

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.

The RIB team and the Database Administrators should consider the following.

- 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 RIB has high transaction rates. Consider this when configuring 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.
- RIB is a client of the AQ database and uses JDBC connections through 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. Explicitly setting AQ_TM_PROCESSES is not required. However, monitoring the workload and making adjustments as necessary is recommended. 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, the recommended value is between 2 and 8. Do not set the value to the maximum allowed value of 10 in Oracle, because 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 two processes for Oracle that can be started to maintain buffered messages.

Optimizing Enqueue/Dequeue Performance

The AQ database performance must be tuned according to Oracle database tuning practices.

To tune the SGA, use tools such as Statspack, Oracle Enterprise Manager and SQL trace to identify bottlenecks. An inefficiently configured SGA slows down enqueue and dequeue transactions.

To tune the Server Resources, check server CPU, memory, I/O, and network utilization. Tools such as nmon, sar, iostat, vmstat, and 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.

Tuning Physical Schema setup entails creating right tablespaces, placements of datafiles, tables, and indexes.

Note: See also Oracle® Database Administrator's Guide 12c Release 2 (12.2.1), Oracle® Streams Advance Queuing User's Guide, and Reference 12c Release 2 (12.2.1)

Sizing Considerations

The RIB team and Database Administrators provide the following considerations for sizing the deployment of RIB on AQ JMS:

- The enqueuing/dequeuing 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) with a high transaction rate can quickly consume available segments. If the tablespace is not sized appropriately, 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 enqueuing versus dequeuing. 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 must 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."

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 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.

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 Connections - Summary

RIB Adapter Type	Total Adapters in RIB
rib-app Subscriber	88
TAFR Subscriber	21
rib-app Polling Publisher	33
rib-app Request-driven Publisher	19
TAFR Publishers	21
Hospital Retry - Polling Publisher	18
Total	200

At any time, depending on deployment options in a non-multiple channel deployment, RIB can have at least 200 AQ connections. The application server may ask for more than 200 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
Subscriber	36
Polling Publisher	23
Hospital Retry - Polling Publisher	3
Total	62

At any time, depending on deployment option, the rib-rms app can have at least 62 AQ connections. The application server may ask for more than 62 from the database.

rib-rwms Connections

RIB Adapter Type	Total Adapters in RIB
Subscriber	16
Polling Publisher	8
Hospital Retry - Polling Publisher	2
Total	26

At any time, depending on deployment option, the rib-rwms application can have at least 27 AQ connections. The application server may ask for more than 27 from the database.

rib-sim Connections

RIB Adapter Type	Total Adapters in RIB
Subscriber	21
Request Driven Publishers	12
Hospital Retry - Polling Publisher	2
Total	35

At any time, depending on deployment option, the rib-sim app can have at least 35 AQ connections. The application server may ask for more than 35 from the database.

rib-tafr Connections

RIB Adapter Type	Total Adapters in RIB
Subscriber	21
Publishers	21
Hospital Retry - Polling Publisher	2
Total	44

At any time, depending on deployment option, the rib-tafr app can have at least 46 AQ connections. The application server may ask for more than 46 from the database.

rib-rpm Connections

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

rib-rfm Connections

RIB Adapter Type	Total Adapters in RIB
Subscriber	1
Polling Publisher	2

RIB Adapter Type	Total Adapters in RIB
Hospital Retry - Polling Publisher	3
Total	6

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

rib-oms Connections

RIB Adapter Type	Total Adapters in RIB
Subscriber	7
Request Driven Publisher	1
Hospital Retry - Polling Publisher	2
Total	10

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

rib-rxm Connections

RIB Adapter Type	Total Adapters in RIB
Subscriber	7
Request Driven Publisher	0
Hospital Retry - Polling Publisher	2
Total	9

At any time, depending on deployment option, the rib-rxm app can have at least nine AQ connections. The application server may ask for more than nine 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 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 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 configured for a given 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

This section describes the process for setting up multiple JMS servers.

Process Overview

The following are basic steps.

1. Determine the family to be configured.
2. Examine the rib-integration-flows.xml to identify all 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

Consider the following 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 and flows through 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_tafr" app-name="rib-tafr"
                   adapter-class-def="ItemsToItemsTL_tafr" type="JmsToJms">
                    <in-topic>etItemsFromRMS</in-topic>
                    <out-topic>etItemsTLFromRIB</out-topic>
             </node>
             <node id="rib-tafr.ItemsToItemsISO_tafr" app-name="rib-tafr"
                   adapter-class-def="ItemsToItemsISO_tafr" 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><port></jms-port>
              <jms-user-alias>aq1</jms-user-alias>
      </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><port></jms-port>
              <jms-user-alias>aq2</jms-user-alias>
              </aq-jms-server>
 </aq-jms-servers>
```

RIB-RMS Application Configuration

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

rib-rms-adapters.xml

For rib-rms-adapters.xml, do the following.

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 the Error Hospital.
- pub_hosp-2.name=PUB Hospital Retry jms2
- pub_hosp-2.desc=Re-publish messages to JMS.
- jms_hosp-2.name=JMS Hospital Retry jms2
- jms_hosp-2.desc=Re-publish messages from the Error Hospital 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

For rib-tafr-adapters.xml, do the following.

1. Edit $RIB_HOME/application-assembly-home/rib-tafr/rib-tafr-adapters.xml, where $RIB_HOME is the rib-home directory.
2. Point the ItemsToItemsTL_tafr_1 adapter to jms-server-id "jms2", as shown below.
3. Point the ItemsToItemsISO_tafr_1 adapter to jms-server-id "jms2", as shown below:

```
<tafrs>
       <message-driven id="ItemsToItemsTL_tafr_1" initialState="stopped"
tafr-business-impl="com.retek.rib.domain.tafr.bo.impl.ItemsToItemsTLFromRibBOIm
pl"  jms-server-id="jms2" />

       <message-driven id="ItemsToItemsISO_tafr_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 the Error Hospital.
- jms_hosp-2.name=JMS Hospital Retry jms2
- jms_hosp-2.desc=Re-publish messages from the Error Hospital 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

For rib-sim-adapters.xml, do the following.

1. Edit $RIB_HOME/application-assembly-home/rib-sim/rib-sim-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 the Error Hospital.
- jms_hosp-2.name=JMS Hospital Retry jms2
- jms_hosp-2.desc=Re-publish messages from the Error Hospital to JMS after JMS is brought up again.

RIB-RWMS Application Configuration

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

rib-rwms-adapters.xml

For rib-rwms-adapters.xml, do the following.

1. Edit $RIB_HOME/application-assembly-home/rib-rwms/rib-rwms-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

Add the following properties to the resources file:

- sub_hosp-2.name=SUB Hospital Retry jms2
- sub_hosp-2.desc=Inject messages into the Error Hospital.
- jms_hosp-2.name=JMS Hospital Retry jms2
- jms_hosp-2.desc=Re-publish messages from the Error Hospital to JMS after JMS is brought up again.

RIB-RFM Application Configuration

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

rib-rfm-adapters.xml

For rib-rfm-adapters.xml, do the following.

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

```
<subscribers>
         <message-driven id="ShipInfo_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-rfm-adapters-resources.properties

Add the following properties to the resources file:

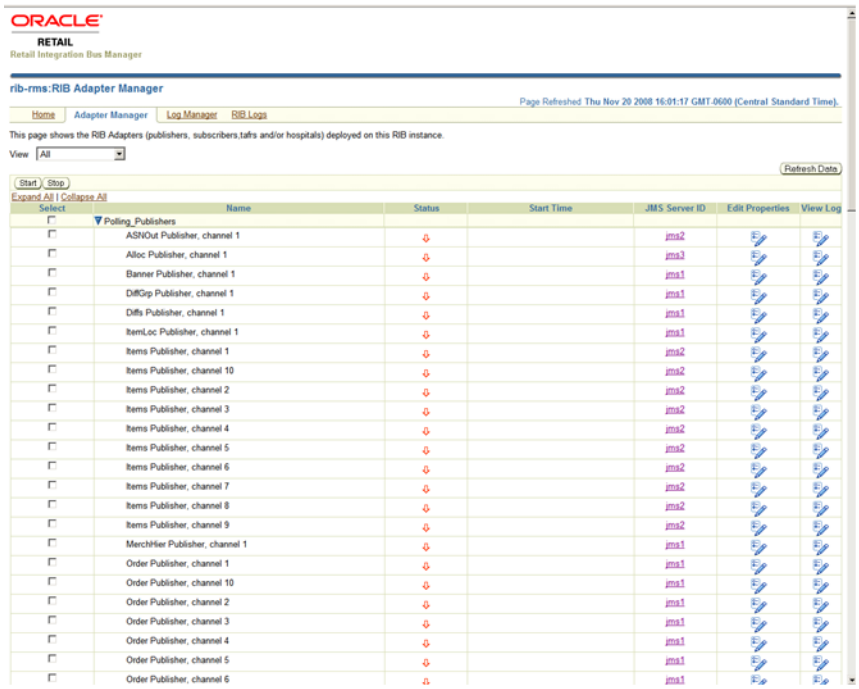
- sub_hosp-2.name=SUB Hospital Retry jms2
- sub_hosp-2.desc=Inject messages into the Error Hospital.
- jms_hosp-2.name=JMS Hospital Retry jms2
- jms_hosp-2.desc=Re-publish messages from the Error Hospital to JMS after JMS is brought up again.

Compile and Deploy

Using the RIB Installer or the RIB Application 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.


ORACLE
RETAIL
Retail Integration Bus Manager
rib-rms:RIB Adapter Manager
Page Refreshed Thu Nov 20 2008 16:01:17 GMT-0600 (Central Standard Time).
Home
Adapter Manager
Log Manager
RIB Logs
This page shows the RIB Adapters (publishers, subscribers,tafrs and/or hospitals) deployed on this RIB instance.
View All
Refresh Data
Start
Stop
Expand All | Collapse All
Select
Name
Status
Start Time
JMS Server ID
Edit Properties
View Log
Polling_Publishers
ASNOut Publisher, channel 1 jms2
Alloc Publisher, channel 1 jms3
Banner Publisher, channel 1 jms1
DiffGrp Publisher, channel 1 jms1
Diffs Publisher, channel 1 jms1
ItemLoc Publisher, channel 1 jms1
Items Publisher, channel 1 jms2
Items Publisher, channel 10 jms2
Items Publisher, channel 2 jms2
Items Publisher, channel 3 jms2
Items Publisher, channel 4 jms2
Items Publisher, channel 5 jms2
Items Publisher, channel 6 jms2
Items Publisher, channel 7 jms2
Items Publisher, channel 8 jms2
Items Publisher, channel 9 jms2
MerchHier Publisher, channel 1 jms1
Order Publisher, channel 1 jms1
Order Publisher, channel 10 jms1
Order Publisher, channel 2 jms1
Order Publisher, channel 3 jms1
Order Publisher, channel 4 jms1
Order Publisher, channel 5 jms1
Order Publisher, channel 6 jms1

7

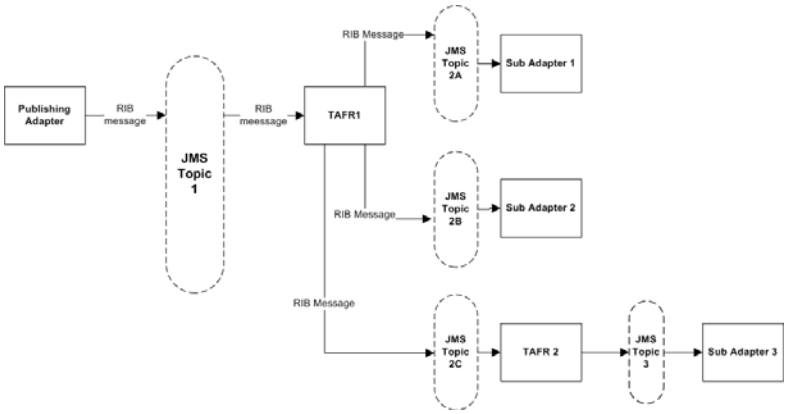
Message Transform, Filtering and Routing (TAFR)

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.

TAFR operation is specific to a message family and its set of subscribers. Multiple TAFRs may process a single message for a specific subscriber, and different specific TAFRs may be present for other subscribers. Separate sets of TAFRs are necessary for the various message families.

Multiple TAFRs may be needed, depending on the types of subscribers. The following diagram shows an example of 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 the 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 Java EE container is handled by the rib-app-builder application. Refer to 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.AllocToStockOrderFromRibBOIm
pl" />
</tafrs>
```

- message-driven—Indicates that the TAFR is deployed as an MDB.
- id—The ID for this particular adapter.
- InitialState—The state of the adapter.
- 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_tafr.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 tafr" 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.

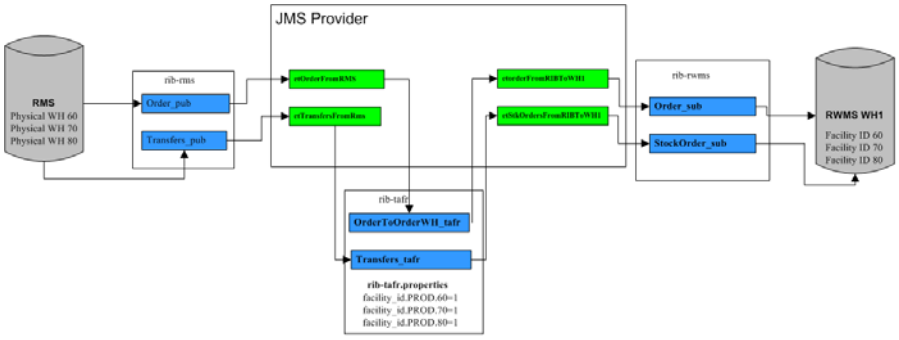
By default the standard RIB configuration is set for a single RWMS instance. This means that all physical warehouses in RMS route directly to a single RWMS instance (in this case denoted as WH1) with each RMS physical warehouse directly correlating to a facility ID in RWMS.

Configuration Process

Complete the following steps.

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 below, 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 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:

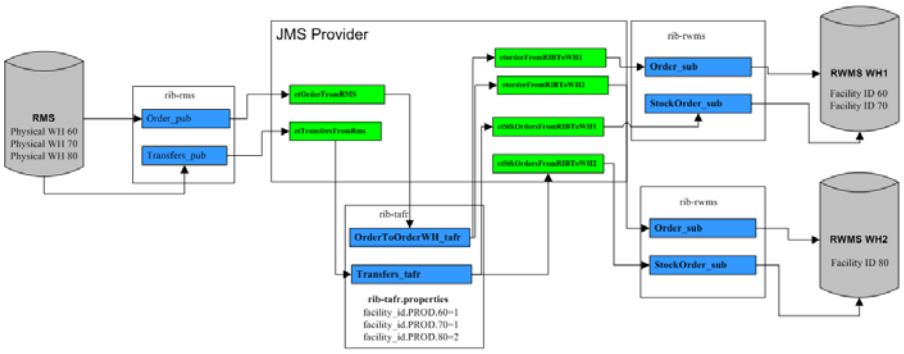
   The new TAFR routing settings must be migrated to the rib-tafr instance. Run the following script found in the $RIB_HOME/deployment-home/bin directory.

   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. RIB TAFRs 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

Complete the following steps.

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, among 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:
   - 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 adapter 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 adapter mappings for the new instance need to point to et<TOPIC_NAME>WH2 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-topic></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

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
- WOOutToWOOutWH

8

RIB in Operation

This chapter address common issues faced while operating the RIB.

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 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

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

Note: See Chapter 4, "RIB and JMX."

How to Configure Alerts and Notification

The RIB code has been instrumented to send alerts to notify interesting internal events like the following:

- Adapter status changes
- Dynamically changing runtime configuration
- Fatal error condition that needs user intervention
- Inconsistent persistence store because of user error

The following are the alerting mechanism supported in RIB:

- JMX – JSR-174, JSR-160
- Email - JavaMail

Any standard JMX client. (E.g. JConsole) can subscibe to RIB JMX notifications and receive notification/alerts. Follow the steps described in chapter 4 “RIB and JMX” to configure jconsole and view notification/alerts in jconsole.

The JMX Client can also be coded to monitor jmx notifications and act upon the notifications (sample code see below).

```
 public class Client {
/**
 * Inner class that will handle the notifications.
```

```
 */
public static class ClientListener implements NotificationListener {
public void handleNotification(Notification notification,Object handback) {
// logic to act upon notifications goes here
echo("\nReceived notification:");
...
echo("\tSource: " + notification.getSource());
echo("\tMessage: " + notification.getMessage());
}
}

public static void main(String[] args) throws Exception {
...
echo("Connecting to the remote server");
String protocol = "t3";
Integer portInteger = Integer.valueOf(portString);
int port = portInteger.intValue();
String jndiroot = "/jndi/";
String mserver = "weblogic.management.mbeanservers.runtime";
JMXServiceURL serviceURL = new JMXServiceURL(protocol, hostname, port,
jndiroot + mserver);
...
jmxc = JMXConnectorFactory.connect(serviceURL, h);
ClientListener listener = new ClientListener();
// Get an MBeanServerConnection
//
mbsc = jmxc.getMBeanServerConnection();
// Construct the ObjectName for adding listener to
ObjectName mbeanName = new ObjectName(
"rib-rms:appName=rib-rms,name=ribLogManager");
// Add notification listener on ribLogManager mbean
//
echo("\nAdd notification listener...");
mbsc.addNotificationListener(mbeanName, listener, null, null);
echo("\nWaiting for notification...");
echo("\nClose the connection to the server");
jmxc.close();
}

}
```

For email alerts, email IDs need to be configured at install time for RIB kernel to send out emails. The RIB deployment config file (rib-deployment-env-info.xml) contains notifications tag for each rib-<app> as below. Set all the values correctly to configure email alert for that app. RIB kernel sends out poison messages to email ids configured at install.

Example:

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

Note: RDMT also sends email alerts. For more information on configuring email alerts in RDMT, see the *RIB Support Tool Guide*.

RIB Log File Monitoring

Because 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 a particular site.

Log File Archive and Purge

RIB uses log4j2 for all of its logging control. It manages the logs size via its control file.

Note: See Apache Software Foundation `http://logging.apache.org/log4j2/2.x/manual/index.html` for details.

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

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 a particular site.

RMS MFQ and RWMS UPLOAD Tables Sizes

The MFQ and Upload table size and growth needs 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 a particular site.

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 components 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 retailer's site requirements.

It is always recommended that the order of startup be SUB, TAFR, PUB and the shutdown be in the reverse order. 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 Application 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

RIB contains code and configurations that are critical to the Enterprise. This version of 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."

RIB has an inventory tracking mechanism that is maintained by the tools in the RIB Application Builder. These tools also manage the application of defect fixes and tracking the defect fixes applied in the inventory.

Note: See "check-version-and-apply-defect-fix."

RIB HA Requirements

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® WebLogic Application Server High Availability Guide

Oracle® Database Administrator's Guide 12c Release 1

RIB Disaster Recovery

In addition to the HA requirements, there is the issue of message retention, auditing and recovery. It is common for an end-point application to experience an issue such as a crash that requires recovery or a 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."

RIB Administration 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 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 administrators who can 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 on 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 WebLogic 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

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

Areas of Responsibility

- Installation of WLS and patches
- Configuration of Oracle WebLogic Application Server
- Installation and configuration of RIB

- Support and configuration of Adapters and patches
- Operational issues such as backup/restore, failure analysis using RIBLOGS and Application 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 addressed.

Triage of messages placed in the RIB Hospital is a time consuming task. This is a difficult 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 dependent 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 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.

Oracle Retail Integration Bus Hospital Administration (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 that RIHA has to maintain 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, "Backend System Administration and Logging ."

9

Testing RIB

The Oracle Retail Integration Bus 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 RIB, a test harness has been developed and made available. The test harness is comprised of these components:

- **plsql-api-stub**—An API simulator of the PL/SQL API applications, RMS, RFM, and RWMS.
- **javaee-api-stubs**—An API simulator of the 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 that allows end-to-end auditing of a message as it passes through all RIB components.

Initially, installing and deploying RIB requires connecting to the Oracle Retail applications to verify that messages could flow end to end. 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 "RIB Logging."

RIB Test Harness

The ability to initially install and deploy 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. 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 testing 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 for RIB installation, configuration, and testing. These were developed to address the requirement for the full application to be present to validate a RIB installation as well as providing a tool for integration and system tests.

Master Checklist

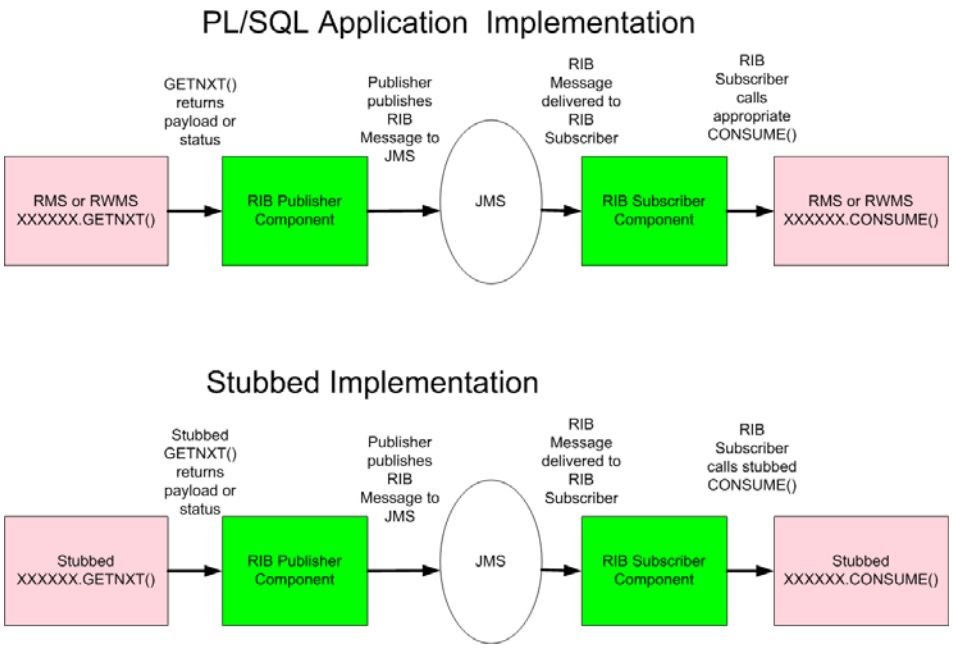
This check list covers all of the sequential steps required to create a stand-alone RIB Test Harness.

Task	Notes
Create the rib-home	Follow the guidelines in the *Oracle Retail Integration Bus Installation Guide* and the *Oracle Retail Integration Bus Implementation Guide* 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 *Oracle Retail Integration Bus Installation Guide*. See note about two stubs.
Deploy the javaee-api-stubs.	Follow the instruction in the tools section.
Install RIB using the stubs as application end-points.	See the *Oracle Retail Integration Bus Installation Guide*.

PL/SQL Application API Stubs

The plsql-api-stubs is an API simulator designed to act in the same manner as when 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 APIs to RIB: RMS, RFM, 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 RIB directly interfaces with. These interfaces map calls to the common subsystem to output parameters or statuses.
- The Stub Administration 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 a sequence called RMSMFM_ORDER_GE_SEQ created in the database.

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 executed 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 APIs internally call the Java functions present in the classes and the PL/SQL OUT parameters are mapped with the Java return types.

So when 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 that uses the common subsystem tables to get messages as a CLOB. It then converts the CLOB to an Oracle Object and maps it with the PL/SQL OUT parameters and returns.

The Stub Administration 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 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 APIs.

Configuration Files

The following are /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 storedprocedureметadatxml specified in the Stubby.properties file. **Note: Do not edit**.

Configuration File	Description
StoredProcedureMetaData_RWMS.xml	**Note: Do not edit**.
StoredProcedureMetaData_RMS.xml	**Note: Do not edit**.

Installation and Setup

Complete the following steps:

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 retail-public-payload-database-object-types.	retail-public-payload-database-object-types-<version>.jar 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. Note: Do not share the plsql-api-stubs schema with any other test-tools.	The user requires no special permissions. CREATE USER *<plsql stub user>* PROFILE "DEFAULT" IDENTIFIED BY *<plsql stub password>* DEFAULT TABLESPACE "USERS" TEMPORARY TABLESPACE "TEMP"; GRANT "CONNECT" TO *<plsql stub user>*; GRANT "RESOURCE" TO *<plsql stub user>*;
This version requires a path to jdk1.8 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 PlsqlApiStubs16.0.0ForAll16.x.xApps_eng_ga.tar	This will create the file folders and place the executables and configuration files. In rib-home/tool-home there is a directory already. It is a placeholder and this will over write it.
Place the database objects file in the scripts subdirectory	

Task	Notes
Extract the retail-public-payload-database-object-types-<version>.jar into the scripts directory. unzip retail-public-payload-database-object-types-<version>.jar	

Edit /conf/stubby.properties to point to the database url and user/password (see prerequisites).
vi stubby.properties
	# Database details
	hostname= <*host name*>
	port= <*port*>
	sid=ora12c
	dbuseralias=rms16dbuseralias

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 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. The PLSQL stubby needs its own schema. It cannot share its tools with the other test tools.
Install Hospital tables	See the *Oracle Retail Integration Bus Installation Guide*.
Enter the complete path for jdk1.7:	This version of stubby and the RDBMS require jdk1.7 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 Enterprise Integration Guide* and the operations guides for the RMS and RWMS applications.

Note: The database credentials setup must be complete before configuring the API.

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.	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. retail-public-payload-xml-samples-<version>.jar 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	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.	

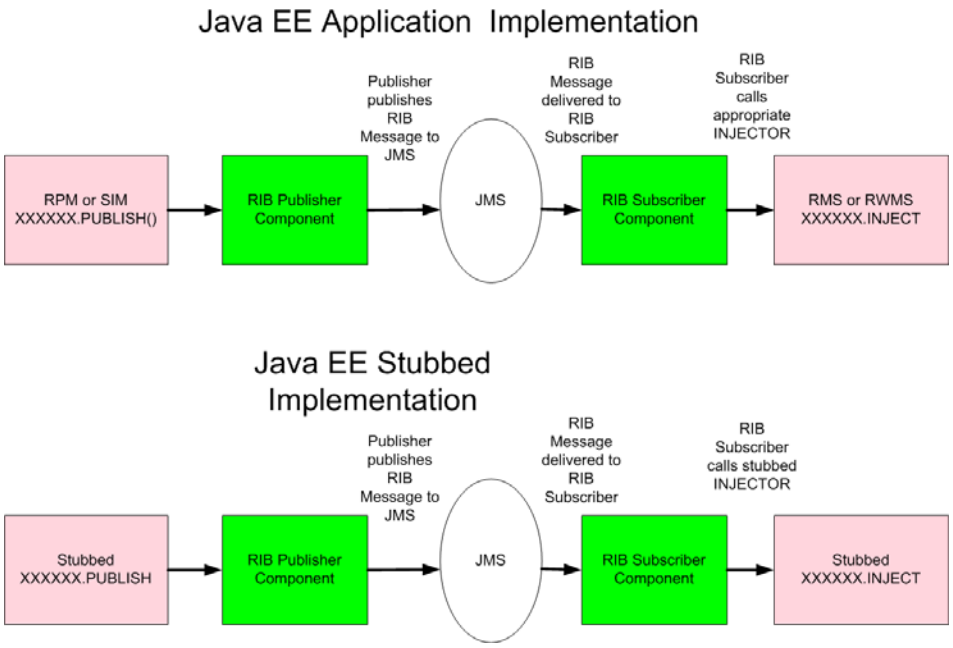
Prompts during configuration of a CONSUME.

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

Java EE Application API Stubs

The javaee-api-stubs is an API simulator designed to acts in the same manner as when 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 RIB: SIM, RPM, and AIP.

Architecture and Design



Installation and Setup

Complete the steps described below.

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-home/tools-home/javaee-api-stubs/rib-home/tools-home/ javaee-api-stubs
Get the latest version of the javaee-api-stubs.	The javaee-api-stubs is packaged as a stand-alone tar.

Task	Notes
Create a database user that will own the javaee-api-stubs objects.	The user requires no special permissions. CREATE USER <*javaee stub user*> PROFILE DEFAULT IDENTIFIED BY <*plsql stub user*> DEFAULT TABLESPACE USERS TEMPORARY TABLESPACE TEMP; GRANT CONNECT TO <*javaee stub user*>; GRANT RESOURCE TO <*javaee stub user*>;

Installation

Task	Notes
Determine the WebLogic instance to which to deploy the javaee-api-stubs-<version>.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-<version>.ear.	See WebLogic deployment documentation for more details on how to deploy a Java EE application.
Using the WebLogic console, configure the database resources for the javaee-api-stubs JDBC resources. ■ Log in to the WebLogic administration console ■ Navigate to the Data Sources sceen 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
Create one more data source named javaee-api-stubs-xa-managed-datasource. 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-xa-managed-datasource JNDI Name: jdbc/OracleRibDs Database Type: Oracle Database Driver: Oracle's Driver (Thin XA) ■ 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 javaee-stubby (for example, javaee-stubby-instance). Click **Finish**. ■ Verify that both data sources are listed on Services > JDBC > Data Sources screen.	
Install Hospital tables	See the *Oracle Retail Integration Bus Installation Guide*.

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

Task	Notes
Decide which rib-<app> to configure for.	The stubbed implementation has been written to insert the payload to a database once inject has been called. Injectors.xml has been configured to include all the SIM subscribing families.

Using RIB Application Builder or the RIB Installer configure and deploy the rib-app using the jndi information of the javaee-api-stubs in place of the app.	`<app id="sim" type="javaee-app">`
	`<jndi>`
	`<url>t3://javaeestubhost.example.com`
	`:<port>/javaee-api-stubs</url>`
	`<factory>weblogic.jndi.WLInitialContextFa`
	`ctory</factory>`
	`<user-alias>sim_jndi_`
	`user-name-alias</user-alias>`
	`</jndi>`
	`</app>`

10

Performance Considerations

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

Performance of RIB within a customer site is critical to the performance of the business, and is determined by factors specific to a given deployment. Because of this is, a Performance Test is recommended as 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 Integration Bus Implementation Guide*.

Performance Factors

The performance of each of these components affects the overall performance of the system:

- Application Server topology and configuration.
- RIB deployment approach.
- Hardware sizing and configuration of the following:
  - RIB hosts
  - Applications connected to RIB
  - JMS provider host
  - RIB Hospital 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

For each RIB message family, volume requirements are almost always described for the end-to-end message flow, from publication to 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 its own 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, because 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 for many Message Families.

The following are examples of 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: Although these examples are for illustration, they 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. There is a RIB 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

RIB is designed to support parallel message handling to increase throughput by way of a mechanism called multi-channel, which logically partitions the flow of messages within the JMS topic so that multiple publishers and subscribers can simultaneously

use the same JMS topic without contention or interference and preserve publication message ordering within the logical channel.

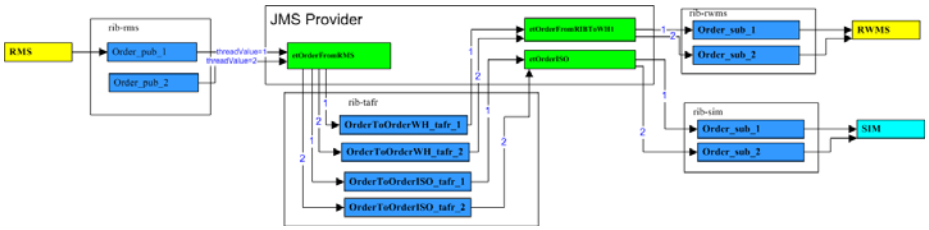
Every adapter instance of a publisher, subscriber, or TAFR configured in RIB is considered to belong to a logical channel for processing messages. Multi-channel adapters are multiple adapter instances for the same message family, each processing messages asynchronously and in parallel. When multiple 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 included with the message. Message properties have types to 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-channeled by making configuration changes to the system. The base RIB configuration 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

RIB 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 straightforward measurement.

Message throughput 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 throughput.

How to Calculate Average Message Size

It is important to understand the average messages in an integration flow. Where interfaces are separated into messages families with differing payloads per message

type, these calculations can be difficult. This section outlines an approach for arriving at averages using the sample XML files that ship with RIB.

RIB delivers sample files generated for each message family.

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 RIB. These messages can then be used to estimate the average sizes.

The RIB message envelope, called a RibMesssages 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.

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 or no payload	1147 Bytes

Each message family is comprised of message type and an associate payload (for example, POCre uses PODesc.xsd). These relationships are defined in the *Oracle Retail Enterprise 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. Be aware that this relationship varies by family and can be complex for some message types (for example, 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 details. This will be the detail size.

Example Message - PODesc	
Header Size (PODesc no detail)	9413

1 Detail Size (PODtl)	1943

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

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 PODtl	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 (PODtl)	1943
Total 1 RibMessages + 1 ribMessage + 1 PODesc + 1 PODtl	12,053

The following is an example using the default settings.

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).

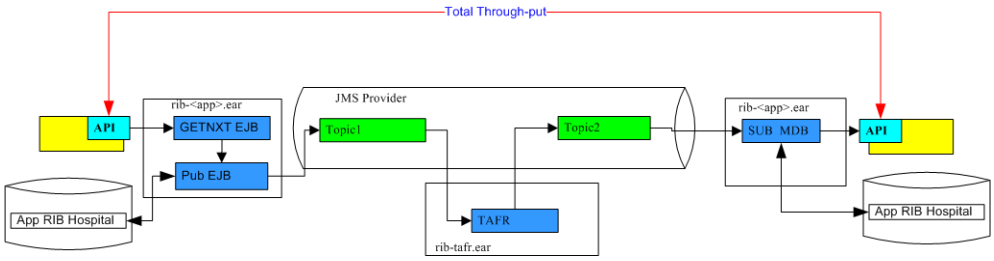
So:

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

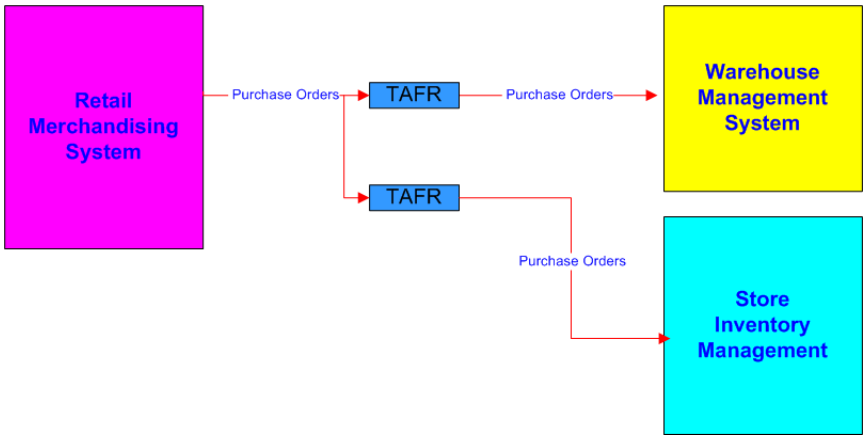
These are end-to-end processing time requirements across the entire message flow from Publisher to Subscription completion.

The following diagram is a generic message flow.

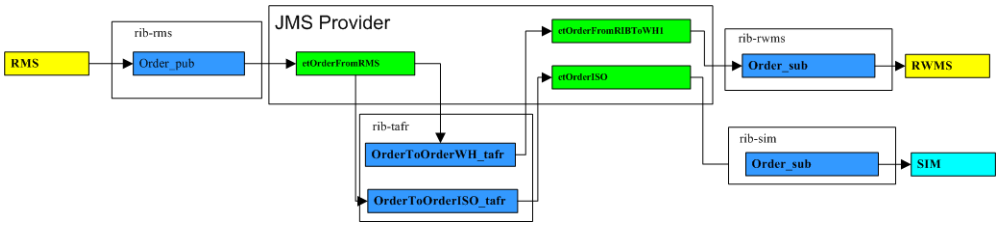


To continue the Purchase Order example, the requirements and timings have to be broken down further. The Purchase Order flow has a TAFR as well as multiple subscribers. For purposes of this example, consider the Subscribers Consume times as equivalent. As the diagram depicts, for a flow like the Purchase 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

RIB performance is a complex subsystem to measure. It involves not only host level performance, but database, network, and application server subsystems performance. To measure the RIB components' timing characteristics available for analysis, the RIB kernel code logs events as it processes them. The logging of these events is through log4j2; timings are logged per adapter. Once the timings are enabled the events log 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 predefined 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.

	Timing Type	Description
T6	SUB_TIME_IN_CONSUME_CALL	Time taken by the CONSUME call to the plsql application.
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 application.
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 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 requires much higher counts, a broad spectrum of time, and system load. Other factors include average size of message.

Key Interfaces to Consider

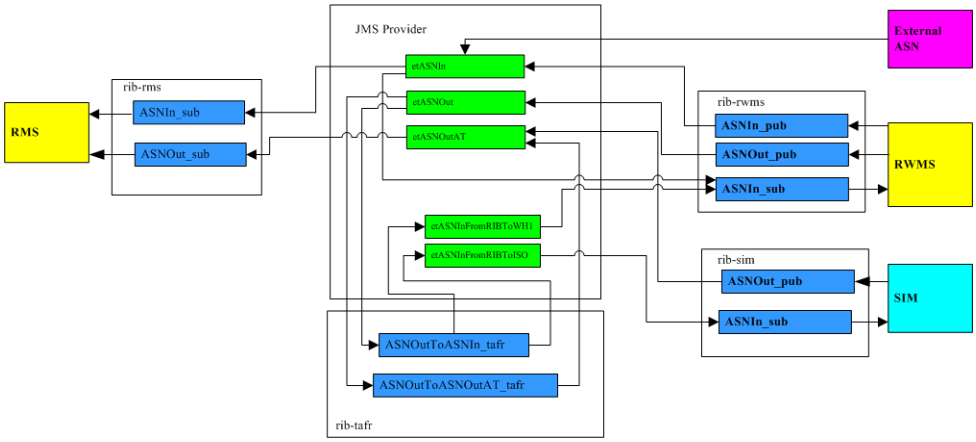
Every customer site has unique requirements and flows, so the ones to focus on will vary. However, there are ones that always make to the list.

- 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).

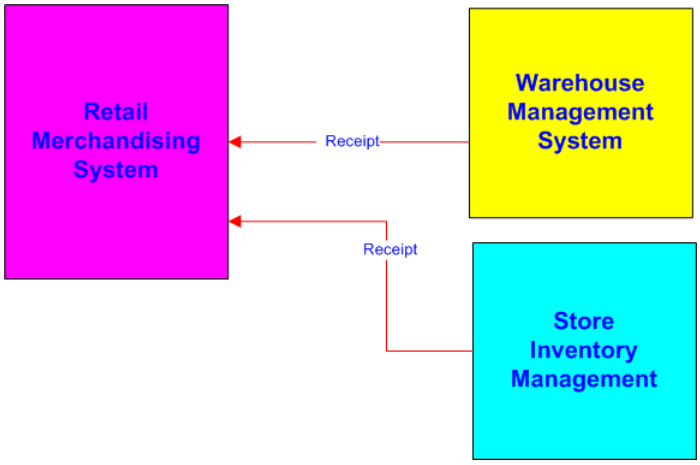
ASN (Inbound/Outbound)

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

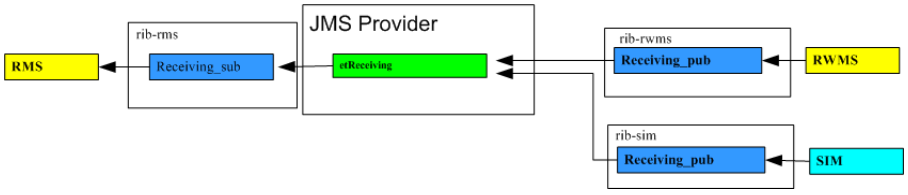


Receipts

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



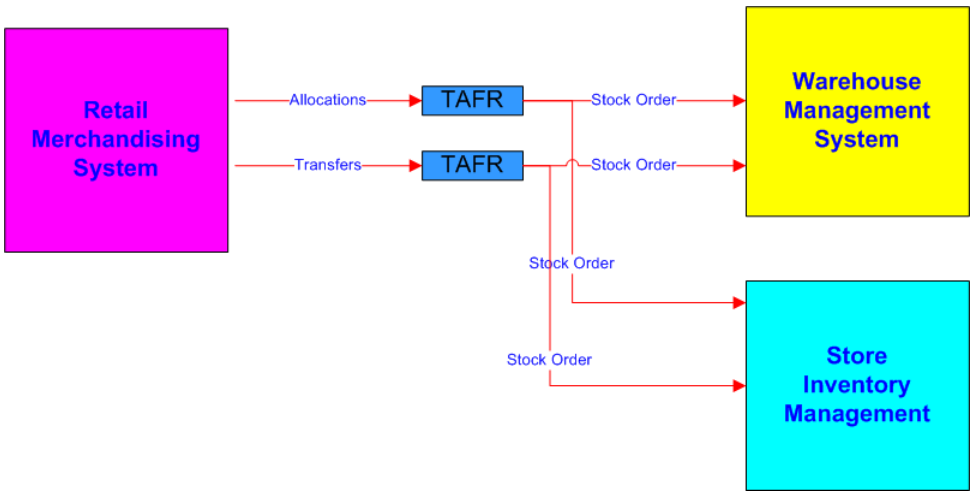
The following diagram is functional, detail view of the Oracle Retail Receipts Flow.



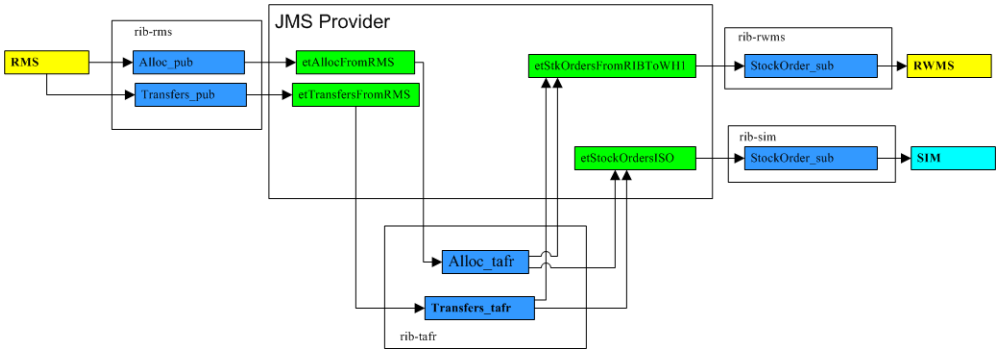
The Receipts message family is transactional data, and often a candidate for performance testing. Receiving consists of appointment and receipt messages that are published to RIB for RMS providing open to buy visibility. An appointment is information about the arrival of merchandise at a location. A 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.

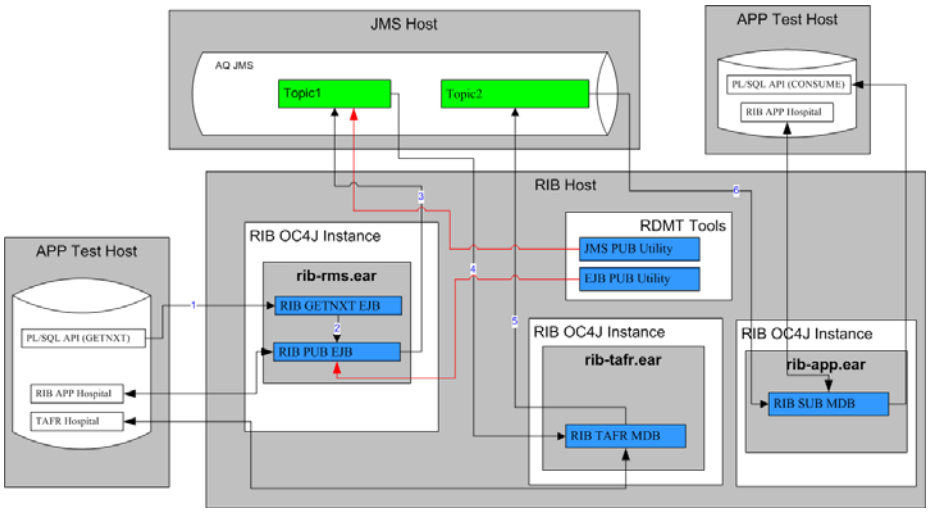
There are tools available in RDMT to assist in this, as well as the audit feature of RIB. By enabling audit on an interface all messages are saved to a file in a form that can be played back by RDMT utilities.

The API Simulators (PL/SQL and Java EE) allow the focus to be on 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 RIB components and the application APIs.

The following illustrates the RIB Performance Test Harness.



Tools supplied 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 for measuring the flow end-to-end.

1. Prepare for the run. Use RIB Administration GUI to do the following.
   - Stop all adapters (PUB, SUB, TAFR).
   - Archive all logs so that the run has clean logs.
   - Enable timings logs (DEBUG) on all adapters.
   - Set all other adapter logs to INFO.
2. Determine how to generate the messages.
   - Using the Oracle Retail Application (for example, 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 Administration 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 creates 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 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 application 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 Application 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.

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 publishing APIs support multi-channel processing:

- ASNOUT Publication API
- Allocations Publication API
- Delivery Slot Publication API
- Fulfill Order Confirmation 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
- Transfers Publication API
- Work Orders in Publication API
- Work Orders out Publication API

The following RMS 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 Thread Value

Each messaging RIB component involved in publishing or subscribing to a logical channel is distinctly identified by a JMS Message property known as "Thread Value" 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 Thread Value property and this value as part of its JMS Durable Subscriber selector and each RIB publisher sets the "Thread Value" 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 Thread Value of 1 (one). The naming convention and the RIB kernel code identify 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

The following is the basic process for configuring a multi-channel flow.

1. Determine the family to configure as 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.

Back up 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

For RIB-RMS, complete the following steps.

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

For RIB-TAFR, complete the following steps.

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_tafr_1.name=AllocToStockOrder TAFR, channel 1
Alloc_tafr_1.desc=TAFR for converting Allocation messages to StockOrders and
routing them to the correct warehouse or store system

Alloc_tafr_2.name=AllocToStockOrder TAFR, channel 2
Alloc_tafr_2.desc=TAFR for converting Allocation messages to StockOrders and
routing them to the correct warehouse or store system

Alloc_tafr_3.name=AllocToStockOrder TAFR, channel 3
Alloc_tafr_3.desc=TAFR for converting Allocation messages to StockOrders and
routing them to the correct warehouse or store system
```

```
Alloc_tafr_4.name=AllocToStockOrder TAFR, channel 4
Alloc_tafr_4.desc=TAFR for converting Allocation messages to StockOrders and
routing them to the correct warehouse or store system

Alloc_tafr_5.name=AllocToStockOrder TAFR, channel 5
Alloc_tafr_5.desc=TAFR for converting Allocation messages to StockOrders and
routing them to the correct warehouse or store system
```

RIB-SIM

For RIB-SIM, complete the following steps.

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"/>
```

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

For RIB-RWMS, complete the following steps.

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"/>
```

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 5 for the Alloc Family.

Compile and Deploy

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

Message Aggregation

To improve message publication throughput within the integration system, 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) apply 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>.

Understand that the bigger the payload size, the bigger the memory requirement. A process (JVM) has 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

To configure message aggregate, complete the following steps.

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>

   Note: The value for <family> must be entered in all capital letters. For example, VENDOR.

3. Using the app-builder tool compile/deploy the application.
   - rib-app-compiler.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
maxNodesPerMessagess=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.

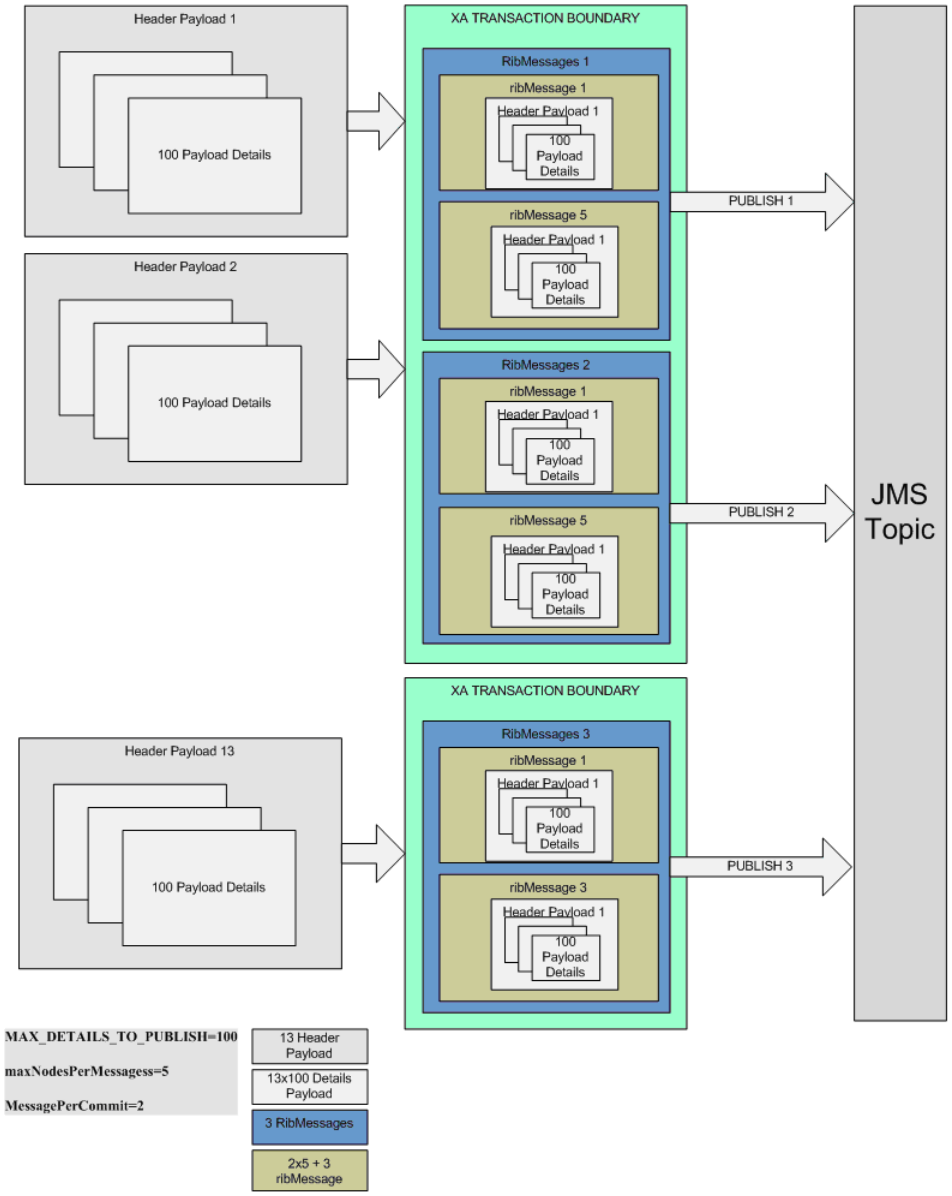
XA transaction 1 = (RibMessages1 + RibMessages2)
RibMessages1 = ribMessage1 + ribMessage2 + ribMessage3 + ribMessage4 + ribMessage5.
ribMessage1 = PayloadHeader + 100 * PayloadDetail
ribMessage2 = PayloadHeader + 100 * PayloadDetail
…..
ribMessage5 = PayloadHeader + 100 * PayloadDetail

RibMessages2 equivalent to RibMessages1

XA Transaction 2 = RibMessages3
RibMessages3 = ribMessage1 + ribMessage2 + ribMessage3

Total = (XA Transaction 1 + XA Transaction 2)
100*5 + 100*5 + 100*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

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 messages with a reason code of SUB only. 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 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

The following is a 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.ErrorHospitalRetryTimerTask"/>
                             </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 Application Builder command line tools, compile and deploy the new rib-<app>.ears.