Oracle® Retail Integration Bus

Operations Guide

Release 13.0

April 2008

ORACLE®

Oracle Retail Integration Bus Operations Guide, Release 13.0

Primary Author: Ross Steinberg

Value-Added Reseller (VAR) Language

(i) the software component known as **ACUMATE** developed and licensed by Lucent Technologies Inc. of Murray Hill, New Jersey, to Oracle and imbedded in the Oracle Retail Predictive Application Server - Enterprise Engine, Oracle Retail Category Management, Oracle Retail Item Planning, Oracle Retail Merchandise Financial Planning, Oracle Retail Advanced Inventory Planning and Oracle Retail Demand Forecasting applications.

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

(iii) the **SeeBeyond** component developed and licensed by Sun MicroSystems, Inc. (Sun) of Santa Clara, California, to Oracle and imbedded in the Oracle Retail Integration Bus application.

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

(v) the software component known as **Crystal Enterprise Professional and/or Crystal Reports Professional** licensed by Business Objects Software Limited ("Business Objects") and imbedded in Oracle Retail Store Inventory Management.

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

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

(viii) the software component known as **Style Report™** developed and licensed by InetSoft Technology Corp. of Piscataway, New Jersey, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(ix) the software component known as **WebLogic™** developed and licensed by BEA Systems, Inc. of San Jose, California, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(x) the software component known as **DataBeacon™** developed and licensed by Cognos Incorporated of Ottawa, Ontario, Canada, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

Contents

3 Backend System Administration and Logging

4 RIB and JMX

5 Admin GUI

6 JMS Provider Management

7 Message Transform, Filtering and Routing (TAFR)

8 Diagnostic and Monitoring Tools

9 The RIB in Operation

10 Testing the RIB

11 Performance Considerations

Preface

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

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

Audience

This operations guide is designed for System Administrators, Developers, and Applications Support personnel. Its purpose is to provide a basic understanding of the Oracle Retail Integration Bus components, how messages flow between them, and the operational activities surrounding these components. It also provides templates for using the RIB as an alternative to FTP batch jobs for transferring files from one system to another.

Related Documents

For more information, see the following documents in the Oracle Retail Integration Bus Release 13.0 documentation set:

- *Oracle Retail Integration Bus Release Notes*
- *Oracle Retail Integration Bus Installation Guide*
- *Oracle Retail Integration Bus Hospital Administration User Guide*
- *Oracle Retail Integration Bus Hospital Administration Online Help*
- *Oracle Retail Integration Bus Data Model*
- *Oracle Retail Integration Bus Integration Guide*
- *Oracle Retail Integration Bus Implementation Guide*

Customer Support

- https://metalink.oracle.com

When contacting Customer Support, please provide:

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

Review Patch Documentation

For a base release (".0" release, such as 13.0), Oracle Retail strongly recommends that you read all patch documentation before you begin installation procedures. Patch documentation can contain critical information related to the base release, based on new information and code changes that have been made since the base release.

Oracle Retail Documentation on the Oracle Technology Network

In addition to being packaged with each product release (on the base or patch level), all Oracle Retail documentation is available on the following Web site:

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

Documentation should be available on this Web site within a month after a product release. Note that documentation is always available with the packaged code on the release date.

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.

Third-Party Open-Source Applications

Oracle Retail Security Manager includes the following third-party open-source applications:

Software Provider: lo4j

Software Name: log4j

Software Version: Unknown

Jar File Name: log4j.jar

Provider Web Site:

http://logging.apache.org/log4j/docs/index.html

License: Apache

Software Provider: Apache XML Project

Software Name: xerces

Software Version: Unknown

Jar File Name: xercesImpl.jar

Provider Web Site:

http://xerces.apache.org/xerces-j

License: Apache

Software Provider: Apache XML Project

Software Name: xerces

Software Version: Unknown

Jar File Name: xml-apis.jar and/or xmlParserAPIs.jar (one and the same, with xmlParserAPIs.jar being deprecated)

Provider Web Site:

http://xerces.apache.org/xerces-j

License: Apache

Software Provider: GNU

Software Name: gsort (renamed from gnu sort)

Software Version: Unknown

Provider Web Site:

http://www.gnu.org/software/textutils/textutils.html

License: GPL

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 Application Server

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

Note: See Oracle® Application Server Administrator's Guide 10g Release 3 (10.1.3.3)

Oracle Retail RIB Supplied Components

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

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

- RIHA is the RIB Hospital administration tool.

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 RIB Implementation Guide - Pre-Implementation Considerations

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

RIB Application Builder Directory Structure

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

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

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

The rib-home is a controlled structure and there are very specific rules for using the tools and the key files 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.

NOT ALLOWED: rib-rms.properties.bak

Directory Structure and Key Files

Example 2–1

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

```
rib-func-artifact-gen
riha
```

RIB App Builder Tools

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

Logging

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

Backup and Archive of Key Files

The rib-app-builder tools will automatically generate a backup when a patch is installed. Additionally, it is recommended that each site develop a backup plan that includes a regular backup at the file system level of the rib-app-builder directory structure.

rib-app-compiler

The rib-complier is the tool that drives the rib-<app>.ear creation process. It performs validation of the input xml files. There are four xml files are used to build the rib-<app>.ear. These input files are:

- 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, and then collects all of the generated files and packages them to create a deployable rib-<app>.ear file.

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

rib-app-deployer

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

Table 2–1 Command Line Options to rib-app-deployer

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" later in this guide.

Table 2–1 (Cont.) Command Line Options to rib-app-deployer

Command Line Option	Description
-deploy-rib-func-artifact-war	Deploys the rib-func-artifact.war to the Java EE application server defined in rib-deployment-env-info.xml. The Java EE server must be running.
-deploy-rib-app-ear rib-<app>	Deploys the rib-<app>.ear to the Java EE application server defined in rib-deployment-env-info.xml. The Java EE server must be running.
-update-remote-rib-app-config-files rib-<app>	Updates the rib-<app> application level configuration files in the remote server where rib-<app>.ear is or will be deployed. The remote server information is defined in rib-deployment-env-info.xml. The Java EE server must be running.
-undeploy-rib-func-artifact-war	Undeploys the rib-func-artifact.war from the Java EE application server defined in rib-deployment-env-info.xml. The Java EE server must be running.
-undeploy-rib-app-ear rib-<app>	Undeploys the rib-<app> from the Java EE application server defined in rib-deployment-env-info.xml. The Java EE server must be running.

Check-version-and-unpack

This tool will verify the version compatibility between the RIB paks and extract the files if they are compatible. 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 that is present 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 same. All should have same number of major and minor versions.

How Verifications work:

1. The tool 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. The tool reads the functional artifact file from rib-home/download-home/rib-func-artifacts.
3. The tool reads the list of all the RIB application packs from the -home/download-home/all-rib-apps directory is read.
4. The tool makes use of the naming convention to check if the kernel version is the same as the functional artifact version. If the version is compatible, the tar file is un-tar'd into the rib-home/application-assembly/ rib-func-artifacts directory.
5. The tool makes use of the naming convention to check if the kernel version is the same as the application packs. If the version is compatible, the tar file is un-tar'd into the rib-home/application-assembly/rib-<app> directory.

check-version-and-apply-defect-fix

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

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

```
------------------------------------------------------------
    Product     : Oracle Retail Integration Bus
    Version #   : 13.0.x
    Defect #                          : 1789
    Date        : 02/27/2008
------------------------------------------------------------

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

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

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

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

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

inventory-management

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

Table 2–2 Command Line Options to inventory-management

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.

Table 2–2 (Cont.) Command Line Options to inventory-management

Command Line Option	Description
-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.

rib-adapter-controller

The rib-adapter-controller a set of tools that perform RIB adapter control functions such as start/stop and subscriber check, The command line options and usage are summarized here. See the section, "RIB Components Start and Stop", in this manual.

Start Flow

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

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

Description:

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

Examples:

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

Stop Flow

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

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

Description:

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

Examples:

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

Start Adapters By Type

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

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

Description:

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

Examples:

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

Stop Adapters by Type

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

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

Description:

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

Examples:

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

Start Adapter

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

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

Description:

1. Checks if durable subscribers exist before starting an adapter.

2. Starts the adapter instance.

Examples:

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

Stop Adapter

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

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

Description:

- Stops the adapter instance.

Examples:

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

Test Durable Subscriber for Adapter

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

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

Description:

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

Examples:

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

Test Durable Subscriber for RIB App

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

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

Description:

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

Examples:

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

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

List RIB App Adapters

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

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

Description:

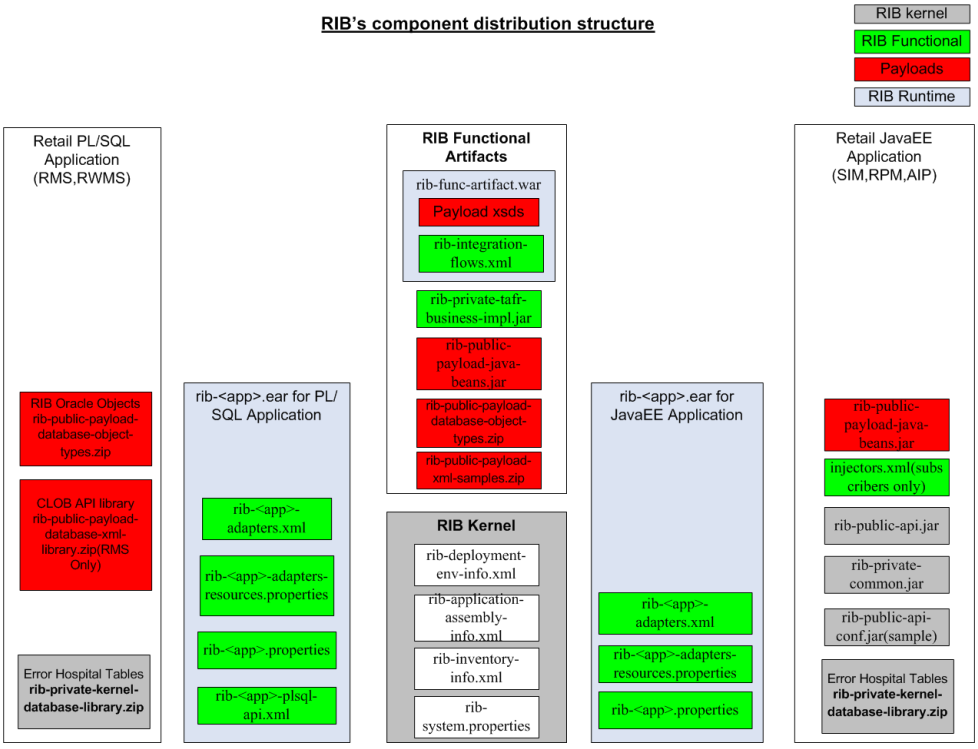
- Lists all adapters that are part of the rib-app.

Examples:

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

3

Backend System Administration and Logging



This figure depicts the actual files that comprise the RIB and where they are located in the deployment picture.

rib-<app>-adapters.xml

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

The file is located in the rib-home/application-assembly/rib-<app> directory. After deployment it is found in the path $application_instance_home/$application_name where $application_instance_home is the oc4j instance path where the application is deployed.

These are the standard RIB defined adapter types.

<subscribers> element

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

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

Note: Running and stopped are the ONLY valid states and are case sensitive.

<publisher> element

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

<timer-driven>

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

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

<request-driven>

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

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

<hospital> element

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

```
<hospitals>
       <timer-driven id="sub_hosp_0" initialState="running" timeDelay="10">
              <timer-task>
                    <class name="com.retek.rib.j2ee.ErrorHospitalRetryTimerTask"/>
                           <property name="reasonCode" value="SUB"/>
              </timer-task>
       </timer-driven>
```

rib-<app>-adapters-resource.properties

These are internationalizable strings for internal rib adapter key names.

Examples:

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

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

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

rib-<app>-plsql-api.xml

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

rib-<app>.properties

These are internationalizable strings for internal rib adapter key names.

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

rib-system.properties

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

Property Name and Default Value	Description
hospital_attempt_max defaultValue = "5";	This property refers to the maximum number of attempts to try to push this record through the RIB automatically. Once this retry count is exceeded, the message remains in the RIB Hospital DB but is no longer retried automatically
hospital_attempt_delay defaultValue = "10";	This property refers to the value (in seconds) used to calculate the next attempt time.
hospital_attempt_ delayIncrement defaultValue = "10";	This property refers to the value (in seconds) used to calculate the next attempt time. The next attempt time is calculated as: hospitalAttemptDelay + (hospitalAttemptDelyIncrement * attempt count). This is done so that the delay between each attempt is longer than the previous delay.
numOfRecordsToRetry defaultValue = "20";	This property refers to the maximum number of RIB Hospital records to be retried in one RIB Hospital retry attempt.
xml_schema_base_url defaultValue = "http://localhost:8888/rib-func-artifact";	This property refers to the location of web application (rib-func-artifact) which has RIB related XML Schema (XSD) files.
mail_smtp_host defaultValue = "mail.smtp.host";	This property is used to identify the smtp host from which to send out emails.
mail_smtp_port defaultValue = "25";	This property is used to identify the smtp port from which to send out emails.
mail_smtp_from defaultValue = "admin@company.com";	This property refers to the email id that the RIB platform needs to use to send the emails for administrative purposes.
war_http_port defaultValue = "9080";	This property refers to the port number used by the web based Hospital Retry Admin Tool.

rib-integration-flows.xml

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

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

Example:

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

```
                adapter-class-def="StockOrder_sub" type="JmsToDb">
                 <in-topic>etStockOrdersISO</in-topic>
                 <out-db>default</out-db>
          </node>
          <node id="rib-rwms.StockOrder_sub" app-name="rib-rwms"
                adapter-class-def="StockOrder_sub" type="JmsToDb">
                 <in-topic>etStkOrdersFromRIBToWH1</in-topic>
                 <out-db>default</out-db>
          </node>
    </message-flow>
```

rib-deployment-env-info.xml

This file is the single source of all values used in the RIB App Builder tools and is the only file that requires editing or should be, for using them. The RIB Installer will gather the appropriate values from the user and will construct the file, and then invoke the appropriate tools.

The RIB Application Builder tools can be executed independent of the RIB installer tool and in some cases it is necessary to edit this file manually.

The rib-deployment-env-info.xml file is divided into four major sections.

app-in-scope-for-integration

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

Example:

```
<app-in-scope-for-integration>
   <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="aip" type="javaee-app"/>
</app-in-scope-for-integration>
```

rib-jms-server

This section defines the JMS server information.

Example:

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

rib-application-server

This section defines the Oracle Application Server information.

Example:

```
<oas-instance-name> AS4.linux1.localdomain </oas-instance-name>
<oas-instance-home>soa1@linux1:/home/soa1/product/10.1.3.1/OracleAS_
6</oas-instance-home>
<oas-opmn-request-port>6003</oas-opmn-request-port>
<oas-ohs-port protocol="http" >7777</oas-ohs-port>
<java-home>/usr/java/jdk1.5.0_01</java-home>
```

rib-javaee-containers

This section defines the oc4j instances for each of your rib-<app> applications that are in-scope.

Example:

```
<oc4j id="rib-rms-app-server-instance">
    <oc4j-instance-name>rib-rms-oc4j-instance</oc4j-instance-name>
    <oc4j-instance-home>soa1@linux1:/home/soa1/product/10.1.3.1/OracleAS_
4/j2ee/rib-rms-oc4j-instance</oc4j-instance-home>
   <oc4j-user>riboc4jadmin</oc4j-user>
   <oc4j-password>riboc4jadmin</oc4j-password>
</oc4j>
```

rib-applications

This section defines the rib-<app> specific information for each of the rib-<app> that is in scope.

Example:

```
<rib-app id="rib-rms" type="plsql-app">
            <url>jdbc:oracle:thin:@linux11521:soa1</url>
            <user>rms13en</user>
            <password>rms13</password>

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

For JavaEE applications you need to define the connecting retail application's JNDI information.

```
          <jndi>

<url>opmn:ormi://linux1:6003:sim-oc4j-instance/javaee-api-stubs</url>
              <factory>oracle.j2ee.rmi.RMIInitialContextFactory</factory>
              <user>oc4jadmin</user>
              <password>welcome1</password>
          </jndi>
```

commons-logging.properties

There are several - insert locations and use. The RIB uses the Apache Commons Logging subsystem as the logging interface. For RIB deployments this file should not be edited.

log4j.xml

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

rib-app-builder-paths.properties

For RIB deployments this file should not be edited.

rib-application-assembly-info.xml

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

RIB Logging

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

For additional information on how the RIB uses log4j, see "log4j.xml".

Log Level Recommendations

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

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

> **Note:** See the section,"RIB Timings Utility" , in this manual.

As a rule the appropriate level will always be INFO.

Changing Logging Levels

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

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

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

Adapter Logging (RIBLOGS)

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

Example:

```
/home/rib/product/10.1.3.3/OracleAS_6/j2ee/rib-rms-oc4j-instance/log/rib-rms
```

The RIBLOG filenames are in the format <adapter-instance-name>.rib.log.

Example:

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

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

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

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

RIB Timing Logs

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

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

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

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

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

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

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

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

Before:

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

After:

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

RIB Audit Logs

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

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

```
<!--Audit Appender for adapterInstance: ASNIn_sub_1-->
    <appender class="org.apache.log4j.FileAppender"
name="appender.rib.sub.audit.ASNIn_sub_1">
        <param name="File" value="/u00/webadmin/product/10.1.3/OracleAS_
1/j2ee/rib-rms-oc4j-instance/log/rib-rms/ASNIn_sub_1.audit.log"/>
        <!--param name="MaxFileSize" value="2048KB"/-->
        <!--param name="MaxBackupIndex" value="1"/-->
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="&lt;audit-entry
audit-time=&quot;%d{yyyy.MM.dd
HH.mm.ss,SSS}&quot;&gt;%n%m%n&lt;/audit-entry&gt;%n"/>
        </layout>
    </appender>

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

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

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

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

Before:

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

After:

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

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

Sample Log Entry:

```
<audit-entry audit-time="2008.01.28 11.37.57,642">
<?xml version="1.0" encoding="UTF-8"?>
<RibMessages
  xmlns="http://www.oracle.com/retail/integration/rib/RibMessages"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oracle.com/retail/integration/rib/RibMessages
http://mspdev85:7777/rib-func-artifact/integration/xsd/RibMessages.xsd" >
<ribMessage><family>Banner</family><type>BannerCre</type>    <id>1</id>
    <ribmessageID>Banner_pub_1|2008.01.28 11:37:57.500|6936</ribmessageID>
    <publishTime>2008-01-28 11:37:57.500 CST</publishTime>
 <messageData>&lt;BannerDesc
xmlns=&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://mspdev81:7777/rib-func-artifact/payload/xsd/BannerDesc.xsd
http://www.oracle.com/retail/integration/payload/RIBDate
http://mspdev81:7777/rib-func-artifact/payload/xsd/RIBDate.xsd&quot;&gt;
&lt;banner_id&gt;1&lt;/banner_id&gt;  &lt;banner_name&gt;B&amp;amp;M&lt;/banner_
name&gt;&lt;/BannerDesc&gt;
</messageData>
<customData></customData><customFlag>F</customFlag>
</ribMessage>
</RibMessages>
```

Other RIB Management Logs

deploy.rib.log This log will track the source rib-app-builder home that pushed the changes to this OC4J instance.

Example:

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

management.rib.log The RIB maintains a management log which is used to keep track of the oc4j instance on the whole.

This is usually written during the startup of an oc4j instance. The recommendation is that each rib-app be deployed in a separate oc4j 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.adapt
ers.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

Overview

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

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

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

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

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

Note: See RIB Implementation Guide – Java Management Extensions (JMX)

RIB JMX Client

The RIB provides a command line interface to the RIB JMX system. The client is shipped as java classes in the jmx-cmd-line-ui.jar. The entry point is JmxClientMain and allows the user to either execute a single JMX command or run JMX commands in an interactive shell. There is a menu selection that invokes the interactive shell feature. See Chapter 8, "Diagnostic and Monitoring Tools" for examples of how to interface to this utility.

User Interface

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

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

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

getattribs ObjectName [attribute1] [attribute2]..
Example: getattribs oc4j:j2eeType=JVM,name=single,J2EEServer=standalone
javaVersion freeMemory

help [command]

invoke ObjectName methodName pramType1 paramValue1 [pramType2 paramValue2]...

Example: invoke
oc4j:j2eeType=JNDINamespace,name=JNDINamespace,J2EEServer=standalone
getAllBindingsAsXMLString
Example: invoke oc4j:j2eeType=JVM,name=single,J2EEServer=standalone getproperty
java.lang.String java.library.path
list objectName
Example: list *:*

close
Example: close

connect jmxServiceUrl jmxUser jmxPassword jmxConnectionProtocolProviderPackage
Example: connect service:jmx:rmi://localhost:23791/oc4j oc4jadmin oc4jadmin
oracle.oc4j.admin.jmx.remote

Previous command successful: JmxCommand(help).
JMXClient>
```

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

5

Admin GUI

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

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

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

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

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

RIB Admin URLs

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

RIB Admin GUI

`http://<server>.us.oracle.com:<http-port>/rib-<app>-admin-gui/`
Replace <server> with the name or IP address of the server in the environment that has the rib-<app> deployed.

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

Replace <app> with either:

- rms
- tafr
- rwms
- sim

RIB Functional Artifacts

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

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

RIB Message Flows

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

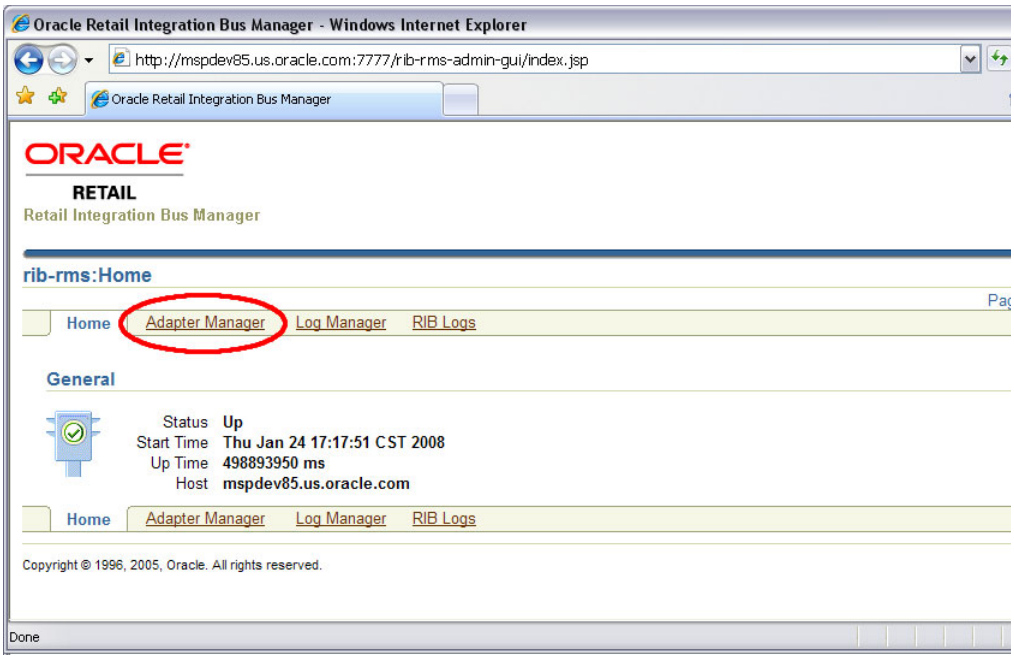
RIB Payloads (xsds)

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

RIB Admin GUI Home

Home Screen

Click the "Adaptor Manager" link to view all adaptors for the given application.



Adapter Manager

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

Adapter Manager Screen

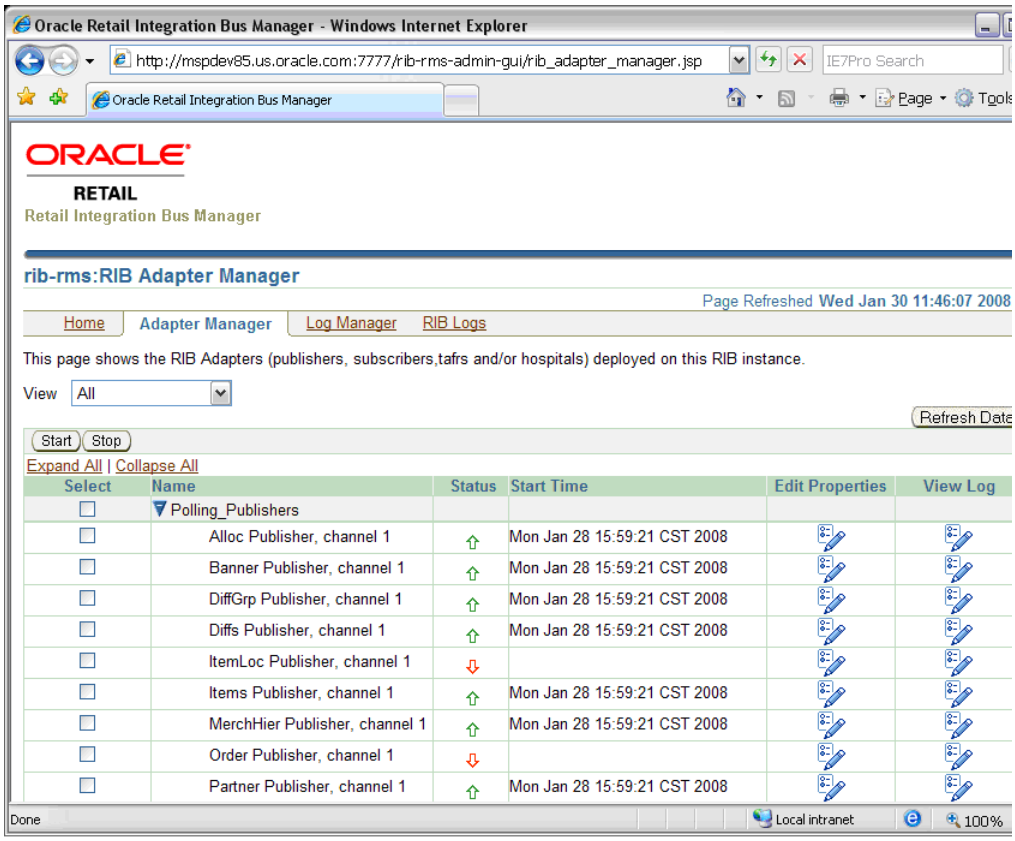
This screen shows you the current status of all adapters for the specified application. A

signifies that an adapter is up and running where as

means that the adapter is offline or has shut itself down. From this screen you can start and stop any listed adapters by selecting the check-box related to the adapter and then using the

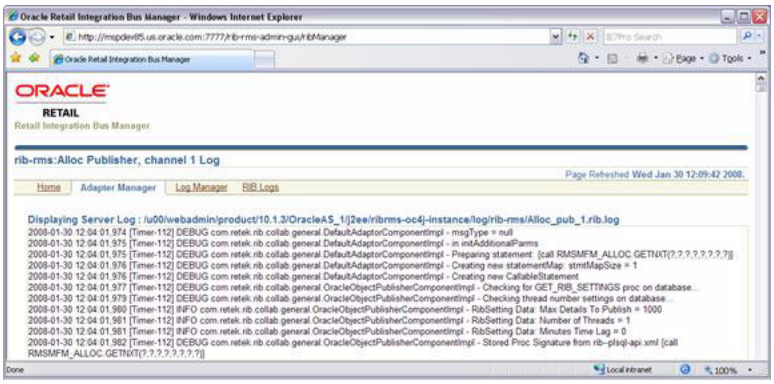
buttons. Clicking the

button in the "View Log" column takes you 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. However it will always be shown.



Log Manager

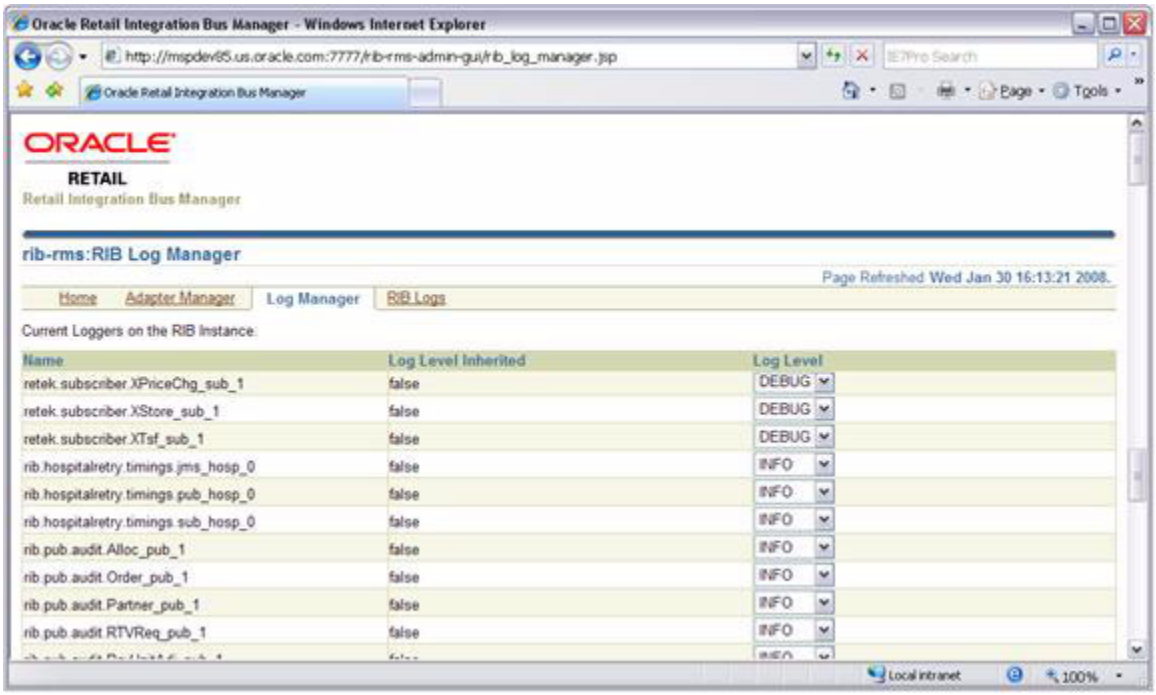
Log Manager Screen

This screen enables the user to change the logging level of the adapters. It also allows the user to enable audit and timings logging.

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

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

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

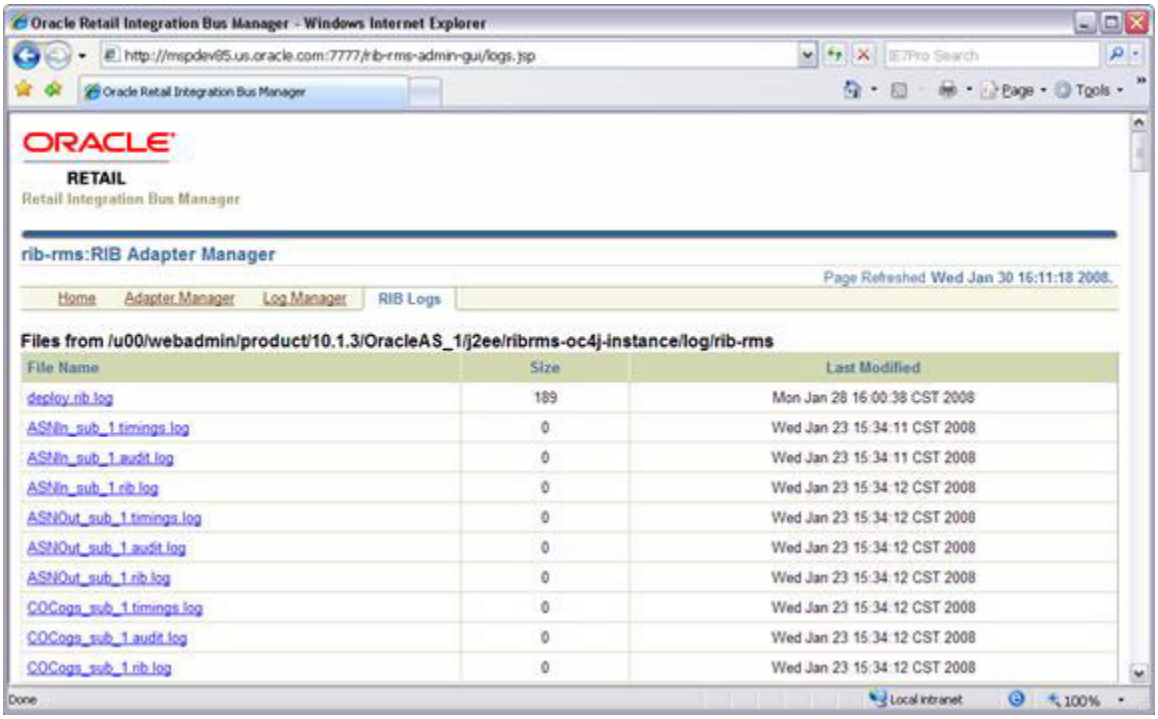


RIB Logs

RIB Logs Screen

This screen can be used to view the regular adapter log file (also accessible by clicking the

button in the "View Log" column on the Adaptor Manager screen) along with the Timings and Audit logs for each adapter, if they have been activated (see instructions in the "Log Manager Screen").



6

JMS Provider Management

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

The RIB is designed to be JMS provider agnostic and will be certified with several, starting with the OEMS JMS Database persistence option; which is the JMS interface to the Oracle Database Streams Advanced Queuing (AQ) feature. Subsequent releases will add certification of the OC4J JMS (for the file and memory-persistence version) that is bundled with the Oracle Application Server, as well as other JMS standard providers.

For more details on OEMS, see the Oracle® Containers for J2EE Services Guide 10g - Using Oracle Enterprise Messaging Service.

The RIB on AQ JMS

The AQ JMS is a database and need to be installed, configured and tuned to support the anticipated transaction loads for a retailers production message volumes. It is beyond the scope of the RIB documentation to provide specific details or guidelines.

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

- AQ is I/O intensive. Pay close attention to the disk layout.
- AQ JMS as used by the RIB will have high transaction rates. Consider this in configuration of the redo logs.

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

Queue Monitor Process Setup

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

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

- aq_tm_processes=4
- alter system set aq_tm_processes=4

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

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

Optimizing Enqueue/Dequeue Performance

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

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

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

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

Note: See also Oracle® Database Administrator's Guide 10g Release 2 (10.2), Oracle® Streams Advance Queuing User's Guide, and Reference 10g Release 2 (10.2)

7

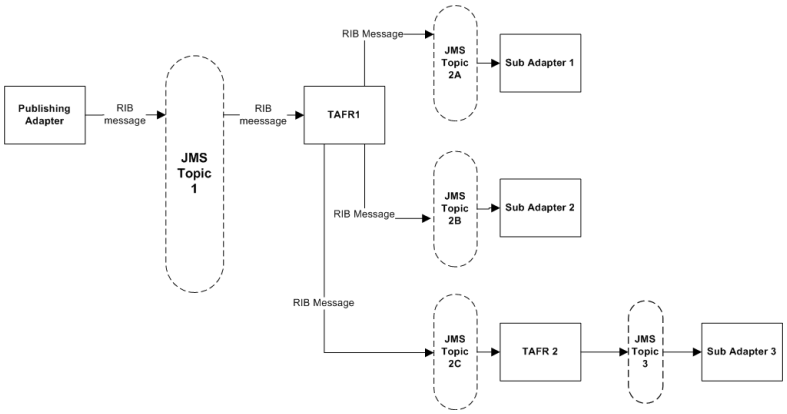
Message Transform, Filtering and Routing (TAFR)

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

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

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

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



TAFR Adapter Process

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

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

Configuration

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

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

- message-driven - Indicates that the TAFR is deployed as an MDB.
- id - Indicates the id for this particular adaptor.
- InitialState - Indicates the state of the adaptor.
- Tafr-business-impl - Indicates 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

This 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,ItemImageMo
d,ItemImageDel,ItemUdaDateCre,ItemUdaDateMod,ItemUdaDateDel,ItemUdaFfCre,ItemUdaFf
Mod,ItemUdaFfDel,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

Description

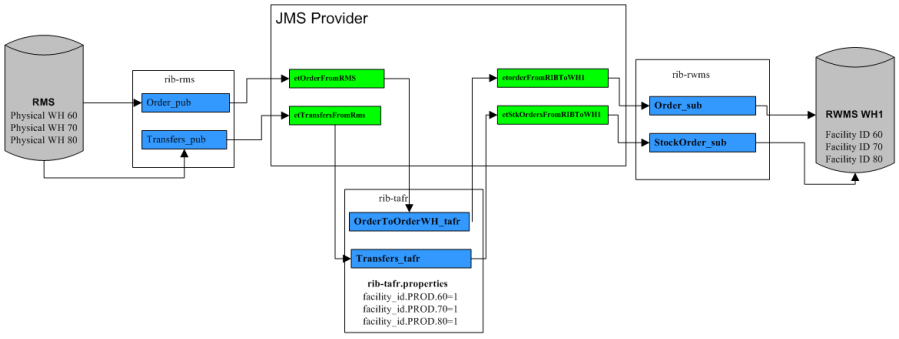
RIB allows stock based transactions to be routed between different RWMS instances. An RWMS instance is assigned to a physical distribution center which may have one or more facilities assigned to it. A company may have one or more distribution centers.

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

Configuration Process

1. Modify the TAFR routing settings:
   - For each physical warehouse set up in RMS there should be a matching entry in the rib-tafr.properties file. This file resides in the $RIB_HOME/application-assembly-home/rib-tafr directory and is used by the TAFR adapters, amongst other things, to route messages by facility ID to the correct RWMS instances.

- The file by default contains the following mappings:
  - facility_id.PROD.1=1
  - facility_id.PROD.2=1
  - facility_id.PROD.3=1
- The routing properties are structured in the following way: facility_id.<FACILITY_TYPE>.<RMS_PHYSICAL_WH_ID>=<RWMS_INSTANCE_NAME>
  - <FACILITY_TYPE> - This should match the facility_type.default value in the rib-tafr.properties file, in most cases this would be left as the default value which is "PROD".
  - <RMS_PHYSICAL_WH_ID> - The physical warehouse ID from RMS.
  - <RWMS_INSTANCE_NAME> - The RWMS installation topic name identifier that the warehouses messages will be routed to.
- These mappings need to be edited so that each physical warehouse in RMS has its own entry. The physical warehouses can be found by running the following query in the RMS schema:
  - SELECT wh FROM wh
    WHERE wh.wh = wh.physical_wh;
- For the example in the diagram above, physical warehouse IDs 60, 70 and 80 would be returned by the query.
- There is only one RWMS instance (WH1) in this example and the RWMS installation topic name identifier is simply "1". This corresponds to the name of the topics that the RIB routes the messages to. This is also the default name suffix of the RWMS topics in the rib-integration-flows.xml file.



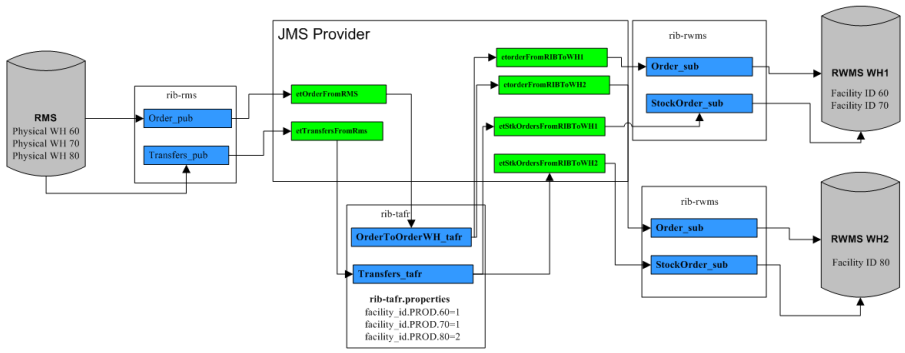
- Therefore, our mapping in the rib-tafr.properties file should read:
  - facility_id.PROD.60=1
  - facility_id.PROD.70=1
  - facility_id.PROD.80=1

2. Deploy the settings to the rib-tafr instance:
   - The new TAFR routing settings 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

3. Configuration should now be complete.

> **Note:** For every new physical warehouse added to RMS, the rib-tafr.properties file requires a new entry and the new settings need to 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 will be shipped to or from a certain warehouse to the distribution center that contains that warehouse (facility).

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

Configuration Process

1. Modify the TAFR routing settings:
   - For each physical warehouse set up in RMS there should be a matching entry in the rib-tafr.properties file. This file resides in the $RIB_HOME/application-assembly-home/rib-tafr directory and is used by the TAFR adapters, amongst other things, to route messages by facility ID to the correct RWMS instances.
   - The file by default contains the following mappings:
     - facility_id.PROD.1=1
     - facility_id.PROD.2=1
     - facility_id.PROD.3=1

- The routing properties are structured in the following way: facility_id.<FACILITY_TYPE>.<RMS_PHYSICAL_WH_ID>=<RWMS_INSTANCE_NAME>
  - <FACILITY_TYPE> - This should match the facility_type.default value in the rib-tafr.properties file, in most cases this would be "PROD".
  - <RMS_PHYSICAL_WH_ID> - The physical warehouse ID from RMS.
  - <RWMS_INSTANCE_NAME> - The RWMS installation topic name identifier that the warehouses messages are routed to.
- These mappings need to be edited so that each physical warehouse in RMS has its own entry. The physical warehouses can be found by running the following query in the RMS schema:
  - SELECT wh FROM wh
    WHERE wh.wh = wh.physical_wh;
- Before editing the file for multiple RWMS instance routing, the user should know which RMS physical warehouses are to be routed to the particular RWMS instances and the RWMS installation topic name identifiers.
- For the example in the diagram above physical warehouse IDs 60 and 70 are routed to RWMS instance WH1 where the RWMS installation topic name identifier is "1" and RMS physical warehouse ID 80 are routed to RWMS instance WH2 where the RWMS installation topic name identifier is "2". To support this the mapping in the rib-tafr.properties file should read:
  - facility_id.PROD.60=1
  - facility_id.PROD.70=1
  - facility_id.PROD.80=2

2. Modify the rib-integration-flows.xml file:
   - The RIB requires information on how to route the messages between the two RWMS instances. This is done by adding new entries to the rib-integration-flows.xml file.
   - By default the file contains entries for the RWMS instance "rib-rwms" and all appropriate warehouse based adaptor mappings point to the et<TOPIC_NAME>WH1 topics. When adding multiple RWMS instances all the entries for RWMS need to be duplicated for the second instance "rib-rwms2" and all adapter mappings for the new instance need to point to et<TOPIC_NAME>WH1 topics.
   - The entire "RWMS PUBLISHERS" section in the integration-flows.xml file needs to be duplicated and all new entries need to be changed to the second RWMS instance name of "rib-rwms2" for example:
     - <node id="rib-rwms2.ASNIn_pub" app-name="rib-rwms2" adapter-class-def="ASNIn_pub" type="DbToJms"><in-db>default</in-db><out-topic>etASNIn</out-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 me 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 me migrated to the rib-func-artifact instance, to do this run the following script found in the $RIB_HOME/deployment-home/bin directory.
     - rib-app-deployer.sh -deploy-rib-func-artifact-war
5. Configuration should now be complete.

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

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

Changes to this configuration affect the following TAFRS.

- AllocToStockOrder
- ASNOutToASNInLoc

- CustOrderToStockOrder
- ItemLocToItemLocLoc
- OrderToOrderWH
- PendReturnToPendReturnWH
- RTVReqToRTVReqLoc
- TransfersToStockOrder
- WOInToWOInWH
- WOOutToWOOutWH

8

Diagnostic and Monitoring Tools

Overview

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

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

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

Functionality

- Support for Oracle AS - OC4J RIB Version.
- Support for both local/remote installation.
- Support for Oracle Streams AQ JMS as the JMS Provider.
- Support for RIB Hospital databases.
- Support for JMX control of all RIB Components
- Support for message Pub/Sub.

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

RDMT and User Roles and Responsibilities

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

The primary target role is the RIB administrator as someone who is responsible for the installation, configuration, and deployment of the RIB components. The ribadmin is also usually then tasked with ongoing RIB Software Life Cycle management as well as production operation support. This person has full permissions on all of the application server directories and has full read and execute permissions on the Oracle Application Server tools such as opmnctl and the OC4J instance subdirectories.

Local or Remote Installations and Capabilities

RDMT can be installed by a user on the system that may or may not have the RIB/OAS environment. RDMT tools support both local and remote OC4J functions via JMX.

In remote installs, some scripts in the toolkit expect the installing user to have read permissions of the OAS home RIB OC4J sub-directories or require execute permission of opmnctl. Therefore, these will return file or permissions errors.

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

RDMT Support jars

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

Sample XML Messages

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

Tools Overview

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

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

RDMT as an Application

SCRIPTDIR

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

Setup

An installation script is provided that determines most of the environment values needed and prompts for the RIB deployment environment specific values. All of the scripts have been designed to be configuration driven by property files. The setup process updates these files.

Current Configuration

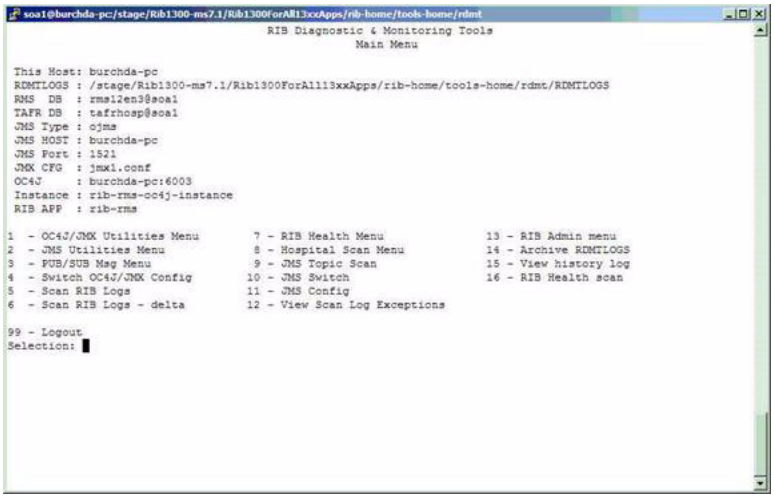
Because there are multiple configurations possible with the fully distributed RIB, all of the tools are designed to work against a set of property files that provide the values need to execute. Collectively these are call "current". In the menu system there are functions that allow configuration of n-number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

RDMTLOGS

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

RDMT Main Menu

This is the main entry point into the RDMT tool kit application. Most selections invoke other submenus, but for convenience several tools, include in other submenus, are directly accessed from this menu.



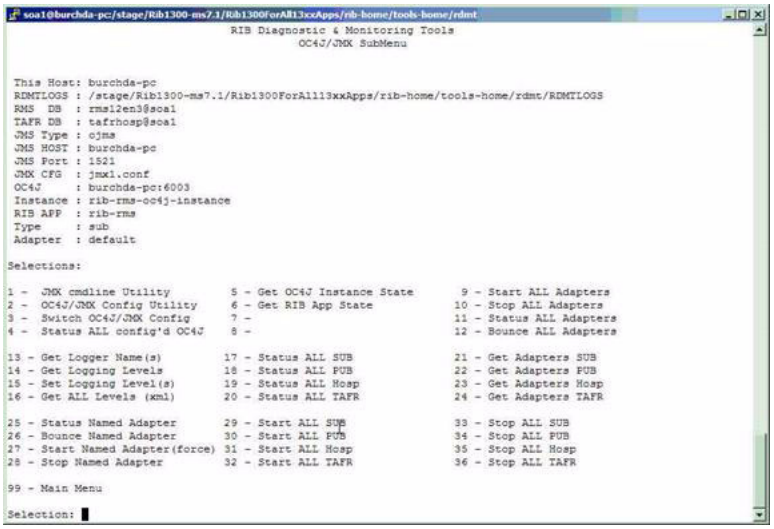
OC4J/JMX Utilities

Script Used:

rdmt_jmx_submenu

Description:

This menu option exposes the various tools that use JMX to interact with the OC4J instance and to control or status the current rib-<app> and its components.



Script	Description
jmx_app_state.sh	status of the currently active rib application
jmxcmdline_interactive.sh	A wrapper to the jmx client support classes. This script directly invoke the interactive functions.
jmxcmdline.sh	General wrapper for other tools to invoke specific jmx functions.
jmxconfig.sh	There are multiple configurations possible with the fully distributed J2EE RIB. This utility is used to manage the configuration files that allow the rdmt tools to access them. This option can also be used to switch/re-configure the previously configured OC4J/JMX configuration.
jmx_get_logger_names.sh	RIB app logging tool
jmx_get_logging_levels_all.sh	RIB app logging tool
jmx_get_logging_levels.sh	RIB app logging tool
jmx_set_logging_levels.sh	RIB app logging tool
jmx_managed_adpaters.sh	Common script used by all jmx tools to interact with the jmx client jar. Many of the menu selections merely set the calling parameters to this tool.
jmx_OC4Jribstatus.sh	Get the run state of the rib-app OC4J instance and application for all configured.
jmx_OC4Jrib_scan.sh	For all configured rib-app scans the state of the instance, app and adapters.
jmx_oc4j_state.sh	Status of the currently active OC4J instance.
jmx_switch_config.sh	This utility is used to switch the active configuration file that the rdmt tools use.
jmx_tester.sh	Test script for testing arbitrary jmx commands within the RDMT framework. This is not a menu selection since it requires user editing.

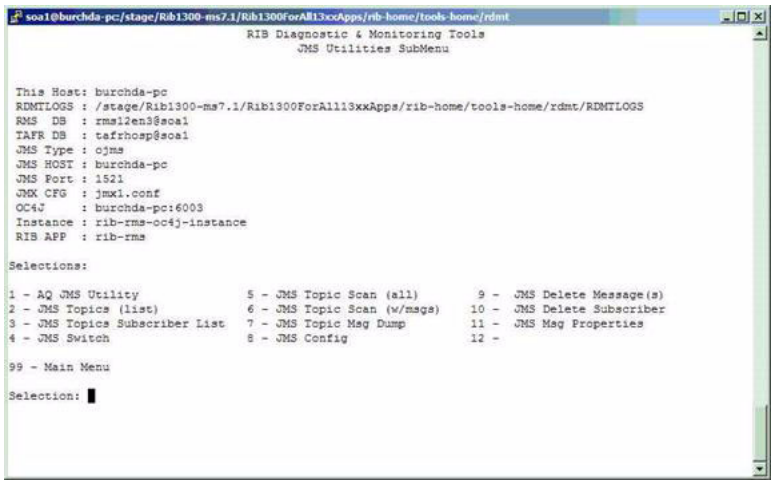
JMS Tools

Script Used:

rdmt_jmsutil_AQ_submenu

Description:

This menu option exposes the various JMS functionalities available in the tool kit. For convenience some tools in other submenus are presented here as well.



Script	Description
aqjmscmdline.sh	Common script used by all JMS tools to interact with the AQ JMS client jar. Many of the menu selections merely set the calling parameters to this tool.
deletemsgAQ.sh	Delete message(s) on a specified JMS topic for a specified subscriber.
dmpmsgAQ.sh	Dump a message(s) on an AQ JMS topic for a specified subscriber.
dmp_msg_statsAQ.sh	Dump properties of message(s) on an AQ JMS topic for a specified subscriber.
jmsconfig.sh	RDMT supports configuration of n-number of JMS Providers. This utility configures the values to support. rdmt_jms_submenu is used to make one the current configuration.
jmstopicsAQ.sh	Query the AQ JMS for all of the topics and the message count on each topic.
jmstopicsAQ_scan.sh	Query the AQ JMS for just the topics with message counts.
jmsutil.sh	This utility provides direct access to the AQ JMS java API utilities.

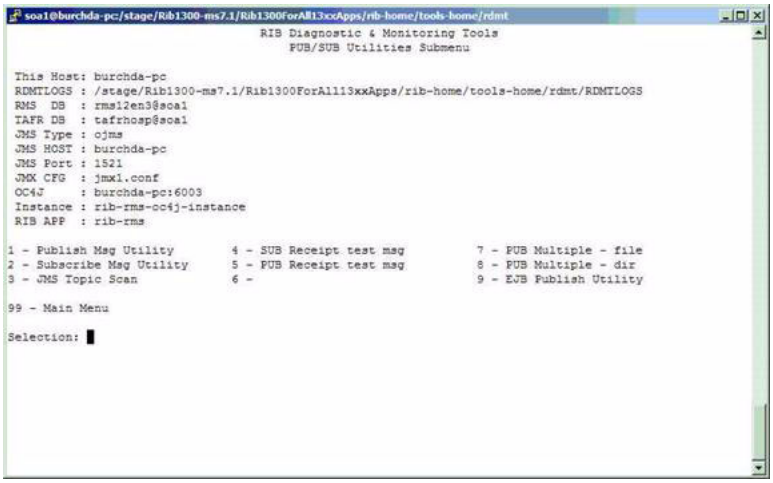
PUB/SUB Msg Tools

Script Used:

rdmt_msgutil_submenu

Description:

All of the tools is this menu are wrappers that expose functions in the java utilities rib-jms-api.jar included in the tool kit library. These are general purpose pub/sub functions that are written to support the various JMS Providers for the RIB, e.g. AQ JMS.



Script	Description
pubmsgutil.sh	Provides direct access the java api by prompting for all of the expected command line values it expects.
submsgutil.sh	Provides direct access the java api by prompting for all of the expected command line values it expects.
submsg.sh	Higher level wrapper that uses the configuration values to shortcut the values needed to call the java api's.
pubmsg.sh	Higher level wrapper that uses the configuration values to shortcut the values needed to call the java api's.
pubmsgutil_multiple.sh	Script that supports multi-message publication.
pubmsgutil_directory.sh	Script that supports multi-message publication.
ejbpub_util.sh	utility to wrapper the EJB Message Publish java api.

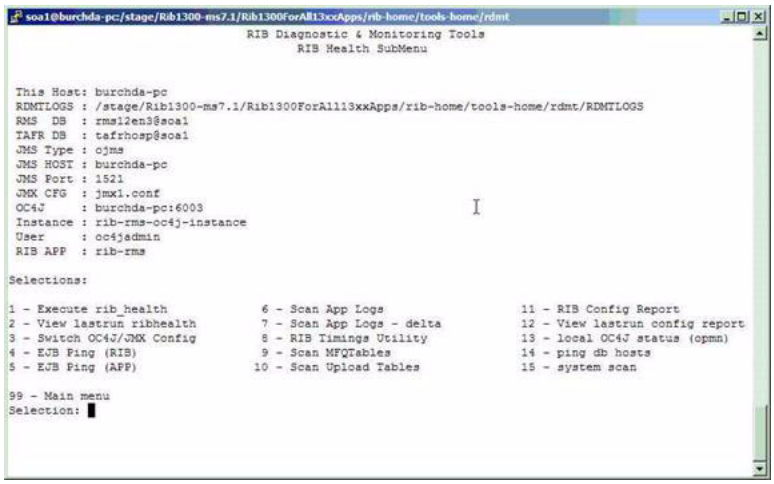
RIB Health Tools

Script Used:

rdmt_ribhealth_submenu

Description:

This option leads the user to the submenu through which the user can get the current RIB health status.



Script	Description
cron_ribhealth.sh	See the section, "RIB Health", in this chapter.
ribejbping.sh	See the section, "EJB Ping (RIB)", in this chapter.
appejbping.sh	See the section, "EJB Ping (RIB)", in this chapter.
loglookoc4j.sh	See the section, "Scan RIB Logs / Scan RIB Logs (Delta)", in this chapter.
loglookoc4j_delta.sh	See the section, "Scan RIB Logs / Scan RIB Logs (Delta)", in this chapter.
timingsutil.sh	See the section, "RIB Timings Utility", in this chapter.
ttestrms.sh	This script scans a list of RMS MFQ tables using a JDBC connection. (see mfqtables.conf).
ttestrdm.sh	This script scans a list of RWMS Upload tables using a JDBC connection. (see uploadtables.conf).
OC4JConfigReport.sh	See the section, "RIB Configuration Report", in this chapter.

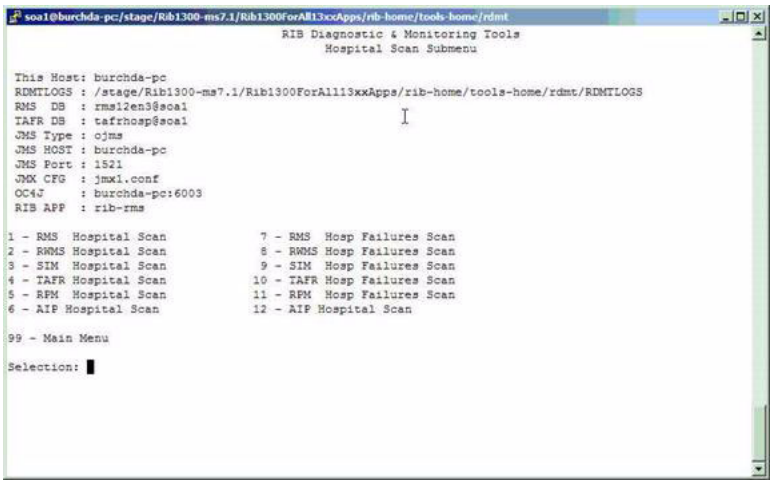
Hospital Scan Tools

Script Used:

rdmt_hosp_submenu

Description:

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



Script	Description
htest.sh	This script calls a java class that uses JDBC to access the database(s) containing the Hospital tables. It scans the Hospital RIB_MESSAGES table and report thing such as: ▪ how many messages (row count), ▪ how many have exceed the retry count, ▪ how many messages of a topic as present.
htest_failures.sh	This script calls a java class to scan the RIB Hospital Message Failure table using a JDBC connection.

RIB Admin Tools

Script Used:

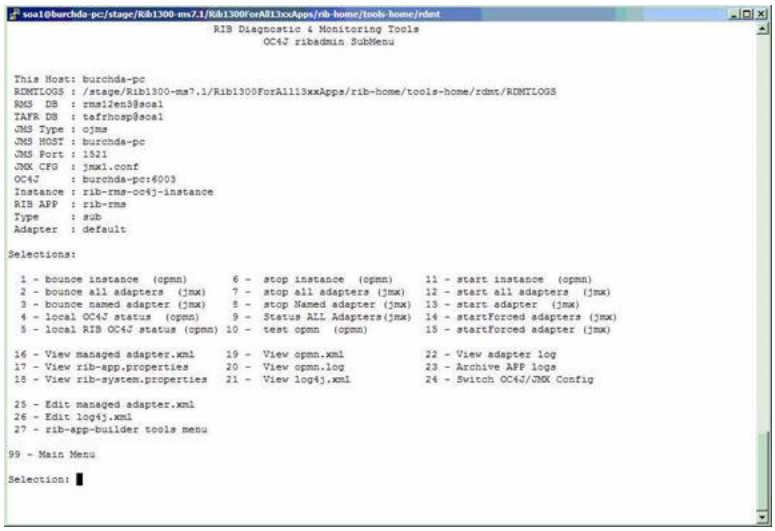
rdmt_ribadmin_submenu

Description:

The ribadmin script was stand-alone in previous RIB releases. In RIB 13.0, those functions have been moved into this menu item. The ribadmin.sh script is sourced to make the existing functions available to the menu items and the variables that the scripts expected have been mapped to rdmt configuration files.

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

If RDMT is installed in the RIB App Builder rib-home and that is accessible and configured, then this menu exposes the rib-app-builder menu selection. A test is performed to verify the rib-home is configured, if not, then the selection will not appear.



Script	Description
ribadmin.sh	This script contains most all of the functions that are exposed by this menu.

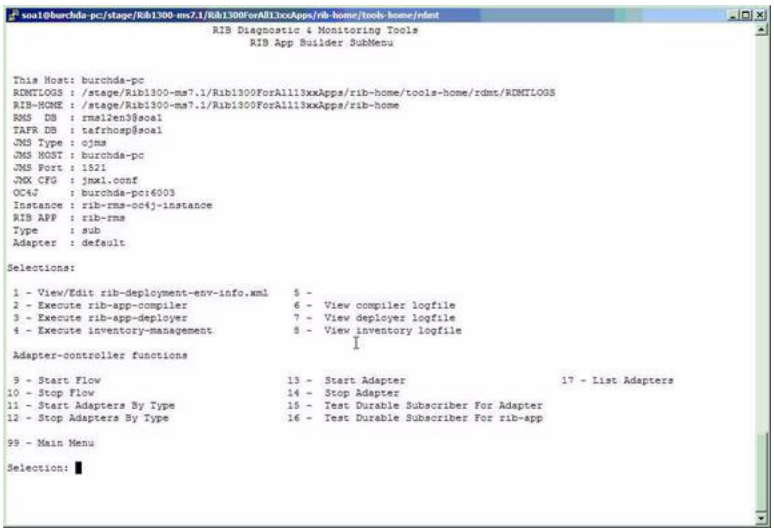
RIB App Builder Tools

Script Used:

rdmt_ribappbuilder_submenu

Description:

This option leads the user to the RIB App Builder tools installed in the rib-home. For a description of the tools and usage, see Chapter 2, "Application Builder".



Scan RIB Logs / Scan RIB Logs (Delta)

Scripts Used:

looklookoc4j.sh, scan_logs.sh, looklookoc4j_delta.sh, scan_logs_delta.sh

Description:

These scripts perform a log scan to look for a pattern ("Exception") in all of the log files in a directory of the currently active OC4J instance. Since they perform file system scans, the RDMT tools must be installed on the host that contains these logs and must have read permissions on the directories and the files.

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

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

RIB Health

Script Used:

cron_ribhealth.sh

Description:

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

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

RIB Configuration Report

Script Used:

OC4JConfigReport.sh

Description:

After the RIB has been installed and configured on OAS, the user can verify all the installations and configurations using RDMT. In order to achieve this functionality a script has been provided and linked in the RDMT menu which scans the installations and configurations of rib applications deployed using the configuration settings in the RDMT configuration files.

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

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

- Provides the local oc4j status using opmnctl.
- Provides the status of all OC4J instances.
- Displays all the RIB instances and their status and PIDS.
- Displays all of the RIB apps and shows the status for each RIB OC4J instance and application.
- Scans the opmn.xml and displays the RIB instances configured and RIB instance settings.
- Provides the directory contents of ORACLE_HOME/j2ee directory.
- Provides the directory contents for each instance configured under the ORACLE_HOME/j2ee directory structure.
- Scans and displays the rib-system.properties for each rib-<app>
- Perform the JMX related functions like scanning configurations for each of jmxX.conf files and displays the status of the adapters, Exceptions is any by scanning the logs and so on.
- Performs checks using JMS configuration.

- Performs checks using Hospital configurations.

RIB Timings Utility

Script Used:

timingsutil.sh

Description:

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

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

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

EJB Publish Utility

Script Used:

ejbpub_util.sh

Class:

RibMessagePublisherClient

Description:

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

Usage:

- " java com.oracle.rib.rdmt.util.RibMessagePublisherClient
- -host <<host | 1>>
- -port <<RMI port -- required>
- -app <<App name -- required>>
- -fa <<family -- required>>
- -ty <<type -- required>>
- -us <<user | optional>>
- -pw <<password | optional>>
- -fi <<file -- required>>

EJB Ping (RIB)

Script Used:

ribejbping.sh

Class:

RibMessageInjectorClient

Description:

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

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

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

EJB Ping (APP)

Script Used:

appejbping.sh

Class:

RibMessagePublisherClient

Description:

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

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

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

Tool Usage Examples

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

1. Using the RDMT Menu system, select the RIB Health SubMenu.
2. Execute RIB Config Report option. This produces the basic report on installation.
   - This scans the installations and configurations of rib applications deployed in oc4j. It finally produces a RIB OC4J configuration report on the console as well as written into a file under the RDMT Temp directory, which contain the status of all the RIB configurations necessary to detect/diagnose any RIB related issues.
3. If you find any discrepancies, refer to the RIB Installation Guide and follow the steps mentioned there.

"How do I know if the local OC4J is running or not?"

1. Using the RDMT Menu system, select the RIB Health SubMenu.
2. Execute local OC4J status (opmn) option. It displays the current OC4j status.
3. If it is not running, start the local OC4J using the start instance (opmn) under the RIB Admin Menu.

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

1. Select RDMT Main Menu.
2. Execute the Scan RIB Logs option. It performs a log scan to look for a /pattern/ ("Exception") in all of the log files in a directory of the currently active OC4J instance.
3. Select JMS Topic Scan. Look for topics with messages stuck.

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

1. Select OC4J/JMX Utilities Menu.
2. Execute Status ALL Adapters option. It displays the status of all the adapters, namely the publishers, subscribers, hospital and TAFR for the currently active OC4J instance.
3. If anything is down, use the Start ALL adapters option and start the same.

"How can I config/switch for a new OC4J instance?"

1. Select OC4J/JMX Utilities Menu.
2. Execute OC4J/JMX Config Utility option.
3. Provide the desired parameters and configure an instance.
4. You can switch to the desired instance using the same option.

"How do I know if the configured OC4J instance is running?"

1. Select OC4J/JMX Utilities Menu.
2. Execute Status all config'd OC4J. It displays the status of all the configured OC4J instances.
3. If it is not running, start the same.

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

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

9

The RIB in Operation

Operational Considerations

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

Alerts and Notifications

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

Note: See Chapter 4, "RIB and JMX" in this manual.

RIB Log File Monitoring

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

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

Note: See the section, "Scan RIB Logs / Scan RIB Logs (Delta)", later in this manual.

Log File Archive and Purge

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

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

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

Note: See Chapter 8, "Diagnostic and Monitoring Tools" in this manual.

Hospital Size and Growth

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

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

Note: See the section, "Hospital Scan Tools", in this manual.

RMS MFQ and RWMS UPLOAD Tables Sizes

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

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

Note: See the section, "Hospital Scan Tools", in this manual.

Remote RWMS

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

RIB Components Start and Stop

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

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

Note: See the section, "RIB App Builder Tools", in this manual.

RIB Operation Support Staff Requirements

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

RIB Components - Source Code Control

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

Note: See Chapter 2, "Application Builder" in this manual.

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

Note: See the section, "check-version-and-apply-defect-fix", in this manual.

RIB HA requirements

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

Note: See RIB Install Guide: The RIB and Oracle Database Cluster (RAC)

Oracle® Application Server High Availability Guide 10g Release 3 (10.1.3.3)

Oracle® Database Administrator's Guide 10g Release 2 (10.2)

RIB Disaster Recovery

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

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

Note: See the section, "RIB Audit Logs", in this manual.

See Chapter 8, "Diagnostic and Monitoring Tools" in this manual.

RIB Admin Roles and Security

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

Note: See RIB Install Guide: Appendix: Creating a RIB OC4J Admin Role

RIB Operation Support Staff Requirements

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

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

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

RIB System Administrator

Technology Background

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

Experience or Training on

- Oracle Application Server
- RIB
- JaveEE concepts
- JMS technology

Areas of Responsibilities:

- Installation or OAS and patches
- Configuration of Oracle Application Server
- Installation and configuration of the RIB

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

RIB Application Administrator

Technology Background

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

Experience or Training on

- RIB
- Oracle Retail Applications
- JMS technology

Areas of Responsibilities:

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

Hospital Monitoring and Maintenance

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

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

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

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

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

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

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

10

Testing the RIB

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

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

- plsql-api-stubs -- An API simulator of the PL/SQL API applications; RMS and RWMS.
- javaee-api-stubs -- An API simulator of those applications exposing JavaEE API's; 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 (xsd's).
- Message auditing - This is a feature of RIB 13 that allows the end-to-end auditing of a message as it passes through all of the RIB components.

The ability to initially install and deploy the RIB has always been difficult because of the need to connect to the Oracle Retail applications to verify that messages could flow end-to-end. The RIB install requires that end-points exist and respond, and to test it requires that the end-points are 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 the section, "RIB Test Harness", later in this chapter.

See Chapter 8, "Diagnostic and Monitoring Tools" in this manual.

See the section, "RIB Logging", in this manual.

RIB Test Harness

Overview

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

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

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

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

Master Checklist

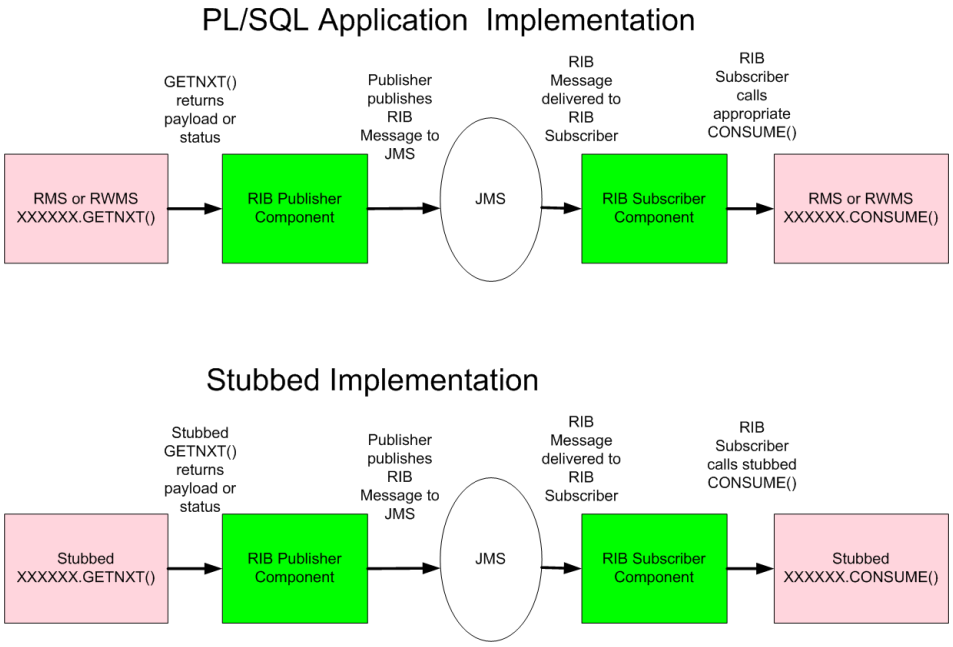
This check list covers all of the sequential steps required to create a stand-alone RIB Test Harness.

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

PL/SQL Application API Stubs

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

Architecture and Design



The tool set contains three main subsystems

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

The Common Subsystem

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

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

For example:

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

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

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

The Thin API layer

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

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

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

The Stub Admin and Setup Functions

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

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

Configuration Files

/conf directory files

Configuration File	Description
stubby.properties	Primary configuration file. Contains database url info and the metadata scripts to load.
commons-logging.properties	Apache logging conf
simplelog.properties	Apache logging conf
SqlToJavaMapper.java	generated from the storedprocedureметadatxml specified in the Stubby.properties file. DO NOT EDIT!

Configuration File	Description
StoredProcedureMetaData_RWMS.xml	DO NOT EDIT!
StoredProcedureMetaData_RMS.xml	DO NOT EDIT!

Installation and Setup

Prerequisite Tasks

Task	Notes
Select a location for the plsql-api-stubs to reside.	Recommended location is in the rib-app-builder/rib-home tree structure: rib-app-builder/rib-home/tools-home
Get the latest version of the plsql-api-stubs.	The plsql-api-stubs is packaged as a stand-alone tar (e.g. PlsqlApiStubs13.0.0ForAll13.x.xApps_eng_ga.tar).
Get the latest version of the rib-public-payload-database-object-types.	rib-public-payload-database-object-types.zip is packaged with the RibFuncArtifacts. (e.g. RibFuncArtifact13.0.0ForAll13.0.0Apps_eng_ga.tar) and should be extracted from there. If this install is in rib-home then the objects will be located in the rib-home/download-home/rib-func-artifacts
Create a database user that will own the plsql-api-stubs schema and the objects.	The user requires no special permissions. CREATE USER "PLSQLSTUB" PROFILE "DEFAULT" IDENTIFIED BY " PLSQLSTUB " DEFAULT TABLESPACE "USERS" TEMPORARY TABLESPACE "TEMP"; GRANT "CONNECT" TO " PLSQLSTUB "; GRANT "RESOURCE" TO " PLSQLSTUB ";
This version requires a path to jdk1.5 for compiling java stored procedures.	Be prepared to specify the path when prompted.

Installation

Task	Notes
Extract the tar file. cd rib-app-builder/rib-home/tools-home tar xvf PlsqlApiStubs13.0.0ForAll13.x.xApps_eng_ga.tar Place the database objects file in the scripts subdirectory	This will create the file folders and place the executables and config files. In rib-home/tool-home there is a directory already. It is a placeholder and this will over write it.

Task	Notes
Extract the rib-public-payload-database-object-types.zip into the scripts directory. unzip rib-public-payload-database-object-types.zip	

Edit /conf/stubby.properties to point to the database url and user/password (see prerequisites).
vi stubby.properties
	# Database details
	hostname=linux1
	port=1521
	sid=ora10g
	username=plsqlstub
	password=plsqlstub

Base Script File names	This is where the selection of either RMS or RWMS objects is made. ONLY one per install.
Execute the install. 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 install performs these actions: Runs a cleanup that will remove any existing RIB related tables, sequences, packages and types installed in the configured user schema. Runs all the scripts files in the udt sub-directory. Runs a drop java utility to remove any existing classes in the configured user schema. Note -- The warnings generated by the drop java can be ignored. Runs the load java utility to load Java classes as objects in the configured user schema. All tables and sequences related to stubby and hospital logic is created in the configured user schema. All the RMS and RWMS packages are created in the configured user schema.
Enter the complete path for jdk1.5:	This version of stubby and the RDBMS requires jdk1.5 for compiling java stored procedures.

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

Configure_api

The next step in using the tool set is to configure the desired behavior of the APIs under test. Use of the tool requires that the user understand the API's involved at enough detail to understand and answer several prompts during the configuration process. See the RIB Integration Guide and the Application Operations Guide (RMS and RWMS).

Prerequisites

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

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

Prompts during configuration of a GETNXT and PUB_RETRY.

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

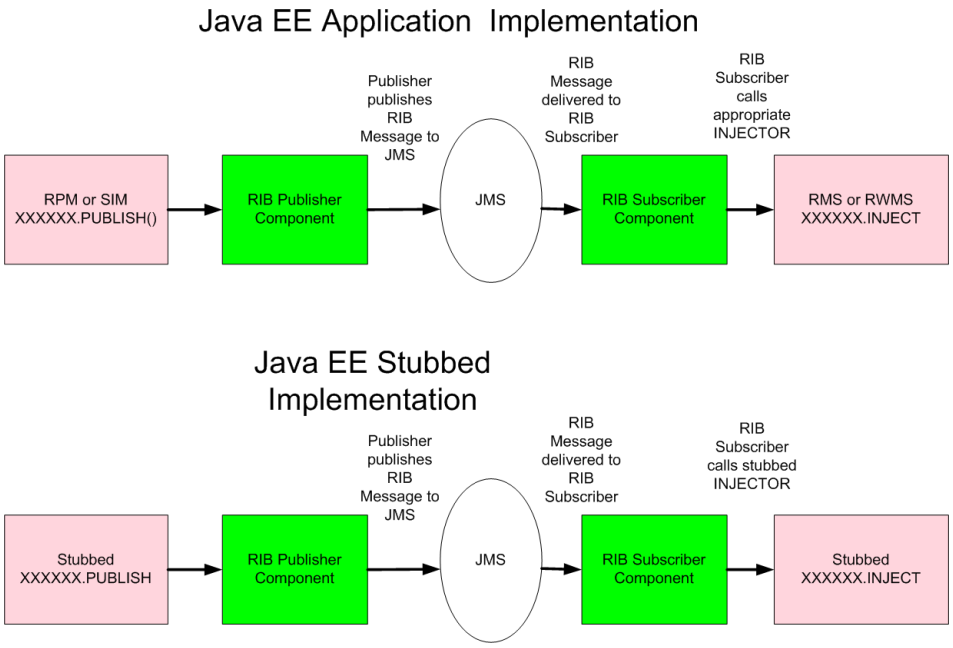
Prompts during configuration of a CONSUME.

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:	

JavaEE Application API Stubs

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

Architecture and Design



Installation and Setup

Prerequisite Tasks

Task	Notes
Select a location for the javaee-api-stubs to reside.	Recommended location is in the rib-app-builder/rib-home tree structure: rib-app-builder/rib-home/tools-home/javaee-api-stubs
Get the latest version of the plsql-api-stubs.	The plsql-api-stubs is packaged as a stand-alone tar (e.g. JavaEeApiStubs13.0.0ForAll13.x.xApps_eng_ga.tar).

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

Installation

Task	Notes
Decide which OC4J instance to deploy the javaee-api-stubs.ear to.	It is recommend but not required that a separate instance be used from the rib-<app> instance.
Using the OAS console, select the oc4j instance and then deploy javaee-api-stubs-.ear.	Refer OAS deployment document for more details on how to deploy a Java EE application.
Verify install	Check logs in $ORACLE_HOME/j2ee/<instance_name>/log/<instance_name_group_name>/oc4j/log.xml
Using the OAS console, configure the database resources for the javaee-api-stubs JDBC resources. ■ Login to the application server console ■ Select Applications Tab ■ Select the javaee-api-stubs application ■ Select Administration Tab	Refer OAS document for more details.
On the JDBC Resources line select "Go to Task".	Select "javaee-api-stubs-db-connection-pool-non-xa" Update database credentials to the javaee-api-stubs user created in the prereqs. Select "javaee-api-stubs-db-connection-pool" Update database credentials to the javaee-api-stubs user created in the prereqs.
Test Connections	Select Test Connection and verify

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 RPM,SIM subscribing families.

Task	Notes
Using RIB App 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.	<jndi> <url>opmn:ormi://linux1:rib-javaeestub-oc4j-instance/javaee-api-stubs</url> <factory>oracle.j2ee.rmi.RMIInitialContextFactory</factory> <user>oc4jadmin</user> <password>welcome1</password> </jndi> </app>

11

Performance Considerations

Performance Factors

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

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

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

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

How to Measure RIB Performance

The performance of the RIB is a complicated subsystem to measure and involves not only host level performance, but database and application server subsystems performance. To make measurement of the RIB components timing characteristics available for analysis, the kernel code has been instrumented to log events as it processes events. The logging of these events is though log4j.

The timings are logged per adapter. Once the timings are enabled the events are logging continuously to the file.

Note: See the section, "RIB Timing Logs", in this manual.

There is a post-processing tool included in RDMT to take the timing file and produce summary reports.

Note: See the section, "RIB Timings Utility", in this manual.

How to Turn-on Timing

Control of the timing log is via the RIB Admin UI or the RIBLOGS log4j control file.

Note: See the section, "RIB Timing Logs", in this manual.

How to Use the Timing Logs

Once the timing logs are created, they need to be post-processed and analyzed. The process to do that is to use the RDMT Timing Log Utility to post-process the adapter.timing.log file. This creates a .csv file that can then be viewed and printed by a spreadsheet tool (e.g. Microsoft Excel).

Note: See the section, "RIB Timings Utility", in this manual.

The row headings that are displayed depend on the API type the adapter is interfaced to, PL/SQL or JavaEE and the adapter type; PUB, SUB, or TAFR. The above example is from a PL/SQL Publisher.

This table lists all of the column headings on the report and the definition of each.

TIMING_TYPE	The type is the predefined interval of time across two or more timings log statements.
COUNT	Count of similar timing_types for this msgtype.
AVERAGE_TIME (sec)	TIME_SUM/COUNT
STANDARD_DEVIATION (sec)	Math.sqrt((TIME_SQUARED_SUM - (TIME_ SUM * TIME_SUM/COUNT)))/(COUNT - 1).
TIME_SUM (sec)	Sum of all time intervals for this timing_type.
TIME_SQUARED_SUM (sec)	Individual squares of timing intervals for this timing_type.
MIN_TIME (sec)	Least time interval for this timing_ type.
MAX_TIME (sec)	Max time interval for this timing_type.
THRESHOLD (sec)	Default Threshold is 10 sec, if any time interval takes more than 10 sec its tracked under over_ threshold.
OVER_THRESHOLD_COUNT	Number of over threshold intervals.
OVER_THRESHOLD_SUM (sec)	Summation of all such intervals that are greater than threshold.
OVER_THRESHOLD_AVG (sec)	Average all such intervals that are greater than threshold.

This table lists the currently pre-defined time intervals interpreted by the timings logs processor utility. They are the row headings in the report under the column heading Timing Types. The description is the definition of interval calculation.

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

Example Timing Analysis - PL/SQL PUB

This a partial extract of a post-processor report on an RMS Order_pub adapter timing log.

Timings for the Period 09:00:00 to 09:59:59

TIMING_TYPE	COUNT	AVG_TIME	STD_DEV	TIME_SUM
PUB_B4_ GETNXT_CALL	220	0.02239	0.00041	4.926
PUB_TIME_IN_ GETNXT_CALL	220	0.15503	0.0067	34.107
PUB_TIME_IN_ EJB_PUBLISH_ CALL	220	1.82395	0.05014	401.269
PUB_TOTAL_ PUBLISH_TIME	220	2.11173	0.04992	464.58

TIMING_TYPE	COUNT	AVG_TIME	STD_DEV	TIME_SUM
PUB_TIME_IN_REAL_JMS_PUBLISH_EJB	220	1.81753	0.05012	399.856

In this example the messages were published between 09:00 - 10:00 AM. In this example the messages were published between 09:00 - 10:00 AM.

T4 - PUB_TOTAL_PUBLISH_TIME is the total time taken for the complete PUB process.

- T1 - PUB_B4_GETNXT_CALL is the time interval between start of the GETNXT EJB and the PL/SQL GETNXT API call.
- T2 - PUB_TIME_IN_GETNXT_CALL is time taken by the PL/SQL API GETNXT call.
- T3 - PUB_TIME_IN_EJB_PUBLISH_CALL is the time taken for the EJB publisher call.
  - T5 - PUB_TIME_IN_REAL_JMS_PUBLISH is the time taken to publish a message to the JMS

T3 = T5 + (RIB Overhead payload creation)

T4 = T1 + T2 + T3 + (Hospital Dependency Checks)

In this example the data points give these insights into the RIB performance.

- "The average time to publish a message is 2.1 seconds.
- The time spent in the PL/SQL API is 0.15 seconds/message
- The time spent in the call to the JMS is 1.8 seconds/message

Note: This is an illustration and not a measure of actual through-put or numbers that can be extrapolated to indicate volume performance. The number of message needed to arrive at a calculation of through-put would require much higher counts and across a broad spectrum of time and system load. Other factor including average size of message used is also a factor.

Multi-Channel Adapters

A channel is a solution approach to maintaining the previous RIB release concept of a "Logical Channel", also known as multi-threading.

Multi-channel is concept to logically partitioning the flow of messages within the JMS topic so that multiple publisher and subscriber can simultaneously use the same JMS topic without any contention or interference with each other and preserving publication message ordering within the logical channel.

Every adapter instance of a publisher, subscriber, or TAFR, configured in the RIB is considered to belong to a logical channel for processing messages. By multi-channel adapters we mean multiple adapter instances for the same message family, each processing messages asynchronously and in parallel.

There are critical rules of behavior that have to 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 make the tasks of configuration of the RIB to support a multi-channel message flow as simple as possible, the process has been integrated into the RIB App Builder tools.

When multiple channels are being considered, 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. The RIB App Builder tools have checks and verification logic to prevent deployment of incomplete flows.

Multi-channels can be a valuable tool to 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. The process of adding multi-channels to a message family should be part of a performance test and tuning process.

Logical Channels and threadValue

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

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

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

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

Alogrithm Used to Calculate Channel

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

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

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

For example:

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

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

Example of a Message Family Flow with a TAFR:

Alloc_pub_1

Alloc_tafr_1

StockOrder_sub_1

How to Configure a Multi-Channel Flow

Generalized Process

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

Example

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

Backup the following files.

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

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

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

```
  <out-db>default</out-db>
  </node>
 <node id="rib-rwms.StockOrder_sub" app-name="rib-rwms"
adapter-class-def="StockOrder_sub" type="JmsToDb">
  <in-topic>etStkOrdersFromRIBToWH1</in-topic>
  <out-db>default</out-db>
  </node>
</message-flow>
```

RIB-RMS

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

   Here is a snippet of rib-rms-adapters.xml

```
   <publishers>
             <timer-driven id="Alloc_pub_1" initialState="running"
timeDelay="10">
                   <timer-task>
                        <class
name="com.retek.rib.app.getnext.impl.GetNextTimerTaskImpl"/>
                              <property name="maxChannelNumber" value="5" />
                   </timer-task>
             </timer-driven>
             <timer-driven id="Alloc_pub_2" initialState="running"
timeDelay="10">
                   <timer-task>
                        <class
name="com.retek.rib.app.getnext.impl.GetNextTimerTaskImpl"/>
                              <property name="maxChannelNumber" value="5" />
                   </timer-task>
             </timer-driven>
             <timer-driven id="Alloc_pub_3" initialState="running"
timeDelay="10">
                   <timer-task>
                        <class
name="com.retek.rib.app.getnext.impl.GetNextTimerTaskImpl"/>
                              <property name="maxChannelNumber" value="5" />
                   </timer-task>
             </timer-driven>
             <timer-driven id="Alloc_pub_4" initialState="running"
timeDelay="10">
                   <timer-task>
                        <class
name="com.retek.rib.app.getnext.impl.GetNextTimerTaskImpl"/>
                              <property name="maxChannelNumber" value="5" />
                   </timer-task>
             </timer-driven>
             <timer-driven id="Alloc_pub_5" initialState="running"
timeDelay="10">
                   <timer-task>
                        <class
name="com.retek.rib.app.getnext.impl.GetNextTimerTaskImpl"/>
                              <property name="maxChannelNumber" value="5" />
                   </timer-task>
             </timer-driven>
```

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

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

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

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

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

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

RIB-TAFR

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

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

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

```
Alloc_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

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

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. Modifyrib-rwms-adapters-properties.properties.

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

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

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

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

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

Edit the RIB_SETTINGS table

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

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

Compile and Deploy

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

Message Aggregation

In order to improve performance of GETNEXT, messages are aggregated before publishing to the JMS. Aggregation of payloads is performed on a per family basis. These properties must be used when RIB is being integrated to the PL/SQL applications. Hence, modifications need to be made for the respective rib-<pl-sql> apps on the RIB end.

maxNodesPerMessages - Maximum number of ribMessages per one RibMessage Envelope.

messagePerCommit - Maximum number of RibMessage's sent for a commit. In order to not overload the JMS with huge payloads this property must be set.

These properties should exist in the rib-<pl-sql>.properties file.

Items.maxNodesPerMessages=2

Meaning 2 ribMessages per RibMessage.

Items.messagePerCommit=5

Meaning 2 ribMessages per commit.

How to Configure Message Aggregate

rib-rms is taken as an example in this configuration.

- Edit the following file in rib-home
  - rib-home/application-assembly-home/rib-rms/rib-rms.properties
- Add the following properties
  - Items.maxNodesPerMessagess=5
  - Items.MessagePerCommit=2
- Using the app-builder tool compile/deploy the application.
  - rib-app-compile.sh
  - rib-app-deployer.sh -deploy-rib-app-ear rib-rms

For example, if there are 15 payloads waiting to be published in RMS, based on the above configuration one should be able to see the following. The scenario is depicted pictorially in the figure below.