

Oracle® Retail Integration Cloud Service

Third Party Integration Guide



Release 23.0.101.0

F76260-01

January 2023

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

Copyright © 2023, Oracle and/or its affiliates.

Primary Author: Nathan Young

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

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

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

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

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

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Send Us Your Comments

Preface

Audience	ix
Customer Support	ix
Improved Process for Oracle Retail Documentation Corrections	ix
Oracle Retail Documentation on the Oracle Technology Network	x
Conventions	x

1 Introduction and Executive Summary

2 Retail Integration Cloud Service Components

Retail Integration Bus (RIB)	2-1
RIB-EXT	2-1
FILEIO	2-2
Oracle Retail Bulk Data Infrastructure (BDI)	2-3
BDI-EXT	2-3
Installation Details	2-4
Retail Financial Integration (RFI)	2-4
Retail Financial Integration (RFI) Products	2-4
Oracle Retail To Financials Application (EBS/PeopleSoft) RFI Process Flow	2-5
RFI for Oracle Cloud Financials	2-6
Universal Service Mapper (USM)	2-6
USM Architecture	2-8
Event Listener	2-8
Service Mapper Orchestration	2-8
Service Provider and External Services	2-9
Oracle Warehouse Management CS (Logfire) Integration	2-9
RICS OWMS Integration Flows	2-10

3 RICS Operations Support, Management, and Monitoring

Customer Access to RICS Operation	3-1
Retail Integration Console (RIC) Overview	3-1
JMS Console	3-2
JMS Admin – Live Monitoring	3-4
JMS Browse	3-4
JMS Manage	3-5
RIB Hospital Administrator (RIHA)	3-5
RIBForXXX Administrator User Interface	3-6

4 Implementing RIB-EXT

How to Send and Receive Messages to/from RIB System using SOAP API's	4-1
External Application as a Publisher (rest-app) using OAuth2	4-1
Create OAuth2 Client Application in IDCS	4-1
External Application as a Subscriber (rest-app)	4-7
How to implement Injector Service (Service Contract) using ReST	4-8
How to Secure Injector Service with Oauth2	4-8
RIB-EXT Side of Configuration to Point to External Application	4-9
How to switch Injector Service app Type at Runtime	4-13
How to Change rib-ext injector-service-app-type from REST to SOAP	4-14
How to change rib-ext injector-service-app-type from SOAP to ReST	4-15
Error Handling	4-16

5 Reference Implementation of Injector Service Using Tomcat

Introduction	5-1
Step-by-step guide for testing rib-injector-service war on Tomcat	5-2
Approach for Writing Custom Implementation for Injectors	5-3

6 Implementing BDI-EXT

BDI External Job Admin as Receiver	6-1
External Importer Job	6-1
Configure External Job Admin as Receiver in the Process Flow	6-2
External BDI Process Flow	6-3

7 Monitoring at Run Time

Instance and Central Repository	7-1
Monitoring Data as XML	7-1

Push Versus Pull	7-2
Service Interfaces	7-2
What is an Event?	7-2
How are Event Count and Messages Count Related?	7-2
Adapter Events	7-2
Application Events	7-3
Event Collection Schedule	7-3
Publisher Versus Subscriber Events	7-3
TAFR Instrumentation	7-3
Data Retention	7-4
Metric Definitions	7-4
Event Counts	7-4
Adapter Execution Time	7-4
API Execution Time	7-4
Adapter Status	7-4
Commits and Rollbacks	7-4
Error Hospital Metrics	7-5
RIB Application Status	7-5

A Sample Files

Sample Application.wadl File	A-1
Sample Resource Class	A-2
ApplicationMessages.xsd	A-3
Rest Publisher Pseudo Code	A-5
payload.properties	A-6
Sample Request/Response for ReST Injector Service	A-7

List of Figures

1-1	RICS Functional Architecture	1-2
2-1	RIBforEXT Information Flow	2-2
2-2	FILEIO Information Flow	2-2
2-3	BDI Third Party Integrations	2-3
2-4	Oracle Retail To Financials Application (EBS/PeopleSoft) RFI Process Flow	2-5
2-5	RFI for Oracle Cloud Financials	2-6
2-6	USM Process Flow	2-7
2-7	USM Architecture	2-8
2-8	WMS-RICS Mappings	2-9
2-9	Retail to LogFire Integration Overview	2-10
3-1	Customer Access to RICS Operation	3-1
3-2	Retail Integration Console	3-2
3-3	JMS Console Architecture	3-2
3-4	JMS Console Workflow	3-3
3-5	JMS Console	3-4
3-6	RIB Hospital Administrator	3-5
3-7	RIBForXXX Administrator User Interface	3-6
6-1	External importer Job	6-1
6-2	Merchandising to External Process Flow	6-3
7-1	RIB Monitoring ARchitecture	7-1

List of Tables

4-1	Publishing Service Pattern	4-1
4-2		4-10
7-1	Schedule of Collection Times	7-3
A-1	Sample Request/Response for ReST Injector Service	A-7

Send Us Your Comments

Oracle Retail Integration Cloud Service Third Party Integration Guide, Release 22.1.401.0

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

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

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

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



Note:

Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the Online Documentation available on the Oracle Technology Network Web site. It contains the most current Documentation Library plus all documents revised or released recently.

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

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

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

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

Preface

This guide provides an overview of third party integration with Oracle Retail Integration Cloud Service.

Audience

This guide is intended for administrators.

This guide describes the administration tasks for Oracle Retail Integration Cloud Services.

Customer Support

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

<https://support.oracle.com>

When contacting Customer Support, please provide the following:

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

Improved Process for Oracle Retail Documentation Corrections

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

Oracle Retail documentation is available on the Oracle Technology Network at the following URL:

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

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

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

Oracle Retail Documentation on the Oracle Technology Network

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

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

(Data Model documents are not available through Oracle Technology Network. You can obtain these documents through My Oracle Support.)

Conventions

The following text conventions are used in this document:

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

1

Introduction and Executive Summary

The Retail Integration Cloud Service (RICS) is the SaaS Cloud deployment of the full Retail Technology's Integration Product Suite and the out-of-box GA application integration flows.

- All of the integration styles and products supported (RIB, RSB, BDI) are deployed in the RICS SaaS Cloud and are accessible to on-premise applications and other cloud applications through service APIs.
- Functionality has been added to round out the integration styles and available flows to support customers of the RICS SaaS offering.
- RICS assumes and supports hybrid cloud topologies; cloud-to-cloud, cloud-to-premise and premise-to-cloud.
- All of the integration styles and products and the applications supported by RICS are integrated via REST web services, with SOAP still available but not recommended.

The RICS Integration Infrastructure products provide standards and guidance and remove the complexity from the business applications and provide accepted solutions to recurring problems within a given integration context. Around these styles and patterns, Oracle Retail has developed a set of Integration Infrastructure products and supporting tools.

These products expose hundreds of Oracle Retail Application API's as contract driven integration points exposed for integrations between Oracle Retail applications and customer 3rd Party and Legacy applications.

- Retail Integration Bus (RIB) - Asynchronous JMS Messaging
- Bulk Data Integration (BDI) - Bulk Data Movement
- Retail Financial Integration (RFI) - Integration to Oracle Financial Products (EBS, People Soft Financials, Oracle Cloud Financials)
- Oracle Warehouse Management Cloud Service integration using the Universal Service Mapper product.



Note:

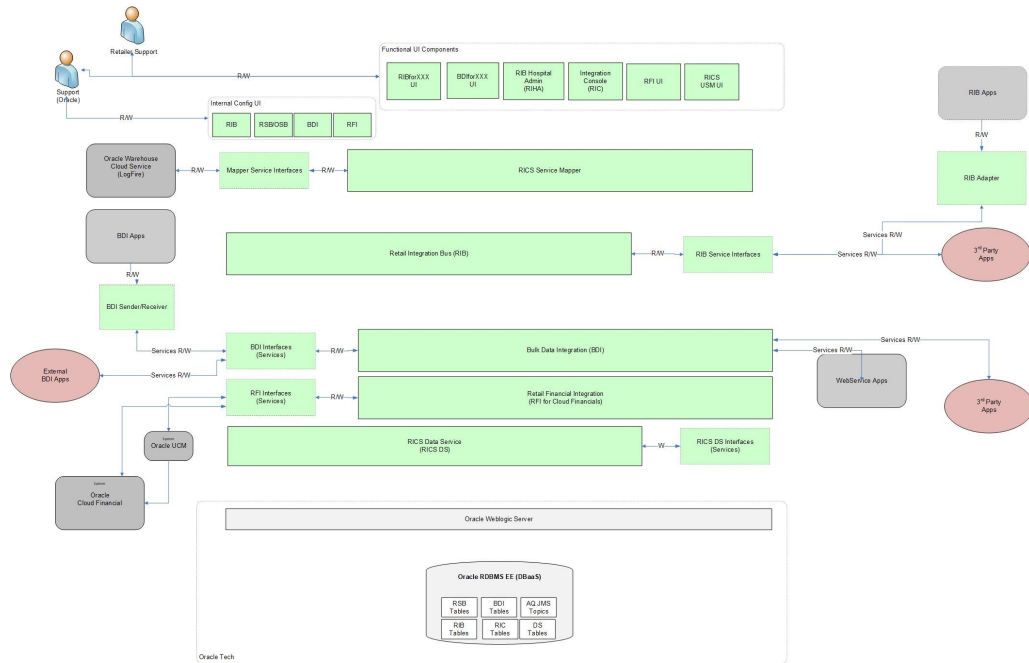
All Integration Infrastructure products are fully cloud enabled and will be deployable as the Oracle Retail Integration Cloud Services and existing integration APIs work without modification for Cloud, Hybrid-Cloud and On-Premise installations.

As used in this document; RICS third party integrations is the common term used generically for any customer's custom legacy systems and/or customer purchased third party applications.

The list of these are as great as the customer list and RICS integration applications have evolved to make those integrations easier and to follow our contracts and integration styles by providing defined endpoints, guidelines, best practices, and support tooling.

Retail Integration Cloud Services Architecture

Figure 1-1 RICS Functional Architecture



2

Retail Integration Cloud Service Components

Retail Integration Bus (RIB)

The Retail Integration Bus (RIB) is a fire-and-forget, asynchronous messaging backbone and designed as a "Pub/Sub" JMS messaging architecture, with additional application functionality added such as intelligent transformation, routing and error handling.

The RIB has been designed, and proven to handle retail volumes of messages, typically millions during a Tier 1 -Tier 1+ retailer's business day.

The design and architecture of the RIB infrastructure and the Oracle Retail Applications API's are based on two key requirements driven by the Oracle Retail Application's Business process models.

- *Preservation of Publication Sequence.* The business event (message) must be delivered to all the subscribing applications in the order (FIFO) the business event (messages) was published by the publishing application.
- *Guaranteed Once-and-only-once Successful Delivery.* The RIB must preserve and persist all business events (messages) until all applications (Subscribers) have looked at the message and have successfully consumed it or decided they do not care about that event (message).

The RIB is designed as an asynchronous publication and subscription messaging integration architecture.

The RIB interfaces are organized by message family. Each message family contains information specific to a related set of operations on a business entity or related business entities. The publisher is responsible for publishing messages in response to actions performed on these business entities in the same sequence as they occur.

Each message family has specific message payloads based on agreed upon business elements between the Oracle Retail applications.

RIB-EXT

Third Party and Legacy Integration has been the corner stone of the Enterprise Integration components. To that end there are cloud focused enhancements to RIB; RIB-EXT and FileIO.

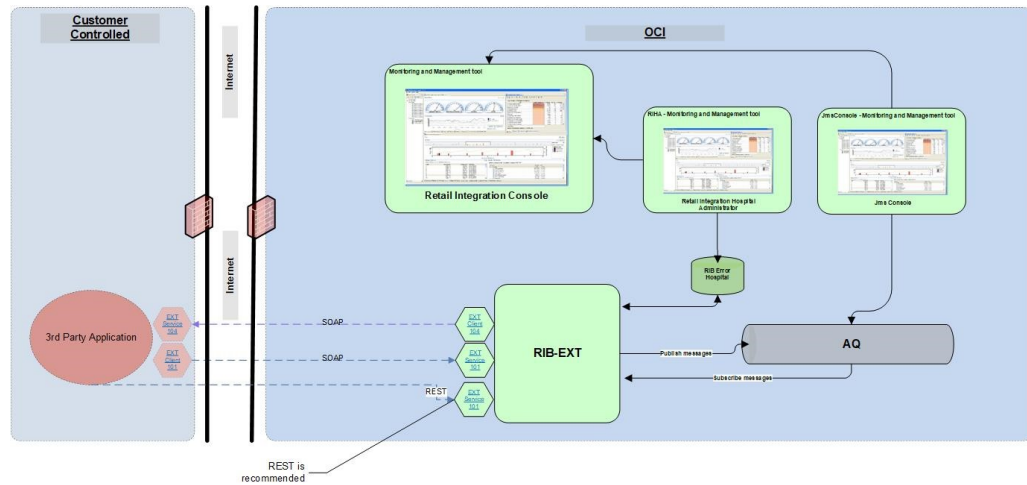
RIB-EXT completes the ability of the Integration Cloud Service to expose the RIB as REST and SOAP services to 3rd Party deployed on-prem or in other cloud deployments.

RIB-EXT is a deployment time configurable component that supports pub/sub to/from the RIB and an external application by exposing the RIB as SOAP services to Retail Applications and third party deployed on-premises or in other cloud deployments.

- RIB-EXT has all of the RIB flows available for the deployment time configuration based on the customer use cases.

- RIB-EXT configuration is performed based on the customers use cases for data to and from the RIB.

Figure 2-1 RIBforEXT Information Flow



FILEIO

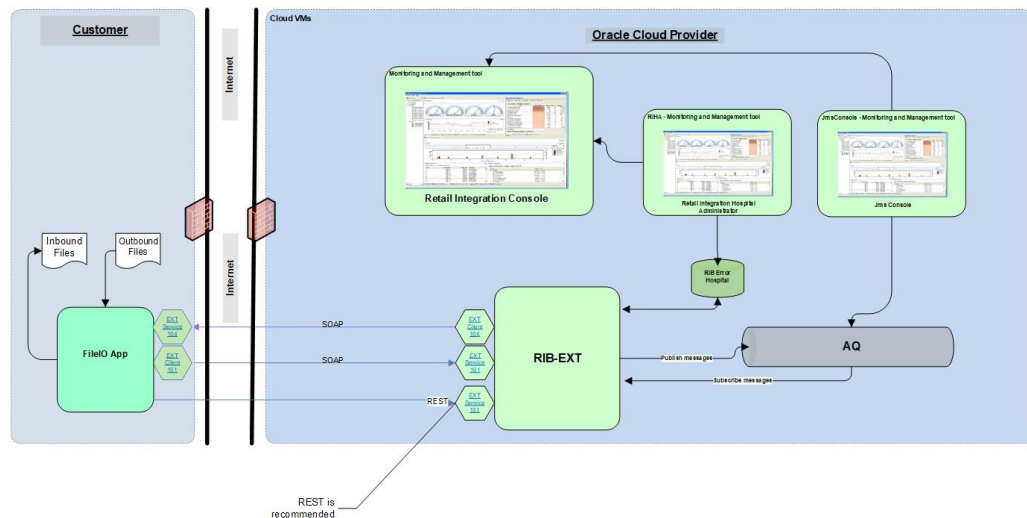
The FileIO app has been enhanced to fill a gap by exposing customer facing REST and SOAP Web Service APIs to FileIO that publishes and subscribes flat files to RIB-EXT that then pub/subs to the RIB's JMS.

FileIO works in conjunction with the RIB-EXT component. The RIB-EXT server side component exposes customer facing REST and SOAP Web Service APIs.

FileIO publishes or subscribes using flat files to RIB-EXT that then pub/subs to the RIB's JMS.

The files are not moved by ftp. RICS transmits the data via web services.

Figure 2-2 FILEIO Information Flow



Oracle Retail Bulk Data Infrastructure (BDI)

Batch (Bulk) data is still a predominant integration style within Oracle Retail and its customers. The movement of bulk data remains important because some work is best suited to being performed in bulk. Batch processing was there in the early days; it is still here today; and it will still be here tomorrow. What has changed is the approach to batch processing.

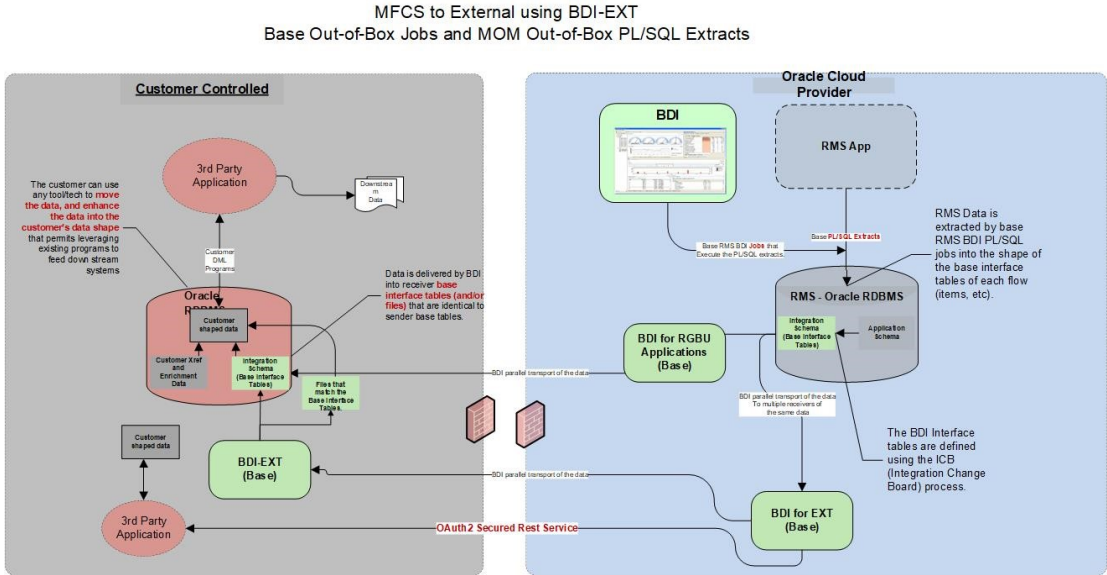
- BDI Architecture and Design is to be On-Premise and Hybrid Cloud ready.
 - Lightweight UI's and services provide full coverage and are both customer facing and operations facing.
- BDI Provides a fully service enabled, fault tolerant, concurrent, high throughput infrastructure
- BDI Transport layer moves data via services, not files.
- Provides integration job process scheduling and is fully instrumented to provide end-to-end visibility and re-startability.
- Automated restart recovery at a granular level.

BDI-EXT

BDI is an integration infrastructure product which can integrate Oracle Retail applications to third party applications. BDI external application is designed to address the complexities for third party integration with Oracle Retail application. In BDI, bulk data movement happens between sender and receiver application. External application can only be a receiver.

Please refer to the *Oracle Retail Bulk Data Integration Guide – Concepts* for the details

Figure 2-3 BDI Third Party Integrations



The technology used by the sender application and the receiver application to extract and fill the interface tables and to upload the data from the interface tables is a decision made by

each of the applications. There are templates and hooks provided to the application's teams to assist in development.

The BDI Infrastructure for each application is an application (BDIforApp) that is built and supplied by the integration team. There is one dedicated to each application and it is responsible for transforming the data to and from the BDI Transport to the integration interface tables or files that are transported to each of the configured receivers.

All BDI Infrastructure components are service enabled.

Installation Details

Please refer to the *Oracle Retail Bulk Data Integration Installation Guide* for the details.

Retail Financial Integration (RFI)

The Oracle Retail Financial Integration (RFI) for E-Business Suite (EBS) / PeopleSoft Financials / CFIN provides integration to a robust enterprise financial system to complement the Oracle Retail Merchandising system in a retail customer environment.

Retail Financial Integration (RFI) Products

On-Premise Merchandising to On-Premise Financial Applications

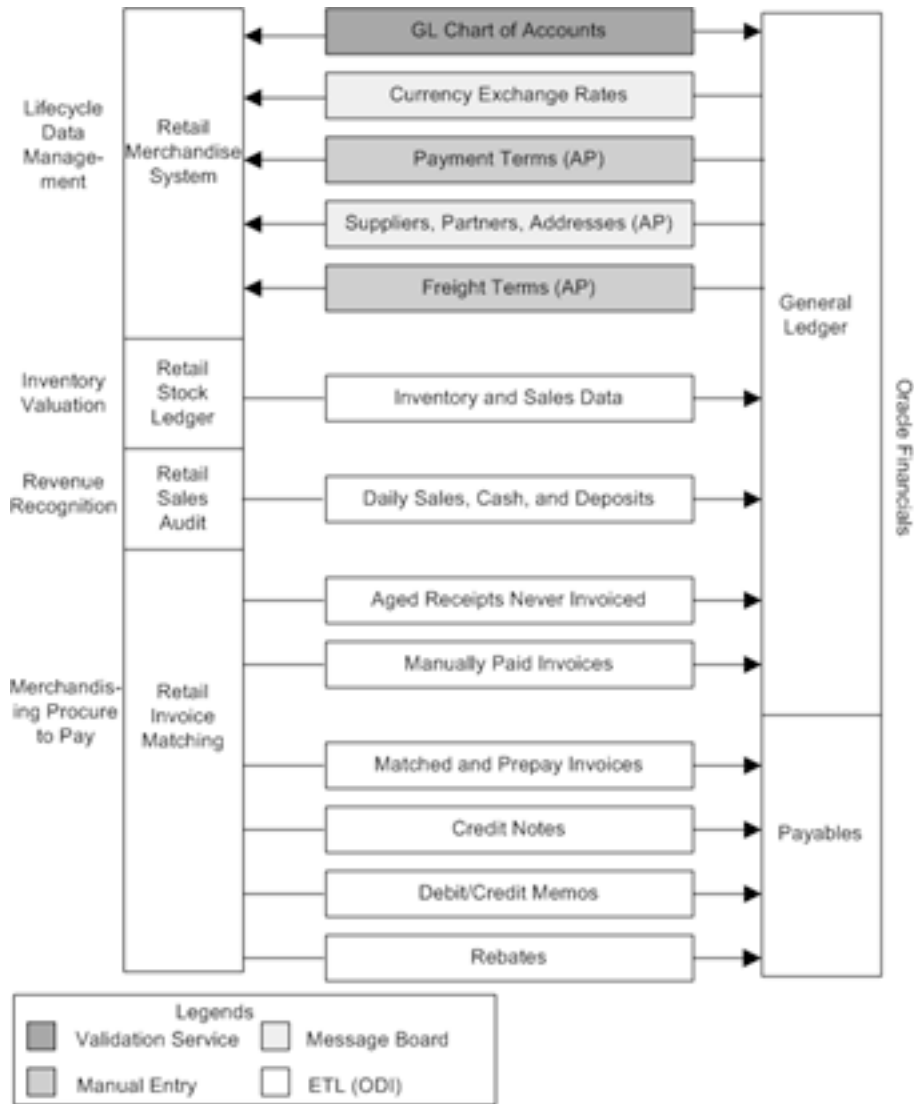
- RFI for PeopleSoft (On-premise)
- RFI for EBS (On-premise)

Merchandise Foundation Cloud Service (MFCS) to Financial Applications

- RFI for MFCS to On-Premise EBS
- RFI for MFCS to Oracle Cloud Financials

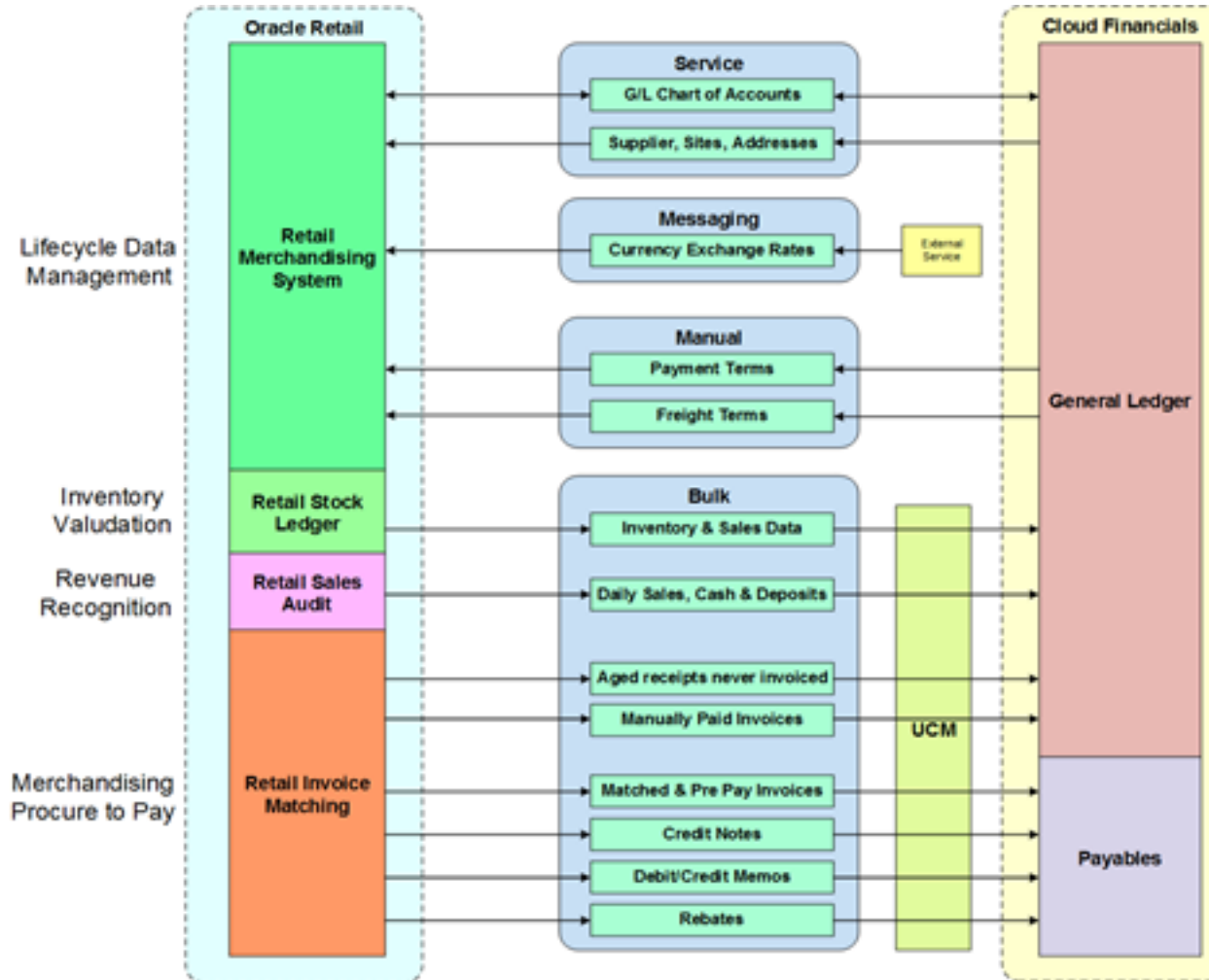
Oracle Retail To Financials Application (EBS/PeopleSoft) RFI Process Flow

Figure 2-4 Oracle Retail To Financials Application (EBS/PeopleSoft) RFI Process Flow



RFI for Oracle Cloud Financials

Figure 2-5 RFI for Oracle Cloud Financials



Universal Service Mapper (USM)

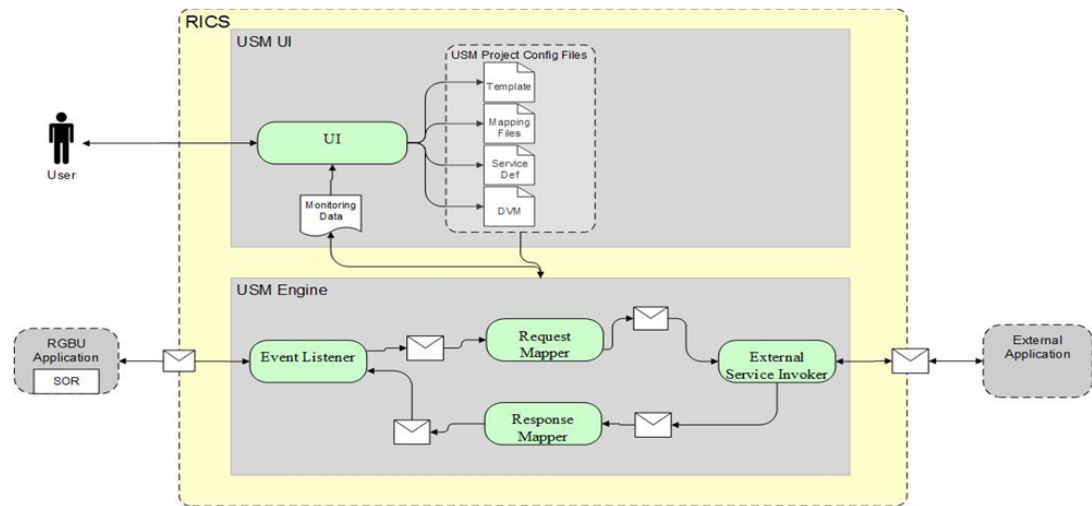
The Universal Service Mapper (USM) is an application component of Retail Integration Cloud Service (RICS) that allows the definition, mapping, and configurations needed to support the integration between two heterogeneous applications. Typically this would be an Oracle Retail Application such as RMS, found in the Merchandise Foundation Cloud Service, and an application external to Oracle Retail such as Oracle Warehouse Management CS.

RICS USM supports two of the styles of integration used within Oracle Retail; message-based and service-based as inputs. Within the RGPU applications the message-based flows are across the RIB. It is typical that external applications are predominately service-based so the output of USM is a call is to an external service.

Service calls from an external service are transformed to the correct style and format of the internal application.

The functional requirement for the USM is to act as the place to transform the Oracle Retail application data style and the data format into the data format expected by the external application, and then to perform the transformations of the external application's response.

Figure 2-6 USM Process Flow



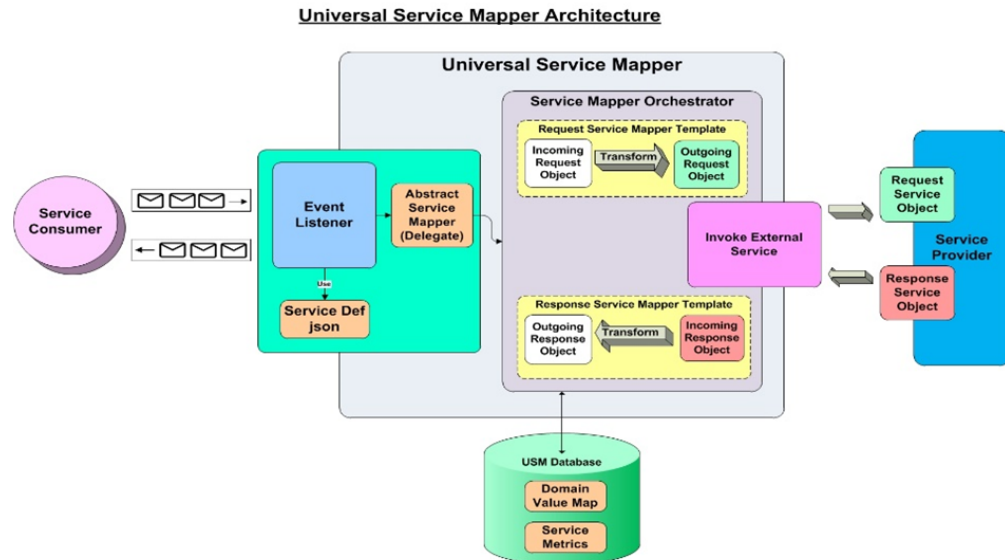
The USM User Interface provides the ability to create and manage Projects in USM and to view app statistics, metrics about the message flow and the system Logs

The USM Engine is the logic part of the system where the data is received from one application, mapped to other data and the mapped data is sent to other applications. Data is communicated through service calls.

- USM hosts all the necessary web-services that are required by both the participating sender and the receiver applications.
- USM has a configuration file of the service URLs to the participating applications.
- USM has templates that contains the mapping information, the code that does the mapping and other configuration files to make the application work.

USM Architecture

Figure 2-7 USM Architecture



Universal Service Mapper has three major components:

- Event Listener [Abstract Service Mapper, Service Def JSON]
- Service Mapper Orchestration [Orchestrator, Template and DVM]
- External Service Invocation and Service Provider

Event Listener

The Event listener is a service hosted by the USM application which is open to receiving data from any application that is connected to it. The application here is either RIB-LGF or WMS Cloud. The applications have the following URL pattern set in their target for USM.

`http://<host>:<port>`

When application sends data, the event listener internally calls the abstract service mapper which determines family, message type and the operation(s) from the message received by referring to the Service Def JSON file.

Service Mapper Orchestration

The abstract service mapper in turn now calls the Service mapper orchestrator which decides which what data is to be populated in the mapper templates. The orchestrator does the mapping, field by field, from the source application to destination application. Certain key-value pairs in the DVM in order to maintain context between the applications.

Service Provider and External Services

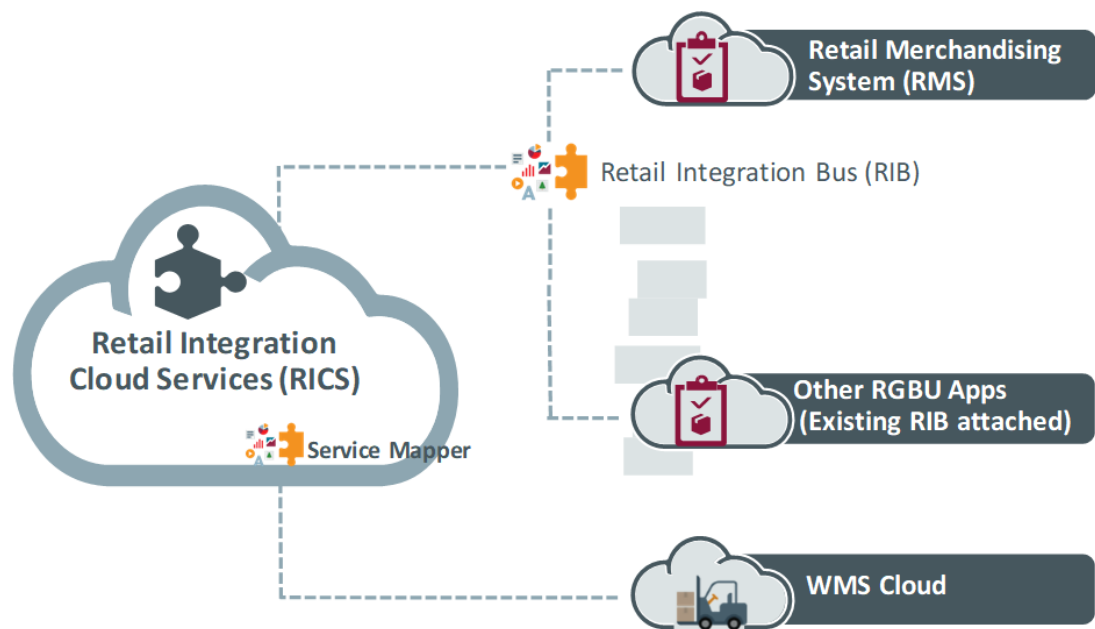
The Service Mapper Orchestrator calls the services hosted by the service providers after the mapping operations are completed. The service providers here are either RIB-LGF or WMS Cloud, which consume these services via USM. The calls are REST calls. USM holds the information necessary for it to call these services in a file with the prefix "external_env_json" for the respective application. These are stored as key-value pairs in JSON file

Oracle Warehouse Management CS (Logfire) Integration

Retail Integration Cloud Service (RICS) is used to integrate MFCS to Oracle Warehouse Management Cloud (WMS - LogFire).

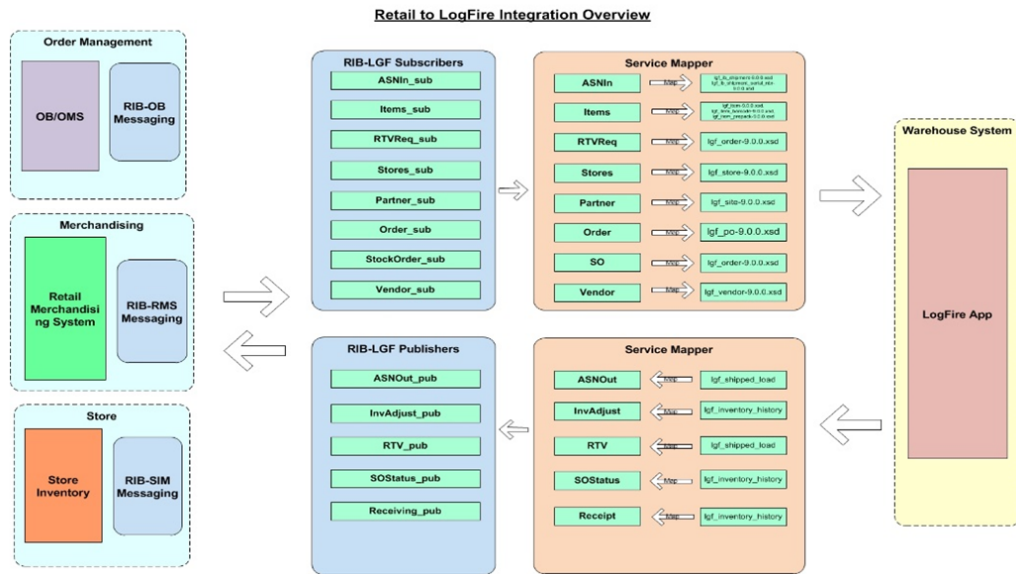
RICS uses Universal Service Mapper (USM) to perform the mappings required to connect RICS payloads/services to LogFire interfaces. USM is used to transform and manage the integration flows in both directions.

Figure 2-8 WMS-RICS Mappings



RICS OWMS Integration Flows

Figure 2-9 Retail to LogFire Integration Overview



Pre-defined flows between Oracle WMS Cloud and Oracle Retail Applications.

3

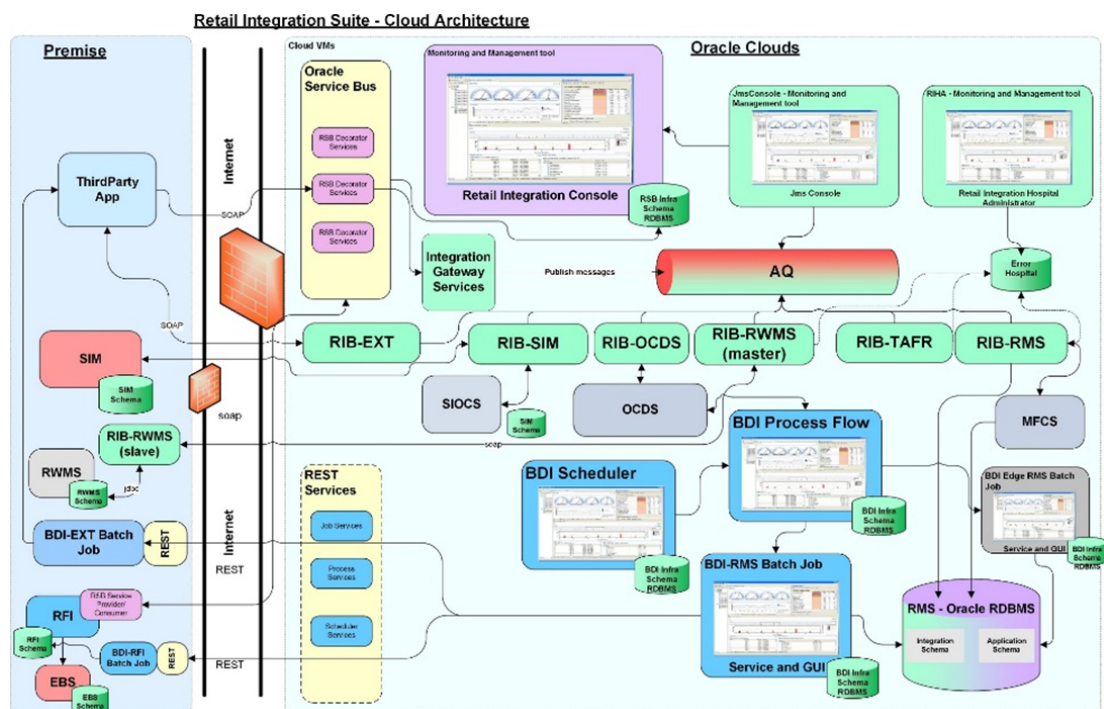
RICS Operations Support, Management, and Monitoring

RICS exposes UI's for the customer and for the Oracle Support Team that provide configuration and run time information as well as logs for operational insights and troubleshooting. This section introduces the primary tools.

Customer Access to RICS Operation

RICS is fully instrumented and exposes metrics via UI's for full operational visibility.

Figure 3-1 Customer Access to RICS Operation



Retail Integration Console (RIC) Overview

RIC is the consolidated Enterprise Integration Monitoring tool and provides full visibility to the Oracle Retail Integration System in a unified view. The audiences for the tool are business analysts, operations people, and integration administrators.

Data is presented in graphical and tabular forms along with business context, so it is more easily understood by business analysts and technical personnel.

RIC directly integrates with the Integration Guide via contextual hyperlinks.

Figure 3-2 Retail Integration Console



JMS Console

The JMS Console is an administration tool to monitor and manage the RIB's JMS Servers.

Figure 3-3 JMS Console Architecture

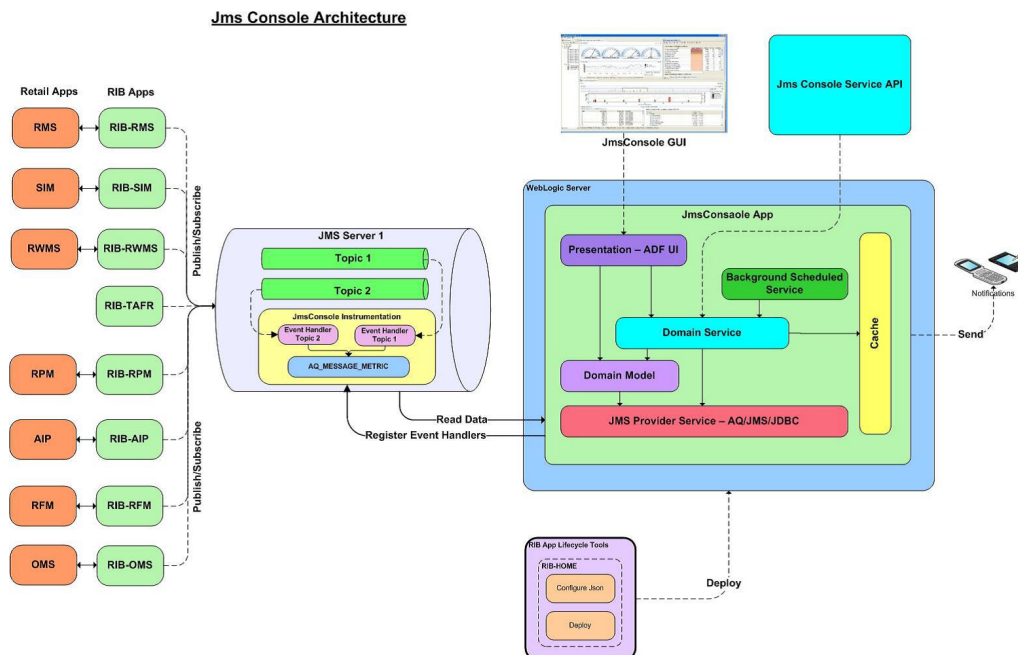
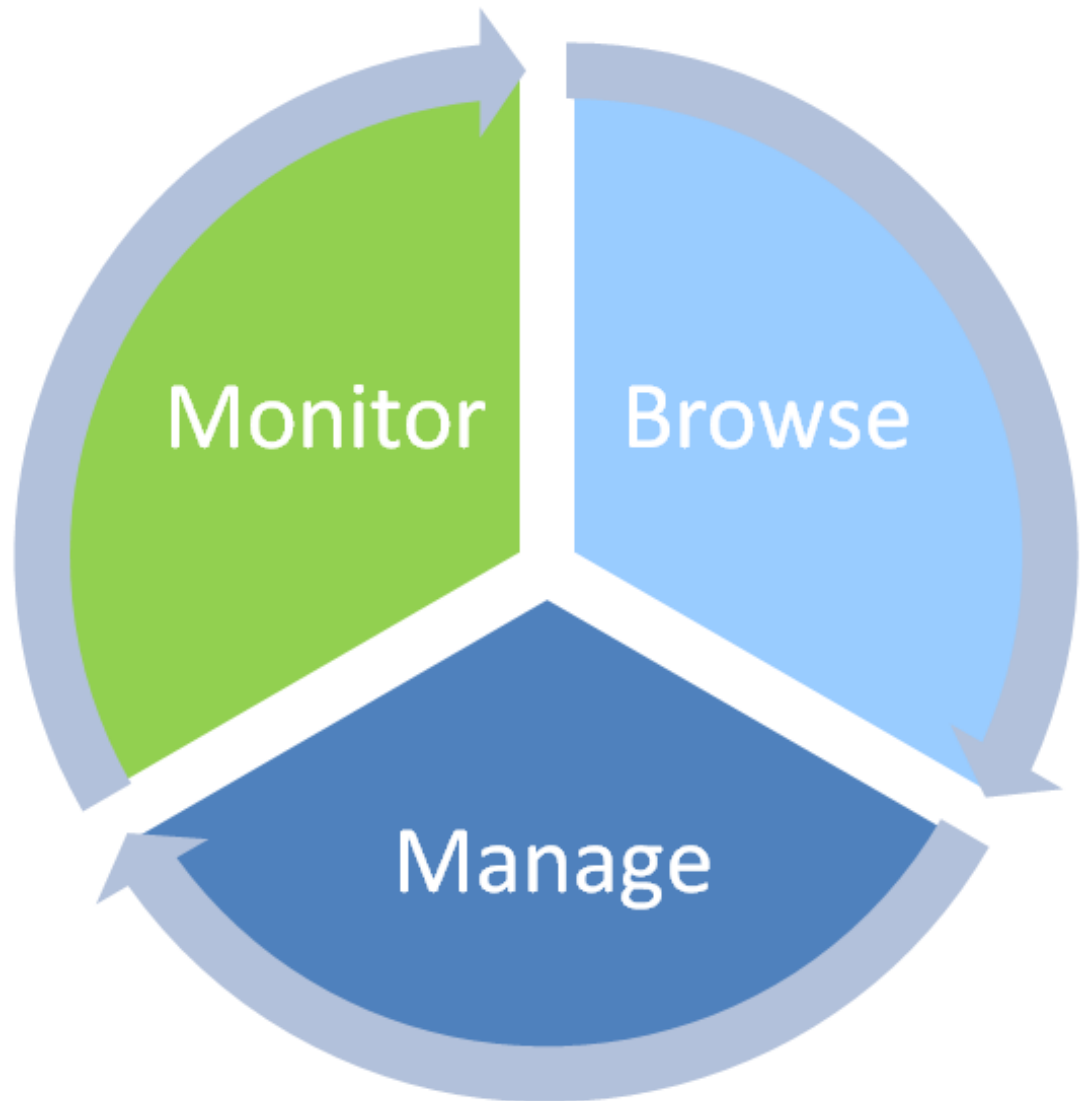


Figure 3-4 JMS Console Workflow

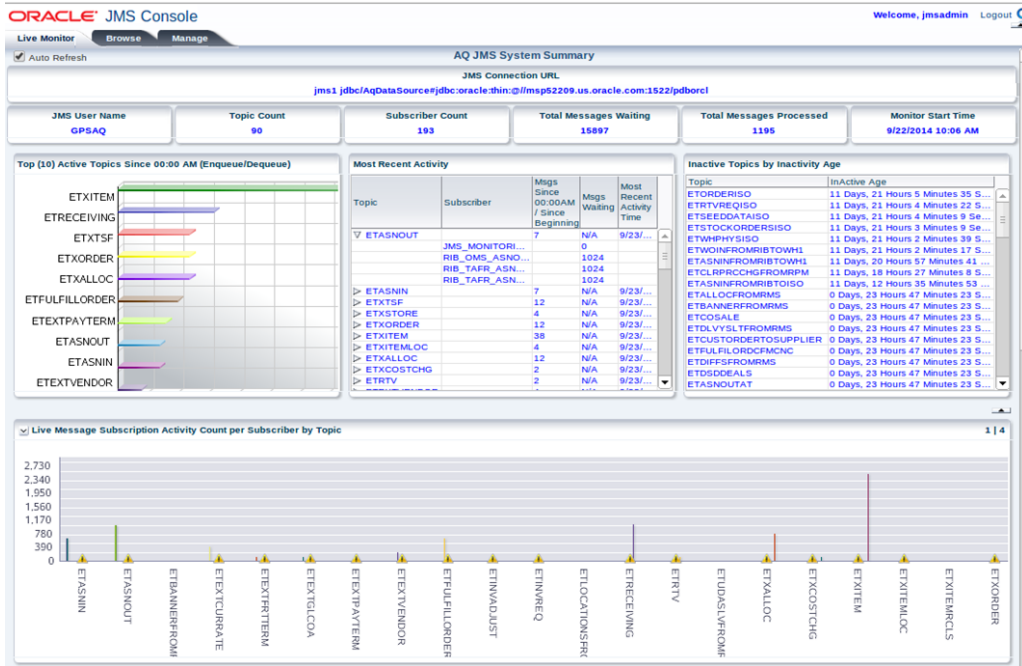


The JMS Admin provides an operational view point of RIB's JMS server.

- **Monitor** - Unattended view of the JMS system.
- **Browse** - Discover and Drill down into various aspects of the JMS Server.
- **Manage** - Provide JMS Server operation management functionality.

JMS Admin – Live Monitoring

Figure 3-5 JMS Console



The Live Monitor tab is the landing page of the JMS Console Application. It provides an unattended high-level view of the JMS Server.

Live Monitoring also provides

- Current message volume information.
- Provides most recent activity details.
- Identify problematic JMS topics and bring to user notice.
- Display a full view of all message activities on all topics.
- Generate alerts based on set thresholds.
- Show visual clues on problematic topics.
- Gives an aggregated view of the system.

The Browse and Manage Tabs provide more specific data metrics and ability to manage messages on the JMS topics.

JMS Browse

The Browse Tab allows you to discover and drill into the internals of JMS Servers.

- Discover all topics and get a listing of them from the server.
- Drill into each topic and discover all subscribers.

- Drill into each subscriber and get message count information.
- Browse message content inside JMS server
- Browse message headers inside JMS server

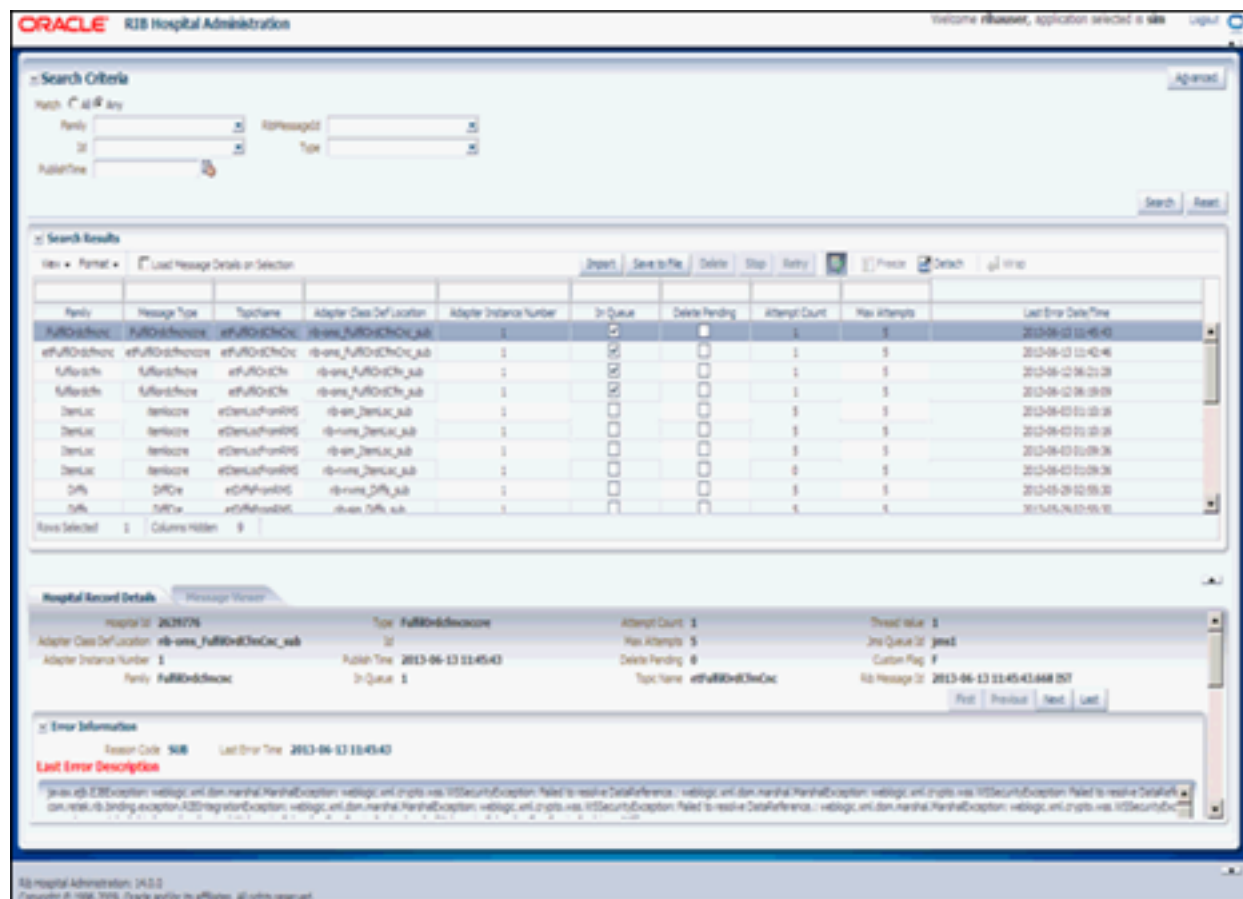
JMS Manage

The Manage tab allows you to interact with the JMS server with some core messaging system functionality.

- Publish messages to JMS topics easily
- Dump messages to files from topics
- Drain messages from topics
- Configure preference for threshold and alerts

RIB Hospital Administrator (RIHA)

Figure 3-6 RIB Hospital Administrator



Oracle Retail Integration Bus Hospital Administration or RIB Hospital Administration (RIHA) is a tool to manage RIB messages in the RIB hospital error tables.

RIHA can search for hospital records, stop a message from being retried, retry a message for which maximum system-set retries have been tried, and delete a message from the tables. RIHA can insert new records into Hospital tables. You can also update existing Hospital records and message payloads.

Using RIHA is the recommended way to perform all RIB Hospital error table operations.

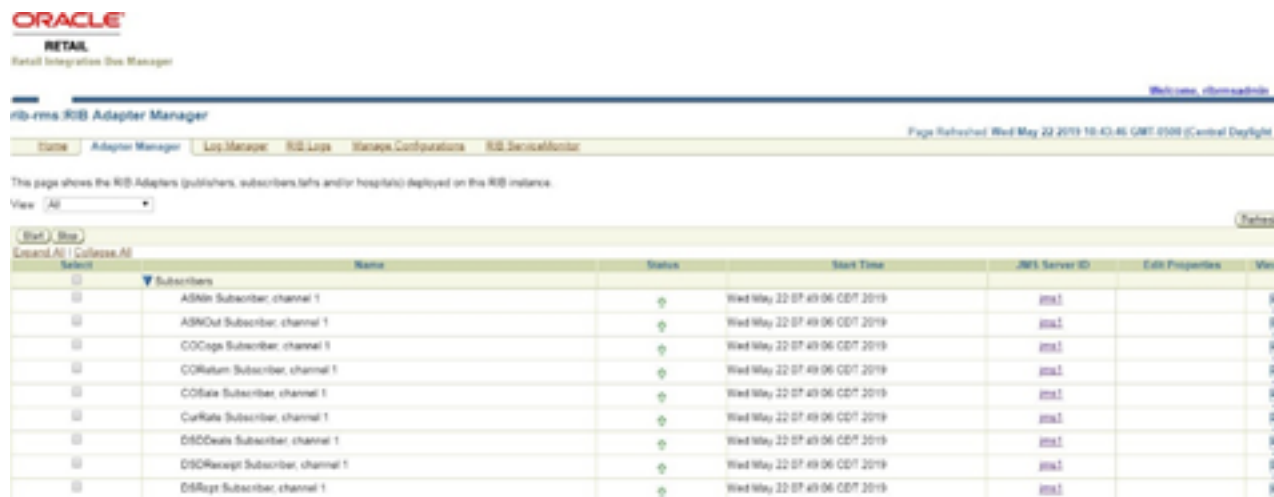
RIHA Operations

- Delete a Message
- Stop a Message
- Retry a Message
- View and Edit a Message
- Save a Message Locally
- Import a New Hospital Record to Hospital Tables
- Update an Existing Hospital Record

RIBForXXX Administrator User Interface

Each of the RIB components (RIB-RMS, RIB-LGF, and so on) has its own UI that exposes run-time and configuration details. The Administrator UI provides operations to turn on or off RIB adapters, control logging levels and to see logs.

Figure 3-7 RIBForXXX Administrator User Interface



4

Implementing RIB-EXT

RIB-EXT is an Oracle Retail Integration Application that provides necessary communication channel for external applications to publish and consume message from RIB's JMS on cloud and premise.

How to Send and Receive Messages to/from RIB System using SOAP API's

External Application as a Publisher (rest-app) using OAuth2

For external applications to publish to the RIB JMS on cloud, it needs to use a publishing webservice provided by rib-ext.

The end point of publishing service follows below pattern:

Table 4-1 Publishing Service Pattern

Resource	HTTP Method	Endpoint
Ping	GET	GET http://<external_LB_url>/<rics-sub-namespace>/rib-ext-services-web/resources/publisher/ping
Publish	POST	http://<external_LB_url>/<rics-sub-namespace>/rib-ext-services-web/resources/publisher/publish

- RIB-EXT publishing service REST endpoints are protected using OAuth2 token-based authentication meaning end points are accessible by sending along an access token.
- Scope will be used for authorization of REST services. Scope for RICS is in the following format- rgbu:rics:RICS-<Environemnt Type><Environment Index> (that is, rgbu:rics:RICS-DEV1).
- Client Credentials grant type is supported.

For getting access to RICS publishing service you need to create a client app in IDCS. IDCS app generates an access token that will be used for making publishing service calls. Follow steps for creating the client app in IDCS.

Create OAuth2 Client Application in IDCS

Use Retail Home for creating the client app in IDCS. Once app is created you will get client id and client secret both of them necessary to get access token. Follow the instructions below for generating the access token and making service call using OAuth2 token.

1. Login into retail home as retail home administrator.

ORACLE Cloud
rgbudev sandbox
Oracle Cloud Account Sign In

User Name
RETAIL_HOME_ADMIN_USER

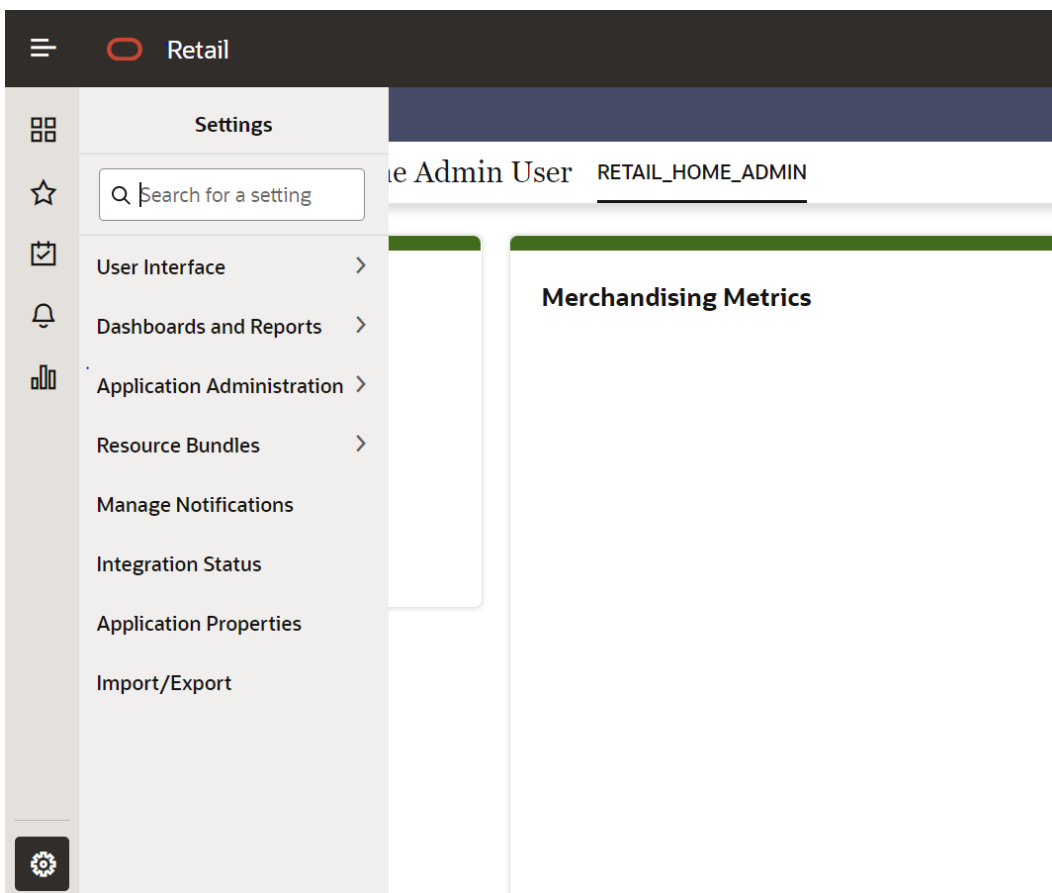
Password
●●●●●●●●●●

Sign In

Need help signing in? [Click here](#)

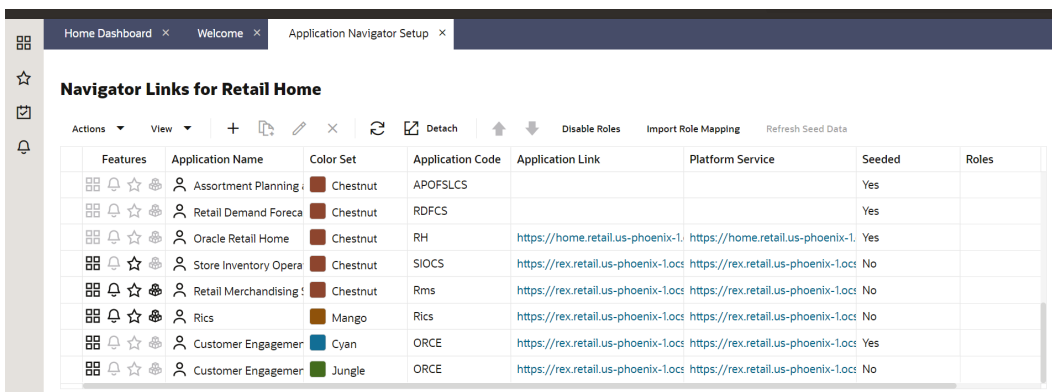
[Cookie Preferences](#)

2. In retail home screen click on Settings menu icon on the left and then click on **Application Administration**.

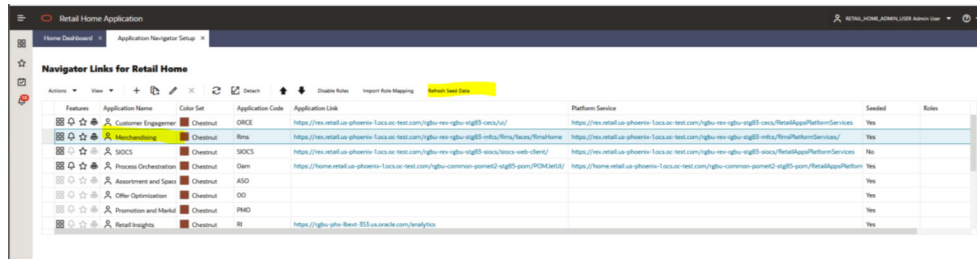


3. On the Application Administration menu click on Application Navigator Setup. Notice all the hosted applications are listed here with their application and platform service url.

Settings -> Application Administration->Application Navigator Settings



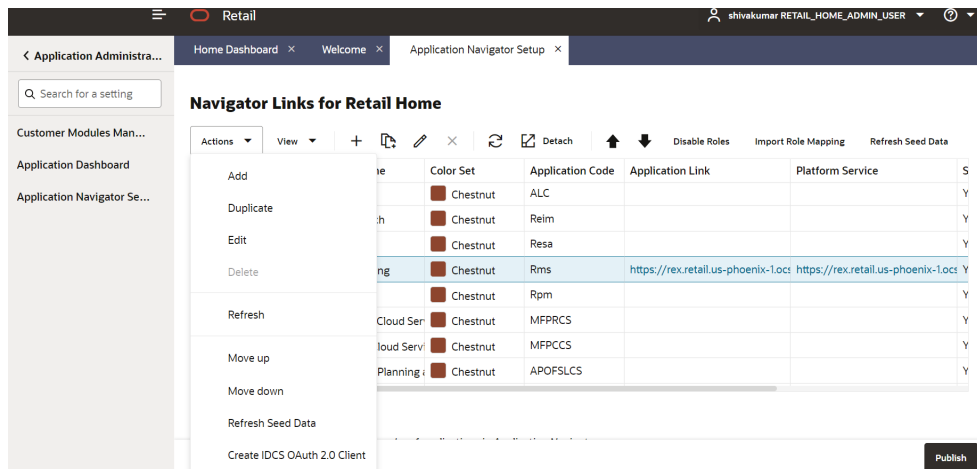
4. Look for application with name RICS. If you are not seeing RICS application try refreshing seed. Steps
 - a. Select the row with the application code as Rms.
 - b. Click the **Refresh Seed Data** button on top right corner of the menu.
 - c. Wait for some time and refresh the screen.
 - d. RICS should reflect now.



- If RICS application is not reflecting even after following step 4. Select the row with the application code as Rms and click on the Actions menu on top left. Select Create IDCS OAuth 2.0 Client. A dialog will open for entering oauth2 client details.

Note:

Create IDCS OAuth 2.0 Client option is available only for applications those have platform service URL mentioned. RICS is making use of merch platform service as both the apps are sharing same IDCS tenancy.



- Skip this step if RICS application is not showing up. One of either Step 5 or Step 6 needs to be followed.

Select a row with application code as RICS. Click on the Actions menu on top left and select Create IDCS OAuth 2.0 Client. A dialog will open for entering oauth2 client details.

Actions	View	+	🔗	✎	✕	🔄	📄	Detach	↑	↓	Disable Roles	Import Role Mapping	Refresh Seed Data
Add	et		Application Code	Application Link	Platform Service	Seeded	Roles						
Duplicate	estnut		APOFSLCS			Yes							
Edit	estnut		RDFCS			Yes							
Delete	estnut		RH	https://home.retail.us-phoenix-1.	https://home.retail.us-phoenix-1.	Yes							
Refresh	estnut		SIOCS	https://rex.retail.us-phoenix-1.ocs	https://rex.retail.us-phoenix-1.ocs	No							
Move up	estnut		Rms	https://rex.retail.us-phoenix-1.ocs	https://rex.retail.us-phoenix-1.ocs	No							
Move down	ingo		Rics	https://rex.retail.us-phoenix-1.ocs	https://rex.retail.us-phoenix-1.ocs	No							
Refresh Seed Data	an		ORCE	https://rex.retail.us-phoenix-1.ocs	https://rex.retail.us-phoenix-1.ocs	Yes							
Create IDCS OAuth 2.0 Client	ngle		ORCE	https://rex.retail.us-phoenix-1.ocs	https://rex.retail.us-phoenix-1.ocs	No							

7. This dialog takes the following values:

App Name is 2-100 characters and will be used as the name in IDCS. Provide unique application name.

Description is a detailed description of the application.

Scope: <Custom environment-specific scope>

The scope pattern that is used in the RICS IDCS app creation template is `rgbu:rics:<SERVICETYPE>-<ENVIRONMENT>` where `SERVICETYPE` is RICS and `ENVIRONMENT` is the environment type (STG, PRD, UAT, DEV1, DEV2, and so on).

For example:

```
"scope": "rgbu:rics:RICS-PRD" "scope": "rgbu:rics:RICS-STG"
```

Create IDCS OAuth 2.0 Client

⚠ Client ID and Client Secret will be displayed once

Submitting this form will provide you with a Client ID and Client Secret which will be displayed one time only

App Name

Description

Scope 1

+ Add Scope

Cancel

OK

- When the application is created, another dialog will open to show the client ID and client secret of the new application. These values should be copied down to a safe location, as they will only be shown once. Retail Home cannot retrieve the credentials again after the dialog is closed.

New IDCS OAuth 2.0 Client

Display Name: RICS_TEST

Client ID: RICS_TEST_APPID

Client Secret: 998e1e1d-f146-45a5-a9a1-99785e3ebf43

Done

- Client ID and Client Secret from previous step will be used for generating access token.

Sample code for generating Access Token:

```

clientId=RICS_TEST_APPID
clientSecret=998e1e1d-f146-45a5-a9a1-99785e3ebf43
idcsUrl=https://
idcs-234e8f7334564936aa0ed93f2c39e9ca.identity.pint.oc9qadev.com
scope=rgbu:rics:RICS-STG99
ec=$(echo -n "$clientId:$clientSecret" | base64 -w 0)

AccessToken=$(curl -iv \
-H "Authorization: Basic $ec" \
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8" \
--request POST $idcsUrl/oauth2/v1/token \
-d "grant_type=client_credentials&scope=$scope" | grep -o -P '(?
<=access_token":).*?(?=",token_type)')

echo $AccessToken

```

- Now service call can be made by passing along the access token generated in previous step.

Here is sample curl command with Bearer token and rib-ext publisher ping

```

ribExtServiceUrl=https://rex.retail.us-phoenix-1.ocs.oc-test.com:443/rgbu-
rex-eit-stg99-rics/rib-ext-services-web/resources/publisher/ping
curl -ivkL --noproxy '*' -H "Authorization: Bearer $AccessToken" -H
"Content-Type: application/xml" -X GET $ribExtServiceUrl

```

Sample response

```

{"message": "ping() was called with input String of: hello"}

```

- Publishing a message using access token.

Here is sample curl for publishing a message

```

ribExtServiceUrl=https://rex.retail.us-phoenix-1.ocs.oc-test.com:443/rgbu-rex-eit-
stg99-rics/rib-ext-services-web/resources/publisher/publish
curl -ivkL --noproxy '*' -H "Authorization: Bearer $AccessToken" -H "Content-
Type: application/xml" -X POST $ribExtServiceUrl --data '<v1:ApplicationMessages
xmlns:v1="http://www.oracle.com/retail/integration/rib/ApplicationMessages/v1">
<v1:ApplicationMessage>
<v1:family>InvAdjust</v1:family>
<v1:type>InvAdjustCre</v1:type>
<v1:payloadXml>&lt;InvAdjustDesc xmlns=&quot;http://www.oracle.com/retail/
integration/base/bo/InvAdjustDesc/v1&quot; xmlns:xsi=&quot;http://www.w3.org/2001/
XMLSchema-instance&quot;
xsi:schemaLocation=&quot;http://www.oracle.com/retail/integration/base/bo/
InvAdjustDesc/v1
http://www.oracle.com/retail/integration/base/bo/InvAdjustDesc/v1/
InvAdjustDesc.xsd&quot;
&gt;&lt;dc_dest_id&gt;DC_ES&lt;/
dc_dest_id&gt;&lt;InvAdjustDtl&gt;&lt;item_id&gt;Aline&lt;/
item_id&gt;&lt;adjustment_reason_code&gt;stri&lt;/
adjustment_reason_code&gt;&lt;unit_qty&gt;22.4&lt;/unit_qty&gt;
&lt;transshipment_nbr&gt;ss&lt;/
transshipment_nbr&gt;&lt;from_disposition&gt;ss&lt;/
from_disposition&gt;&lt;to_disposition&gt;sss&lt;/
to_disposition&gt;&lt;from_trouble_code&gt;sss&lt;/from_trouble_code&gt;
&lt;to_trouble_code&gt;ss&lt;/to_trouble_code&gt;&lt;from_wip_code&gt;aaa&lt;/
from_wip_code&gt;&lt;to_wip_code&gt;sss&lt;/
to_wip_code&gt;&lt;transaction_code&gt;4&lt;/
transaction_code&gt;&lt;user_id&gt;TestUser&lt;/user_id&gt;
&lt;create_date&gt;1999-10-23T20:27:56.32&lt;/
create_date&gt;&lt;po_nbr&gt;PratapOrd96&lt;/po_nbr&gt;&lt;doc_type&gt;P&lt;/
doc_type&gt;&lt;aux_reason_code&gt;string&lt;/aux_reason_code&gt;
&lt;weight&gt;12.4&lt;/weight&gt;&lt;weight_uom&gt;smn&lt;/
weight_uom&gt;&lt;unit_cost&gt;20.4&lt;/
unit_cost&gt;&lt;InvAdjustUin&gt;&lt;uin&gt;123&lt;/uin&gt;
&lt;status&gt;4&lt;/status&gt;&lt;/InvAdjustUin&gt;&lt;/InvAdjustDtl&gt;&lt;/
InvAdjustDesc&gt;</v1:payloadXml>
</v1:ApplicationMessage>
</v1:ApplicationMessages>'

```

Sample response

```
{"message": "Publish done"}
```

External Application as a Subscriber (rest-app)

For an external application to consume the message from the RIB's JMS on cloud, it has to host the Injector Service. Injector Service is a ReST webservice that is made available as a pluggable jar.

A pluggable jar is provided which contains all the wrapper classes to help in implementing injector service. rib-injector-services-web-22.1.302.0.war is the pluggable jar which can be included into the external application deployable file for example, ext-app.ear/lib. Once pluggable jar is added, endpoint for injector service will be exposed as follows:

```
http://<external-app-host>:<port>/ rib-injector-services-web/resources/injector/inject
```

Pluggable jar is provided for reference however customer can choose to write their own injector service by adhering to REST service contract detailed in next section.



Note:

For information on pluggable jar, see the Client Connector For Oracle Retail Integration Cloud Service 22.1.302.0 (Patch) available on My Oracle Support.

How to implement Injector Service (Service Contract) using ReST

Here is the Rest service contract detail:

1. Keep the path as Injector/inject.

```
@Path("/injector")
```

2. Use POST for this service. As the input message object itself has identifier (message type- CRE/MOD) they don't need to use the PUT/PATCH. they can use message type to build the implementation logic.

```
@POST
@Path("/inject")
@Consumes({MediaType.APPLICATION_XML})
```

3. The input would be MediaType.APPLICATION_XML and the structure would be 'ApplicationMessage' object. (file attached for reference).

```
<xs:element name="ApplicationMessage">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="family" type="string25"/>
      <xs:element name="type" type="string30"/>
      <xs:element name="businessObjectId" type="string255" minOccurs="0"/>
      <xs:element ref="ApplicationMessageRoutingInfo" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="payloadXml" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4. Customer can utilize the payload.properties file for validation of message family and type.
5. Return type should be JSON, see below example:

```
String message = "{\"message\": \"Inject successful.\"}";
return Response.ok(message, MediaType.APPLICATION_JSON).build();
```

6. For exception response customer needs to follow the structure of exceptionVO.

How to Secure Injector Service with OAuth2

Primary authentication mechanism in the cloud is OAuth2 using IDCS authenticator. RIB uses IDCS OAuth2 for authentication of ReST calls both inbound and outbound (publisher/injector restful services). Hence Injector service exposed by external service should be secured with OAuth2. This chapters covers the key points that should be taken into consideration while protecting the resources exposed by external application.

Prerequisites

- IDCS should be same as RICS.
- Use Client Credentials grant type with scope to provide access to resource.
- Following is the screen shot of a sample IDCS app with scope added

The screenshot shows the Oracle Cloud Identity and Access Management console for an application named RGBU_RICS_DEV23. The application is in an ACTIVE state. The configuration includes application information, OAuth configuration, and resource server configuration. A scope named RICS_DEV23 is defined with a protected status of 'Yes' and a requirement of 'No'.

Application Information:

- Description: Oracle Real Integration Cloud Service
- Custom sign-in URL: -
- Custom error URL: -
- Display in My Apps: No
- User can request access: No
- Enforce grants as authorization: Disabled

OAuth configuration:

- Resource server configuration: Resource server configuration for this application is enabled.
- Configure application APIs that need to be OAuth protected
- Access token expiration (seconds): undefined
- Allow token refresh: Disallowed
- Primary audience: rgburics

Secondary audience:

Secondary audience	Protected
No secondary audience added	Protected

Scopes:

Scope	Protected	Display name	Description	Required
RICS_DEV23	Yes			No



Note:

Follow IDCS documentation for detailed instruction on setup.

RIB-EXT Side of Configuration to Point to External Application

Below are the steps to point rib-ext to the correct injector service.

Table 4-2

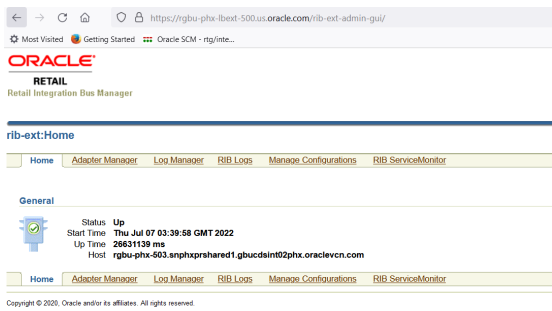
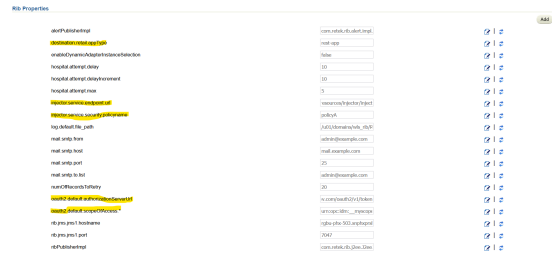
Category	Step	Comment
Access RIB Admin GUI	Access the rib admin GUI at <a href="https://<external-load-balancer>/rib-ext-admin-gui">https://<external-load-balancer>/rib-ext-admin-gui Log in with the admin user.	
Verify Configuration and update	<p>Navigate to Manage Configurations -> System options</p> <p>Search for and verify the following:</p> <ol style="list-style-type: none"> 1. destination.retail.appType: rest-app 2. Update the value for InjectorService URL (injector.service.endpoint.int.url). URL should point to inject service provided by external application. (e.g.- <a href="https://<host>:<port>/rib-injector-services-web/resources/injector/inject">https://<host>:<port>/rib-injector-services-web/resources/injector/inject) 3. Security Policy (injector.service.security.policyname) : policyA 4. IDCS OAuth Server URL (oauth2.default.authorizationServerUrl): <a href="https://<idcs-tenant>/oauth2/v1/token">https://<idcs-tenant>/oauth2/v1/token 5. OAuth2 Token Scope: Default (i.e.- urn:opc:idm:___myscopes___) 	
Supported Grant Types for OAuth2	There are two grant type supported. Based on the use case follow steps below.	

Table 4-2 (Cont.)

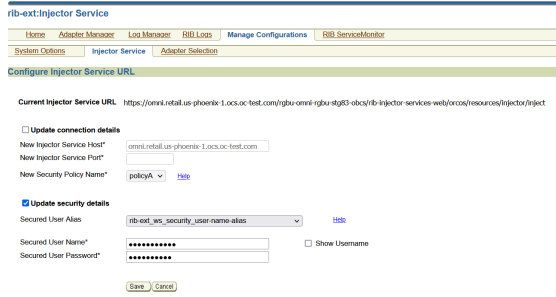
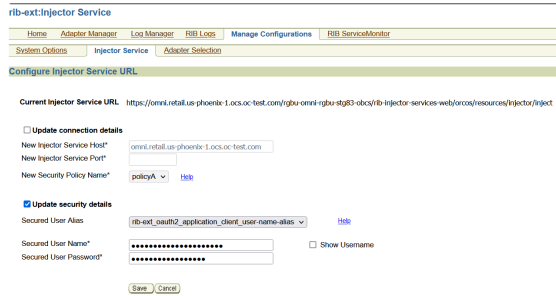
Category	Step	Comment
Update username and password	Steps for grant_type as password	
	Navigate to Manage Configurations - > Injector Service	
	Update details	
<ol style="list-style-type: none"> 1. Choose rib-ext_ws_security_user_name_alias from drop down. 2. Update rib-ext_ws_security_user_name_alias with valid username/password. 3. Click on Save. 		
Update ClientID/ Secret needed for token generation	Navigate to Manage Configurations - > Injector Service	
	Choose rib-ext_oauth2_application_client_user-name-alias from drop down and update details	
	<ol style="list-style-type: none"> 1. Provide a valid Client ID in username. 2. Provide a valid Client Secret in password. 3. Click on Save. 	

Table 4-2 (Cont.)

Category	Step	Comment
How to verify provided injector service details are correct	Verify if the provided injector service URL and credentials are correct for grant_type password using curl.	<p>Execute the following curl commands for grant_type password:</p> <pre> ClientId=56c7eb72f11b43bb98bf2570fa2353eb ClientSecret=bb18aa22-4bb4-41d1-9ed4-fea276651e28 IDCSUrl=https:// idcs-24e4baae56764e91be371e6a2060d66e.identity.c9dev2.oc9qadev.com un=integrationUser pw=Welcome@1234 AccessToken=\$(curl -i -X POST \ --user \$ClientId:\$ClientSecret \ -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8" \ \$IDCSUrl/oauth2/v1/token \ -d "grant_type=password&username=\$un&password=\$pw&scope=urn:opc:idm:__myscopes__" grep -o -P '(?<=access_token":).*?(? =","token_type)') echo \$AccessToken ribExtServiceUrl=https://<external-app-load-balancer>/rib-injector-services-web/resources/injector/ping curl -ivkL --noproxy '*' -H "Authorization: Bearer \$AccessToken" -H "Content-Type: application/xml" -X GET \$ribExtServiceUrl </pre> <p>if you get a 200 response, then the configuration is correct</p> <p>if you get 401 unauthorized, then Client ID and secret are incorrect</p>

Steps for grant_type as client credentials

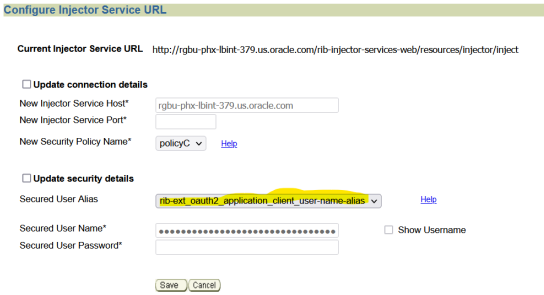

Update username and password to empty

Navigate to Manage Configurations -> Injector Service

Update details.

1. Choose "rib-(app)_ws_security_user-name-alias" as Secured User Alias.
2. Update the Secured User Name with a blank userName.
3. Update the Secured User Password with a blank password.
4. Click on Save.

Table 4-2 (Cont.)

Category	Step	Comment
Update ClientID/ Secret	<p>Navigate to Manage Configurations -> Injector Service</p> <p>Update details</p> <ol style="list-style-type: none"> Choose "rib-(app)_oauth2_application_client_user-name-alias" as Secured User Alias. Update the Secured User Name with clientID. Update the Secured User Password with clientSecret. 	
Ping Test	<p>Navigate to Manage Configurations -> RIB Service Monitor</p> <ol style="list-style-type: none"> Click on ping It should return success 	
How to verify provided injector service details are correct	Verify if the provided injector service URL and credentials are correct.	<p>Execute the following curl commands</p> <pre> ClientId=56c7eb72f11b43bb98bf2570fa2353eb ClientSecret=bb18aa22-4bb4-41d1-9ed4-fea276651e28 IDCSUrl=https:// idcs-24e4baae56764e91be371e6a2060d66e.identity.c9dev2.oc9gadev.com AccessToken=\$(curl -i -X POST \ --user \$ClientId:\$ClientSecret \ -H "Content-Type: application/x-www-form-urlencoded; charset=UTF-8" \ \$IDCSUrl/oauth2/v1/token \ -d "grant_type=client_credentials&scope=urn:opc:ldm:_myscopes_" grep -o -P '(?<=access_token:").*(?=", "token_type)') ribExtServiceUrl=https://rgbu-phx-lbext-351.us.oracle.com/rib-injector-services-web/resources/injector/ping curl -ivkL --noproxy '*' -H "Authorization: Bearer \$AccessToken" -H "Content-Type: application/xml" -X GET \$ribExtServiceUrl </pre>

How to switch Injector Service app Type at Runtime

RIB-EXT is a rest-app by default for CFS and expects injector service also to be of Rest type. ONLY for egress/migration customers who already have injector service SOAP

implementation in GBUCS they should follow these steps to switch from rest to soap based injector calls and vice-versa.

How to Change rib-ext injector-service-app-type from REST to SOAP

1. Open rib-ext admin gui. Go to Manage Configurations > System Options, observe new prop-erty i.e. injector-service-appType added to allow switching injector service app-type at runtime.

By default rib-ext is deployed as rest-app so injector-service-appType is defaulted to.

ORACLE
RETAIL
Retail Integration Bus Manager

Welcome, rics.admin Logout

rib-ext: System Options Page Refreshed Tue Jan 18 2022 10:53:42 GMT+0530 (India Standard Time).

Home Adapter Manager Log Manager RIB Logs Manage Configurations RIB ServiceMonitor

System Options Injector Service Adapter Selection

Rib Properties Add

IsRibAppEnabled	true	
alertPublisherImpl	com.retek.rib.alert.impl	
destination retail appType	rest-app	
disableLogLevelUpdatesForAdapter	Customer_pub,Custome	
enable global email alert	false	
enableDynamicAdapterInstanceSelection	true	
for Alloc_sub drop-messages-of-types	ALLOCFULREP	
for ISORTVReq_sub drop-messages-of-types	RTVREQFULREP	
for ISOStockOrder_sub drop-messages-of-types	SOFULREP	
for Items_sub drop-messages-of-types	ITEMFULREP	
for Order_sub drop-messages-of-types	POFULREP	
for RTVReq_sub drop-messages-of-types	RTVREQFULREP	
for Transfers_sub drop-messages-of-types	TRANSFERFULREP	
for Vendor_sub drop-messages-of-types	VENDORFULREP	
for WHStockOrder_sub drop-messages-of-types	SOFULREP	
hospital attempt delay	10	
hospital attempt delayIncrement	10	
hospital attempt max	5	
injector.service.appType	rest-app	
injector.service.endpoint.url	http://examplehost.com	
injector.service.security.policyname	policyC	
log default file_path	/u01/oracle/user_projec	
mail smtp from	admin@oracle.com	
mail smtp host	mail-router-svc.rgbr-re	
mail smtp port	2501	
mail smtp to list	rtg_dev_us_grp@oracle	
numOfRecordsToRetry	20	
repave notification lead time	30	
request domain	rex.retail.us-phoenix-1.1	
request referer	rex.retail.us-phoenix-1.1	
rib adapter shutdown repave lead time	5	
rib.jms.jms1.hostname	jms1host.example.com	
rib.jms.jms1.port	1521	
ribPublisherImpl	com.retek.rib.j2ee.J2ee	
war http port	8010	
war http protocol	http	
ws.wallet file location	/u01/oracle/user_projec	
ws.wallet.map.name	rib-ext-ws1	
ws.wallet.user.alias	rib-ext_ws_user-name-	
xml_schema_base_url	http://rib-func-artfact-s	

System Options Injector Service Adapter Selection

Home Adapter Manager Log Manager RIB Logs Manage Configurations RIB ServiceMonitor

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

2. Edit injector-service-appType and update this to soap-app. Save the changes.

for WHStockOrder_sub drop-messages-of-types

hospital attempt delay

hospital attempt delayIncrement

hospital attempt max

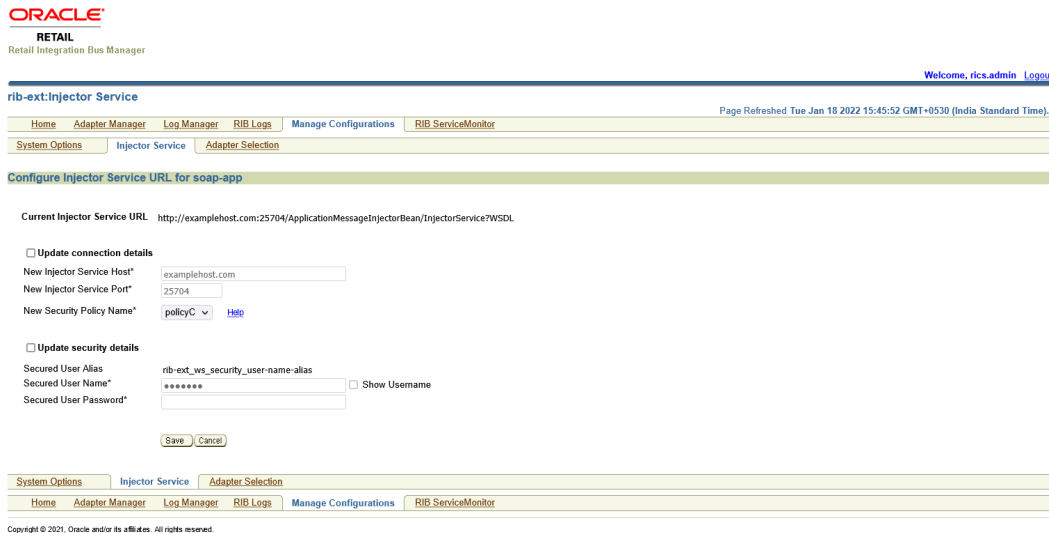
injector.service.appType

injector.service.appType

SOAP-APP

3. Navigate to Manage Configurations > Injector Service tab. Check for the correctness of injector service URL, ensure it points to correct ext-app injector service.

Update rib-ext_ws_security_user-name-alias with correct username/password needed to make inject call.



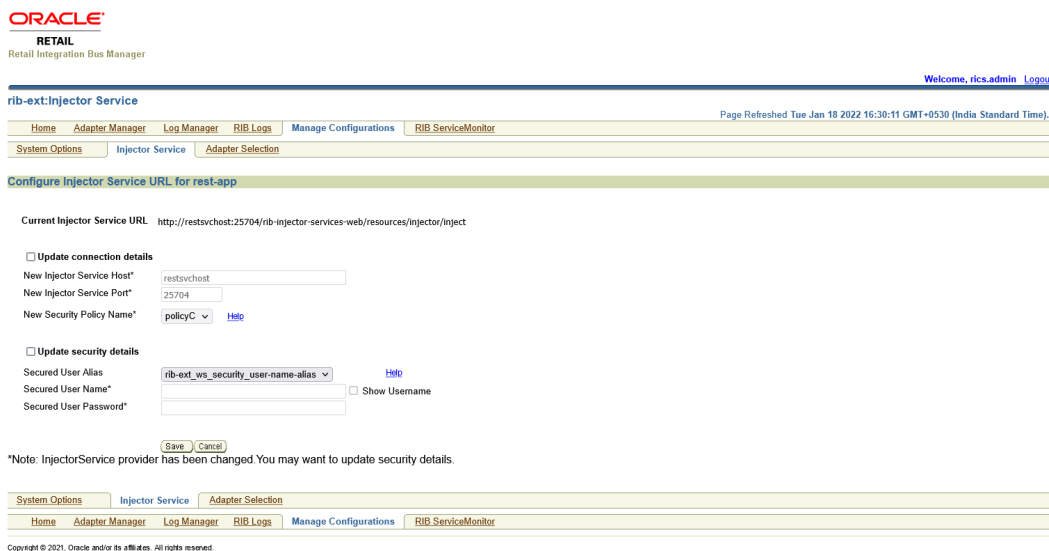
4. Setup is ready now. Do a ping test from RIB ServiceMonitor tab.

How to change rib-ext injector-service-app-type from SOAP to ReST

1. Navigate to Manage Configurations > System Options from admin GUI. Look for injector-service-appType, update this property to switch from SOAP to ReST. Save the changes.



2. Navigate to Injector Service tab. Update host/port and security credentials (rib-ext_ws_security_user-name-alias) if needed.



3. Setup is ready now. Do a ping test from RIB Service Monitor tab.
See [Sample Files](#) (application.wadl, Resource file, Sample request/response etc.).



Note:

A Reference implementation for injector service is provided, See [Reference Implementation of Injector Service Using Tomcat](#) for details.

Error Handling

The RIB infrastructure provides a mechanism called RIB error hospital to handle and manage the error messages. When the publishing or subscription of a message fails in the rib-ext for some reason, it lands in error hospital with a reason code. The retry adapters in the rib-ext application are responsible for retrying the messages in error hospital.

Oracle RIB Hospital Administration (RIHA) is a Weblogic application that allows the management of messages in error hospital. Some of the RIHA operations include:

- Viewing error messages
- Editing error messages
- Retrying error messages
- Stopping error messages

For more information, see the Oracle Retail Integration Bus Hospital Administration Guide.

5

Reference Implementation of Injector Service Using Tomcat

Introduction

For an external application to consume the message from the RIB's JMS on cloud, it has to host the Injector Service. Subscriber adapters in rib-ext makes a ReST call to Injector service to send the message to the external application. This document contains detailed information that can be used for implementing Rest inject service.



Note:

Tomcat is the certified application server here but provided injector service war should work on any standard app server.

Important Notes

- Provided rib-injector-service war (inside RibExtConnectorServiceImplPak22.0.000_eng_ga.zip) runs on tomcat and has all the dependencies for rib in order to consume the message as individual application using RIB-EXT flow.
- No container-managed transaction capability is required.
- Authentication and authorization will be adjusted by the consuming application by editing web.xml to match their own requirements.
- rib-injector-services-light-web war works as an standalone utility, this war is provided as a reference implementation for injector service. After the war is deployed, injector service will be made available as ReST End Point. Service contract WADL should be accessible at `http://<app-host>:<port>/rib-injector-services-web/resources/application.wadl`.
- This pluggable war can be added in to the external application deployable file (for example, ext-app.ear/lib). After deployment, Injector service should be available for access at the following:
`http://<external-app-host>:<port>/rib-injector-services-web/resources/injector/inject`
- The customer can choose to write their own injector service implementation without using rib-injector-services war as long as they adhere to the service contract for Injector. Detailed information is documented in [How to implement Injector Service \(Service Contract\) using ReST](#).
- The external application has to write their own implementation logic for the injectors. However, an implementation jar (injector-sample-impl-22.0.000.jar) is provided for reference. Customers can write custom implementation logic inside injector-sample-impl-22.0.000.jar or can choose to implement on their own.

Step-by-step guide for testing rib-injector-service war on Tomcat

1. Copy jersey jars into <tomcat>/lib folder. App server used here is tomcat which is a web container and doesn't have Jersey libraries packaged inside. In case of full stack Java EE app server such as weblogic this step is not required.

```
cp <ribExtConnectorPak>/Rest-Injector/rib-injector-services-web/jersey-jars
<tomcat>/lib
```

 **Note:**

Jersey jars are packaged inside zip.

 **Note:**

Jars packaged here are for reference purpose. You may use different versions of Jersey jars that may be compatible with external application.

 **Note:**

application-messages-bo-22.0.000.jar is added with jersey zip in order to make this available inside <tomcat>/lib. This jar contains request/response business object and is needed for injector service to work.

2. Deploy the rib-injector-services-light-web-22.0.000.war into tomcat by using the following command.

 **Note:**

Change the name to remove the version number and the word light.

```
cp rib-injector-services-light-web-21.1.000.war <tomcat>/webapps/rib-
injector-services-web.war
```

3. Start the application if not already started.
4. Copy injector-sample-impl-22.0.000.jar inside <tomcat-apache>\webapps\rib-injector-services-web\WEB-INF\lib
5. Test ping. Ping should return response as "Got hello from server."

```
curl -ivl4 --user tomcat:tomcat1 http://localhost:8080/rib-injector-
services-web/resources/injector/ping
```

6. Test inject call using a sample payload data. Create a file called app-message.data with the content as follows.

```
<ApplicationMessage xmlns="http://www.oracle.com/retail/integration/rib/
ApplicationMessages/v1"><family>WH</family><type>WHCRE</
type><payloadXml>&lt;WHDesc xmlns="http://www.oracle.com/retail/integration/
base/bo/WHDesc/v1" &lt;wh>10</wh>&lt;wh_name>g</wh_name>&lt;/
WHDesc&gt;</payloadXml></ApplicationMessage>
```

7. Call inject with the above (app-message.data) data. For a successful inject call, response should be "Inject successful." With the implementation jar provided here, message will get written to log.

```
curl -ivl4 --user tomcat:tomcat1 -H 'Content-Type: application/xml' -d @app-
message.data http://localhost:8080/rib-injector-services-web/resources/injector/
inject
```

Approach for Writing Custom Implementation for Injectors

injector-sample-impl-22.0.000.jar is provided as reference implementation for injector service however customer can choose to write their own custom implementation logic. Steps listed here will help customer to write their own injector classes.

1. To start with implementation, create a file with name injectors.xml. This file contains mapping for the injector implementation class, which will be looked for the given family and msgType. InjectorFactory looks for the injectors.xml, this should be present in classpath. Look at inject-tor-sample-impl-22.0.000.jar/retail/injectors.xml for reference.

Sample Code: For Diffs family and DiffCre message type, injector implementation class is Diffs

```
<injector_config>
<family name="Diffs">
<injector class="oracle.retail.rib.javaee.api.stubs.injector.file.impl.Diffs">
<type>DIFFCRE</type>
</injector>
<injector class="oracle.retail.rib.javaee.api.stubs.injector.file.impl.Diffs">
<type>DIFFDEL</type>
</injector>
<injector class="oracle.retail.rib.javaee.api.stubs.injector.file.impl.Diffs">
<type>DIFFMOD</type>
</injector>
</family>
..
.
</injector_config>
```

2. In the given jar, all the injectors class extends SampleInjector. This is the class where logic for handling the payload will be written. You can write your own implementation class and Diffs can extend that class.

Sample Code:

```
public final class Diffs extends SampleInjector{
..
}
```

3. Custom Implementation class should implement injector interface (contract for inject method).

Sample code:

```
import oracle.retail.rib.common.exception.RetailBusinessException;
import oracle.retail.rib.common.exception.RetailSystemException;
import com.oracle.retail.integration.payload.Payload;
```

```
import com.retek.rib.binding.injector.Injector;
public class SampleInjector implements Injector {

    // dummy impl for Injector
    public void inject(String type, Payload payload)
        throws RetailBusinessException, RetailSystemException {
        // Write logic here
        System.out.println("Inject executed successfully...");
        LOG.info("Inject executed successfully...");
    }
}
```

4. Copy custom implementation jar in-side <tomcat-apache>\webapps\rib-injector-services-web\WEB-INF\lib for it to work.

6

Implementing BDI-EXT

BDI External Job Admin as Receiver

For example, sender application is RMS and receiver is a third party application. There will be external application for the integration to happen as External edge application. External edge application organizes all the importer jobs. External edge application provides GUI and CLI tool to manage jobs like start/stop/restart jobs.

The External Importer Job imports data set for an Interface Module from Inbound Interface Tables into application specific transactional tables. Importer jobs are application specific jobs.

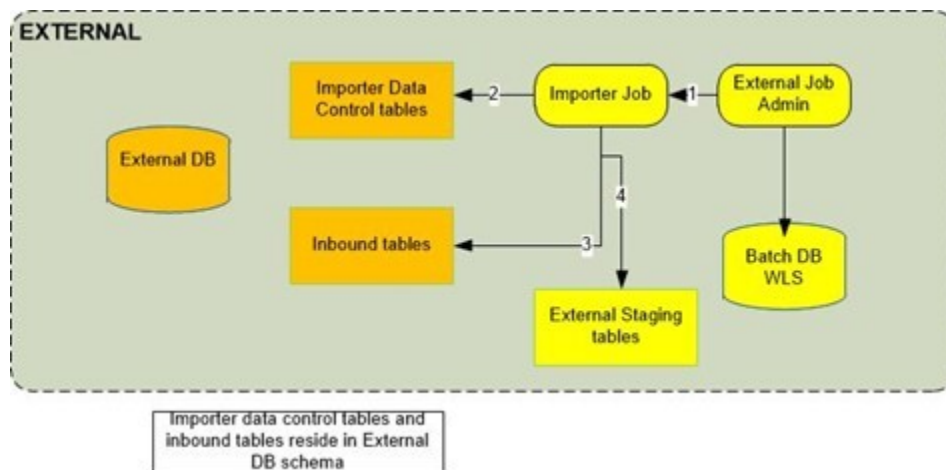
External Importer Job

The tables `BDI_IMPRTR_IFACE_MOD_DATA_CTL` and `BDI_IMPORTER_IFACE_DATA_CTL` act as a handshake between the receiver service and importer jobs. When the Receiver Service completes processing a data set successfully, it creates an entry in these tables.

An entry in the table `BDI_IMPRTR_IFACE_MOD_DATA_CTL` indicates to the Importer Job that a data set is ready to be imported.

The Importer job imports a data set for an Interface Module from inbound tables into application specific transactional tables. Importer jobs are application (for example SIM/ RPAS/EXTERNAL) specific jobs. It uses the Importer Data Control Tables to identify whether a data set is ready for import or not.

Figure 6-1 External importer Job



For each required interface, implement the logic in the "import" function of the "<InterfaceModule_Name>_Importer_Body.sql" file as in the indicated section below. The sql file is located in <bdi-edge-external-job-home>/setup-data/ddl/ folder.

```

create or replace PACKAGE BODY Brand_Fnd_Importer AS
    interfaceShortNames bdi_InDataControl.interfaceShortNamesType :=
        bdi_InDataControl.interfaceShortNamesType('Brand');
FUNCTION import(
    appName          IN VARCHAR2,
    interfaceModule IN VARCHAR2,
    dataSetType     IN VARCHAR2,
    I_job_context   IN VARCHAR2,
    dataSetId       IN NUMBER DEFAULT NULL,
    O_error_message OUT VARCHAR2)
    return NUMBER IS
    interfaceModuleDataControlId NUMBER;
    beginSequenceNumber NUMBER;
    endSequenceNumber NUMBER;
    returnCode NUMBER := 0;
    errorMessage VARCHAR2(1000);
BEGIN
    --Implement business logic to purge the data from the transaction tables.
    --purge_Brand_Fnd(returnCode, errorMessage);
    --Call BDI package to read data from inbound table and write to given transactional/staging
    bdi_inDataControl.beginImport(interfaceModuleDataControlId, interfaceModule,
        appName, I_job_context, dataSetId, errorMessage);
    --dms_output.put_line('beginImport complete. ');
    for i IN interfaceShortNames.FIRST .. interfaceShortNames.LAST LOOP
        bdi_inDataControl.getDataSetInfoForInterface(interfaceModuleDataControlId,
            interfaceShortNames(i), beginSequenceNumber, endSequenceNumber);
        --dms_output.put_line('getDataSetForInterface is complete');
        --dms_output.put_line('begin seq num: ' || beginSequenceNumber);
        --dms_output.put_line('end seq num: ' || endSequenceNumber);
        --Implement business logic to move data to transaction tables.
        --move_Brand_Fnd(interfaceShortNames(i), beginSequenceNumber, endSequenceNumber, returnCode,
        IF returnCode = 1 THEN
            --dms_output.put_line('move_Brand_Fnd return code 1');
            EXIT;
        END IF;
END IF;

```

External Importer

1. Importer job is run from App B EXTERNAL Job Admin application through REST or UI.
2. Importer job checks for data sets in importer data control tables.
3. If data set is available for import, importer job downloads data from inbound table.
4. Importer job loads data to App B EXTERNAL staging tables.

Configure External Job Admin as Receiver in the Process Flow

System options properties in bdi-process-flow-admin-deployment-env-info.json allow you to configure the available destination apps and appsInScope.

allAvailableDestinationApps property mentions all the applications available as destination.

The appsInScope property mentions the applications that are in scope. Add an external application in the appsInScope property to make it available as a receiver.

```

"SystemOptions": [
  {"name": "allAvailableDestinationApps", "value": "SIM, RPAS, EXTERNAL, OCDS, RFI, RMS"},
  {"name": "appsInScope", "value": "SIM, RPAS, OCDS, RFI, EXTERNAL"}
]

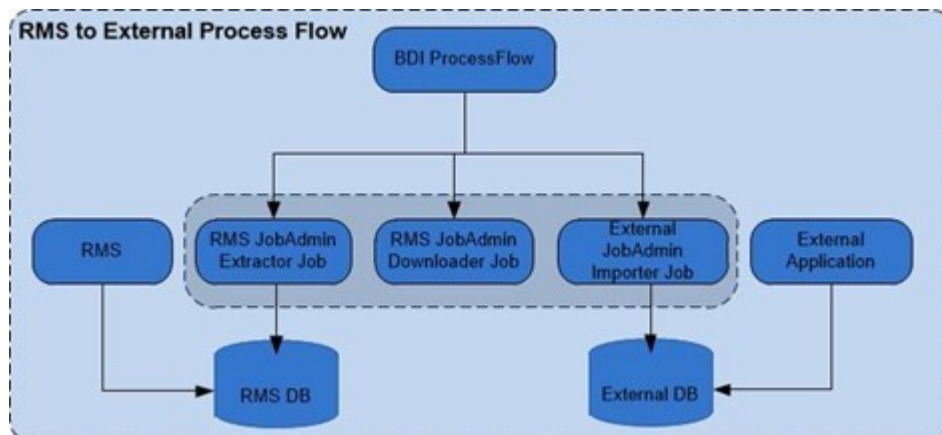
```

External BDI Process Flow

A process flow is a generic concept and is not limited to BDI. However all the out-of-box process flows are for data transfers from a retail application to one or more retail applications.

There are process flow dsl files for each interface from RMS to external and External to other applications which have all the activities for the particular interface as depicted in below pictures. Scheduler will trigger the process flow to execute the activities within the dsl file. Process flows are available out of box to move data end to end. The only thing to implement is extractor or importer packages or both.

Figure 6-2 Merchandising to External Process Flow



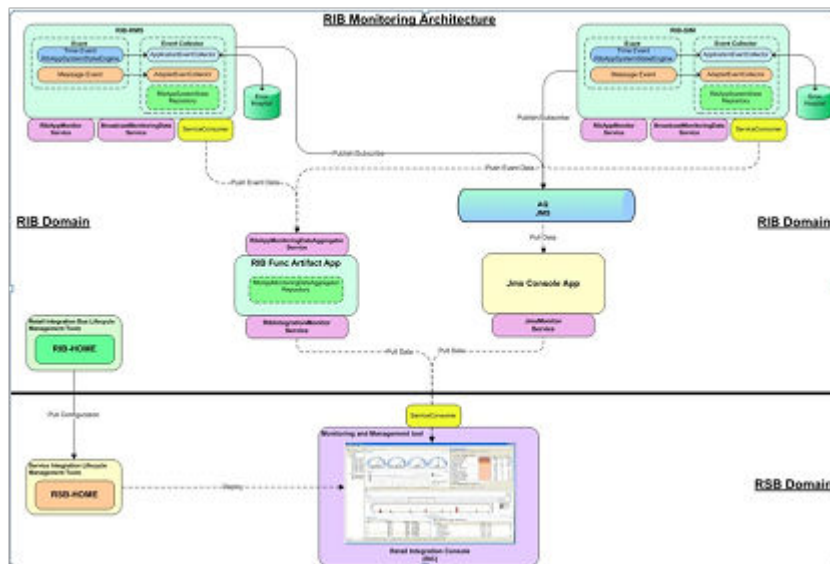
7

Monitoring at Run Time

RIB runtime monitoring enables you to monitor the state and volume of messages running through the RIB system. It also provides the status of various components of the system. The current RIB system and message flows are interrogated transparently to collect useful metrics that immensely enable business users and system administrators to review the state and health of the system. The monitoring enhancement collects application and adapter statuses, message event counts, transaction counts, error hospital statistics, and server resource utilization statistics.

The following graphic describes the architecture of the system:

Figure 7-1 RIB Monitoring ARchitecture



Instance and Central Repository

The monitoring metric data is collected in the rib-<app> instances. The data collected from all rib-<app> instances are consolidated in the central location. Both the collection and consolidation server instances store the data in in-memory repositories. Various pieces of data are collected at different times based on the nature of data and performance considerations. At any point of time, the repository data shows a complete picture of the state as of the last data collection time.

Monitoring Data as XML

The collected data is reported in a defined format. The monitoring data is exchanged between components that produce and consume in XML format. rib-<app> instances produce

the data and the central repository and Retail Integration console (RIC) (or third-party tools) consume the monitoring data.

Push Versus Pull

Sometimes, data is collected by scheduled background jobs. Message related data is collected asynchronously as the messages are consumed/published by adapters. The collected metric data is kept in a local repository in the rib-<app> instance. This information is pushed to a central repository (in memory) on a scheduled frequency (every two minutes). If any rib-<app> is down, the central repository does not receive data from that instance. The Central repository does not poll for data nor pull data from the rib-<app> instances. This way the central repository has no dependency on the rib-<app>s.

While each rib-<app> has its own monitoring data, the central repository holds the consolidated data from all the rib-<app> instances.

Service Interfaces

The monitoring data in the rib-<app> instances and the central repository are made available to RIB monitoring system as well as the third-party tools via SOAP web services running in the respective server instances.

What is an Event?

RIB messages flow from the publishing apps to subscribing apps, TAFRs, and error hospital in the RIB system. Sometimes, messages can be rolled back due to application or system errors. Each attempted delivery, whether successful or not, is called an Event. The RIB monitoring system counts the events which include both successful and failed delivery of messages. Also, any changes in the adapter status, error hospital data, server resource utilization etc. is considered an event.

There are two types of events - Adapter Events and Application Events.

How are Event Count and Messages Count Related?

Event count includes both successful and failed message counts. There is no reliable way of getting the exact successful message count without affecting the performance of the system. Hence, the RIB monitoring system collects event counts instead of message counts. For the most part, they are similar, but not exact.

Adapter Events

Adapter events are adapter level events like message flows (subscription, publishing) and adapter statuses. In the RIB monitoring system, message related adapter events are collected in real-time. Adapter status events are collected by scheduled background threads.

Application Events

Application events are application level events like server resource (CPU, Memory) utilization, application status, error hospital data, etc. These metrics are collected by scheduled background threads.

Event Collection Schedule

Various events in the system are collected at various times.



Note:

There is a difference between the collection time and reporting time. For example, even though the event counts are collected in real-time, they are not available in the central repository immediately.

The following is a complete schedule of collection times:

Table 7-1 Schedule of Collection Times

Metric	Event Type	Schedule
Event Count	Adapter	Real time
Adapter Execution Time	Adapter	Real time
API Execution Time	Adapter	Real time
Adapter Status	Adapter	Every three minutes
Application Status	Application	At startup
Error Hospital Statistics	Application	Every five minutes
CPU Utilization	Application	Every five minutes
Memory Utilization	Application	Every five minutes

Publisher Versus Subscriber Events

The publishing event does not collect certain metrics, like the API Execution Time, since it is not possible to find out the API execution time once the message is published. It collects only the Adapter Execution time, which is the time taken to publish the message.

TAFR Instrumentation

TAFRs are monitored for collecting various time metrics. Measuring the time for the TAFR API execution begins as soon as the TAFR starts transforming the inbound message to an outbound message and ends when the message get transformed. Collecting Adapter Execution Time begins as soon as the message is available for the rib-tafr to transform and ends after routing the message to the destination topic.

Data Retention

The monitoring data is collected in rib-<app> repositories and a central repository in the functional artifact app. These are in-memory repositories. The information in the repositories is lost when the application is restarted. Additionally, the repositories are not purged, so the data collects as long as the applications run. The monitoring data is collected in hourly buckets. There can only be a maximum of 24 records per day. This strategy reduces the chances of the system going out of memory.

Metric Definitions

The following sections describe the metrics that are collected by the system.

Event Counts

When a message is subscribed or published, an event is generated to increment the event count for the hour of the day.

Adapter Execution Time

For a subscriber adapter, the time is noted as soon as the message arrives. At the end of the `onMessage` method the difference is calculated. An Adapter Execution Time event is created, which is used (if applicable) to set the minimum, maximum, and last adapter execution time for the hour of the day.

For a publishing adapter, the time is noted at the beginning and end of the publishing method, and the difference is calculated. An Adapter Execution Time event is created, which is used (if applicable) to set the minimum, maximum, and last adapter execution time for the hour of the day.

API Execution Time

For a subscriber adapter, the time is noted around the API call and the difference is calculated. An API Execution Time event is created, which is used (if applicable) to set the minimum, maximum, and last API execution time for the hour of the day.

For publishing adapter, there is no API execution time.

Adapter Status

A scheduled background job collects the Adapter status and updates the local repository. If the RIB application is down, since the job cannot run the status of the adapter in the central repository will be the last known status until the cache expires. After the cache expiry it will be "Unknown" until the status is reset by the rib-<app>.

Commits and Rollbacks

The commit and rollback count is the same information maintained by WebLogic server for the EJBs transactions. RIB monitoring system interrogates the JMX MBeans for the commit and rollback counts and updates the local repository. A message flow

may result in more than one commit and rollback, depending on various scenarios of failures.

Error Hospital Metrics

Error hospital data for the RIB application is queried by a scheduled background thread and the following information is collected:

- Total Messages in Error Hospital: Total number of messages in the Error Hospital for the application
- Total Messages in Error Hospital due to dependency: Total number of dependent messages in the Error Hospital
- Message Family: Message family of the family-wise statistics
- Adapter class Definition: Adapter information for the message family
- Error count: Number of error messages for the message family
- Dependency count: Number of the dependent messages for the message family

RIB Application Status

Status of the RIB application, e.g., RUNNING, STOPPED etc.

A

Sample Files

Sample Application.wadl File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ns0:application xmlns:ns0="http://wadl.dev.java.net/2009/02">
  <ns0:doc ns1:generatedBy="Jersey: 2.22.4 2016-11-30 13:33:53" xmlns:ns1="http://
jersey.java.net/" />
  <ns0:doc ns2:hint="This is simplified WADL with user and core resources only. To
get full WADL with extended resources use the query parameter detail. Link: http://
abc.us.oracle.com:8003/rib-injector-services-web/resources/application.wadl?
detail=true" xmlns:ns2="http://jersey.java.net/" />
  <ns0:grammars>
    <ns0:include href="application.wadl/xsd0.xsd">
      <ns0:doc title="Generated" xml:lang="en" />
    </ns0:include>
  </ns0:grammars>
  <ns0:resources base="http://abc.us.oracle.com:8003/rib-injector-services-web/
resources/">
    <ns0:resource path="discover">
      <ns0:method id="discoverAllResources" name="GET">
        <ns0:response>
          <ns0:representation mediaType="application/json" />
        </ns0:response>
      </ns0:method>
    </ns0:resource>
    <ns0:resource path="/injector">
      <ns0:resource path="/inject">
        <ns0:method id="injectMessage" name="POST">
          <ns0:request>
            <ns0:representation mediaType="application/xml"
element="ns3:ApplicationMessage" xmlns:ns3="http://www.oracle.com/retail/
integration/rib/ApplicationMessages/v1" />
          </ns0:request>
          <ns0:response>
            <ns0:representation mediaType="*/" />
          </ns0:response>
        </ns0:method>
      </ns0:resource>
    <ns0:resource path="/ping">
      <ns0:method id="ping" name="GET">
        <ns0:request>
          <ns0:param name="pingMessage" default="hello"
type="xsd:string" xmlns:xsd="http://www.w3.org/2001/XMLSchema" style="query" />
        </ns0:request>
        <ns0:response>
          <ns0:representation mediaType="application/json" />
        </ns0:response>
      </ns0:method>
    </ns0:resource>
  </ns0:resources>
</ns0:application>
```

Sample Resource Class

```
package com.oracle.retail.rib.integration.services.applicationmessageinjector;

import javax.ejb.EJB;
import javax.ejb.Stateless;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import com.oracle.retail.integration.rib.applicationmessages.v1.*;
import com.retek.rib.binding.exception.InjectorException;
import com.retek.rib.binding.injector.Injector;
import com.retek.rib.binding.injector.InjectorFactory;
import com.retek.rib.domain.payload.PayloadFactory;
import javax.ws.rs.DefaultValue;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.oracle.retail.integration.payload.Payload;

@Stateless
@Path("/injector")

public class ApplicationMessageInjectorResource {

    private static Log LOG =
        LogFactory.getLog(ApplicationMessageInjectorResource.class);

    @GET
    @Path("/ping")
    @Produces({MediaType.APPLICATION_JSON})
    public Response ping(@DefaultValue("hello") @QueryParam("pingMessage")
String pingMessage) {
        String message = "{\"message\": \"Got \" + pingMessage + \" from
server.\"}";
        return Response.ok(message, MediaType.APPLICATION_JSON).build();
    }

    @POST
    @Path("/inject")
    @Consumes({MediaType.APPLICATION_XML})
    public Response injectMessage(ApplicationMessage applicationMessage) throws
InjectorException{

        verifyNotNull(applicationMessage, "applicationMessage");

        invokeInjectForMessageType(applicationMessage.getFamily(),
applicationMessage.getType(), applicationMessage.getBusinessObjectId(),
applicationMessage.getPayloadXml());

        String message = "{\"message\": \"Inject successful.\"}";
        return Response.ok(message, MediaType.APPLICATION_JSON).build();
    }
}
```

```

private void invokeInjectForMessageType(String family, String messageType, String
businessObjectId, String retailPayload) throws InjectorException{

    try {

        verifyNotNull(family, "family");
        verifyNotNull(messageType, "messageType");
        verifyNotNull(retailPayload, "retailPayload");

        Payload payload = PayloadFactory.unmarshalPayload(family, messageType,
retailPayload);

        Injector injector = InjectorFactory.getInstance().getInjector(
??         family, messageType);
        if (injector == null) {
            final String eMsg = "Unknown message"
                + " family/type: " + family + "/" + messageType;
            LOG.error(eMsg);
            throw new InjectorException(eMsg);

        }
        if(LOG.isDebugEnabled()){
            LOG.debug("Received inject call for family("+family+")
type("+messageType+") businessObjectId("+businessObjectId+) with payload:\n" +
payload.toString());
        }

        injector.inject(messageType, businessObjectId, payload);
        LOG.debug("Inject call for family("+family+") type("+messageType+")
businessObjectId("+businessObjectId+) return.");

    ??    } catch (InjectorException e) {
        final String eMsg = "Exception calling inject.";
        LOG.error(eMsg, e);
        throw e;
    } catch (Exception re) {
        final String eMsg = "Exception calling inject.";
        LOG.error(eMsg, re);
        throw new RuntimeException(eMsg, re);
    }

}

private void verifyNotNull(Object field, String fieldName){
    if(field == null){
        final String eMsg = fieldName + " cannot be null.";
        LOG.error(eMsg);
        throw new IllegalArgumentException(eMsg);
    }
}

}

```

ApplicationMessages.xsd

```

<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.oracle.com/retail/integration/rib/ApplicationMessages/v1"
    xmlns:rib="http://www.oracle.com/retail/integration/rib/

```

```

ApplicationMessages/v1"
    xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
    xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
    jaxb:extensionBindingPrefixes="xjc"
    jaxb:version="2.0"
    targetNamespace="http://www.oracle.com/retail/integration/rib/
ApplicationMessages/v1"
    elementFormDefault="qualified" attributeFormDefault="unqualified">

    <xs:annotation>
        <xs:appinfo>

            <jaxb:globalBindings
                fixedAttributeAsConstantProperty="false"
                choiceContentProperty="true"
                enableFailFastCheck="true"
                generateIsSetMethod="true          "
                enableValidation="true">
                <!--xjc:javaType name="java.util.Calendar"
                    xmlType="xs:dateTime"

adapter="com.oracle.retail.integration.rib.rib_integration_runtime_info.datatypepa
dapter.CalendarAdapter"/ -->
                <jaxb:serializable uid="1"/>
            </jaxb:globalBindings>

            <!--jaxb:schemaBindings>
                <jaxb:package
name="com.oracle.retail.integration.rib.ribintegrationruntimeinfo" />
            </jaxb:schemaBindings-->
        </xs:appinfo>
    </xs:annotation>

    <xs:element name="ApplicationMessages">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="ApplicationMessage" maxOccurs="unbounded" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="ApplicationMessage">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="family" type="string25"/>
                <xs:element name="type" type="string30"/>
                <xs:element name="businessObjectId" type="string255"
minOccurs="0"/>
                <xs:element ref="ApplicationMessageRoutingInfo" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="payloadXml" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="ApplicationMessageRoutingInfo">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="string25"/>
                <xs:element name="value" type="string25"/>
                <xs:element ref="ApplicationMessageRoutingInfoDetail"

```

```

minOccurs="0" maxOccurs="2"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="ApplicationMessageRoutingInfoDetail">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="string25"/>
      <xs:element name="value" type="string300"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:simpleType name="string255">
  <xs:restriction base="xs:string">
    <xs:maxLength value="255" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="string25">
  <xs:restriction base="xs:string">
    <xs:maxLength value="25" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="string30">
  <xs:restriction base="xs:string">
    <xs:maxLength value="30" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="string300">
  <xs:restriction base="xs:string">
    <xs:maxLength value="300" />
  </xs:restriction>
</xs:simpleType>

</xs:schema>

```

Rest Publisher Pseudo Code

```

//Import required classes
import com.oracle.retail.integration.base.bo.fulfilorddesc.v1.*
import com.oracle.retail.integration.payload.Payload
import com.retek.rib.domain.payload.PayloadFactory
import com.oracle.retail.integration.rib.applicationmessages.v1.ApplicationMessage;
import com.oracle.retail.integration.rib.applicationmessages.v1.ApplicationMessages;
//Create new instance of your FulfilOrdDesc object and populate it.
FulfilOrdDesc fulfilOrdDesc = new FulfilOrdDesc()
fulfilOrdDesc.setCustomerOrderNo(123)
//Get a string version of the payload
String payloadXml = PayloadFactory.marshalPayload(fulfilOrdDesc)

//Prepare the header message section
ApplicationMessages ams = new ApplicationMessages();
ApplicationMessage am = new ApplicationMessage();
am.setFamily("FULFILORD");
am.setType("FULFILORDPOCRE");

```

```
am.setBusinessObjectId("abc"); //optional
//Set the payload xml into the message
am.setPayloadXml(payloadXml);
ams.getApplicationMessage().add(am);

//Call rest url with ams

String ribPublisherRestUrl = "http://<host>:<port>/rib-ext-services-web/
resources/publisher/publish"

Client client = ClientBuilder.newClient();
WebTarget webTarget = client.target(ribPublisherRestUrl);

String userName = "user";
char[] password = "passed";

String usernameAndPassword = userName + ":" + new String(password);
String authorizationHeaderValue = "Basic " +
java.util.Base64.getEncoder().encodeToString( usernameAndPassword.getBytes() );

Invocation.Builder invocationBuilder =
webTarget.request().header("Authorization", authorizationHeaderValue);

Response response = invocationBuilder.post(Entity.entity(ams,
MediaType.APPLICATION_XML));

log.debug("Publish call response(" + response + ").");
```

payload.properties

```
ASNIN.ASNINCRE=com.oracle.retail.integration.base.bo.asnindesc.v1.ASNInDesc
ASNIN.ASNINDEL=com.oracle.retail.integration.base.bo.asninref.v1.ASNInRef
ASNIN.ASNINMOD=com.oracle.retail.integration.base.bo.asnindesc.v1.ASNInDesc

WH.WHCRE=com.oracle.retail.integration.base.bo.whdesc.v1.WHDesc
WH.WHDEL=com.oracle.retail.integration.base.bo.whref.v1.WHRef
WH.WHMOD=com.oracle.retail.integration.base.bo.whdesc.v1.WHDesc
```

Sample Request/Response for ReST Injector Service

Table A-1 Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
http://localhost:7001/rib-injector-services-web/resources/injector/inject	POST	application/xml Request are xml only and response are json only.	A valid user that is part of Integration Group.	<pre><ApplicationMessage xmlns="http:// www.oracle.com/ retail/ integration/rib/ ApplicationMessages/ v1"> <family>Vendor</ family> <type>VendorCre</ type> <businessObjectId>10 111011</ businessObjectId> <payloadXml>&lt;Vend orDesc xmlns="http:// www.oracle.com/ retail/integration/ base/bo/ VendorDesc/v1" xmlns:xsi="http:// www.w3.org/2001/ XMLSchema- instance"&gt;&lt;ns1 :VendorHdrDesc xmlns:ns1="http:// www.oracle.com/ retail/integration/ base/bo/ VendorHdrDesc/v1" &gt;&lt;ns1:supplier &gt;10111011&lt;/ ns1:supplier&gt;&lt; ns1:sup_name&gt;supp lier site</pre>	<pre>HTTP/1.1 200 OK Date: Thu, 10 May 2018 16:33:11 GMT Content-Length: 33 Content-Type: application/json X-ORACLE-DMS-ECID: 4a8e5d3f-1aae-43d7- ba84- c6b9c60563c7-0000003 9 X-ORACLE-DMS-RID: 0 Set-Cookie: JSES- SIONID=hsFK5jW4B1Qti pC9zhng-- or1WL7ywxCuXsJeVwdgP pnv6oNUnde! 233126712; path=/; HttpOnly {"message": "In- ject successful."}</pre>	Success

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
				<pre> </> ns1:sup_name<>< ns1:contact_name<> G Srilekha</> ns1:contact_name<> <ns1:contact_phon e<>1234567</> ns1:contact_phone< ><ns1:sup_status< >A</> ns1:sup_status<>< t;ns1:qc_ind<>N< t; ns1:qc_ind<><ns 1:vc_ind<>N< ns1:vc_ind<><ns 1:currency_code<>P LN< ns1:currency_code< ><ns1:terms<>Ne t_07</> ns1:terms<><ns1 :freight_terms<>F_ 01</> ns1:freight_terms< ><ns1:ret_allow_i nd<>N< ns1:ret_allow_ind< ><ns1:ret_auth_re q<>Y</> ns1:ret_auth_req< ><ns1:edi_po_ind< t;N< ns1:edi_po_ind<><l t;ns1:edi_po_chg< t;N< ns1:edi_po_chg<><l t;ns1:edi_po_confirm <>N< ns1:edi_po_confirm< ><ns1:edi_ </pre>		

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
				<pre> asn>>N<</ nsl:edi_asn>>&lt;n s1:edi_supp_availabl e_ind>>N<</ nsl:edi_supp_availab le_ind>>&lt;nsl:ed i_contract_ind>>N<< lt;/ nsl:edi_contract_ind >>&lt;nsl:edi_invc _ind>>N<</ nsl:edi_invc_ind>> &lt;nsl:cost_chg_pct _var>>0<</ nsl:cost_chg_pct_var >>&lt;nsl:cost_chg _amt_var>>0<</ nsl:cost_chg_amt_var >>&lt;nsl:replen_a pproval_ind>>N<< / nsl:replen_approval_ ind>>&lt;nsl:settl ement_code>>E<</ nsl:settlement_code& gt;&lt;nsl:pre_mark_ ind>>N<</ nsl:pre_mark_ind>> &lt;nsl:auto_appr_in vc_ind>>N<</ nsl:auto_appr_invc_i nd>>&lt;nsl:dbt_me mo_code>>Y<</ nsl:dbt_memo_code&gt ;&lt;nsl:freight_cha rge_ind>>N<</ nsl:freight_charge_i nd>>&lt;nsl:auto_a ppr_dbt_memo_ind>> N<</ nsl:auto_appr_dbt_me mo_ind>>&lt;nsl:in v_mgmt </pre>		

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
				<pre> _lvl>S</ ns1:inv_mgmt_lvl> <ns1:backorder_in d>N</ ns1:backorder_ind> <ns1:vat_region> 1002</ ns1:vat_region><l t;ns1:prepay_invc_in d>N</ ns1:prepay_invc_ind> <ns1:service_p erf_req_ind>N</ / ns1:service_perf_req _ind><ns1:addi nvc_gross_net >N</ ns1:addinvc_gross_ne t><ns1:deliver y_policy>NEXT</ / ns1:delivery_policy> <ns1:bracket_c osting_ind>N</ ns1:bracket_costing_ ind><ns1:dsd_s upplier_ind>N</ / ns1:dsd_supplier_ind ><ns1:sup_qty_ level>CA</ ns1:sup_qty_level> <ns1:supplier_pa rent>1011101</ ns1:supplier_parent> <ns1:final_des t_ind>N</ ns1:final_dest_ind> </ ns1:VenVendorHdrDesc ><ns1:VendorAd drDesc </pre>		

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
				<pre> xmlns:ns1="http:// www.oracle.com/ retail/integration/ base/bo/ VendorAddrDesc/v1" ><ns1:module> t;SUPP</ns1:module> <ns1:key_value_1>101 11011</ns1:key_value_1> <ns1:seq_no>1</ns1:seq_no> <ns1:addr_type>01</ns1:addr_type> <ns1:primary_addr_ind>Y</ns1:primary_addr_ind> <ns1:add_1>3/4/678</ns1:add_1> <ns1:city>AE</ns1:city> <ns1:country_id>PL</ns1:country_id> <ns1:country_id> <ns1:VendorAddrDesc> t;</ns1:VendorAddrDesc> xmlns:ns1="http:// www.oracle.com/ retail/integration/ base/bo/ VendorAddrDesc/v1" ><ns1:module> t;SUPP</ns1:module> <ns1:key_value_1>101 11011</ns1:key_value_1> <ns1:seq_no>1</ns1:seq_no> <ns1:addr_type>03</ns1:addr_type> <ns1:primary_addr_ind>Y</ns1:primary_addr_ind> </pre>		

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
				<pre> <ns1:add_1> <3/4/678>/ <ns1:add_1><ns1:city>AE</ns1:city><ns1:country_id>PL</ns1:country_id><ns1:VendorAddrDesc><ns1:VendorAddrDesc </pre>		
				<pre> xmlns:ns1="http://www.oracle.com/retail/integration/base/bo/VendorAddrDesc/v1" <ns1:module>SUPP</ns1:module><ns1:key_value_1>10111011</ns1:key_value_1><ns1:seq_no>1</ns1:seq_no><ns1:addr_type>04</ns1:addr_type><ns1:primary_addr_ind>Y</ns1:primary_addr_ind><ns1:add_1> <3/4/678>/ <ns1:add_1><ns1:city>AE</ns1:city><ns1:country_id>PL</ns1:country_id><ns1:VendorAddrDesc><ns1:VendorAddrDesc </pre>		

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
				<pre> xmlns:ns1="http:// www.oracle.com/ retail/integration/ base/bo/ VendorAddrDesc/v1" ><ns1:module> t;SUPP</ns1:module> </ns1: key_value_1>10111 011</key_value_1> <ns1:seq_no>1</ns1:seq_no> <ns1:addr_type>05</ns1:addr_type> <ns1:primary_addr_ind>Y</ns1:primary_addr_ind> <ns1:add_1>3/4/678</ns1:add_1> <ns1:city>AE</ns1:city> <ns1:country_id>PL</ns1:country_id> <ns1:VendorAddrDesc> t;ns1:VendorAddr Desc </pre>		

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
				<pre> xmlns:ns1="http:// www.oracle.com/ retail/integration/ base/bo/ VendorAddrDesc/v1" ><ns1:module> <SUPP</ns1:module> <ns1:key_value_1>10111011</ns1:key_value_1> <ns1:seq_no>1</ns1:seq_no> <ns1:addr_type>06</ns1:addr_type> <ns1:primary_addr_ind>Y</ns1:primary_addr_ind> <ns1:add_1>3/4/678</ns1:add_1> <ns1:city>AE</ns1:city> <ns1:country_id>PL</ns1:country_id> <ns1:VendorAddrDesc> <ns1:VendorOUDe sc </pre>		
				<pre> xmlns:ns1="http:// www.oracle.com/ retail/integration/ base/bo/ VendorOUDesc/v1" ><ns1:org_unit_id>1</ns1:org_unit_id> <ns1:primary_pay_site_ind>N</ns1:primary_pay_site_ind> <ns1:VendorOUDesc> </ns1:VendorOUDesc> VendorDesc</ payloadXml> </ ApplicationMessage> </pre>		

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
				<pre> <stockholding_ind>a< /stockholding_ind> <item_id>nbYDUFLqAcT sBUnhYuhpcæ±</ item_id> <origi- nal_item_id>UxrgzyAg zDgTDbHfMBjbtæ±</ original_item_id> <order_line_nbr>3</ order_line_nbr> <unit_qty>12.4</ unit_qty> <status>a</ status> <us- er_id>CAswTBGUzTaNjw gDwWXEgqCjEmæ±</ user_id> <updat- ed_date>2013-06-13T1 4:20:35</ updated_date> </SOStatusDt1> <context_type>vRæ±</ context_type> <con- text_value>oDHGRuOeD mvFPytxgiiJyæ±</ context_value> <inventory_type>kwæ± </inventory_type> <cust_order_nbr>cwFL uXBqFPBvkxVmTSBrhovr ROJAZYCfYncVEhfub- mAYæ±</ cust_order_nbr> </pre>		

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
				<pre> <fulfill_order_nbr>q SzQUPkqbEFboWQFxFPSqo Z- NOEJotCMnqbWzXTqRVkV kLæ±</ fulfill_order_nbr> </SOStatusDesc>]]></ payloadXml> </ ApplicationMessage> </pre>		

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
			If user in not added in Integration Group	<pre><v1:ApplicationMessage xmlns:v1="http://www.oracle.com/retail/integration/rib/ApplicationMessages/v1"> <v1:family>WH</v1:family> <v1:type>WHCR</v1:type> <!--Optional:--> <v1:businessObjectId>?</v1:businessObjectId> <!--Zero or more repetitions:--> <v1:ApplicationMessageRoutingInfo> <v1:name>?</v1:name> <v1:value>?</v1:value> <!--Zero or more repetitions:--> <v1:ApplicationMessageRoutingInfoDetail> <v1:name>?</v1:name> <v1:value>?</v1:value> </v1:ApplicationMessageRoutingInfoDetail> </v1:ApplicationMessageRoutingInfoDetail> </v1:ApplicationMessageRoutingInfo> <v1:payloadXml>&lt;WHDesc xmlns="http://www.oracle.com/retail/integration/base/bo/WHDesc/v1" > <v1:wh_name>?</v1:wh_name> <v1:wh_name_desc>?</v1:wh_name_desc> </v1:payloadXml> </v1:ApplicationMessage></pre>	<pre>HTTP/1.1 403 Forbidden Date: Thu, 05 Aug 2021 10:25:26 GMT Content-Length: 1166 Content-Type: text/html; char-set=UTF-8 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Draft//EN"> <HTML> <HEAD> <TITLE>Error 403--Forbidden</TITLE> </HEAD> <BODY bgcolor="white"> <BR CLEAR=all> <TABLE border=0 cellpadding=5><TR><TD><BR CLEAR=all> <H2>Error 403--Forbidden</H2> </TD></TR> </TABLE> <TABLE border=0 width=100% cellpadding=10><TR><TD VALIGN=top WIDTH=100% BGCOLOR=white><H3>From RFC 2068 <i>Hypertext Transfer Protocol -- HTTP/1.1</i> </H3> <H4>10.4.4 403 For-bidden</H4> </pre>	Failure

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
					<pre> <P>The server understood the request, but is refusing to fulfill it. Authorization will not help and the request SHOULD NOT be repeated. If the request method was not HEAD and the server wishes to make public why the request has not been ful-filled, it SHOULD de-cribe the reason for the refusal in the entity. This status code is commonly used when the server does not wish to reveal exactly why the request has been refused, or when no other response is ap-plica-ble.</ FONT></P> </TD></TR> </TABLE> </BODY> </HTML> </pre>	

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
			Wrong User/pass	<pre><v1:ApplicationMessage xmlns:v1="http://www.oracle.com/retail/integration/rib/ApplicationMessages/v1"> <v1:family>WH</v1:family> <v1:type>WHCR</v1:type> <!--Optional:--> <v1:businessObjectId>?</v1:businessObjectId> <!--Zero or more repetitions:--> <v1:ApplicationMessageRoutingInfo> <v1:name>?</v1:name> <v1:value>?</v1:value> <!--Zero or more repetitions:--> <v1:ApplicationMessageRoutingInfoDetail> <v1:name>?</v1:name> <v1:value>?</v1:value> </v1:ApplicationMessageRoutingInfoDetail> </v1:ApplicationMessageRoutingInfo> <v1:payloadXml>&lt;WHDesc xmlns="http://www.oracle.com/retail/integration/base/bo/WHDesc/v1" >&lt;wh&gt;t;10&lt;/wh&gt;&lt;wh_name&gt;g&lt;/wh_name&gt;&lt;/WHDesc&gt;</v1:payloadXml> </v1:ApplicationMessage></pre>	<pre>HTTP/1.1 401 WWW-Authenticate: Basic realm="Authentication required" Content-Type: text/html; charset=utf-8 Content-Language: en Content-Length: 669 Date: Thu, 05 Aug 2021 05:08:40 GMT Keep-Alive: timeout=20 Connection: keep-alive <!doctype html><html lang="en"><head><title>HTTP Status 401 â€" Unauthorized</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1,h2,h3,b {color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:#525D76;border:none;}</style></head><body><h1>HTTP Status 401 â€" Unauthorized</h1><hr class="line" /><p>Type</p><p>Description</p><p>The request has not been applied because it lacks valid</pre>	Failure

Table A-1 (Cont.) Sample Request/Response for ReST Injector Service

End Point	Method	Media Type	User/ Password	Request xml	Response	Comments
					<pre> authentication credentials for the target resource.</ p><hr class="line" / ><h3>Apache Tomcat/ 8.5.64</h3></body></ html> </pre>	