

Oracle® Retail Bulk Data Integration

Implementation Guide

Release 16.0

E81413-01

December 2016

Oracle Retail Integration Bus Implementation Guide, Release 16.0

E81413-01

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Primary Author: Sanal Parameswaran

Contributing Author: Rakhee Prabhudesai

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

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

If this is software or related 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 installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. 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 and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron 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.

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

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

- (i) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.
- (ii) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Mobile Store Inventory Management.
- (iii) the software component known as **Access Via**[™] licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.
- (iv) the software component known as **Adobe Flex**[™] licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.

You acknowledge and confirm that Oracle grants you use of only the object code of the VAR Applications. Oracle will not deliver source code to the VAR Applications to you. Notwithstanding any other term or condition of the agreement and this ordering document, you shall not cause or permit alteration of any VAR

Applications. For purposes of this section, "alteration" refers to all alterations, translations, upgrades, enhancements, customizations or modifications of all or any portion of the VAR Applications including all reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

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

Contents

Send Us Your Comments	ix
Preface	xi
Audience	xi
Documentation Accessibility	xi
Related Documents	xi
Customer Support	xi
Review Patch Documentation	xii
Improved Process for Oracle Retail Documentation Corrections	xii
Oracle Retail Documentation on the Oracle Technology Network	xii
Conventions	xii
1 Introduction	
Oracle Retail Enterprise Integration Products and Styles	1-1
Standards and Specifications	1-2
Java Platform Enterprise Edition (Java EE)	1-2
Java Batch – JSR 352	1-2
Java EE Server	1-3
Java Batch Overview	1-3
2 Job Administrator	
Job Admin Core Components	2-1
Extractor Job	2-1
Downloader-Transporter job	2-2
Receiver Service	2-8
Uploader Job	2-8
Uploader Job Configuration	2-11
Importer Job	2-12
3 Job Admin Services	
Job Admin RESTful Services	3-1
Receiver Service	3-1
Batch Service	3-7
Data Service	3-10

Configuration of Job Admin.....	3-12
---------------------------------	------

4 Job Admin UI

Job Admin UI Security	4-1
Authentication	4-1
Authorization.....	4-1
Monitoring Batch Jobs Using BDI Job Admin	4-2
Batch Summary Tab.....	4-2
Manage Jobs Tab	4-2
Job Executions.....	4-3
Job Launch.....	4-3
Job Details.....	4-4
System Logs Tab.....	4-4
Diagonostics Tab	4-6
Outbound Job Execution Errors	4-6
Inbound Job Execution Errors	4-7
Trace Data.....	4-7
Manage Configurations	4-10
Outbound Interface Controls	4-11
Inbound Interface Controls.....	4-11
System Options.....	4-12
Job Admin Troubleshooting	4-12
BDI apps deployment Error.....	4-12
BDI Job Admin runtime WSMException.....	4-13
REST Service from SOAP UI for Downloader and Transporter job	4-13
BDI Job Admin not able to find UploaderJob.xml file.....	4-13
Job Fails and Job Admin Log Files Contain No Details of the Failure	4-14

5 Process Flow

Process Flow	5-1
DSL (Domain Specific Language).....	5-2
Process Flow Instrumentation.....	5-3
Process Flow Monitor Web Application.....	5-3
Process Restart	5-8
Process Security	5-8
Customizing Process Flows	5-9
Process Flow DSL.....	5-9
APIs	5-9
How to modify a Process Flow	5-10
Sub Processes	5-10
Process Schema.....	5-10
REST Interface.....	5-11
Troubleshooting	5-11

6 BDI Scheduler

Scheduler Core Concepts	6-1
--------------------------------------	-----

Schedule Types.....	6-1
Scheduling Mechanisms.....	6-2
Schedule Frequency.....	6-3
Schedule Action.....	6-4
Schedule Action Type.....	6-5
Schedule Status.....	6-6
Scheduler Runtime	6-6
Scheduler Startup	6-6
Schedule Runtime Execution.....	6-7
Schedule Execution - BDI Process Flows.....	6-7
Schedule Execution - Async Action.....	6-8
Schedule Execution - Sync Action	6-9
Schedule Execution Failover.....	6-10
Schedule Notification	6-10
Scheduler Infrastructure Schema	6-10
Scheduler REST Services	6-11
Scheduler Console	6-11
..... Schedule Summary	6-12
Manage Schedules.....	6-13
Scheduler Security Considerations.....	6-20
Scheduler Operational Considerations	6-21
Scheduler Customization.....	6-23
Customizing Schedule Actions	6-24
Scheduler Troubleshooting.....	6-25
Scheduler Known issues	6-26
7 CLI Tools	
BDI CLI Job Executor	7-1
Tool Setup.....	7-1
Tool Usage.....	7-2
BDI CLI Transmitter	7-2
Tool Setup.....	7-2
Tool Usage.....	7-4
File Processing	7-6
Output Logs	7-6
Error Reprocessing.....	7-7
8 BDI Data Integration Topologies	
Point to Point Topology.....	8-1
Sender side split	8-2
Receiver Side Split.....	8-3
9 Pre-implementation Considerations	
BDI Software Lifecycle Management	9-1
Preparation Phase	9-1
Application Assembly Phase.....	9-1

Deployment Phase	9-1
Operation Phase	9-1
Maintenance Phase.....	9-1
Physical Location Considerations	9-2
High Availability Considerations.....	9-2
WebLogic Server Cluster Concepts	9-2
bdi-<app> application and WebLogic Application Server Cluster	9-3

10 Deployment Architecture and Options

Recommended Deployment Options.....	10-1
Distributed	10-1
Centralized	10-2

11 Implementation Process

12 Performance Considerations

Performance Tuning Downloader-Transporter Jobs.....	12-1
Performance Tuning Uploader Jobs.....	12-2

A Process Schema

B Process Flow REST Endpoints

C System Setting Service

D Sample Extractor - PL/SQL application code that calls procedures in PL/SQL package

E Glossary

Send Us Your Comments

Oracle Retail Bulk Data Integration Implementation Guide, Release 16.0

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

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

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

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

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

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

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

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

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

Preface

The Oracle Retail Bulk Data Integration Implementation Guide provides detailed information that is important when implementing BDI.

Audience

The Implementation Guide is intended for the Oracle Retail Bulk Data Integration application integrators and implementation staff, as well as the retailer's IT personnel.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Related Documents

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

- *Oracle Retail Bulk Data Integration Installation Guide*
- *Oracle Retail Bulk Data Integration Release Notes*

Customer Support

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

<https://support.oracle.com>

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)

- Detailed step-by-step instructions to re-create
- Exact error message received
- Screen shots of each step you take

Review Patch Documentation

When you install the application for the first time, you install either a base release (for example, 16.0) or a later patch release (for example, 16.0.1). If you are installing the base release and additional patch releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch releases can contain critical information related to the base release, as well as information about code changes since the base release.

Improved Process for Oracle Retail Documentation Corrections

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

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

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

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

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

Oracle Retail Documentation on the Oracle Technology Network

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

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

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

Conventions

The following text conventions are used in this document:

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

Introduction

Bulk Data Integration (BDI) is the Oracle Retail Enterprise Integration Infrastructure product designed to address the complexities of the movement of bulk data between Oracle Retail applications and third-party applications.

BDI is designed to provide the bulk data integration to meet the modern needs of cloud and on-premise movement of large data sets in the deployments of the Oracle Retail applications and fully support both on-premise configurations and on-cloud configurations in a hybrid cloud-premise deployment.

Oracle Retail Enterprise Integration Products and Styles

There is no one integration approach that addresses all criteria equally well. Therefore, multiple approaches for integrating applications have evolved over time. Oracle Retail has focused on three main integration styles:

- Asynchronous JMS Pub/Sub Fire-and-Forget (Retail Integration Bus - RIB)
- Request/Response (Retail Service Backbone - RSB)
- Bulk Data Integration - BDI

Batch (Bulk) data is still a predominant integration style within Oracle Retail and its Customers.

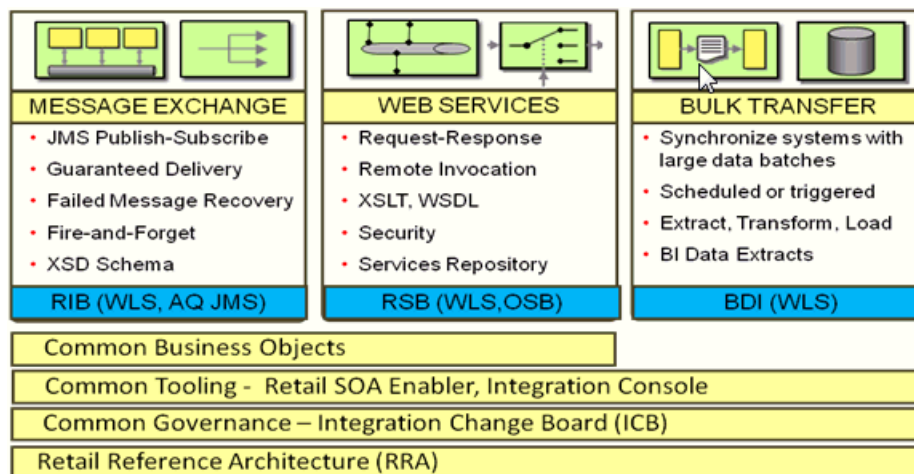
The movement of bulk data remains important because some work is best suited to being performed in bulk. Batch processing was there in the early days; it's still here today; and it will still be here tomorrow. What has changed is the approach to batch processing.

Batch processing is typified by bulk-oriented, non-interactive, background execution. Frequently long running, it may be data or computationally intensive, executed sequentially or in parallel, and may be initiated through various invocation models, including ad hoc, scheduled, and on-demand.

Batch applications have common requirements including logging, checkpoint, and parallelization. Batch workloads have common requirements such as operational control, which allow for initiation of, and interaction with, batch instances; such interactions include stop and restart.

BDI is the latest Oracle Retail Integration product to be released to productize Oracle Retail bulk data flows for delivery to customers to meet these requirements, and provide the tooling that is required to automate the creation and packaging of the configurations and to manage the full life cycle.

Oracle Retail now has integration products designed and built to satisfy all three of the integration styles used by our customers today.



Standards and Specifications

BDI, such as RIB and RSB, relies on industry standards and specifications and leverages the features of the WebLogic Application Server.

In 2011, a working group was formed to study and design an open standard for Java batch processing. Representatives from many companies, including Oracle, developed a draft standard. The initial release of the standard was released in 2013. The standard, known as 352, is now included as part of the Java EE 7 open standard.

BDI is designed and built on these Java EE 7 and Java Batch (JSR 352) specifications and standards.

Java Platform Enterprise Edition (Java EE)

Java Platform Enterprise Edition (Java EE) is an umbrella standard for Java's enterprise computing facilities. It bundles together technologies for a complete enterprise-class server-side development and deployment platform in java.

Java EE specification includes several other API specifications, such as JDBC, RMI, Transaction, JMS, Web Services, XML, Persistence, mail, and others and defines how to coordinate among them. Java EE also features some specifications unique to enterprise computing. These include Enterprise JavaBeans (EJB), servlets, portlets, Java Server Pages (JSP), Java Server Faces (JSF) and several Web service technologies.

A Java EE application server manages transactions, security, scalability, concurrency, pooling, and management of the EJB/Web components that are deployed to it. This frees the developers to concentrate more on the business logic/problem of the components rather than spending time building scalable, robust infrastructure on which to run on.

Java Batch – JSR 352

JSR 352 is a Java specification for building, deploying, and running batch applications. Batch is an industry metaphor for background bulk processing. A myriad business processes depend on batch processing and demand powerful standards-based facilities for enabling this essential workload type.

JSR 352 specifies the layers, components and technical services commonly found in robust, maintainable systems used to address the creation of simple to complex batch applications.

JSR 352 addresses three critical concerns: a batch programming model, a job specification language, and a batch runtime. This constitutes a separation of concerns.

- Application developers have clear, reusable interfaces for constructing batch style applications
- Job writers have a powerful expression language for how to execute the steps of a batch execution
- Solution integrators have a runtime API for initiating and controlling batch execution

JSR 352 defines a Job Specification Language (JSL) to define batch jobs, a set of interfaces that describes the artifacts that comprise the batch programming model to implement batch business logic, and a batch runtime for running batch jobs, according to a defined life cycle.

The batch runtime is a part of the Java EE 7 runtime and has full access to all other features of the platform, including transaction management, persistence, messaging, and more.

Java EE Server

The Oracle WebLogic Server implements the Java EE specification and is the Java EE server vendor for BDI. The WebLogic server provides many additional services beyond the standard services required by the Java EE specification.

Note: Review the WebLogic Application Server documentation for more information:

<http://docs.oracle.com/middleware/12211/wls/index.html>

[HTTP://WWW.ORACLE.COM/TECHNETWORK/MIDDLEWARE/FUSION-MIDDLEWARE/DOCUMENTATION/INDEX.HTML](http://WWW.ORACLE.COM/TECHNETWORK/MIDDLEWARE/FUSION-MIDDLEWARE/DOCUMENTATION/INDEX.HTML)

Java Batch Overview

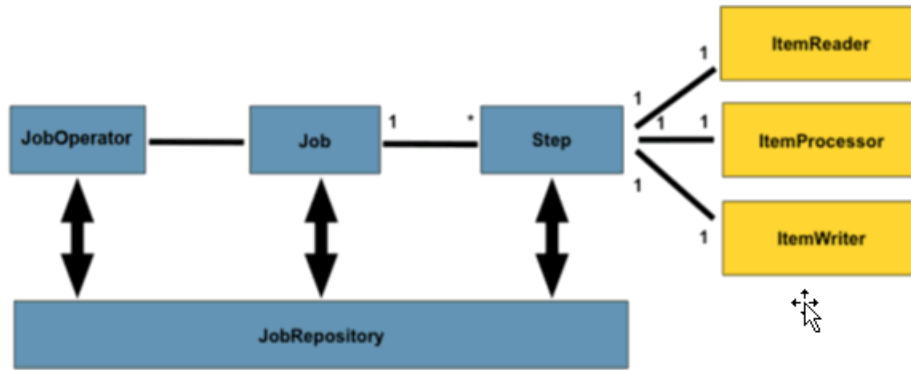
Batch processing for Java platform was introduced in Java EE 7. It provides a programming model for batch applications and a runtime to run and manage batch jobs.

Batch processing is the execution of a series of programs ("jobs") on a computer without manual intervention.

JSR 352 defines:

- **Implementation:** A programming model for implementing the artifacts
- **Orchestration:** A Job Specification Language, which orchestrates the execution of a batch artifact within a job
- **Execution:** A runtime environment for executing batch application, according to a defined lifecycle

The diagram below is a simplified version of the batch reference architecture that has been used for decades. It provides an overview of the components that make up the domain language of batch processing.



- The Job Operator provides an interface to manage all aspects of job processing, including operational commands, such as start, restart, and stop, as well as job repository related commands, such as retrieval of job and step executions.
- The Job Repository holds information about jobs currently running and jobs that ran in the past.
- A step contains all the necessary logic and data to perform the actual processing.
- A chunk-style step contains ItemReader, ItemProcessor, and ItemWriter.

A job encapsulates the batch process. A job contains one or more steps. A job is put together using Job Specification language (JSL) that specifies the sequence in which steps must be executed.

A step is a domain object that encapsulates an independent, sequential phase of a batch job. Therefore, every Job is composed entirely of one or more steps. A step contains all of the information necessary to define and control the actual batch processing.

ItemReader is an abstraction that represents the retrieval of input for a step, one item at a time. When the ItemReader has exhausted the items it can provide, it will indicate this by returning null.

ItemWriter is an abstraction that represents the output of a step, one batch or chunk of items at a time. Generally, an item writer has no knowledge of the input it will receive next, only the item that was passed in its current invocation.

The remainder of this document describes the implementation of the BDI product using Java Batch and JavaEE.

Job Administrator

BDI Job Admin is a web application that provides the runtime and GUI for managing batch jobs. It provides the following high level functionality.

- RESTful service to start/restart, check status and so on of a job.
- RESTful service to stream data from source system to destination system.
- The Infrastructure for various bulk data integration jobs. This includes the database for keeping track of data and the batch database for holding information about jobs.
- The User Interface provides ability to:
 - Start/restart, and track status of jobs
 - Trace data
 - View Diagnostic Errors
 - Manage options at job and system level
 - View the logs

BDI uses instances of Job Admin to run the downloader and uploader jobs. For example; RMS uses an instance of Job Admin to run extractor jobs whereas RXM uses an instance of Job Admin to run importer jobs.

Job Admin Core Components

The BDI Job Admin contains the batch jobs for moving bulk data from source (senders) systems (for example RMS) to destination (receiver) systems (for example SIM, RXM and so on). A bulk integration flow moves data for one family from source to destination application(s).

An Integration Flow is made up of the multiple activities: Extractor, Downloader, Transporter, Uploader, and Importer. These activities are implemented as batch jobs.

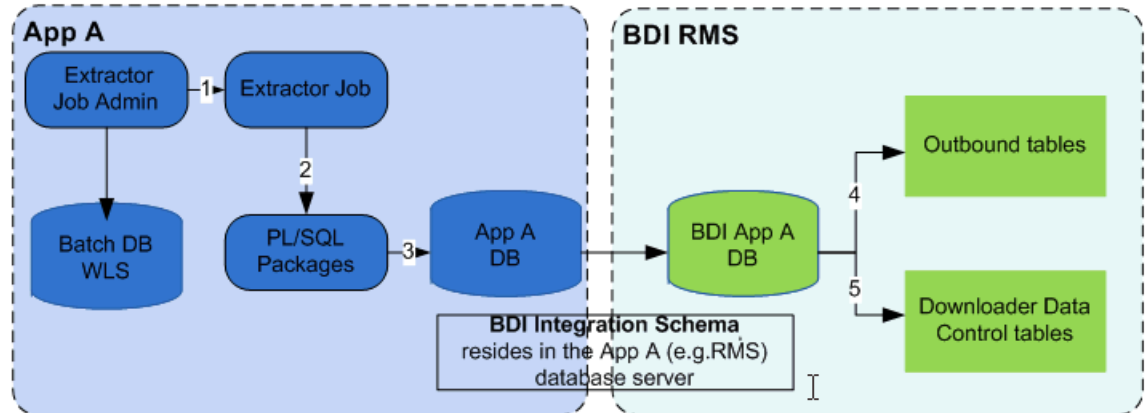
An Extractor Job extracts data for a Family from a source system and moves data to the outbound Interface Tables.

Outbound Interface Tables typically exist in the integration database schema and the schema resides in the source system database.

Extractor Job

The Extractor Job uses a Batchlet and PL/SQL stored procedures to move data from transactional tables of source system (for example RMS) to outbound tables. A PL/SQL stored procedure calls BDI PL/SQL stored procedure to insert data set

information in the outbound data control tables. Extractor jobs are currently implemented to provide full data (not delta) for an interface.



BDI Extractor (PL/SQL Application)

1. The Extractor job is run from App A (for example RMS) Extractor Job Admin application through REST or UI.
2. The Extractor job invokes PL/SQL stored procedure in App A database.
3. A PL/SQL stored procedure is run in the App A database.
4. The PL/SQL stored procedure moves data from transactional tables to the outbound tables in the BDI schema.
5. The PL/SQL stored procedure inserts entries in downloader data control tables to indicate the data set is ready for download.

Note: Review [Appendix D](#).

Sample Extractor – PL/SQL application code that calls procedures in PL/SQL package.

The Downloader Data Control Tables act as a handshake between the Extractor and the Downloader. There are two Outbound Data Control Tables:

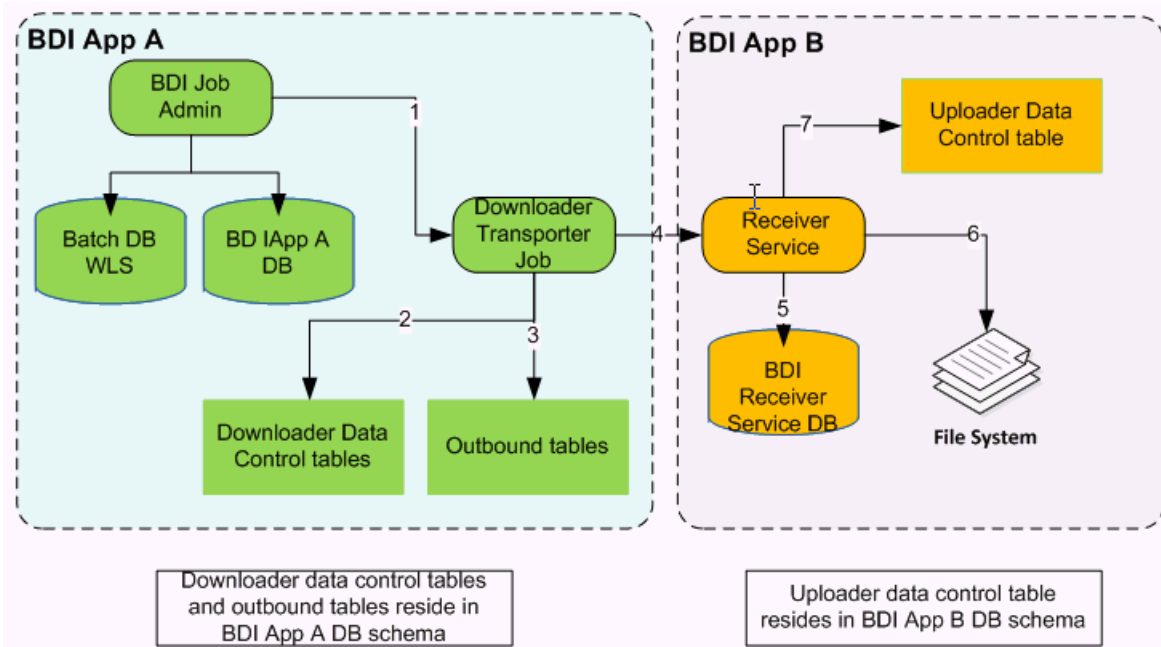
- BDI_DWNLDR_IFACE_MOD_DATA_CTL
- BDI_DWNLDR_IFACE_DATA_CTL

The Extractor job inserts entries in the downloader data control tables to indicate that data is ready to be downloaded after it completes moving data to outbound interface tables.

Downloader-Transporter job

A Downloader-Transporter job downloads the data set from outbound interface tables for an Interface Module (family) and streams data to a BDI destination application using the Receiver Service.

If there are multiple Interfaces for an Interface Module, data for all interfaces for that interface module are downloaded and streamed concurrently to the Receiver Service of BDI destination application.



BDI Downloader Transporter

1. The Downloader Transporter job is run from the BDI App A Job Admin application through REST or UI.
2. The Downloader Transporter job checks for new data sets in Downloader Data Control Tables.
3. If a Data Set is available for download, the Downloader Transporter job downloads a block of data from the outbound table.
4. The Downloader Transporter job streams downloaded blocks to Receiver Service.
5. The Receiver Service stores meta data in Receiver Service database.
6. The Receiver Service creates a file for every block that it receives. Steps 3, 4, 5, and 6 repeat until there is no more data to download.
7. The Receiver Service inserts an entry in the uploader data control table indicating that the data set is ready for upload.

Rules for processing a data set by Downloader Job

1. A full data set is available for download, if it is not processed by a downloader job yet and if a newer full data set is not processed successfully.
2. If data set id is passed through job parameters (for example `jobParameters=dataSetId=1`) to downloader job, it will use the data set if it is available as per the above rule. Otherwise job will fail.
3. If the data set id is not passed through job parameters to downloader job, it will identify the next available data set if there is one. Otherwise job completes without processing any data set.
4. If the downloader-transporter job fails for whatever reason, the data set that it tried to download can only be downloaded by restarting the job after fixing the issues.
5. If the downloader-transporter job is started instead of a restart, it will either pick up a new data set or none.

Downloader Data Sets

A Data Set consists of rows between a begin and end sequence number (bdi_seq_id column) in the Outbound Interface Table. The BDI_SEQ_ID column is automatically incremented when data is inserted into the outbound table.

The Downloader-Transporter job downloads a single data set that is not downloaded yet from the outbound interface tables.

If a data set id is passed as job parameter (for example jobParameters=dataSetId=1) to Downloader-Transporter job, it will use that data set if it is available for download. Job Parameters as a query parameter. Job Parameters is a comma separated list of name value pairs. This parameter is optional.

If there are multiple data sets in the outbound tables that are available for download, then the Downloader-Transporter job picks up the oldest data set.

If there is no data set available in the outbound tables, the Downloader-Transporter job completes without downloading any data.

If a newer data set is processed by the Downloader-Transporter job, then older data set cannot be processed.

A data set is divided into Logical Partitions and data in each partition is downloaded by a separate thread. The Downloader-Transporter job tries to allocate data equally between partitions. Data in each partition is divided into blocks based on the "item-count" value in the job and each block is retrieved from an outbound table and streams it to the destination application using the Receiver Service.

A data set is divided into logical partitions based on the number of partitions specified in the BDI_DWNLDR_TRANSMITTR_OPTIONS table and the number of rows in the data set.

The number of rows is calculated by subtracting the begin sequence number from the end sequence number provided in the BDI_DWNLDR_IFACE_DATA_CTL table. The number of rows may be approximate as there can be gaps in sequence numbers.

The number of rows allocated to each logical partition is calculated by dividing the approximate row count with the number of partitions.

Example 1:

```
Begin Sequence number = 1
End Sequence number = 100
Number of partitions = 2
```

```
Approximate row count = 100 - 1 + 1
Items for partition = 100/2 = 50
Data assigned to partition 1
  Begin Sequence number = 1
  End Sequence number = 1 + 50 - 1 = 50
Data assigned to partition 2
  Begin Sequence number = 51
  End Sequence number = 51 + 50 - 1 = 100
```

Example 2:

```
Begin Sequence number = 1
End Sequence number = 75
Number of partitions = 2
```

```
Approximate row count = 75 - 1 + 1
Items for partition = 75/2 = 37
Extra items = 75 % 2 = 1
Data assigned to partition 1
```

```

Begin Sequence number = 1
End Sequence number = 1 + 37 - 1 = 37
Data assigned to partition 2
Begin Sequence number = 38
End Sequence number = 38 + 37 + 1 - 1 = 75

```

The Downloader-Transporter job deletes data from outbound tables after the successful completion of the job if AUTO_PURGE_DATA flag in BDI_DWNLDR_TRNSMITTR_OPTIONS table is set to TRUE. The default value for this flag is TRUE. If sender side split topology is used, this flag needs to be changed to FALSE. Otherwise all destination applications may not get the data.

When a Downloader-Transporter job fails, the error information such as stack trace gets stored in BDI_JOB_ERROR and BDI_DOWNLOADER_JOB_ERROR tables. Errors are displayed in the “Diagnostics” tab of the Job Admin GUI. The error information can be used to fix the issues before restarting the failed job. Note that if there are exceptions in Batch runtime, then those exceptions won't show up in the Job Error tables and so in the Diagnostics tab of the Job Admin GUI.

Downloader-Transporter Job Configuration

Seed data for the Downloader-Transporter jobs is loaded to the database during the deployment of Job Admin. Some of the seed data can be changed from the Job Admin GUI.

BDI_SYSTEM_OPTIONS

During the installation of Job Admin, the following information is provided by the user and that information is loaded into the BDI_SYSTEM_OPTIONS table.

Table 2–1 System Options

Column	Type	Comments
VARIABLE_NAME	VARCHAR2(255)	Name of the system variable
APP_TAG	VARCHAR2(255)	The application name
VARIABLE_VALUE	VARCHAR2(255)	Value of the variable

<app>JobAdminBaseUrl - Base URL for Job Admin of destination applications such as sim/rxm

<app>JobAdminBaseUrlUserAlias - User alias for Job Admin of destination applications such as sim/rxm

<app> - Destination application name (for example sim or rxm)

Example:

```

insert into BDI_SYSTEM_OPTIONS VALUES('rxmJobAdminBaseUrl',
'com.oracle.retail.integration_bdi-batch-job-admin_war_16.0.0',
'http://rxmhost:7001/bdi-rxm-batch-job-admin')

```

BDI_INTERFACE_CONTROL

During the design time, seed data for the BDI_INTERFACE_CONTROL table is generated for all interface modules (aka families) for a job type (DOWNLOADER, UPLOADER) so that interface modules are active.

Table 2–2 Interface Control

Column	Type	Comments
ID	NUMBER	Primary Key
INTERFACE_CONTROL_COMMAND	VARCHAR2(255)	ACTIVE or IN_ACTIVE
INTERFACE_MODULE	VARCHAR2(255)	Name of interface module
SYSTEM_COMPONENT_TYPE	VARCHAR2(255)	DOWNLOADER or UPLOADER

Example:

```
insert into BDI_INTERFACE_CONTROL (ID, INTERFACE_CONTROL_COMMAND, INTERFACE_MODULE, SYSTEM_COMPONENT_TYPE) values (1, 'ACTIVE', 'Diff_Fnd', 'DOWNLOADER')
```

BDI_DWNLDR_TRNSMITTR_OPTIONS

Seed data for BDI_DWNLDR_TRNSMITTR_OPTIONS specifies various configuration options for the Downloader-Transmitter job. Seed data is generated during design time and executed during deployment.

Table 2–3 Transmitter Options

Column	Type	Comments
ID	NUMBER	Primary Key
INTERFACE_MODULE	VARCHAR2(255)	Name of interface module
INTERFACE_SHORT_NAME	VARCHAR2(255)	Name of the interface
RECVR_END_POINT_URL	VARCHAR2(255)	Name of the URL variable in BDI_SYSTEM_OPTIONS table
RECVR_END_POINT_URL_ALIAS	VARCHAR2(255)	Name of the URL alias variable in BDI_SYSTEM_OPTIONS table
PARTITION.	NUMBER	Number of partitions used by Downloader-Transporter job. Default value is 10. This value can be changed through Job Admin GUI
THREAD	NUMBER	Number of threads used by Downloader-Transporter job. Default value is 10. This value can be changed through Job Admin GUI.
QUERY_TEMPLATE	VARCHAR2(255)	Query to be run by downloader job
AUTO_PURGE_DATA	VARCHAR2(255)	This flag indicates Downloader-Transporter job whether to clean data set in the outbound table after the job successfully downloads the data set. Default value is set to True. This value need to be changed based on the deployment topology used for bulk data integration.

Example:

```
insert into BDI_DWNLDR_TRNSMITTR_OPTIONS (ID, INTERFACE_MODULE, INTERFACE_SHORT_NAME, RECVR_END_POINT_URL, RECVR_END_POINT_URL_ALIAS, PARTITION, THREAD, QUERY_TEMPLATE, AUTO_PURGE_DATA) values (1, 'DiffGrp_Fnd', 'Diff_Grp', 'rxmJobAdminBaseUrl', 'rxmJobAdminBaseUrlUserAlias', 10, 10, 'select * from InterfaceShortName where (bdi_seq_id between ? and ?) QueryFilter order by bdi_seq_id', 'False')
```

```
insert into BDI_DWNLDR_TRNSMITTR_OPTIONS (ID, INTERFACE_MODULE, INTERFACE_SHORT_NAME, RECVR_END_POINT_URL, RECVR_END_POINT_URL_ALIAS, PARTITION, THREAD, QUERY_TEMPLATE, AUTO_PURGE_DATA) values (2, 'DiffGrp_Fnd', 'Diff_Grp_Dtl', 'rxmJobAdminBaseUrl', 'rxmJobAdminBaseUrlUserAlias', 10, 10, 'select * from InterfaceShortName where (bdi_seq_id between ? and ?) QueryFilter order by bdi_seq_id', 'False')
```

Downloader-Transporter Job Properties

The following job properties can be changed in the Downloader-Transporter jobs to tune the performance.

item-count

Item Count is an attribute of the “chunk” element in the Downloader-Transporter job. The default value for “item-count” is set to 1000. The Downloader job retrieves 1000 rows of data from the database before it sends data to the Receiver Service.

```
<chunk checkpoint-policy="item" item-count="1000">
```

The default value is 1000. This value can be changed to fine tune the performance of the Downloader-Transporter job. You need to manually change the value in the job xml files in bdi-<app>-home/setup-data/job/META-INF/batch-jobs folder and reinstall the app. Increasing the item count will increase memory utilization.

fetchSize

The Fetch Size is a property in the Downloader-Transporter job.

FetchSize property is used by JDBC to fetch n number of rows and cache them. The default value is set to 1000. Typically “item-count” and “fetchSize” values are identical to get better performance.

```
<property name="fetchSize" value="1000"/>
```

This value can be changed to fine tune the performance of the Downloader-Transporter job. You need to manually change the value in the job xml files.

Cleanup

The Downloader-Transporter job deletes data from outbound tables after the successful completion of the job if the AUTO_PURGE_DATA flag in BDI_DWNLDR_TRNSMITTR_OPTIONS table is set to TRUE. The default value for this flag is TRUE. If sender side split topology is used, this flag needs to be changed to FALSE. Otherwise all destination applications may not get the data.

Error Handling

When a Downloader-Transporter job fails, error information like the stack trace gets stored in the BDI_JOB_ERROR and BDI_DOWNLOADER_JOB_ERROR tables. Errors are displayed in the “Diagnostics” tab of the Job Admin GUI. The error information can be used to fix the issues before restarting the failed job.

Note: If there are exceptions in Batch runtime, then those exceptions won't show up in Job Error tables and so in Diagnostics tab of Job Admin GUI.

BDI_DOWNLOADER_JOB_ERROR

Table 2–4 *Downloader Job Error*

Column	Type	Comments
DOWNLOADER_JOB_ERROR_ID	NUMBER	Primary key
PARTITION_INDEX	VARCHAR2(255)	Partition number of data set
BLOCK_NUMBER	NUMBER	Block number in the partition
BEGIN_SEQ_NUM_IN_BLOCK	NUMBER	Begin sequence number in the block
END_SEQ_NUM_IN_BLOCK	NUMBER	End sequence number in the block
JOB_ERROR_ID	NUMBER	Foreign key to JOB_ERROR table

BDI_JOB_ERROR

Table 2–5 *Job Error*

Column	Type	Comments
JOB_ERROR_ID	NUMBER	Primary key
TRANSACTION_ID	VARCHAR2(255)	Transaction Id of the job
INTERFACE_MODULE	VARCHAR2(255)	Name of the interface module
INTERFACE_SHORT_NAME	VARCHAR2(255)	Name of the interface
DESCRIPTION	VARCHAR2(1000)	Error description
STACK_TRACE	VARCHAR2(4000)	Stack trace

Receiver Service

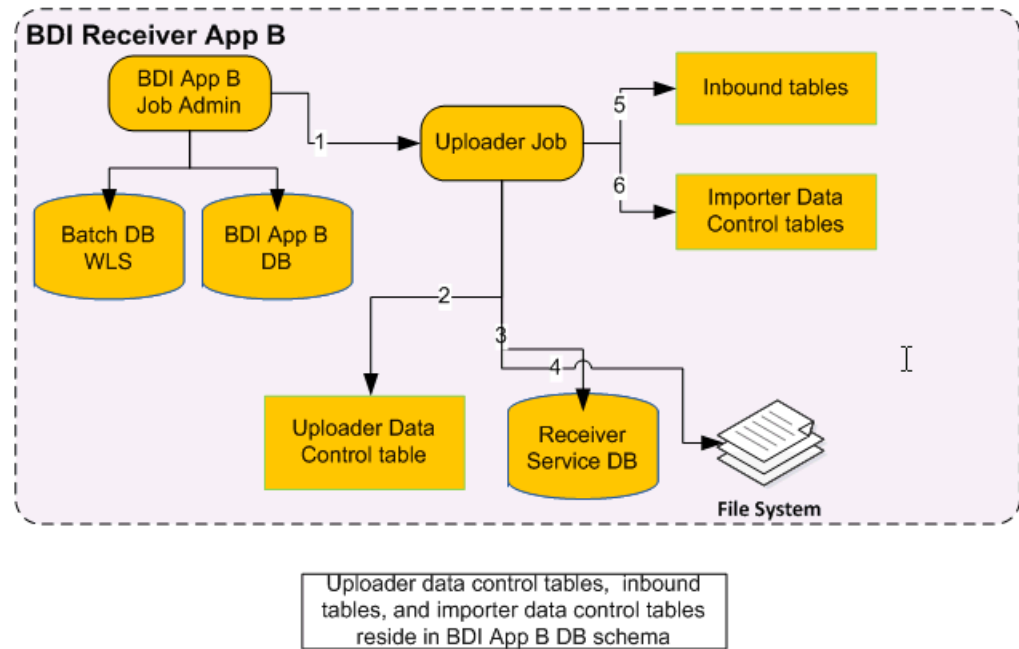
The Receiver Service is a RESTful service that provides various endpoints to send data transactionally.

The Receiver Service is part of Job Admin. It stores data as files and keeps track of metadata in the database. The Receiver Service also supports various merge strategies for merging files at the end.

The Receiver Service is used by the Downloader-Transporter job to transmit data from source to destination.

Uploader Job

An Uploader Job uploads data from CSV files into Inbound Tables for an Interface Module. It divides files into logical partitions and each partition is processed concurrently. If a data set is already present in inbound tables, and the Uploader Job tries to upload the same data set, then it will overwrite the existing data set.



BDI Uploader

1. The Uploader job is run from the BDI App B Job Admin application through REST or UI.
2. The Uploader job checks for data sets in the uploader data control table.
3. If the data set is available for upload, the uploader job finds the location of the files from Receiver Service database.
4. The Uploader job retrieves a block of data from the file(s).
5. The Uploader job inserts/updates data in inbound tables.
6. The Uploader job inserts entries in importer data control tables to indicate that a data set is ready for import.
7. Steps 4, 5, and 6 are repeated until there is no more data to upload.

Rules for processing a data set by an Uploader Job

1. A data set is available for upload, if it is not processed by an uploader job yet and if a newer data set is not processed successfully.
2. If the data set id is passed through job parameters (for example `jobParameters=dataSetId=1`) to the uploader job, it will use the data set if it is available as per the above rule. Otherwise job will fail.
3. If the data set id is not passed through job parameters to the uploader job, it will identify the next available data set if there is one. Otherwise the job completes without processing any data set.
4. If an uploader job fails for whatever reason, the data set that it tried to upload can only be uploaded by restarting the job after fixing the issue. If uploader job is started instead of a restart, it will either pick up a new data set or none.

Uploader Data Set

A Data Set consists of files created by the Receiver Service.

- The Uploader job uploads a single data set that is not uploaded to inbound tables yet. It uses BDI_UPLDER_IFACE_MOD_DATA_CTL and Receiver Service tables to identify the data set to be uploaded.
- If source data set id is passed through the job parameters, then the Uploader job uses that data set if it is available for upload.
- If there are multiple data sets available, then the Uploader job picks up the oldest data set available for upload.
- If a newer data set is processed by the Uploader job, then the older data set cannot be processed.
- If there is no data set available, the Uploader job completes without uploading any data.

The Uploader job tries to allocate files equally between different partitions. Data is read from files in each partition based on the “item-count” value in the job and data is uploaded to inbound tables. The job continues to read data from files in a partition until there is no more data to be read.

A data set is divided into logical partitions based on the number of partitions specified in the BDI_UPLOADER_OPTIONS table and number of files in the data set. The number of files allocated to each logical partition is calculated by dividing file count with number of partitions.

Example 1:

Number of files = 100
Number of partitions = 2

Files for partition = $100/2 = 50$
Files assigned to partition 1
 Begin File index = 0
 End File index = 49
Data assigned to partition 2
 Begin File Index = 50
 End File Index = 99

Example 2:

Number of files = 75
Number of partitions = 2

Files for partition = $75/2 = 37$
Extra files = $75 \% 2 = 1$
Files assigned to partition 1
 Begin File index = 0
 End File index = 36
Files assigned to partition 2
 Begin File index = 37
 End File index = $37 + 37 = 74$

Uploader Cleanup Jobs

The Cleanup job cleans data set(s) for an interface module from outbound tables or receiver files. Cleanup jobs are generated during design time. They are included in the sender side or receiver side split flows.

There are two types of cleanup jobs - extractor and receiver cleanup jobs.

The Extractor cleanup job deletes data set(s) from outbound tables. It identifies data sets that have been successfully downloaded and transmitted to destination applications and deletes the identified data sets from outbound tables.

The Receiver cleanup job deletes CSV files from the receiver file system. It identifies the files that have been successfully uploaded to inbound tables and deletes the files from the file system.

Uploader Job Configuration

BDI_INTERFACE_CONTROL

During the design time, seed data for the BDI_INTERFACE_CONTROL table is generated for all interface modules (aka families) for a job type of UPLOADER so that interface modules are active. DML runs during deployment.

Example:

```
insert into BDI_INTERFACE_CONTROL (ID, INTERFACE_CONTROL_COMMAND,
INTERFACE_MODULE, SYSTEM_COMPONENT_TYPE) values (1, 'ACTIVE', 'Diff_
Fnd', 'UPLOADER')
```

BDI_UPLOADER_OPTIONS

Seed data for BDI_UPLOADER_OPTIONS specifies various configuration options for uploader job. Seed data is generated during design time and executed during deployment.

Table 2–6 Uploader Options

Column	Type	Comments
ID	NUMBER	Primary Key
INTERFACE_MODULE	VARCHAR2(255)	Name of interface module
INTERFACE_SHORT_NAME	VARCHAR2(255)	Name of the interface
PARTITION	NUMBER	Number of partitions used by Uploader job. Default value is 10. This value can be changed through Job Admin GUI.
THREAD	NUMBER	Number of threads used by Uploader job. Default value is 10. This value can be changed through Job Admin GUI.
MERGE_STRATEGY	VARCHAR2(255)	Merge strategy used by Receiver Service - NO_MERGE, MERGE_TO_PARTITION_LEVEL, MERGE_TO_INTERFACE_LEVEL
AUTO_PURGE_DATA	VARCHAR2(255)	This flag indicates Uploader job whether to clean the files after uploader job successfully uploads the data to inbound tables. Default value is True. This value needs to be changed based on the deployment topology used for bulk data integration.

Example:

```
insert into BDI_UPLOADER_OPTIONS (ID, INTERFACE_MODULE, INTERFACE_SHORT_NAME,
```

MERGE_STRATEGY, PARTITION, THREAD, AUTO_PURGE_DATA) values (1, 'Diff_Fnd', 'Diff', 'NO_MERGE', 10, 10, 'False')

Uploader Job Properties

The following job property can be changed in uploader jobs to tune the performance.

item-count

Item Count is an attribute of “chunk” element in the Uploader job. The default value for “item-count” is set to 1000. Uploader job reads 1000 rows from the file(s) before it inserts/updates that data in the inbound tables.

```
<chunk checkpoint-policy="item" item-count="1000">
```

This value can be changed to fine tune the performance of an Uploader job. You need to manually change the value in the the job xml files in the bdi-<app>-home/setup-data/job/META-INF/batch-jobs folder and reinstall the app for changes to take place.

Cleanup

The Uploader job deletes CSV files from the receiver file system after the successful completion of the job if the AUTO_PURGE_DATA flag in BDI_UPLOADER_OPTIONS table is set to TRUE. The default value for this flag is TRUE. If receiver side split topology is used, this flag needs to be changed to FALSE. Otherwise all destination applications may not get the data in inbound tables.

Error Handling

When Uploader job fails, error information like stack trace gets stored in BDI_UPLOADER_JOB_ERROR table. Errors are displayed in the “Diagnostics” tab of Job Admin GUI. The error information can be used to fix the issues before restarting the failed job.

Note: If there are exceptions in Batch runtime, then those exceptions won't show up in Job Error tables and so in Diagnostics tab of Job Admin GUI.

BDI_UPLOADER_JOB_ERROR

Table 2-7 Uploader Job Error

Column	Type	Comments
UPLOADER_JOB_ERROR_ID	NUMBER	Primary key
FILE_NAME	VARCHAR2(255)	File in which error occurred
BEGIN_ROW_NUMBER	NUMBER	Beginning row number in the file
END_ROW_NUMBER	NUMBER	Ending row number in the file
JOB_ERROR_IDNUMBER	NUMBER	Foreign key to JOB_ERROR table

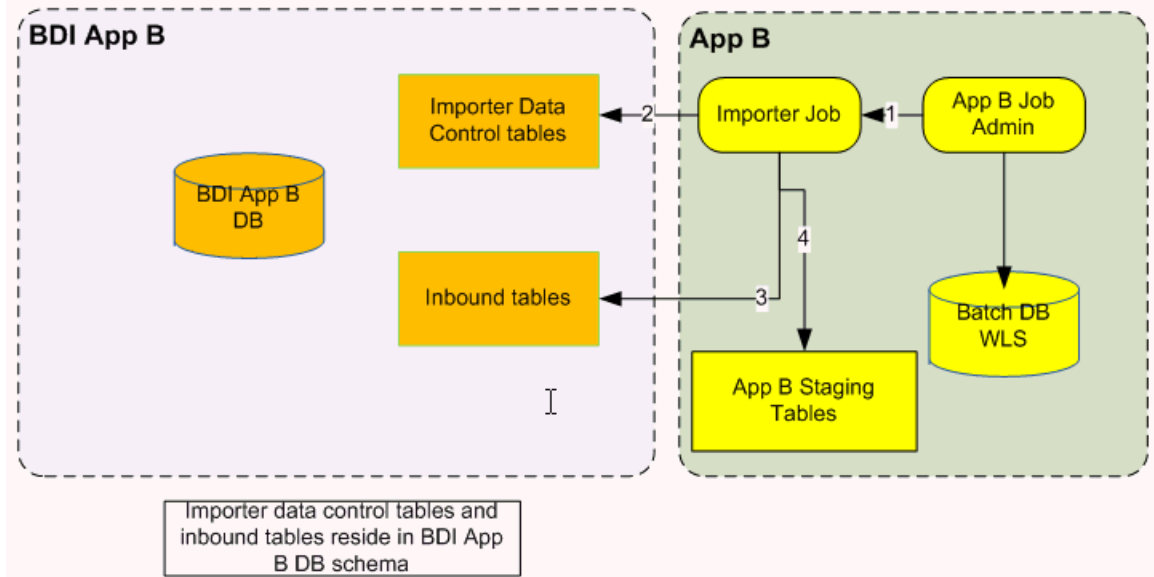
Importer Job

The tables BDI_IMPRTR_IFACE_MOD_DATA_CTL and BDI_IMPORTER_IFACE_DATA_CTL act as a handshake between the uploader and importer jobs. When the

Uploader Job 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/RXM) specific jobs. It uses the Importer Data Control Tables to identify whether a data set is ready for import or not.



RXM Importer

1. Importer job is run from App B 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 staging tables.

BDI_IMPRTR_IFACE_MOD_DATA_CTL

Table 2–8 Importer Data

Column	Type	Comments
IMPORTER_IFACE_MOD_DATACTL_ID	NUMBER	Primary key
INTERFACE_MODULE	VARCHAR2(255)	Name of the interface module
SOURCE_SYSTEM_NAME	NUMBER	Name of the source system
SOURCE_DATA_SET_ID	NUMBER	Source data set id
SRC_SYS_DATA_SET_READY_TIME	TIMESTAMP	Time when data set was ready in outbound tables

Table 2–8 (Cont.) Importer Data

Column	Type	Comments
DATA_SET_TYPE	VARCHAR2(255)	Type of data set (FULL or PARTIAL)
DATA_SET_READY_TIME	TIMESTAMP	Time when data set was available in inbound tables
UPLOADER_TRANSACTION_ID	NUMBER	Transaction id of the uploader job

BDI_IMPORTER_IFACE_DATA_CTL**Table 2–9 Importer Data**

Column	Type	Comments
IMPORTER_IFACE_DATA_CTL_ID	NUMBER	Primary key
INTERFACE_SHORT_NAME	VARCHAR2(255)	Name of the interface
INTERFACE_DATA_BEGIN_SEQ_NUM	NUMBER	Beginning sequence number of the data set in the inbound table
INTERFACE_DATA_END_SEQ_NUM	NUMBER	Ending sequence number of the data set in the inbound table
JIMPORTER_IFACE_MOD_DATACTL_ID	NUMBER	Foreign key to BDI_IMPORTER_IFACE_MOD_DATA_CTL table

Job Admin Services

This chapter discusses the Job Admin Services.

Job Admin RESTful Services

Job Admin provides below RESTful services. These services are secured with SSL and basic authentication.

- Batch Service - Ability to start/stop/restart, check status, and so on of jobs. This service is typically used by the BDI Process Flow engine.
- Receiver Service - Ability to stream data from one system to another system. This service is used by the Downloader-Transporter job.
- System Setting Service - Ability to view, change system settings, and credentials. Refer to Appendix D for details on System Setting REST resources.
- Data Service - Ability to get data set information using job information such as job name, execution id or instance id.

Receiver Service

The Receiver Service is a RESTful service that provides various endpoints to send data transactionally. Receiver Service is part of Job Admin. It stores data as files and keeps track of metadata in the database. The Receiver Service also supports various merge strategies for merging files at the end. The Receiver Service is used by the Downloader-Transporter job to transmit data from source to destination.

Seed data for Receiver Service is generated during design time and loaded during deployment of the Job Admin application.

BDI_RECEIVER_OPTIONS

The Receiver Service options can be configured at interface level.

Table 3-1 Receiver Options

Column	Type	Comments
ID	NUMBER	Primary key
INTERFACE_MODULE	VARCHAR2(255)	Name of the interface module
INTERFACE_SHORT_NAME	VARCHAR2(255)	Name of the interface

Table 3–1 (Cont.) Receiver Options

Column	Type	Comments
BASE_FOLDER	VARCHAR2(255)	This is the base folder for storing files created by Receiver Service. Receiver Service creates a subfolder “bdi-data” under base folder. Base folder can be changed from “Manage Configurations” tab of Job Admin GUI.
FOLDER_TEMPLATE	VARCHAR2(255)	Folder template provides the folder structure for storing files created by Receiver Service. Default value is “\${basefolder}/\${TxId}/\${Tid}/\${BId}/”. TxId - Transaction Id Tid - Transmission Id BId - Block Id This value can’t be changed.
MERGE_STRATEGY	VARCHAR2(255)	The strategy for merging files. The default value is “NO_MERGE”. The valid values are NO_MERGE, MERGE_TO_PARTITION_LEVEL, and MERGE_TO_INTERFACE_LEVEL. MERGE_TO_PARTITION_LEVEL Merges all files for that partition and creates the merged file in “\${Tid}” folder. MERGE_TO_INTERFACE_LEVEL Merges all files for interface and creates the merged file in “\${TxId}” folder. MERGE strategies are only supported in cases where the Uploader is not used.

Endpoints**Ping**

This endpoint can be used to check whether the Receiver Service is up or not.

HTTP Method: GET

Path: receiver/ping

Response: alive

Begin Receiver Transaction

This is the first endpoint to be called by the client (for example Downloader-Transporter job) before sending any data. It stores the following metadata in the BDI_RECEIVER_TRANSACTION table and returns the response in JSON format.

Parameter	Description
Transaction Id	Transaction Id (Tx#<Job Instance Id> of the Downloader-Transporter job)
Interface Module	Name of the interface module (for example Diff_Fnd)
Source System Name	Name of the source system (for example RMS)
sourceSystemId	ID of the source system (for example URL)
sourceDataSetId	ID of the data set
Data Set Type	Type of data set (FULL or PARTIAL)
Source System Data Set Ready Time	Time when source data set was ready in the outbound tables

HTTP Method: POST

Path:

receiver/beginTransaction/{transactionId}/{interfaceModule}/{sourceSystemName}/{sourceSystemId}/{sourceDataSetId}/{dataSetType}/{sourceSystemDataSetReadyTime}

Sample Response

```
{
  "receiverTransactionId": "1",
  "transactionId": "Tx#1",
  "sourceSystemName": "RMS",
  "interfaceModule": "Diff_Fnd",
  "sourceSystemId": "",
  "sourceDataSetId": "",
  "dataSetType": "FULL",
  "sourceSystemDataSetReadyTime": "",
  "dir": "",
  "fileName": "",
  "receiverTransactionStatus": "",
  "receiverTransactionBeginTime": "",
  "receiverTransactionEndTime": ""
}
```

Begin Receiver Transmission

This end point needs to be called by client (for example Downloader-Transporter job) before sending any data for a partition. It stores the following metadata in BDI_RECEIVER_TRANSMISSION table and returns response in JSON format.

Parameter	Description
TransmissionId	Generated for each partition
InterfaceModule)	Name of the interface module (for example Diff_Fnd)
InterfaceShortName)	Name of the interface (for example Diff

Parameter	Description
partitionName	Partition number
partitionBeginSeqNum	Begin sequence number in the partition
partitionEndSeqNum	End sequence number in the partition
beginBlockNumber	Begin block number

HTTP Method: POST**Path:**

receiver/beginTransmission/{transactionId}/{transmissionId}/{sourceSystemName}/
 {interfaceModule}/{interfaceShortName}/{partitionName}/{partitionBeginSeqNum}/
 {partitionEndSeqNum}/{beginBlockNumber}

Sample Response:

```
{
  "transmissionId": "1",
  "interfaceModule": "Diff_Fnd",
  "interfaceShortName": "Diff",
  "sourceSystemPartitionName": "1",
  "sourceSystemPartitionBeginSequenceNumber": "1",
  "sourceSystemPartitionEndSequenceNumber": "100",
  "beginBlockNumber": "1",
  "endBlockNumber": "",
  "dir": "",
  "fileName": "",
  "receiverTransmissionStatus": ""
}
```

Update Data Block

Clients use this endpoint to send data. This endpoint is typically called by the client multiple times until there is no more data. It creates a csv file with the data it received at the below location.

`${BASE_FOLDER}/bdi-data/${TxId}/${Tid}/${Bid}`

BASE_FOLDER - Obtained from the BDI_RECEIVER_OPTIONS table

TxId - Transaction Id of the remote Downloader-Transporter job

Tid - Transmission Id associated with the transaction id

Bid - Block Id associated with transmission id

It also stores the following metadata in the RECEIVER_TRANSMISSION_BLOCK table.

Parameter	Description
BlockNumber	Number of the block
ItemCountInBlock	Number of items in the block
Dir	Directory where file is created
FileName	Name of the file
ReceiverBlockStatus	Status of the block
CreateTime	Time when the block is created

HTTP Method: POST**Path:**

receiver/uploadDataBlock/{transactionId}/{transmissionId}/{sourceSystemName}/{interfaceModule}/{interfaceShortName}/{blockNumber}/{itemCountInBlock}

Sample Response

```
{
  "blockId": "1",
  "transmissionId": "1",
  "blockNumber": "1",
  "blockItemCount": "100",
  "dir": "",
  "fileName": "",
  "receiverBlockStatus": "",
  "createTime": ""
}
```

End Transmission

This end point ends transmission for a partition. It updates “endBlockNumber” and “receiverTransmissionStatus” in the RECEIVER_TRANSMISSION table.

HTTP Method: POST**Path:**

receiver/endTransmission/{transmissionId}/{sourceSystemName}/{interfaceModule}/{interfaceShortName}/{numBlocks}

Sample Response

```
{
  "transmissionId": "1",
  "interfaceModule": "Diff_Fnd",
  "interfaceShortName": "Diff",
  "sourceSystemPartitionName": "1",
  "sourceSystemPartitionBeginSequenceNumber": "1",
  "sourceSystemPartitionEndSequenceNumber": "100",
  "beginBlockNumber": "1",
  "endBlockNumber": "",
  "dir": "",
  "fileName": "",
  "receiverTransmissionStatus": ""
}
```

End Transaction

This end point ends the transaction and called once by the client. It updates “receiverTransactionStatus” and “receiverTransactionEndTime” in the RECEIVER_TRANSACTION table. If “mergeStrategy” is set to “MERGE_TO_PARTITION_LEVEL” or “MERGE_TO_INTERFACE_LEVEL”, then it merges the files and creates the merged file(s) at the appropriate directory. It creates an entry in the BDI_UPLOAD_INTERFACE_MOD_DATA_CTL table so that Uploader job can pick up the data.

HTTP Method: POST**Path:**

receiver/endTransaction/{transactionId}/{sourceSystemName}/{interfaceModule}

Sample Response

```
{
```

```

"receiverTransactionId": "1",
"transactionId": "Tx#1",
"sourceSystemName": "RMS",
"interfaceModule": "Diff_Fnd",
"sourceSystemId": "",
"dataSetType": "FULL",
"sourceSystemDataSetReadyTime": "",
"dir": "",
"fileName": "",
"receiverTransactionStatus": "",
"receiverTransactionBeginTime": "",
"receiverTransactionEndTime": ""
}

```

Uploader Interface Module Data Control Table

The BDI_UPLDER_IFACE_MOD_DATA_CTL table acts as a handshake between the downloader and uploader jobs. When the downloader-transporter job calls endTransaction on Receiver Service, the Receiver Service creates an entry in this table if it successfully received data and created files.

An entry in this table indicates to the uploader job that a data set is ready to be uploaded.

BDI_UPLDER_IFACE_MOD_DATA_CTL

Table 3–2 *Module Data Control*

Column	Type	Comments
UPLOADER_IFACE_MOD_DATA_CTLID	NUMBER	Primary key
INTERFACE_MODULE	VARCHAR2(255)	Name of the interface module
REMOTE_TRANSACTION_ID	VARCHAR2(255)	Transaction Id of Downloader-Transporter job
SOURCE_DATA_SET_ID	NUMBER	NUMBERID of the source data set
SRC_SYS_DATA_SET_READY_TIME	TIMESTAMP	Source Data Set Ready Time
DATA_SET_TYPE	VARCHAR2(255)	Type of data set (FULL or PARTIAL)
DATA_SET_READY_TIME	TIMESTAMP	Time when data set was available in the outbound tables
DATA_FILE_MERGE_LEVEL	VARCHAR2(255)	Merge level for the files (NO_MERGE, MERGE_TO_PARTITION_LEVEL, MERGE_TO_INTERFACE_LEVEL)
SOURCE_SYSTEM_NAME	VARCHAR2(255)	Name of the source system (for example RMS)

Receiver Side Split for Multiple Destinations

If there are multiple destinations that receive data from a source, one of the options is to use the Receiver Service at one destination to receive data from the sender and then multiple destinations use the data from one Receiver Service to upload to inbound tables. The requirements for the Receiver Side Split are such that:

- The Receiver Service database schema is shared by all the destinations

- The File system is shared by all destinations

The performance of BDI can be improved by using the receiver side split if there are multiple destinations.

Batch Service

Batch service is a RESTful service that provides various endpoints to manage batch jobs in the bulk data integration system. Batch Service is part of Job Admin.

Table 3-3 Batch Service

REST Resource	HTTP Method	Description
/batch/jobs	GET	Gets all available batch jobs
/batch/jobs/{jobName}	GET	Gets all instances for a job
/batch/jobs/{jobName}/executions	GET	Gets all executions for a job
/batch/jobs/executions	GET	Gets all executions
/batch/jobs/currently-running-jobs	GET	Gets currently running jobs
/batch/jobs/{jobName}/{jobInstanceId}/executions	GET	Gets job executions for a job instance
/batch/jobs/{jobName}/{jobExecutionId}	GET	Gets job instance and execution for a job execution id
/batch/jobs/{jobName}	POST	Starts a job asynchronously
/batch/jobs/executions/{jobExecutionId}	POST	Restarts a stopped or failed job
/batch/jobs/executions	DELETE	Stops all running job executions
/batch/jobs/executions/{jobExecutionId}	DELETE	Stops a job execution
/batch/jobs/executions/{jobExecutionId}	GET	Gets execution steps with details
/batch/jobs/executions/{jobExecutionId}/steps	GET	Gets execution steps
/batch/jobs/executions/{jobExecutionId}/steps/{stepExecutionId}	GET	Gets step details
/batch/jobs/job-def-xml-files	GET	Gets all job xml files

Key End Points

Start Job

This end point starts a job asynchronously based on a job name and returns the execution id of the job in the response.

Path: /batch/jobs/{jobName}

HTTP Method: POST

Inputs

Job Name as path parameter

Job Parameters as a query parameter. Job Parameters is a comma separated list of name value pairs. This parameter is optional.

Sample Request

`http://localhost:7001/bdi-batch-job-admin/resources/batch/jobs/DiffGrp_Fnd_ImporterJob?jobParameters=datasetId=1`

Successful Response

XML

```
<executionIdVo targetNamespace="">
<executionId>1</executionId>
<jobName>DiffGrp_Fnd_ImporterJob</jobName>
</executionIdVo>
```

JSON

```
{
  "executionId": 1,
  "jobName": "DiffGrp_Fnd_ImporterJob"
}
```

Error Response

XML

```
<exceptionVo targetNamespace="">
<statusCode>404</statusCode>
<status>NOT_FOUND</status>
<message>HTTP 404 Not Found</message>
<stackTrace></stackTrace> <!-- optional -->
</exceptionVo>
```

JSON

```
{
  "statusCode": "404",
  "status": "NOT_FOUND",
  "message": "HTTP 404 Not Found",
  "stackTrace": ""
}
```

Restart Job

This end point restarts a job asynchronously using the job execution id and returns the new job execution id.

Path: `/batch/jobs/executions/{executionId}`

HTTP Method: POST

Inputs

executionId as path parameter

Sample Request

`http://localhost:7001/bdi-batch-job-admin/resources/batch/jobs/executions/2`

Successful Response

XML

```
<executionIdVo targetNamespace="">
<executionId>2</executionId>
```



```
<jobName>DiffGrp_Fnd_ImporterJob</jobName>
</executionIdVo>
```

JSON

```
{
  "executionId": 2,
  "jobName": "DiffGrp_Fnd_ImporterJob"
}
```

Error Response

XML

XML

```
<exceptionVo targetNamespace="">
  <statusCode>500</statusCode>
  <status>INTERNAL_SERVER_ERROR</status>
  <message>Internal Server Error</message>
  <stackTrace></stackTrace> <!-- optional -->
</exceptionVo>
```

JSON

```
{
  "statusCode": "500",
  "Status": "INTERNAL_SERVER_ERROR",
  "Message": "Internal Server Error",
  "stackTrace": ""
}
```

Check Status of a Job

This endpoint returns the status of a job using the job name and execution id.

Path: /batch/jobs/jobName/{jobExecutionId}

HTTP Method: GET

Inputs

jobName as path parameter

jobExecutionId as path parameter

Sample Request

http://localhost:7001/bdi-batch-job-admin/resources/batch/jobs/DiffGrp_Fnd_ImporterJob/1

Successful Response

XML

```
<jobInstanceExecutionsVo targetNamespace="">
  <jobName>DiffGrp_Fnd_ImporterJob</jobName>
  <jobInstanceId>1</jobInstanceId>
  <resource>http://localhost:7001/bdi-batch-job-admin/resources/batch/jobs/DiffGrp_
  Fnd_ImporterJob/1</resource>
  <jobInstanceExecutionVo>
    <executionId>1</executionId>
    <executionStatus>COMPLETED</executionStatus>
    <executionStartTime>2016-07-11 15:45:27.356</executionStartTime>
    <executionDuration>10</executionDuration>
  </jobInstanceExecutionVo>
```

```

</jobInstanceExecutionsVo>

JSON
{
  "jobName": "DiffGrp_Fnd_ImporterJob",
  "jobInstanceId": 1,
  "resource":
"http://localhost:7001/bdi-batch-job-admin/resources/batch/jobs/DiffGrp_Fnd_
ImporterJob/1",
  ["jobInstanceExecutionVo": {
    "executionId": 1,
    "executionStatus": "COMPLETED",
    "executionStartTime": "2016-07-11 15:45:27.356",
    "executionDuration": "10"
  }]
}

```

Error Response

```

XML

<exceptionVo targetNamespace="">
<statusCode>500</statusCode>
<status>INTERNAL_SERVER_ERROR</status>
<message>Internal Server Error</message>
<stackTrace></stackTrace> <!-- optional -->
</exceptionVo>

```

```

JSON

{
  "statusCode": "500",
  "Status": "INTERNAL_SERVER_ERROR",
  "Message": "Internal Server Error",
  "stackTrace": ""
}

```

Data Service

Data Service is a RESTful service that provides end points to get data set information based on job level information.

Table 3–4 Data Service

REST Resource	HTTP Method	Description
/data/dataset/{jobName}/ executions/{jobExecutionId}	GET	Gets a data set based on job name and job execution id
/data/dataset/{jobName}/i nstances/{jobInstanceId}	GET	Gets a data set based on job name and job instance id
/data/dataset/{jobName}ne xtPending	GET	Gets next pending data set based on job name

Get Data Set for job name and execution id

Job name - Extractor or downloader-transmitter or uploader job name

Execution id - Job execution id

This endpoint is used by a process flow to get the data set id after the extractor job is run successfully.

Sample Request

http://localhost:7001/bdi-batch-job-admin/resources/data/dataset/Diff_Fnd_ExtractorJob/executions/1

Sample Response

Returns response in either XML or JSON format.

```
<jobDataSetVo>
  <interfaceModule>Diff_Fnd</interfaceModule>
  <interfaceModuleDataControlId>2</interfaceModuleDataControlId>
  <jobName>Diff_Fnd_ExtractorJob</jobName>
  <jobDataSetInstance>
    <jobInstanceId>1</jobInstanceId>
    <jobDataSetExecutions>
      <jobExecutionId>1</jobExecutionId>
    </jobDataSetExecutions>
  </jobDataSetInstance>
</jobDataSetVo>
```

Get Data Set for job name and instance id

Job name - Extractor or downloader-transmitter or uploader job

Instance id - Job instance id

Sample Request

http://localhost:7001/bdi-batch-job-admin/resources/data/dataset/Diff_Fnd_ExtractorJob/instances/1

Sample Response

```
<jobDataSetVo>
  <interfaceModule>Diff_Fnd</interfaceModule>
  <interfaceModuleDataControlId>2</interfaceModuleDataControlId>
  <jobName>Diff_Fnd_ExtractorJob</jobName>
  <jobDataSetInstance>
    <jobInstanceId>1</jobInstanceId>
    <jobDataSetExecutions>
      <jobExecutionId>1</jobExecutionId>
    </jobDataSetExecutions>
  </jobDataSetInstance>
</jobDataSetVo>
```

Get next pending data set for job name

This endpoint is applicable only to the downloader-transporter or uploader jobs.

Sample Request

http://localhost:7001/bdi-batch-job-admin/resources/data/dataset/Diff_Fnd_DownloaderAndTransporterToRxmJob/nextPending

Sample Response

```
<jobDataSetVo>
  <interfaceModule>Diff_Fnd</interfaceModule>
  <interfaceModuleDataControlId>9</interfaceModuleDataControlId>
</jobDataSetVo>
```

Configuration of Job Admin

During the deployment of Job Admin, seed data gets loaded to various tables. Seed data files are located in the bdi-<app>-home/setup-data/dml folder. If seed data is changed, Job Admin need to be reinstalled and redeployed. For loading seed data again during the redeployment, LOADSEEDDATA flag in the BDI_SYSTEM_OPTIONS table need to be set to TRUE.

Jobs

The following job properties can be changed to improve the performance of the jobs.

Item-count - Number of items read by a job before it writes. Default size is 1000.

fetchSize - Number of items cached by JBDC driver. Default size is 1000.

Job xml files are located in bdi-<app>-home/setup-data/job/META-INF/batch-jobs folder. Job xml files can be changed with new values in this folder. If Job xml files are changes, Job Admin need to be reinstalled and redeployed.

Receiver Service

The Receiver Service allows maximum number of blocks for transaction based on the following system option in BDI_SYSTEM_OPTIONS table.

receiverMaxNumBlocks - Default value is set to 10000

Seed data need to be changed to update the maximum number of blocks for the Receiver Service. To update the seed data, set the LOADSEEDDATA flag to TRUE, reinstall and redeploy Job Admin. The Value of the LOADSEEDDATA flag can be changed from the Job Admin Manage Configurations Tab.

Job Admin UI

The BDI Job Admin UI is a web application that provides the GUI for managing batch jobs and runtime.

The User Interface provides ability to:

- Start/restart, and track status of jobs
- Trace data
- View diagnostic errors
- Manage options at job and system level
- View the logs

Job Admin UI Security

Security in the integration layer is a big concern for every retail enterprise. The security system should be open enough to allow trusted remote applications to integrate easily and, at the same time, lock down unauthorized remote access. To address security concerns, the Job Admin utilizes the security models allowed in the Oracle middleware and database systems.

Authentication

Both the Job Admin UI and REST Services are secured with SSL and basic authentication.

Authorization

The below mentioned roles are defined to restrict access to operations in Job Admin.

- BdiJobAdminRole
- BdiJobOperatorRole
- BdiJobMonitorRole

There are three categories of users in Job Admin: Job Administrators, Job Operators, and Job Monitors. Batch jobs can be run from Job Admin UI or through the Batch REST service. Here are the operations that can be performed by the users based on their role.

Function	Admin Role	Operator Role	Monitor Role
Edit configuration from UI	Yes	No	No

Function	Admin Role	Operator Role	Monitor Role
Create/update/delete system options	Yes	No	No
Create/update/delete system credentials	Yes	No	No
View credentials	Yes	No	No
Run Jobs	Yes	Yes	No
Monitor jobs	Yes	Yes	Yes

Monitoring Batch Jobs Using BDI Job Admin

Batch jobs can be monitored using the Job Admin UI.

Batch Summary Tab

Figure 4–1 Batch Summary Tab

The screenshot shows the Oracle BDI RXM Batch Job Admin interface. At the top, there's a navigation bar with tabs: Batch Summary (selected), Manage Batch Jobs, Trace Data, Diagnostics, Manage Configurations, and System Logs. Below the navigation bar is the System Summary section, which contains several summary cards: Batch Application (BDI-RXM), System Health (green checkmark), Total Jobs (24), Total Executions (21), Total Successful Executions (21), and Total Failed Executions (0). Below the System Summary is the Latest Job Executions section, which features a search bar and a table of job executions.

Job Name	Family	Instance Id	Execution Id	Start Time	Status
OrgHier_Fnd_UploaderJob	OrgHier	163	169	Fri Oct 07 01:26:00 PDT 2016	COMPLETED
MerchHier_Fnd_UploaderJob	MerchHier	151	157	Fri Oct 07 01:20:00 PDT 2016	COMPLETED
ItemLoc_Fnd_UploaderJob	ItemLoc	149	155	Fri Oct 07 00:51:00 PDT 2016	COMPLETED
ItemHdr_Fnd_UploaderJob	ItemHdr	147	153	Fri Oct 07 00:40:00 PDT 2016	COMPLETED
InvAvailWh_Tx_UploaderJob	InvAvailWh	145	151	Fri Oct 07 00:35:00 PDT 2016	COMPLETED
DiffGrp_Fnd_UploaderJob	DiffGrp	141	147	Fri Oct 07 00:21:01 PDT 2016	COMPLETED
Diff_Fnd_UploaderJob	Diff	137	143	Fri Oct 07 00:16:04 PDT 2016	COMPLETED
ItemHdr_Fnd_UploaderJob	ItemHdr	125	131	Thu Oct 06 03:42:45 PDT 2016	COMPLETED
Diff_Fnd_UploaderJob	Diff	123	129	Thu Oct 06 03:36:55 PDT 2016	COMPLETED
OrgHier_Fnd_UploaderJob	OrgHier	111	117	Thu Oct 06 01:26:01 PDT 2016	COMPLETED
MerchHier_Fnd_UploaderJob	MerchHier	99	105	Thu Oct 06 01:21:00 PDT 2016	COMPLETED

This tab shows the summary of the system and details about the latest batch job executions. It can be used to quickly find out whether the latest jobs are successful or not. The last section of this page displays the step summary of the selected job.

Manage Jobs Tab

This tab displays the list of available jobs with their details and allows you to Start a job, Restart failed jobs, list the executions of a job.

Figure 4–2 Manage Jobs Tab

The screenshot displays the Oracle BDI RMS Batch Job Admin interface. The top section, 'All Jobs Definition', contains a table with the following data:

Job Name	Family	Job Description	Execution Count	Action
CodeDetail_Fnd_DownloaderAndTransporterToSimJob	CodeDetail	SIM bound CodeDetail_Fnd Downloader and Transporter Job	0	Launch View Executions
CodeHead_Fnd_DownloaderAndTransporterToSimJob	CodeHead	SIM bound CodeHead_Fnd Downloader and Transporter Job	0	Launch View Executions
DeliverySlot_Fnd_DownloaderAndTransporterToSimJob	DeliverySlot	SIM bound DeliverySlot_Fnd Downloader and Transporter Job	0	Launch View Executions
DiffGp_Fnd_DownloaderAndTransporterToSimJob	DiffGp	RXM bound DiffGp_Fnd Downloader and Transporter Job	10	Launch View Executions
DiffGp_Fnd_DownloaderAndTransporterToSimJob	DiffGp	SIM bound DiffGp_Fnd Downloader and Transporter Job	0	Launch View Executions
DiffGp_Fnd_ExtractorCleanupJob	DiffGp	DiffGp_Fnd Extractor Cleanup Job	0	Launch View Executions
Diff_Fnd_DownloaderAndTransporterToSimJob	Diff	RXM bound Diff_Fnd Downloader and Transporter Job	18	Launch View Executions
Diff_Fnd_DownloaderAndTransporterToSimJob	Diff	SIM bound Diff_Fnd Downloader and Transporter Job	0	Launch View Executions
Diff_Fnd_ExtractorCleanupJob	Diff	Diff_Fnd Extractor Cleanup Job	0	Launch View Executions
FinisherAddr_Fnd_DownloaderAndTransporterToSimJob	FinisherAddr	SIM bound FinisherAddr_Fnd Downloader and Transporter Job	0	Launch View Executions
ImvAvailStore_Tx_DownloaderAndTransporterToSimJob	ImvAvailStore	SIM bound ImvAvailStore_Tx Downloader and Transporter Job	3	Launch View Executions

The bottom section, 'Job Executions', shows a table with the following data:

Job Name	Instance Id	Execution Id	Job Parameters	Start Time	End Time	Duration	Status
DiffGp_Fnd_DownloaderAndTransporterToSimJob	358	359	url=http://hr00abk.lic.oracle.com:7302/bdi-mo-batch-job-admin/resources/batch/jobs/DiffGp_Fnd_DownloaderAndTransporterToSimJob	Thu Sep 29 23:42:03 PDT 2016	Thu Sep 29 23:42:03 PDT 2016	0 Hours 0 Minutes 0 Seconds	COMPLETED

Job Executions

This tab shows the executions of the selected jobs. It can be used to restart the failed executions of a job. The Restart button is available only for restartable executions in the status column. When the user clicks the restart button it is redirected to the job launch tab with the restart option and pre-populated value of the job parameters from last run of the execution. User can edit the value of the existing parameters except the dataSetId and enter new parameters in comma separated format.

Note: Editing the dataSetId during restart can result in errors.

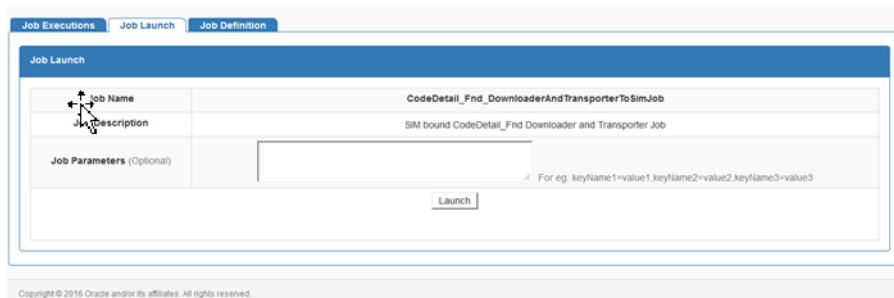
Note: The url is a infrastructure parameter, the user is not allowed to change its value. n

DataSetId in job parameter is not supposed to be edited, and updating same during restart can result into errors.

Job Launch

This tab can be used to launch the jobs. Job Parameters is an optional input to launch the jobs. Multiple job parameters can be entered in comma separated value format. On restart, the user is redirected to the Job Launch tab, and the launch button is replaced with the restart button. The Job parameters values are pre-populated from the last failed run of the instance. The user has an option to add or update existing key values, except dataSetId.

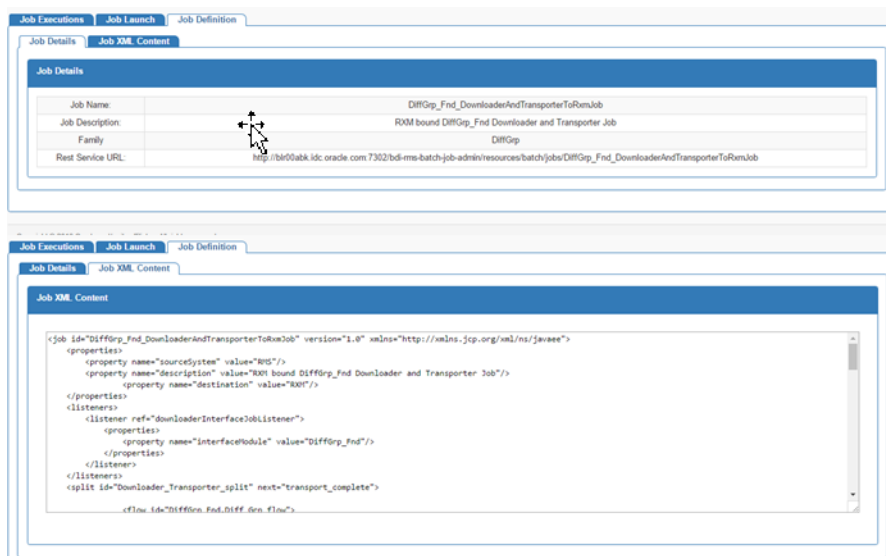
Figure 4–3 Job Launch



Job Details

This tab shows the details of the selected job such as Job Description, Family, Rest Service Url and Job Xml content.

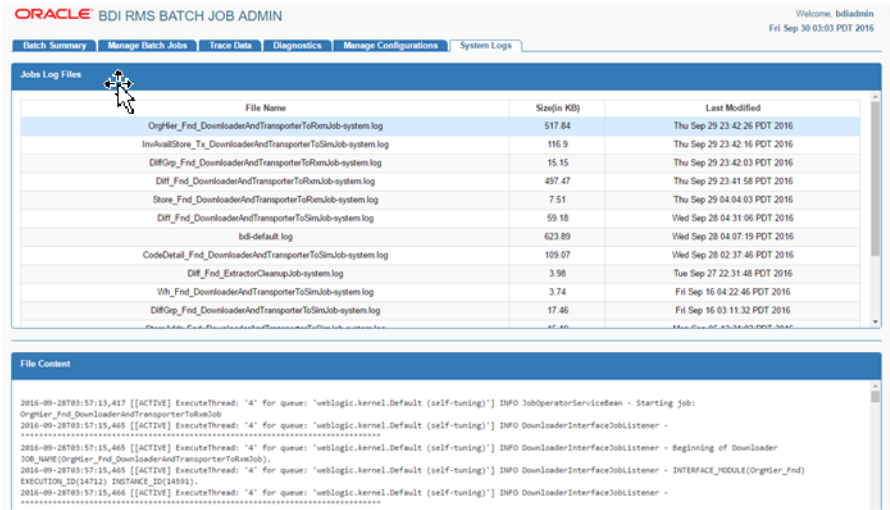
Figure 4–4 Job Details



System Logs Tab

This tab shows logs at job and system level. If a job fails, the job level log provides details about the failure. Information about a job in the log file starts and ends with a banner that shows details such as job name, instance id, execution id and so on.

Figure 4-5 System Logs Tab



Sample Begin Job Banner

```

*****
2016-08-03T02:15:00,764 [[ACTIVE] ExecuteThread: '13' for queue:
'weblogic.kernel.Default (self-tuning)'] INFO DownloaderInterfaceJobListener -
Beginning of Downloader JOB_NAME(Diff_Fnd_
DownloaderAndTransporterToSimJob).
2016-08-03T02:15:00,764 [[ACTIVE] ExecuteThread: '13' for queue:
'weblogic.kernel.Default (self-tuning)'] INFO DownloaderInterfaceJobListener -
INTERFACE_MODULE(Diff_Fnd) EXECUTION_ID(3844) INSTANCE_ID(3844).
2016-08-03T02:15:00,764 [[ACTIVE] ExecuteThread: '13' for queue:
'weblogic.kernel.Default (self-tuning)'] INFO DownloaderInterfaceJobListener -
*****
    
```

Sample End Job Banner

```

*****
2016-08-03T02:15:02,080 [[ACTIVE] ExecuteThread: '13' for queue:
'weblogic.kernel.Default (self-tuning)'] INFO DownloaderInterfaceJobListener - End of
Downloader JOB_NAME(Diff_Fnd_DownloaderAndTransporterToSimJob).
2016-08-03T02:15:02,080 [[ACTIVE] ExecuteThread: '13' for queue:
'weblogic.kernel.Default (self-tuning)'] INFO DownloaderInterfaceJobListener -
INTERFACE_MODULE(Diff_Fnd) EXECUTION_ID(3844) INSTANCE_ID(3844).
2016-08-03T02:15:02,081 [[ACTIVE] ExecuteThread: '13' for queue:
'weblogic.kernel.Default (self-tuning)'] INFO DownloaderInterfaceJobListener - Total
time for Downloader job: 1 seconds.
2016-08-03T02:15:02,081 [[ACTIVE] ExecuteThread: '13' for queue:
'weblogic.kernel.Default (self-tuning)'] INFO DownloaderInterfaceJobListener -
*****
    
```

It also shows whether it processed a data set or not. Here are the keywords that can be used to search the job level log files.

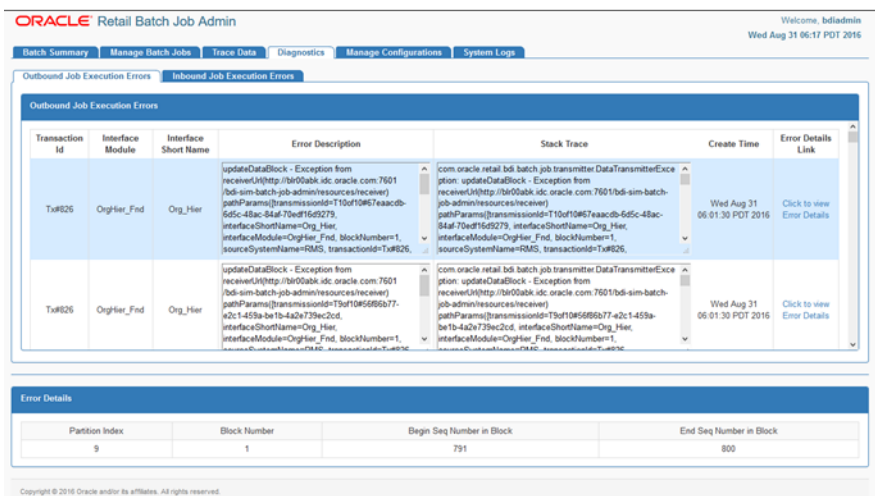
Key Word	Description
JOB_NAME	Name of the job
EXECUTION_ID	Execution Id of the job
INSTANCE_ID	Instance Id of the job
TRANSACTION_ID	Transaction Id (Tx#<Job Instance Id>)
INTERFACE_MODULE	Name of the interface module
INTERFACE_SHORT_NAME	Name of the interface
PROCESSING_DATA_SET	Indicates and shows the details about the data set that job is processing
DATA_SET_PROCESSED	Indicates that the job successfully processed the data set
DATA_SET_FAILED	Indicates that the job failed to process the data set

Diagonostics Tab

This tab shows general job level error information such as error description, stack trace etc as well as job specific error information. Use this tab to identify where a job failed and fix the issue.

Outbound Job Execution Errors

Figure 4–6 Outbound Job Executions Errors



The following information is displayed for outbound job execution errors. There can be multiple outbound job execution errors for a job instance.

Field Name	Description
Partition Index	Partition in which the error occurred
Block Number	Block in which the error occurred
Begin Sequence Number	Beginning sequence number in the block
End Sequence Number	Ending sequence number in the block

Inbound Job Execution Errors

The following information is displayed for inbound job execution errors. There can be multiple inbound job execution errors for a job instance.

Field Name	Description
File Name	Name of the file in which the error occurred
Begin Row Number	Beginning row number in the file
End Row Number	Ending row number in the file

Trace Data

This tab shows data movement in the BDI. Use this tab to verify that data moved from the sender to destination inbound tables.

Sender Data

Figure 4-7 Sender Data

The screenshot shows the Oracle BDI RMS Batch Job Admin interface. The 'Sender Data' tab is active, displaying a table of Outbound Job Executions. The table has the following columns: Tx Id, Interface Module, Data Set Id, Data Set Ready Time, Data Set Type, Status, Transaction Duration, and Job Name. The data rows show various transactions, all with a status of 'DOWNLOADER_TRANSMITTER_COMPLETED'. Below the table, there are controls for the Downloader/Transporter Data Control and Options for the selected interface (OrgHsr_Fnd).

Tx Id	Interface Module	Data Set Id	Data Set Ready Time	Data Set Type	Status	Transaction Duration	Job Name
Tx#362	OrgHsr_Fnd	12	Wed Aug 03 15:05:13 PDT 2016	FULL	DOWNLOADER_TRANSMITTER_COMPLETED	0 Hours 0 Minutes 1 Seconds	OrgHsr_Fnd_DownloaderAndTransporterToRxmJob
Tx#347	Difl_Fnd	7	Tue Aug 09 20:31:14 PDT 2016	FULL	DOWNLOADER_TRANSMITTER_COMPLETED	0 Hours 0 Minutes 1 Seconds	Difl_Fnd_DownloaderAndTransporterToRxmJob
Tx#336	OrgHsr_Fnd	11	Wed Aug 03 15:05:13 PDT 2016	FULL	DOWNLOADER_TRANSMITTER_COMPLETED	0 Hours 0 Minutes 1 Seconds	OrgHsr_Fnd_DownloaderAndTransporterToRxmJob
Tx#325	OrgHsr_Fnd	10	Wed Aug 03 15:05:13 PDT 2016	FULL	DOWNLOADER_TRANSMITTER_COMPLETED	0 Hours 0 Minutes 0 Seconds	OrgHsr_Fnd_DownloaderAndTransporterToRxmJob
Tx#314	InvAvalStore_Tx	20	Tue Aug 09 20:38:01 PDT 2016	FULL	DOWNLOADER_TRANSMITTER_COMPLETED	0 Hours 0 Minutes 0 Seconds	InvAvalStore_Tx_DownloaderAndTransporterToSimJob
Tx#303	Difl_Fnd	6	Tue Aug 09 20:31:12 PDT 2016	FULL	DOWNLOADER_TRANSMITTER_COMPLETED	0 Hours 0 Minutes 0 Seconds	Difl_Fnd_DownloaderAndTransporterToRxmJob
Tx#292	InvAvalStore_Tx	19	Tue Aug 09 20:37:59 PDT 2016	FULL	DOWNLOADER_TRANSMITTER_COMPLETED	0 Hours 0 Minutes 0 Seconds	InvAvalStore_Tx_DownloaderAndTransporterToSimJob

Interface Short Name	Begin Sequence Number	End Sequence Number	Receiver Endpoint URL	Data Partition	Thread	Auto Purge
Org_Hsr	401	500	http://br00atk.ldc.oracle.com:8013/bdi-rms-batch-job-admin/resources/receiver	10	10	TRUE

This tab shows the following information about the sender data by the sender side Job Admin (for example bdi-rms-batch-job-admin)

Field Name	Description
Transaction Id	Transaction Id (Tx#<Job Instance Id>) of the job
Interface Module	Name of the interface module
Job Name	Name of the batch job

Field Name	Description
Data Set Ready Time	Time when sender moved the data outbound tables
Data Set Type	Type of data set (FULL or PARTIAL)
Status	Status of the job (COMPLETED or FAILED)
Transaction Duration	Duration of the job

Receiver Data

Figure 4–8 Receiver Data

The screenshot shows the Oracle BDI RXM Batch Job Admin interface. The 'Receiver Data' tab is selected, displaying a table of receiver transactions. Below the table, there is a section for 'Receiver Transmission Details - Partition Level' with a table of transmission details.

Source Transaction Id	Source System	Source Data Set Id	Source Data Set Ready Time	Family	Transaction Status	Duration	Source System URL
Tx#362	RMS	12	Wed Aug 03 15:05:13 PDT 2016	OrgHier	RECEIVER_COMPLETED	0 Hours 0 Minutes 1 Seconds	http://bdi00abk.idc.oracle.com:7302/bdi-rms-batch-job-admin/resources/batchjobs/OrgHier_Fnd_DownloaderAndTransporterToRxmJob
Tx#347	RMS	7	Tue Aug 09 20:31:14 PDT 2016	Diff	RECEIVER_COMPLETED	0 Hours 0 Minutes 1 Seconds	http://bdi00abk.idc.oracle.com:7302/bdi-rms-batch-job-admin/resources/batchjobs/Diff_Fnd_DownloaderAndTransporterToRxmJob
Tx#336	RMS	11	Wed Aug 03 15:05:13 PDT 2016	OrgHier	RECEIVER_COMPLETED	0 Hours 0 Minutes 1 Seconds	http://bdi00abk.idc.oracle.com:7302/bdi-rms-batch-job-admin/resources/batchjobs/OrgHier_Fnd_DownloaderAndTransporterToRxmJob
			Wed Aug			0 Hours 0	

Transmission Id	Family	Interface Short Name	Source System Partition Name	Partition Begin Sequence Number	Partition End Sequence Number	Begin Block Number	End Block Number	Status	Duration

This tab shows the following information about the receiver data by the destination application (for example bdi-rxm-batch-job-admin).

Receiver Transactions

Field Name	Description
Source Transaction Id	Transaction Id of the sender job
Source System	Name of the sender
Family	Name of the interface module
Transaction Status	Status of the transaction (COMPLETED or FAILED)
Duration	Time it took to send data to Receiver
Source System URL	URL of the source job

Receiver Transmission Details - Partition Level

Field Name	Description
Transmission Id	Transmission Id of the partition
Family	Name of the interface module
Interface Short Name	Name of the interface

Field Name	Description
Source System Partition Name	Partition Number
Partition Begin Sequence Number	Beginning sequence number in the partition
Partition End Sequence Number	Ending sequence number in the partition
Begin Block Number	Beginning block number in the partition
End Block Number	Ending block number in the partition
Status	Status of the transmission (COMPLETED or FAILED)
Duration	Time it took to send data for a partition

Receiver Transmission Details - Block Level

Field Name	Description
Block Number	Block Number in a partition
Block Item Count	Number of items in the block
Block Status	Status (COMPLETED or FAILED)
File Location	Location of the file for the block

Uploader Data

Figure 4-9 Uploader Data

The screenshot shows the Oracle BDI SIM BATCH JOB ADMIN interface. The top navigation bar includes tabs for Batch Summary, Manage Batch Jobs, Trace Data, Diagnostics, Manage Configurations, and System Logs. The Uploader Data tab is selected, showing a table of Inbound Job Executions. Below the table is a section for Importer Data Control and Options for Interface: MerchHier_Fnd.

Transaction Id	Remote Tx ID	Status	Interface Module	Source System	Source Data Set Id	Data Set Ready Time	Source Sys Data Set Ready Time	Data Set Type	File Merge Level
Tx#1394	Tx#4069	UPLOADER_COMPLETED	MerchHier_Fnd	RMS	225	Thu Sep 29 13:56:20 EDT 2016	Thu Sep 29 10:55:07 EDT 2016	FULL	NO_MERGE
Tx#1370	Tx#4044	UPLOADER_COMPLETED	DiffGrp_Fnd	RMS	223	Thu Sep 29 13:34:35 EDT 2016	Thu Sep 29 10:33:25 EDT 2016	FULL	NO_MERGE
Tx#1359	Tx#4032	UPLOADER_COMPLETED	Store_Fnd	RMS	222	Thu Sep 29 13:14:43 EDT 2016	Thu Sep 29 10:12:33 EDT 2016	FULL	NO_MERGE
Tx#1335	Tx#3994	UPLOADER_COMPLETED	Store_Fnd	RMS	218	Thu Sep 29 12:16:05 EDT 2016	Thu Sep 29 09:13:55 EDT 2016	FULL	NO_MERGE
Tx#1324	Tx#3982	UPLOADER_COMPLETED	ItemLoc_Fnd	RMS	217	Thu Sep 29 12:01:48 EDT 2016	Thu Sep 29 08:59:33 EDT 2016	FULL	NO_MERGE
Tx#1301	Tx#3945	UPLOADER_COMPLETED	DiffGrp_Fnd	RMS	56	Thu Sep 29 06:07:41 EDT 2016	Thu Sep 22 13:17:34 EDT 2016	FULL	NO_MERGE

Interface Short Name	Begin Sequence Number	End Sequence Number	Data Partition	Thread	Merge Strategy	Auto Purge Data
MerchHier_Fnd	40000	40010	10	10	NO_MERGE	FALSE

This tab shows inbound job execution details and importer data controls.

Inbound job Executions

Field Name	Description
Transaction Id	Transaction Id (Tx#<Job Instance Id>) of the uploader job
Remote Transaction Id	Transaction Id of the downloader job

Field Name	Description
Interface Module	Name of the interface module (for example Diff_Fnd)
Source System	Name of the source system (for example RMS)
Data Set Type	Type of data set (FULL or PARTIAL)
Source Sys Data Set Ready Time	Time when source system moved data set to outbound tables
Data Set Ready Time	Time when uploader job uploaded data set to inbound tables
File Merge Level	Merge level of the file (NO_MERGE, MERGE_TO_PARTITION_LEVEL, MERGE_TO_INTERFACE_LEVEL)
Status	Status of uploader job (COMPLETED or FAILED)

Importer Data Control

Field Name	Description
Interface Short Name	Name of the interface
Begin Sequence Number	Begin sequence number of data set in the inbound table
End Sequence Number	End sequence number of data set in the inbound table
Data Partition	Number of partitions used by uploader job
Thread	Number of threads used by uploader job
Merge Strategy	Merge strategy used for merging files
Auto Purge Data	Flag that indicates whether files need to be cleaned up or not

Manage Configurations

This tab allows you to view and edit configurations for the BDI jobs, and it also allows the user to view, edit and create System Options.

Outbound Interface Controls

Figure 4–10 Outbound Interface Controls

The screenshot shows the Oracle BDI RMS Batch Job Admin interface. The top navigation bar includes 'Batch Summary', 'Manage Batch Jobs', 'Trace Data', 'Diagnostics', 'Manage Configurations', and 'System Logs'. The 'Outbound Interface Controls' tab is selected, showing a table with the following data:

Interface Module	Component Type	Status
CodeDetail_Fnd	DOWNLOADER	Active
CodeHead_Fnd	DOWNLOADER	Active
DeliverySlot_Fnd	DOWNLOADER	Active
Diff_Fnd	DOWNLOADER	Active
DiffGrp_Fnd	DOWNLOADER	Active
InvAvailStore_Tx	DOWNLOADER	Active
InvAvailWh_Tx	DOWNLOADER	Active
ItemHdr_Fnd	DOWNLOADER	Active
ItemImage_Fnd	DOWNLOADER	Active
ItemLoc_Fnd	DOWNLOADER	Active
ItemSupplier_Fnd	DOWNLOADER	Active

Below the table, there is a section for 'Downloader Transmitter Options' with a 'View/Edit Downloader Transmitter Options' button.

This tab allows the user to manage the outbound interfaces and downloader and transmitter options for BDI jobs. The user with Admin privileges can edit the configurations.

Inbound Interface Controls

Figure 4–11 Inbound Interface Controls

The screenshot shows the Oracle BDI RXM Batch Job Admin interface. The top navigation bar includes 'Batch Summary', 'Manage Batch Jobs', 'Trace Data', 'Diagnostics', 'Manage Configurations', and 'System Logs'. The 'Inbound Interface Controls' tab is selected, showing a table with the following data:

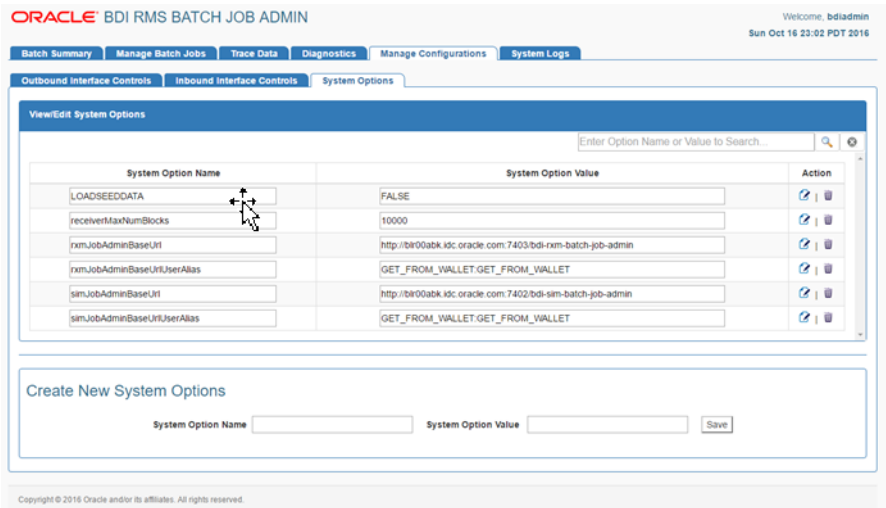
Interface Module	Component Type	Status
Diff_Fnd	UPLOADER	Active
DiffGrp_Fnd	UPLOADER	Active
InvAvailWh_Tx	UPLOADER	Active
ItemHdr_Fnd	UPLOADER	Active
ItemImage_Fnd	UPLOADER	Active
ItemLoc_Fnd	UPLOADER	Active
MerchHier_Fnd	UPLOADER	Active
OrgHier_Fnd	UPLOADER	Active
RelatedItem_Fnd	UPLOADER	Active
Store_Fnd	UPLOADER	Active
StoreAddr_Fnd	UPLOADER	Active

Below the table, there are sections for 'Receiver Options' and 'Uploader Options', each with a 'View/Edit Receiver Options' button.

This tab allows the user to manage the inbound interfaces, receiver and uploader options for BDI jobs. The user with Admin privileges can edit the configurations.

System Options

Figure 4–12 System Logs



This tab allows the user to view, edit and create system options. This page displays the list of system options of the application. The user can modify the value of the existing system options, create new system options and delete the existing system options. The user need admin privileges for editing and creating system options. The Search option based on system options name and value is also provided on this page.

Job Admin Troubleshooting

This section describes the job admin errors and its troubleshooting.

BDI apps deployment Error

Issue:

Bdi Job Admin deployment can run into this error if database credentials are invalid:

```
Caught: javax.management.RuntimeMBeanException:
java.lang.RuntimeException:
weblogic.management.provider.EditFailedException:
java.lang.NullPointerException

javax.management.RuntimeMBeanException:
java.lang.RuntimeException:
weblogic.management.provider.EditFailedException:
java.lang.NullPointerException

    at weblogic.utils.StackTraceDisabled.unknownMethod()

Caused by: java.lang.RuntimeException:
weblogic.management.provider.EditFailedException:
java.lang.NullPointerException

    ... 1 more

Caused by: weblogic.management.provider.EditFailedException:
java.lang.NullPointerException

    ... 1 more
```


Caused by: java.lang.NullPointerException

... 1 more

Solution:

Undo all changes in the Weblogic domain session. Redeploy app with setting up new credentials and verify deployment is successful.

BDI Job Admin runtime WSMEException

Issue:

Log files contain this exception:

```
oracle.wsm.common.sdk.WSMEException: WSM-07620 : Agent cannot enforce policies due to either failure in retrieving polices or error in validations, detail= "WSM-02557 The documents required to configure the Oracle Web Services Manager runtime have not been retrieved from the Policy Manager application (wsm-pm), possibly because the application is not running or has not been deployed in the environment. The query "&(@appliesTo~="REST-CLIENT()")(policysets:global:*)" is queued for later retrieval.
```

Solution:

Follow BDI Installation guide, and verify WSM- policy manager is configured for admin server URL.

Open weblogic domain console and Target wsm-pm app to Admin Server.

Bounce Admin server and verify wsm-pm app is in Active State.

REST Service from SOAP UI for Downloader and Transporter job

Issue:

Diff_Fnd_DownloaderAndTransporterJob is successful, Job status is "completed" but data not transferred from outbound to inbound table and .csv file not created

Rest call to DownloaderAndTransporterJob is successful, Job status is "completed" but data not transferred from the outbound to inbound table and .csv file not created

Solution:

1. Verify the receiverEndpointUrl for the Table DownloaderTransmitterOptions is updated to point to where receiver app (for eg: RXM) is deployed in my case 'blr00abi.idc.oracle.com:7001' in bdi_rms_seed_data.sql.
2. Verify the values in the Interface table DownloaderInterfaceDataControl such as begin and end sequence number matches with the values mentioned in bdi_rms_seed_data.sql.
3. Verify the values in the interface table in DB DownloaderTransmitterOptions, the receiverEndpointUrl is updated to match with bdi_rms_seed_data.sql.

BDI Job Admin not able to find UploaderJob.xml file

Issue:

BDI App B (SIM) Job Admin GUI is showing this exception:

Caused By: java.lang.RuntimeException: Could not find jobName(OrgHier_Fnd_UploaderJob) xml file. You may have renamed the job file or your job repository has more jobs than your application. To resolve the issue either delete the job repository or add the correct job xml file to the app.

Managed server log contains:

Truncated. see log file for complete stacktrace

Caused By: java.lang.RuntimeException: Could not find jobName(OrgHier_Fnd_UploaderJob) xml file. You may have renamed the job file or your job repository has more jobs than your application. To resolve the issue either delete the job repository or add the correct job xml file to the app.

at
com.oracle.retail.bdi.batch.job.operator.JobOperatorServiceBean.
allAvailableBatchJobs(JobOperatorServiceBean.java:167)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)

at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessor
Impl.java:62)

at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethod
AccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:498)

Truncated. see log file for complete stacktrace

Solution:

The process flow has changed and part or all of a flow has been removed, but batch-db has not been updated to match. Either log in to the database and delete references to the job from all tables, or recreate the batch-db using RCU and redeploy BDI.

Job Fails and Job Admin Log Files Contain No Details of the Failure

Issue:

A job fails and the Job Admin log files contain no evidence of or details about the failure.

Solution:

Take a look at the WebLogic Server log files to identify the root cause of the job failure. One example of this is improper data source configuration.

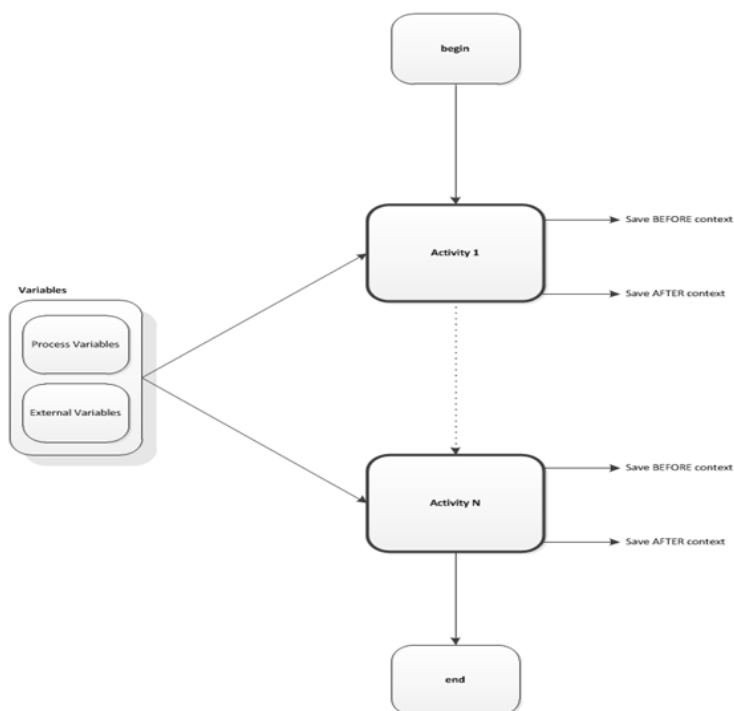
Process Flow

A process flow is a composition of one or more activities. It is written in a DSL script that contains all the activities that make a data flow from source to destination complete.

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.

A process flow encapsulates a sequence of activities. An activity can be synchronous or asynchronous. In BDI some of these activities are invocations of batch jobs.

Figure 5-1 *Process Flow*



Process Flow

This section describes the process flow definitions.

DSL (Domain Specific Language)

Process flow definition is specified in a Domain Specific Language (DSL) built on the top of Groovy. Since Groovy is built on the top of Java Virtual Machine (JVM) Groovy can understand Java and Groovy language constructs. Hence the process flow DSL can understand the DSL, Groovy and Java language constructs. A process is a list of activities. "begin", "end" and "activity" are the main DSL keywords used in process flow definition. These are described in detail below.

Begin Activity

The "begin" activity in process flow definition appears as the first activity. There should be only one "begin" activity. The out of the box process flows may not contain any executable statements in this activity. This activity is intended to be the one used for any initialization needed for the process flow.

Activity

Activity has two parts. Name and Action. Name attribute is mandatory and should be used to name the activity.

The Action section is where the executable code should reside. Any Groovy or Java code can be coded in this section.

There can be one or more Activities in a process.

End Activity

The "end" activity in the process flow definition appears as the last activity. There should be only one "end" activity. The out-of-the-box process flows may not contain any executable statements in this activity. This activity is intended to be the one used for any finalization needed for the process flow.

Process Variables

Variables used between activities can be created and stored in the processVariables map. The process engine also uses some of the variables for its own working in the process variable map. These variables are prefixed with "bdi_internal_". These variables must not be modified inside any custom code.

Here is how you can use the process variable map for your own use.

```
// Set Variable
processVariables["VariableName"] = "Some Value"
// Use a variable value
def anotherVariable = processVariables["VariableName"]
```

External Variables

Some of the system level configuration values are available in the externalVariables map. These values are read-only. The process flow DSL can use these values, but should not attempt to change it.

```
For example,
externalVariables["rxmJobAdminBaseUrlUserAlias"]
```

Statuses

Each activity instance and the process instance maintain the status of execution in the process schema. The following are the possible values for Activities and Process.

At the "begin" activity, the process is marked as PROCESS_STARTED. If any activity fails, the process is marked as PROCESS_FAILED. After the "end" action is completed, the process is marked PROCESS_COMPLETED.

Similar to process statuses, each activity has also a status. There values are ACTIVITY_STARTED, ACTIVITY_FAILED, ACTIVITY_COMPLETED. All the runtime status are persisted in the process schema at runtime when the DSL is executed.

Process Flow Instrumentation

When the process engine executes the process flow, the before and after snapshots of the Activity is recorded in the process schema.

The information is reported through Process Flow Monitor application. This is useful for tracking the process flows as well as troubleshooting. The snapshots also help in case of restarting a failed process. From the schema, the process engine can recreate the context to execute a restart and can resume execution from the activity that failed in the previous run.

Process Flow Monitor Web Application

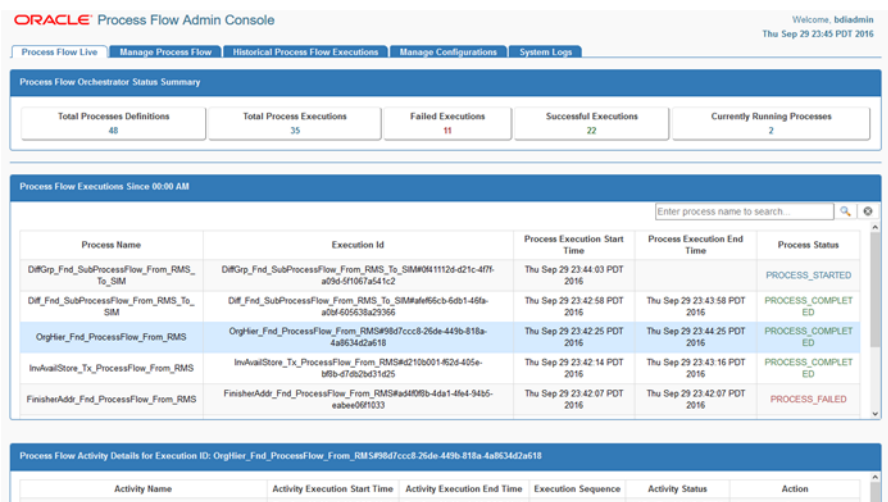
Process Flow (Admin UI) is a web user interface provided by Process Flow where users can view and execute processes, including managing, updating process flow, manually running processes, viewing process executions and process flow logs.

The following describes various functions available in the Process Flow UI in the current release.

Note: It is recommended to use the Chrome web browser to access Process Flow UI since the calendar widget for datetime fields are supported by Chrome browser and not by Firefox or IE as of now.

Process Flow Live tab

Figure 5–2 Process Flow Live Tab

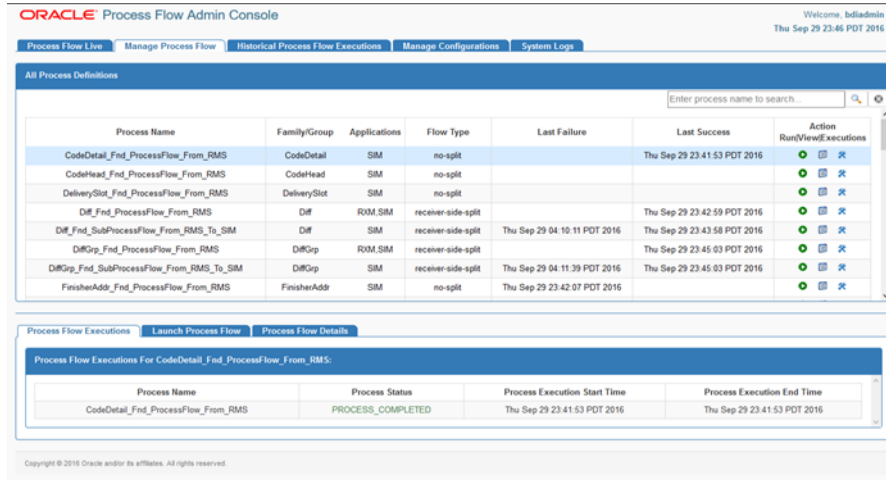


The Process Flow Live tab shows the details of the currently running processes. The first section shows the summary of all processes running in the system. The next section shows the list of all processes running since midnight. The last section shows

the activity details of the selected process. Users also have the option to search for a process by its name.

Manage Process Flow Tab

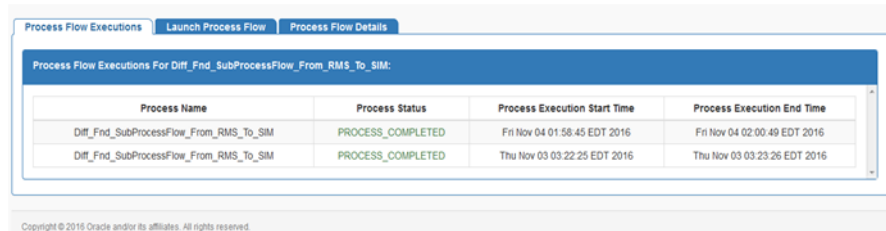
Figure 5–3 Manage Process Flow Tab



The Manage Process Flow tab allows the user to Start a process flow, Restart a failed process flow, View/Edit a process flow, List the executions instances of a process flow. User can search process details on this tab. A failed process flow instance can be restarted only if it is the latest failed instance and there are no successful executions after that. A process flow can be edited only by a user with Admin privileges.

Process Flow Executions

Figure 5–4 Process Flow Executions



This tab shows the executions of the selected process. It can be used to restart the failed executions of a process. The **Restart** button is available only for restartable executions in the status column. When the user clicks the restart button it is redirected to the process launch tab.

Launch Process Flow

Figure 5–5 Launch Process Flow

This tab can be used to launch the selected process. Process Parameters is an optional input from the user to launch the process. Process parameter acts as query parameter and refers to a key value pair. Multiple process parameters can be entered in comma separated value format.

Process Flow Details

Figure 5–6 Process Flow Details

```

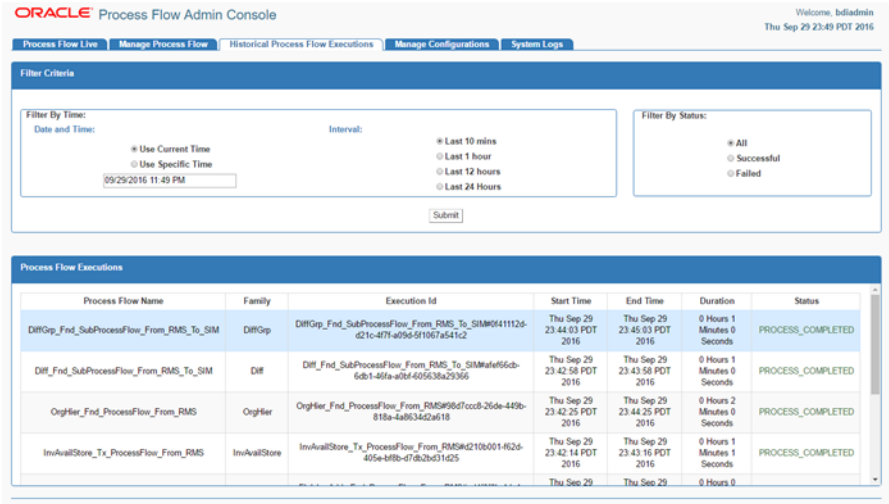
//DO NOT DELETE THE BELOW COMMENT.
//destinationSystem="SIM"
process {
  name "CodeDetail_Fnd_ProcessFlow_From_RMS"
  var ([processFlowType:"no-split"])
  begin{
    action{
      "okay"
    }
    on "okay" moveTo "CodeDetail_Fnd_ExtractorActivity"
  }
}

```

This tab shows process definition in form of a DSL file configured during deployment of the selected process. The Admin user also has the option to modify process DSL. Once updated the process DSL from the UI, changes will take into effect into the BDI_PROCESS_DEFINITION table and no need for process redeployment.

Historical Process Flow Executions Tab

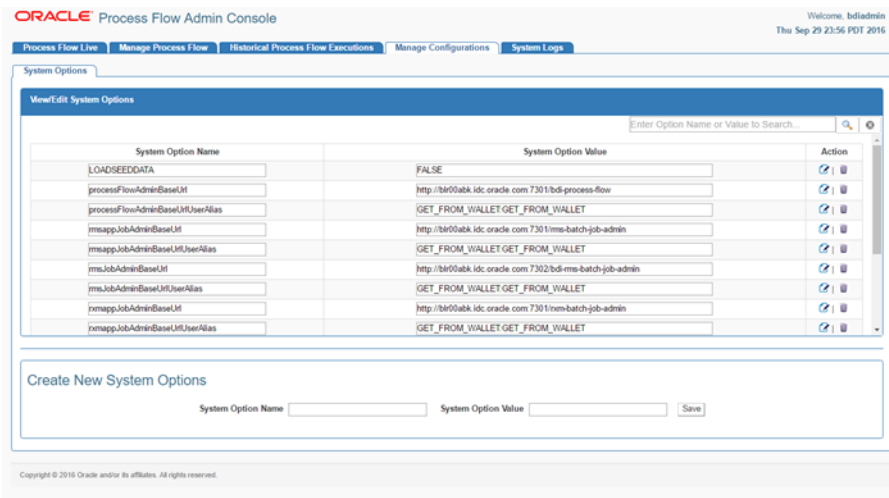
Figure 5–7 Historical Process Flow Executions Tab



The Historical Process Flow Execution tab allows the user to look at the history of process flow executions. The user can specify a date, a time interval and process status. The application will list all the process flow executions matching the criteria. The User can select any of the flow to see the activities details of that execution instance. The page also provides the option to view the before and after values of all process variables for each activity.

Manage Configurations Tab

Figure 5–8 Manage Configurations Tab



The Manage Configurations tab allows users to view, edit and create system options. This page displays the list of system options of the application. The User can modify the value of the existing system options, create new system options and delete the existing system options. The User needs admin privileges for editing and creating system options. The Search option based on the system options name and value is also provided on this page.

System Logs Tab

Figure 5–9 System Logs Tab

Process Log Files

File Name	Size (in KB)	Last Modified
DIRGrp_Fnd_ProcessFlow_From_RMS-system log	147.06	Thu Sep 29 23:45:03 PDT 2016
DIRGrp_Fnd_SubProcessFlow_From_RMS_To_SIM-system log	138.04	Thu Sep 29 23:45:03 PDT 2016
OrgGrp_Fnd_ProcessFlow_From_RMS-system log	208.42	Thu Sep 29 23:44:25 PDT 2016
DIR_Fnd_SubProcessFlow_From_RMS_To_SIM-system log	185.97	Thu Sep 29 23:43:58 PDT 2016
InvAvailStore_Tx_ProcessFlow_From_RMS-system log	72.87	Thu Sep 29 23:43:16 PDT 2016
Dir_Fnd_ProcessFlow_From_RMS-system log	342.32	Thu Sep 29 23:42:59 PDT 2016
FinanceAddr_Fnd_ProcessFlow_From_RMS-system log	216.86	Thu Sep 29 23:42:07 PDT 2016
CodeDetail_Fnd_ProcessFlow_From_RMS-system log	1210.76	Thu Sep 29 23:41:53 PDT 2016
Store_Fnd_ProcessFlow_From_RMS-system log	33.63	Thu Sep 29 04:06:03 PDT 2016
Other_End_OrAProcessFlow_Exec_ORM_Tx_ORM-system log	43.63	Thu Sep 29 04:06:03 PDT 2016

File Content

```
2016-09-27T21:18:19,783 [ACTIVE] ExecuteThread: '19' for queue: 'weblogic.kernel.Default (self-tuning)'] INFO ProcessExecutorServiceBean - Starting new process(DiffGrp_Fnd_ProcessFlow_From_RMS) appName(bdi-process-flow-16.0.0.user).
2016-09-27T21:18:19,813 [ACTIVE] ExecuteThread: '19' for queue: 'weblogic.kernel.Default (self-tuning)'] INFO ProcessExecutorServiceBean - Finished new process(DiffGrp_Fnd_ProcessFlow_From_RMS) appName(bdi-process-flow-16.0.0.user).
2016-09-27T21:18:19,997 [Thread-166] DEBUG GeneratedMethodAccessor4255 - processName(DiffGrp_Fnd_ProcessFlow_From_RMS).
2016-09-27T21:18:19,998 [Thread-166] DEBUG GeneratedMethodAccessor4255 - processOrLabName([jobExecutionId, processFlowType:receiver-side-split]).
2016-09-27T21:18:19,999 [Thread-166] DEBUG Logger$debug - checkBeginCalled@firstOnlyOnce.
2016-09-27T21:18:20,000 [Thread-166] DEBUG Logger$debug - *****
2016-09-27T21:18:20,000 [Thread-166] DEBUG GeneratedMethodAccessor4255 - begin : start
2016-09-27T21:18:20,010 [Thread-166] DEBUG GeneratedMethodAccessor4255 - begin : will run.
2016-09-27T21:18:20,060 [Thread-166] DEBUG GeneratedMethodAccessor4255 - begin executed with return value(okay).
2016-09-27T21:18:20,060 [Thread-166] DEBUG Logger$debug - begin : test result true
```

The System Logs tab shows all the log files created by the process flow execution. Clicking on the View icon will show the log file contents in the screen.

Process Flow Notification Feature

The Process Flow notification options can be set in the System Options of the Process Flow. This can be done either at deployment time (through seed data) or at runtime (through the Manage Configuration tab of the Process Flow Monitoring application)

The options available for notification are:

- processFlowNotification.<scope>.enable - value must be True or false. This is for global enabling or disabling of process flow notification.
- processFlowNotification.<scope>.onStart - value must be True or false. True means notification will be sent at the start of the process.
- processFlowNotification.<scope>.onRestart - value must be True or false. True means notification will be sent at the restart of the process.
- processFlowNotification.<scope>.onCompletion - value must be True or false. True means notification will be sent at the completion of the process.
- processFlowNotification.<scope>.onFailure - value must be True or false. True means notification will be sent when the process fails.
- processFlowNotification.<scope>.recipients - list of recipient email ids
- processFlowNotification.<scope>.subject – Template of the email subject line
- processFlowNotification.<scope>.content – template of email content

where <scope> value is global or the Process Name.

If Process Name is specified, the global notification option is ignored for that process. For Subject and Content, if nothing is specified either at the global or process scope, an internal default format is used.

If Mail Session is not setup in WebLogic, notifications will not be sent. If processFlowNotification.<scope>.recipients is not set, the value from mail.to property in the WebLogic Mail Session is used.

For Subject and Content template, following variables can be used. The variable is case sensitive and the format must match exactly as given below. For multi-line content, \n can be used to indicate line breaks.

`${processUrl}`

`${processName}`

`${processExecutionId}`

`${processStartTime}`

`${processEndTime}`

`${processStatus}`

Process Restart

When the activities within a process flow fail, the process status is marked as failed. A failed process flow can be restarted. If there are multiple failed processes, only the latest failed instance can be restarted.

Note: Restart is for an already run and failed instance. This is different from running a new instance of the process flow.

When a process flow is restarted, the system knows the activity that failed in the previous run. During restart, the process engine will skip all the activities prior to the failed activity. It will restore the context for the activity and resume execution at the failed activity.

Process flow execution does not keep the activity history at restart. It will overwrite the activity records on restart.

Process Security

The Process Flow Application uses basic authentication to access the system. The user must belong to the BdiProcessAdminGroup or BdiProcessOperatorGroup or BdiProcessMonitorGroup to use the process flow REST services and process flow admin application.

There are three authorization roles designed for process flow application; Admin Role, Operator Role and Monitor Role. The Admin role has permissions to use all the functions provided by the process flow application. The Operator Role has limited access compared to Admin. The Monitor role has the least access permissions from all roles, as identified in the table below.

Service/Action	Monitor Role	Operator Role	Admin Role
Update Process DSL	No	No	Yes
Start/Restart Process	No	Yes	Yes
All other services	Yes	Yes	Yes

Customizing Process Flows

This section describes the customizing process flows.

Process Flow DSL

The Process Flow is written in a custom DSL for process. This DSL allows a limited set of keywords to define a process. These keywords are identified in the table below. The execution section (Action keyword) can be written in Groovy or Java, since the DSL is developed on the top of Groovy.

Keyword	Description
process	Identifies the process flow. Only one keyword in a process flow.
name	Used for naming processes and activities
var	Used for initializing process variables
begin	A special activity that occurs at the beginning of the process execution. Only one begin activity per process flow
action	The main executable section of the Activity. The body of Action can be in Groovy or Java
on .. moveTo ..	Jump to a specific activity on matching the condition.
activity	The executable component of the process flow. A process flow is composed of many activities.
end	A special activity that occurs at the end of the process execution. Only one begin activity per process flow

APIs

The process flow engine also provides a few APIs specific to BDI batch jobs. The DSL writers can use these in the activity section of the script.

startOrRestartJob (base URL, Job Name, Credentials, jobParameters)

This API is used to start (or restart) a Retail Batch job. If the job has failed in the previous run of the process, the job will be restarted.

When an activity in a process fails, the activity name is persisted. On the subsequent restart of the process, all the activities ahead of the failed activity are skipped.

The execution begins at the failed activity. If the activity is calling a Retail Batch job, it will call the "restart", instead of new start, in this case. The batch job will then resume execution from the previously failed position.

```
startOrRestartJob(
externalVariables["rxmJobAdminBaseUrl"],
"WhAddr_Fnd_UploaderJob",
externalVariables["rxmJobAdminBaseUrlUserAlias"],processVariables["processDataSetI
d")
```

waitForJobCompletedOrFailed (Activity Name, URL, Credentials)

This API will cause the process instance to wait until the job it is waiting for returns a FAILED or COMPLETED status.

If the job does not return either of the statuses, this API will wait until JTA timeout set in the server.

If the job returns a FAILED status, the activity whose name is specified in the second parameter, will be marked as failed.

For all other errors, including system exceptions, the activity containing the `waitForJobCompletedOrFailed` API will be marked as failed.

```
waitForJobCompletedOrFailed(
  "ItemSuppUom_Fnd_UploaderActivity",
  externalVariables["externalJobAdminBaseUrl"]
+ "/resources/batch/jobs/ItemSuppUom_Fnd_UploaderJob/"
+ processVariables["jobExecutionId"],
  externalVariables["externalJobAdminBaseUrlUserAlias"])
```

How to modify a Process Flow

A process flow can be modified at deployment time. At deployment through the Process Flow Admin app the flow files that come with the application are in the `setup-files/dsl/available_process_flow_options` folder. These files have an extension ".flo". The user can edit these files in any text editor.

After editing the file save the file to the `setup-files/dsl/flows-in-scope` folder. The deployment script will take the process flow file and save in the process flow schema.

After deployment, the process flow can be edited by the Admin user through the Process Flow Admin application. The changes will be picked up at the next run.

It is recommended to make any permanent changes at deployment time, since the change through the Admin App may get overwritten at redeployment.

Note: For security reasons, usage of certain keywords are not allowed in the Process Flow DSL. When defining the process action in the process flow UI, any such forbidden keywords if used will prevent the process from being created or updated. A process cannot be saved or run if such keyword is present in the process action definition.

Sub Processes

In multi-destination process flows, one process may invoke one or more processes asynchronously. All the processes may run at the same time.

In order to identify these subprocesses they are named accordingly. Once invoked, the main process has no control over the sub processes. Each of the process will run in the same way as they are invoked independently.

Process Schema

The process instrumentation captures the state of the process at the beginning and end of each activity. This information is persisted into the process schema. For each activity there will be two records, one for before activity and the other for after activity. The schema details are in the Appendix B.

Table Name	Description
BDI_PROCESS_DEFINITION	This table stores all the process flow definitions. It is loaded at deployment time.
BDI_PROCESS_EXEC_INSTANCE	This table tracks all the process flow executions. There is a row for each process flow execution.

Table Name	Description
BDI_ACTIVITY_EXEC_INSTANCE	This table tracks all the activity executions. There are 2 rows for each activity execution. One to store the before context and one to store after context
BDI_SYSTEM_OPTIONS	This table has all the system level information such as URLs, credential aliases and so on. It is loaded at deployment time.

REST Interface

Process Flow services are exposed as REST endpoints for the use of other applications. The list of REST endpoints are given in the Appendix C

Troubleshooting

Since the process flow can be written in Groovy and DSL, it is prone to programmer's mistakes. Any custom DSL must be properly tested before deploying. At present, the process flow engine can detect syntax errors only at runtime. So it is possible to load an incorrect process flow and fail during runtime.

At the end of an activity, the process engine invokes the next activity depending on the result of activity execution (The "moveTo" statement). If you have empty activities (possibly because you commented out the existing invocation statements), make sure the activity result is valid (for example, "okay")

If any activity fails, the process is marked as failed. So in case of process failure, look at the activity details to find out which activity failed. Once the failed activity is identified, the process variables can be inspected to look for any issues. Next step would be to look at the logs, through the Process Flow Monitor application to see the details of the issue. Once the issue is fixed, either restart or a new run of the process flow can be used depending on the requirement.

BDI Process flow runtime XML UnmarshalException

Error

BDI Process Flow fails and GUI is showing this exception:

Runtime Process Flow exception

```
[Thread-55] ERROR Logger$error$0 - Error calling activity.
javax.ws.rs.ProcessingException: Unable to unmarshall json
object to java object.
```

OR

Caused by: javax.xml.bind.UnmarshalException - with linked
exception:

```
[Exception [EclipseLink-25004] (Eclipse Persistence Services -
2.6.1.v20150916-55dc7c3):
org.eclipse.persistence.exceptions.XMLMarshalException.
```

Exception Description: An error occurred unmarshalling the
document

```
Internal Exception: javax.json.stream.JsonParsingException:
Unexpected char 73 at (line no=1, column no=1, offset=0)]
```

Reason

Process flow deployed with wrong credentials for apps.

Solution

Delete existing process flow deployment from weblogic domain. Redeploy process flow with setting up new credentials.

BDI Process flow stuck in running state

Issue:

BDI Process Flow keeps in running status and does not end with failed or completed state. This even does not allow to cancel an existing running process or start a new process.

Reason:

This happens because of default JTA Timeout in domain configuration, and resource connections not able to timeout. There are instructions in BDI installation guide "How to Set JTA Timeout".

Resolution:

Follow the instructions in the BDI Implementation guide and set JTA timeout. Redeploy the processflow app to stop the running flow and rebound the server.

BDI Scheduler

The Scheduler application in the Bulk Data Integration (BDI) product suite assists in scheduling of batch processes to run at predefined configured intervals of times. A schedule determines when a job or a process or any program needs to be executed and the frequency of execution.

The Scheduler application runtime is based on the container-managed Java EE timer service to execute the schedules and utilizes the Oracle WebLogic Server's implementation and management of the timer service when deployed on the WebLogic server.

The Scheduler supports various schedules ranging from simple interval schedules such as hourly, daily, and so on, to advanced cron-like scheduling.

Scheduler currently supports calling of REST services. The application out-of-the-box contains schedules to run BDI Process Flows in scope for this release.

The Scheduler Console (Admin UI) enables runtime monitoring and administration of schedules where the user can view, create, edit, delete schedules, manually run a schedule, enable/disable schedule, set up notifications for schedules and so on.

Scheduler Core Concepts

This section describes the scheduler core concepts.

Schedule Definition

A schedule definition comprises details of a schedule such as Schedule Name, and Schedule Group which indicates logical or functional grouping of schedules, and Schedule Description.

Schedule Execution

A schedule execution is an instance of scheduled run of a schedule at the specified frequency.

Schedule Types

A schedule can be an interval-based schedule or calendar-based schedule.

Interval Schedules

An interval-based schedule is a schedule that repeats at fixed interval of time starting from a specific time. For example, hourly, daily, weekly, every 5 minutes and so on.

Calendar Schedules

A calendar-based schedule is a cron-type of schedule that specifies different times that the schedule runs. More complex schedules that can be specified as cron expression are defined as calendar-based schedules.

The following parameters define a calendar-based schedule, same as the parameters in a cron expression: Minutes, Hours, Day of Week, Day of Month and Month.

Note: The Scheduler does not currently support Seconds and Year parameters in a calendar schedule.

Scheduling Mechanisms

This section describes the scheduling mechanisms.

Simple Scheduling

Simple schedules are predefined schedule frequencies that are available as options for the user to choose readily. The following are the simple schedules that the Scheduler supports.

- Hourly
- Daily
- Weekly
- Monthly
- Weekday (Monday-Friday)
- Weekend (Saturday-Sunday)
- Saturday
- Sunday
- First day of every month
- Last day of every month
- One time only (run once), and
- User-specified frequency with interval in the units of:
minutes, hours, days or weeks

Advanced Scheduling

BDI Scheduler supports advanced scheduling which is cron-like scheduling. Calendar-based schedules that can be expressed in cron-format can be setup with the advanced scheduling capability of the Scheduler. Advanced scheduling is defined with the following parameters (similar to that of cron expression) and the corresponding range of values:

- Minutes : 0-59
- Hours : 0-23 (12:00 a.m. - 11:00 p.m.)
- Day of Week : Monday - Sunday
- Day of Month : 1-31
- Month : 1-12 (January - December)

If a schedule is created with multiple values for the above parameters, then the schedule will repeat at all those specified times.

Schedule Frequency

The schedule frequency defines the frequency at which a schedule has to be repeated at the configured time and interval starting from a given point of time. The schedule frequency has the following parameters that determines when the schedule has to be run.

Schedule Start Datetime

It specifies the start date and time when a particular schedule has to start executing.

For interval based schedules, this is the first time the schedule runs and then repeats based on the specified interval.

For example, a schedule with start datetime as 2016-08-15 10:00a.m. and repeat 'Daily' will first run at 2016-08-15 10:00 a.m. and next run at 2016-08-16 10:00 a.m. and so on.

For calendar schedules (cron schedules), this defines the time from when the schedule will become effective and starts executing based on the frequency. So it is not necessarily the first run of the schedule, though it very well may be.

For example, a schedule with start datetime as 2016-08-15 10:00 a.m. (which is a Monday) but repeat every Thursday, will first run at 2016-08-18 10:00 a.m. (Thursday) and subsequently next run at 2016-08-25 10:00 a.m. (Thursday) and so on.

So the Start Datetime here signifies the datetime the schedule becomes effective and that it will not run before that datetime. However, here the Start Datetime can very well be specified as 2016-08-18 10:00 a.m. (Thursday) and repeat every Thursday.

So in summary, for interval-based schedules, first run of the Schedule equals Schedule Start Datetime. For calendar-based schedules, first run of the Schedule may or may not be equal to the Schedule Start Datetime, based on the schedule recurrence specified.

Schedule End Datetime

It specifies an end date and time when the schedule should stop executing and no longer run. When a schedule has no end datetime specified, it runs indefinitely.

Note: The end datetime is inclusive for the schedule execution, meaning if the schedule recurrence coincides with the end datetime, the schedule will execute at the end datetime and only then does not repeat.

For example, say, Schedule Start Datetime: 2016-08-15 10:00 a.m., repeat 'Hourly', Schedule End Datetime: 2016-08-15 11:00 a.m., then the schedule will run at 10:00 a.m. and also at 11:00 a.m. before ceasing to run.

Recurrence / Repeat Interval

This specifies the frequency at which the schedule repeats. This is same as described in Simple and Advanced Scheduling.

Schedule Next Run Datetime

This indicates the date and time of the next occurrence of the schedule, obtained based on the configured schedule frequency.

Schedule Timezone

All the date and times in the Scheduler are based on the timezone of the server (JVM) where the application is deployed.

The Scheduler Console (UI) displays the server's current date and time with timezone (the current time displayed gets refreshed when the UI is refreshed).

Note: When creating or updating a schedule and in monitoring schedule executions in the Scheduler Console, the date and time are as per the timezone setup in the application server and not the local timezone.

Schedule Action

The Schedule Action defines what is executed when the schedule runs at the specified frequency. It is a DSL (domain specific language) that is based on Groovy. The schedule action has a simple syntax as follows.

```
action {
//Define what needs to be executed here. Say invoke a REST service.
}
```

Currently Schedule Action supports calling REST services. BDI process flows are called by the Scheduler as REST services.

For example, to trigger a BDI process flow named Store_Fnd_ProcessFlow_From_RMS, the following schedule action is defined:

```
action {

(POST[externalVariables.processFlowAdminBaseUrl +
"/resources/batch/processes/operator/Store_Fnd_ProcessFlow_From_
RMS"]^externalVariables.processFlowAdminBaseUrlUserAlias) as String

}
```

- POST denotes the REST method.
- processFlowAdminBaseUrl is an entry key in 'externalVariables' map variable used by the Scheduler runtime and specifies the BDI Process Flow Admin's base URL. The value for processFlowAdminBaseUrl is specified during install time and gets stored in the BDI System Options. The value of processFlowAdminBaseUrl will be like `https://<host>:<port>/bdi-process-flow`.
 - For example, `https://example.com:8001/bdi-process-flow`
 - The value for processFlowAdminBaseUrl is specified during install time and gets stored in the BDI System Options.
- `/resources/batch/processes/operator/Store_Fnd_ProcessFlow_From_RMS` is the relative REST URL to call the process flow.
 - It is of the form `/resources/batch/processes/operator/<process flow name>`.
- processFlowAdminBaseUrlUserAlias is an entry key in 'externalVariables' map variable used by the Scheduler runtime and specifies the alias name for BDI Process Flow Admin's user credentials to access the process flow REST service.
 - The value for processFlowAdminBaseUrlUserAlias is specified during install time and gets stored in the BDI System Options.

Basic authentication is used to access the BDI process flows. The Scheduler uses `processFlowAdminBaseUrlUserAlias` to lookup the credentials in the runtime secure wallet where the credentials specified at install-time are stored.

Scheduler by itself does not manage executions of process flows called from within the schedule action and any dependencies associated thereof. The Scheduler only triggers process flows. The execution of process flows is done by the Process Flow engine.

For any dependencies between execution of process flows to be managed, it is recommended that such dependencies are defined in the BDI Process Flow Admin and not in the Schedule Action. For example, if `process-flow-2` needs to be run after `process-flow-1` completes, use Process Flow Admin to define this dependency and not in the Schedule Action.

It is recommended to avoid time based dependency management in execution of process flows from within the Scheduler, but rather use Process Flow Admin to coordinate such dependency execution requirements.

Note: For security reasons, usage of certain keywords are not allowed in the Schedule Action DSL. When defining the schedule action in the Scheduler UI, any such forbidden keywords if used will prevent the schedule from being created or updated. A schedule cannot be run if such keywords are present in the schedule action definition.

Schedule Action Type

There are two types of Schedule Action - Sync and Async. When creating a schedule and defining a schedule action, the user needs to specify whether the schedule action is sync or async. The Scheduler determines the action execution statuses according to the action type specified.

Sync Action

Executes synchronously and returns a result after its successful or failed completion (however long the action may run).

Async Action

The action is asynchronous and returns a response immediately when triggered, but will continue to execute. The actual process completes at a later time. The end result of the action is not known to Scheduler in this case.

Schedule Action Execution Status

Indicates the status of execution of the schedule action when the schedule has run at the configured frequency of time.

A schedule execution can be in one of the following statuses depending upon the Schedule Action Type and its execution.

- Triggered (applicable only for 'Async' action)
- Started (applicable only for 'Sync' action)
- Failed (applicable for both 'Async' and 'Sync' actions)

Schedule Action Type and Execution Status

The Schedule action type determines the schedule action status during the execution lifecycle.

Sync Action Execution Statuses

Executes synchronously and returns a result after it's successful or failed completion (however long the action may run).

- When sync action starts, the Schedule Action status will be marked 'STARTED'.
- When the action completes and returns a successful result, the status will be marked 'COMPLETED'.
- When the action does not complete because of an exception or returns a failed response (return value = "FAILED"), then the status will be marked 'FAILED'.

Async Action Execution Statuses

The schedule action status will only be 'TRIGGERED' when the Scheduler successfully invokes the schedule action.

In case there is an exception in invoking the action itself, then the status is 'FAILED'.

By default, all BDI process flows are asynchronous that return an execution Id when triggered, but continue to run to invoke the batch jobs that complete at a later time.

How the Action Execution Statuses are determined

- The Scheduler marks the Action Execution Status as 'FAILED' when there is an exception in executing the action or when an exception is thrown from the schedule action. In order for the Scheduler to mark the execution of the schedule action as 'FAILED' when the action has been executed, the action should either throw an exception or return a value as 'FAILED'.
- If the schedule action returns null or any other return value, the action execution status will be marked 'TRIGGERED' for async action and 'COMPLETED' for sync action and the returned response is stored as such in the Schedule Action Execution Log.

Schedule Status

A schedule can be in one of the following statuses:

- Active: An Active schedule indicates that the schedule is running at the specified frequency.
- Inactive: An Inactive schedule indicates that the schedule has reached its end datetime and no longer runs.
- Disabled: A Disabled schedule indicates that the user has disabled the schedule to not run at its specified frequency.

Scheduler Runtime

This section describes the scheduler runtime.

Scheduler Startup

As the Scheduler is deployed and the application starts up, the Scheduler service performs the following actions:

- Loads the schedules defined in the seed data sql script in the installer. This means schedule definitions are inserted in the corresponding Scheduler infrastructure table.
- Loads the schedule action dsl for each corresponding schedule from the *_Action.sch files in the installer. Each schedule definition in the table is updated to include its corresponding schedule action.
- The Scheduler service sets up the default runtime timers for each schedule.

When the application is deployed first-time, all schedules will be setup new. However, when the application is redeployed or the application server is restarted, the schedule timers that are already created and exist, will not be recreated.

All seed data schedules need to be specified with status as 'Active'. This ensures that the schedule timers are created at startup and the schedules start to run as per the frequency defined.

When a schedule action dsl contains any restricted keyword, the schedule will be 'Disabled' at startup and will not run. The User has to correct the schedule action definition from the Scheduler UI and enable the schedule to make it active.

Schedule Runtime Execution

The Scheduler uses the application server's implementation of Java EE compliant timer service to execute the schedules at runtime. When a schedule is created, Scheduler sets up a timer in the application server based on the schedule frequency configured. At each scheduled time, the application server invokes a callback method where the Scheduler will execute the schedule action.

Each schedule timer executes in a separate thread, so schedule executions do not block each other. Each schedule execution itself is run synchronously in its own thread, that is, the execution is blocked until it completes. But the schedule action can be specified to be asynchronous (Async action) or synchronous (Sync action) based on the action dsl defined for the schedule.

It is appropriate to specify a schedule action as 'async' when all the service calls made within the schedule action are non-blocking asynchronous calls and the action defined runs in different thread from that of the Scheduler.

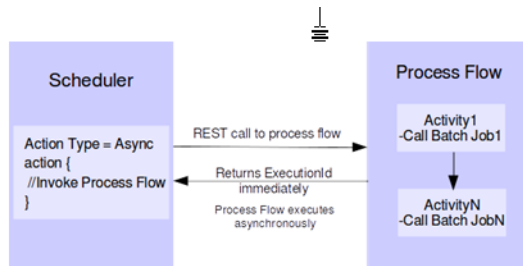
If any of the service calls within the schedule action is a blocking synchronous call and the action is not defined to run in a separate thread, then the action type should be 'sync'.

Specifying the schedule action type 'async' or 'sync' based on the action dsl definition determines the runtime execution behavior and statuses of the schedule execution. This is explained below.

Schedule Execution - BDI Process Flows

BDI process flow invocations (REST service calls) are asynchronous by default and the corresponding schedule actions are specified 'Async'.

Figure 6–1 Schedule Execution - BDI Process Flow

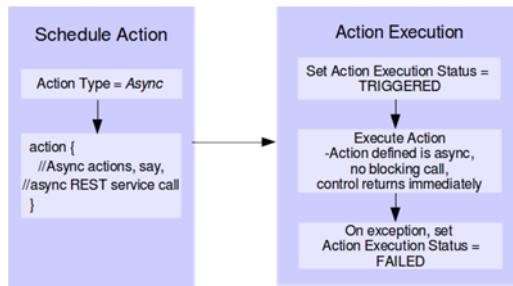


The Process Flow does not block the calling thread (that is, the Scheduler) and returns immediately. The Process flow returns an Execution ID of the process execution instance as a future handle, but will continue to execute the activities defined in the DSL. The activities in BDI process flows may run for longer duration. Here, the Scheduler simply acts as a trigger for the process flows at the scheduled frequency of execution.

The actual status of the process flow instance may be completed or failed after its execution, but the status of schedule execution in Scheduler will remain 'TRIGGERED'. The execution of the process flow and the eventual status thereof will not be known to the Scheduler.

Schedule Execution - Async Action

Figure 6–2 Schedule Execution - Async Action

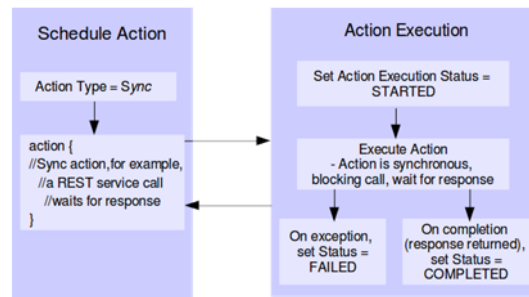


When the schedule action execution starts for async action, the action execution status is set to 'TRIGGERED', and the action is executed. As the action type is specified 'Async', the action should be non blocking, either returning a response immediately or not returning a response and continuing execution, but runs in a separate thread returning the control immediately.

The execution of the action and the eventual status thereof will not be known to the Scheduler. Once the control is returned, the schedule action execution ends but the status remains 'TRIGGERED'. In case of an exception when the action is triggered, the status is set to 'FAILED' and the execution ends.

Schedule Execution - Sync Action

Figure 6–3 Schedule Execution - Sync Action



When the schedule action execution is started for 'sync' action, the action execution status is set to 'STARTED'. As the action type is specified 'sync', the action is blocking and runs in the same thread as the schedule execution.

The schedule execution ends only when the action completes returning a response or throws an exception, thereby releasing the execution thread.

After the schedule action completes successfully, the status is set to 'COMPLETED'. But if the action return value is 'FAILED' or the action returns throwing an exception, the status is set to 'FAILED'.

For sync actions, the action execution status in Scheduler can indicate the actual execution status (either completed or failed) of the process that was executed.

Schedule Execution Failover

All schedule timers created by the Scheduler are persistent. This enables a failover feature that in case of unexpected server shutdown or downtime, the missed schedules will be run once the server is back up. That is, the schedules that should have been run during the downtime, will be run as soon as the server is back up and the application is in running state.

Note: A missed schedule will be run only once, not as many times as it was missed during the downtime. For example, if a schedule is scheduled to run every 5 minutes and the application server is down for 15 minutes and restarted, the schedule will be run only one time and not 3 times. This is a feature supported by the Java EE container.

Schedule Notification

The Scheduler supports email notification of scheduled runs at runtime. The available options of events for notifications on a scheduled run are:

- Notify when the schedule action execution begins
 - This occurs when the schedule action execution is 'Started' for sync action and before triggering of action execution for async action.
- Notify when the schedule action execution ends successfully
 - This occurs when the schedule action execution status is 'Triggered' for async actions and 'Completed' for sync actions.
- Notify when the schedule action execution fails
 - This occurs when the status of schedule action execution is 'Failed', when one of the following occurs: An exception is caught in the Scheduler service itself, when an exception is thrown by the schedule action dsl, when the schedule action dsl returns the string 'FAILED'.

Scheduler Infrastructure Schema

The Scheduler infrastructure relies on the following schema to store the schedule definitions and schedule executions.

Table Name	Description
BDI_SCHEDULE_DEFINITION	This table contains all the schedule definitions created, including schedule frequency, schedule notification information and schedule action dsl for each schedule. Seed data schedules are loaded in this table at deployment time during application startup.
BDI_SCHEDULE_EXECUTION	All schedule executions at runtime are persisted in this table.
BDI_SYSTEM_OPTIONS	This table contains system-level global parameters as key-value pairs used by the Scheduler at runtime, such as, Process Flow Admin Base URL, Process Flow Admin User Alias, which are configured at install time by the user. User can also add system parameters to be made available to the schedule actions.

The Scheduler service captures all schedule executions at runtime and persist the execution instances in the corresponding infrastructure table.

Scheduler REST Services

Scheduler provides certain RESTful services to retrieve information about schedules and run the schedule manually.

All the below REST resources can be accessed by Monitor, Operator and Admin role users, except for the run-schedule-now service that can be accessed by Admin and Operator role users, but not by users with only the monitor role.

REST Resource	Method	Description
/resources/discover	GET	Lists all the available Scheduler REST resources
/batch/schedules	GET	Returns all the schedules in the application (including active, inactive and disabled schedules)
batch/schedules/{scheduleName}	GET	Returns the schedule definition of the specified schedule
/batch/schedules/upcoming-schedules/days/{days}	GET	Returns the upcoming schedules from now to next number of {days} specified
/batch/schedules/upcoming-schedules	GET	Returns the upcoming schedules for the next 1 day from now
/batch/schedules/executions/{scheduleName}	GET	Returns all the historical schedule executions of the given schedule since the beginning
/batch/schedules/executions/past/days/{days}	GET	Returns the historical schedule executions of the given schedule for past number of {days}
/batch/schedules/executions/failed	GET	Returns all the failed executions for all the schedules since the beginning
/batch/schedules/executions/today	GET	Returns today's schedule executions starting from midnight today (12:00 a.m.) to now
/batch/schedules/executions/today/completed	GET	Returns today's schedule executions that are either in 'Triggered' status (for async actions) or in 'Completed' status (for sync actions), starting from midnight today (12:00 a.m.) to now
/batch/schedules/executions/today/failed	GET	Returns today's schedule executions that are in 'Failed' status, starting from midnight today (12:00 a.m.) to now
/batch/schedules/operator/run-schedule-now/{scheduleName}	POST	Runs the specified schedule, that is, executes the Schedule Action of the schedule and returns the Schedule Execution detail response. This is synchronous invocation, so client needs to wait for the response.

Scheduler Console

The Scheduler Console (Admin UI) is a web user interface provided by the Scheduler where users can monitor and manage schedules, including creating, updating, deleting, disabling or enabling schedules, manually running schedules, viewing schedule executions and schedule logs.

The following describes various functions available in Scheduler Console in the current release.

Note: It is recommended to use Chrome web browser to access Scheduler Console since the calendar widget for datetime fields are supported by Chrome browser and not by Firefox or IE as of now.

Schedule Summary

This is the home page that provides the overall summary of the scheduler runtime. It displays the following information.

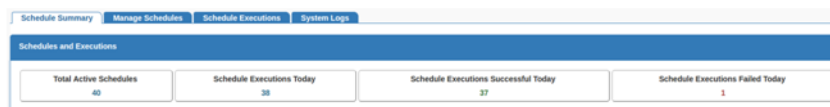
Schedules and Executions

This displays the total count of:

- Active Schedules
- Schedule Executions today
- Schedule Executions that were successful today
- Schedule Executions that failed today

Note: Today here indicates the duration from midnight to now.

Figure 6–4 Schedules and Executions



Upcoming Schedules

Lists the future schedules that are expected to run in the next 24 hours from now.

Figure 6–5 Upcoming Schedules

Schedule ID	Schedule Group	Schedule Name	Schedule Next Run	Schedule Status
1	CodeDetail	CodeDetail_Fnd_From_RMS_Schedule	Sat Aug 20 00:00:00 PDT 2016	ACTIVE
2	CodeHead	CodeHead_Fnd_From_RMS_Schedule	Sat Aug 20 00:05:00 PDT 2016	ACTIVE
3	DeliverySlot	DeliverySlot_Fnd_From_RMS_Schedule	Sat Aug 20 00:10:00 PDT 2016	ACTIVE
4	DIF	DIF_Fnd_From_RMS_Schedule	Sat Aug 20 00:15:00 PDT 2016	ACTIVE
5	DIF	DIFGrp_Fnd_From_RMS_Schedule	Sat Aug 20 00:20:00 PDT 2016	ACTIVE
6	FinisherAddr	FinisherAddr_Fnd_From_RMS_Schedule	Sat Aug 20 00:25:00 PDT 2016	ACTIVE
7	Inventory	InvAvailStore_Tx_From_RMS_Schedule	Sat Aug 20 00:30:00 PDT 2016	ACTIVE
8	Inventory	InvAvailMkt_Tx_From_RMS_Schedule	Sat Aug 20 00:35:00 PDT 2016	ACTIVE
9	Item	ItemHd_Fnd_From_RMS_Schedule	Sat Aug 20 00:40:00 PDT 2016	ACTIVE
10	Item	ItemImage_Fnd_From_RMS_Schedule	Sat Aug 20 00:45:00 PDT 2016	ACTIVE

Schedule Executions Failed Today

Lists the schedule executions that have failed today (from midnight to now).

Figure 6–6 Schedule Executions Failed today

Schedule Execution Id	Schedule Id	Schedule Name	Schedule Execution Datetime	Schedule Action Execution Status	Schedule Action Execution Log
1354	8	InvAvailWh_Tx_From_RMS_Schedule	Wed Aug 31 04:11:40 PDT 2016	FAILED	SCHEDULED RUN: Action triggered at Wed Aug 31 04:11:40 PDT 2016 Action Type: ASYNC Action Status: FAILED Action Response: Exception: HTTP 500 Internal Server Error

Schedule Executions Completed / Triggered Today

Lists the schedule executions that are completed or triggered today (from midnight to now). A status of 'Completed' represents sync actions and the status of 'Triggered' represents async actions.

Schedule Executions In Progress Today

Lists the scheduled executions that were started but have not completed and in progress today (from midnight to now). This is applicable only for sync actions that are in a 'Started' status.

Schedules Past Due

Lists the schedules that failed to run at the scheduled time (that is, schedules whose next run time is before the current time are displayed here). Ideally, there should be no missed schedules unless there may be an internal server issue that the schedule timer failed to run.

Manage Schedules

The Manage Schedules page displays a list of all schedules and details of each schedule in the Schedule Detail view and corresponding schedule executions in the Schedule Executions view for the schedule.

The schedules list provides options to filter schedules based on the Schedule Name, Schedule Group, Schedule Status, Schedule Frequency. There is also an option to filter upcoming schedules based on date range.

The 'Create Schedule' function will be available in this page for admin users.

Figure 6–7 Manage Schedules

Schedule Id	Schedule Name	Schedule Group	Schedule Start	Schedule Frequency	Schedule Next Run	Schedule Status	Schedule End
1	CodeDetail_Fnd_From_RMS_Schedule	CodeDetail	Sat Mar 12 00:00:00 GMT-06:00 2016	Daily	Fri Oct 07 00:00:00 GMT-06:00 2016	Active	Never
2	CodeHead_Fnd_From_RMS_Schedule	CodeHead	Sat Mar 12 00:05:00 GMT-06:00 2016	Daily	Fri Oct 07 00:05:00 GMT-06:00 2016	Active	Never
3	DeliverySlot_Fnd_From_RMS_Schedule	DeliverySlot	Sat Mar 12 00:10:00 GMT-06:00 2016	Daily	Fri Oct 07 00:10:00 GMT-06:00 2016	Active	Never
4	Dif_Fnd_From_RMS_Schedule	Dif	Sat Mar 12 00:15:00 GMT-06:00 2016	Daily	Fri Oct 07 00:15:00 GMT-06:00 2016	Active	Never
5	DifCip_Fnd_From_RMS_Schedule	Dif	Sat Mar 12 00:20:00 GMT-06:00 2016	Daily	Fri Oct 07 00:20:00 GMT-06:00 2016	Active	Never

Creating Schedule

The 'Create Schedule' option displays one page where the user can enter and save all required information to create a schedule. The page displays input fields under four sections as follows.

Figure 6–8 Creating Schedule

Basic Info

Schedule Name, Schedule Group and Schedule Description are entered under Basic Info. Schedule Name and Schedule Group are required fields.

Schedule Name must be unique. The User can choose an existing Schedule Group or add a new group name for the schedule.

There is limitation for the number of characters that these fields can accept.

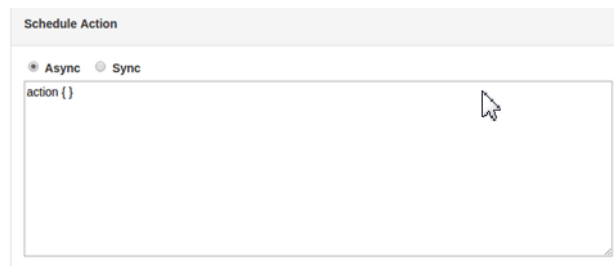
Schedule Action

Specify a valid schedule action definition here that will get executed when the schedule runs.

If any restricted keyword is present in the action definition, the schedule cannot be saved, and when saving the schedule an error highlighting the restricted keyword will be displayed.

Also choose here whether the schedule action is 'Async' (which is the default selected option) or 'Sync'.

Note: The schedule action is not validated or compiled for syntax when creating the schedule, so any syntax or programming error in the action definition will result in an exception at runtime and the schedule execution will fail.

Figure 6–9 Schedule Action**Schedule Frequency**

It consists of Schedule Start Datetime, End Datetime and Schedule Recurrence.

The Schedule End Datetime is 'Never' by default meaning the schedule never ends and repeats indefinitely. If the schedule has an end datetime, the user can enter a specific datetime.

The Start Datetime defaults to 5 minutes from the current time and the End Datetime defaults to 6 minutes from the current time when chosen.

The Start and End datetimes should be future dates. The Schedule End datetime if specified should be after the scheduled start datetime. These validations will be done when saving the schedule.

Scheduler provides two options to specify recurrence of the schedule - Simple Scheduling and Advanced Scheduling. Use the options tab to toggle between Simple and Advanced Scheduling options.

Simple Scheduling provides the following predefined schedules that the user can choose from a drop-down list.

- Hourly
- Daily (selected by default)
- Weekly
- Monthly
- Every Weekday [Mon-Friday]
- On Weekends [Sat-Sunday]
- Every Saturday
- Every Sunday
- First day of every month
- Last day of every month
- One time only
- Specify a different frequency

The User can use this option to specify a recurring interval in minutes, hours, days or weeks, for example, 30 minutes, 2 hours, 3 days, and so on.

Advanced Scheduling enables the user to specify complex schedules similar to a cron expression. The User can choose multiple values for Hours, Minutes, Day of Week, Day of Month and Month options using the multi-select lists.

The default schedule frequency here is daily midnight (Hours: 12 a.m., Minutes: 0 are the values selected by default).

Figure 6–10 Schedule Frequency

The screenshot shows the 'Advanced Scheduling' section of the Scheduler Console. It features five dropdown menus: 'Hours' (ranging from 12 a.m. to 11 a.m.), 'Minutes' (ranging from 0 to 11), 'Day of Week' (Sun, Mon, Tue, Wed, Thu, Fri, Sat), 'Day of Month' (ranging from 1 to 12), and 'Month' (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec). The 'Advanced Scheduling' tab is selected and highlighted in blue.

Schedule Notification

Use the schedule notification option to enable email notification for the schedule when schedule execution starts or fails or is completed.

Enter valid email addresses for notification. When enabled, email alerts will be sent based on the options selected.

Starts:

When this option is chosen, email will be sent when the schedule execution starts, that is, when the schedule runs at the scheduled interval, and just before the execution of the schedule action.

Fails:

An Email will be sent when there is an exception in the schedule execution or when the schedule action throws an exception or returns a 'Failed' response. This means the schedule action execution will be in 'Failed' status.

Triggered / Completed:

An Email will be sent when the schedule action execution status is 'Triggered' (for async actions) and 'Completed' (for sync actions). This essentially means the schedule execution is successful.

Figure 6–11 Schedule Notification

The screenshot shows the 'Notification' section of the Scheduler Console. It includes three checkboxes: 'When schedule execution Starts', 'Fails', and 'Triggered / Completed'. Below these is an 'Email' input field with the placeholder text 'Enter email (separate multiple emails by comma)'. The 'Starts' checkbox is currently selected.

Note: For schedule notification to work, the mail session needs to have been configured in the WebLogic server. Refer the BDI Installation Guide for details on the configuration of the mail session.

Updating Schedule

A schedule can be updated by selecting the schedule from the Manage Schedules page and using the Edit option in Schedule Detail view.

The Edit page is the same as that of the Create Schedule page with the schedule information populated. Update the values as required in the relevant sections as explained previously for creating the schedule. Only admin users can edit a schedule.

Note: The updating schedule frequency will validate schedule start datetime and end datetime (if specified) similar to when creating a schedule.

Updating any other details other than schedule frequency will not validate the existing schedule frequency as the schedule will continue to run at the already defined frequency and only the other details of the schedule definition will get updated as modified by the user.

When changing the schedule action definition, it will be verified for any restricted key-words.

Figure 6–12 Updating Schedule

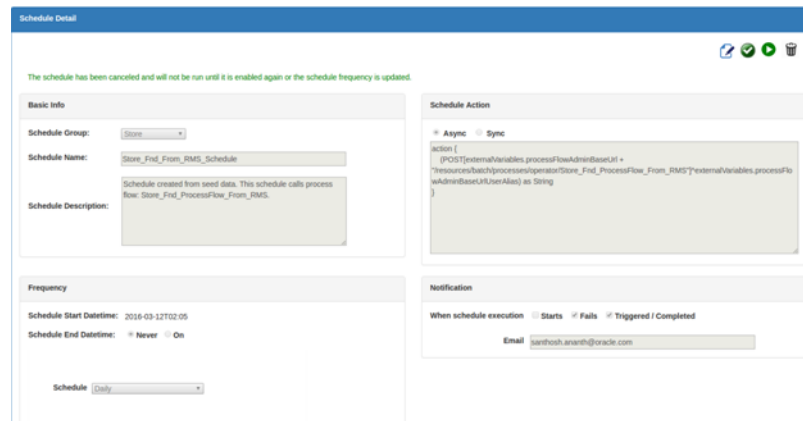
Disabling Schedule

A schedule can be disabled by selecting the schedule from the Manage Schedule page and using the 'Disable schedule' option in the Schedule Detail view. Only admin and operator users can disable a schedule.

Disabling a schedule will change the schedule status to 'Disabled' and the schedule will no longer run at the specified frequency. However the schedule can be manually run using the 'Run Schedule Now' option.

Note: The **Inactive** schedule cannot be disabled, as an inactive schedule has reached its end already and no longer runs.

Figure 6–13 Disabling Schedule

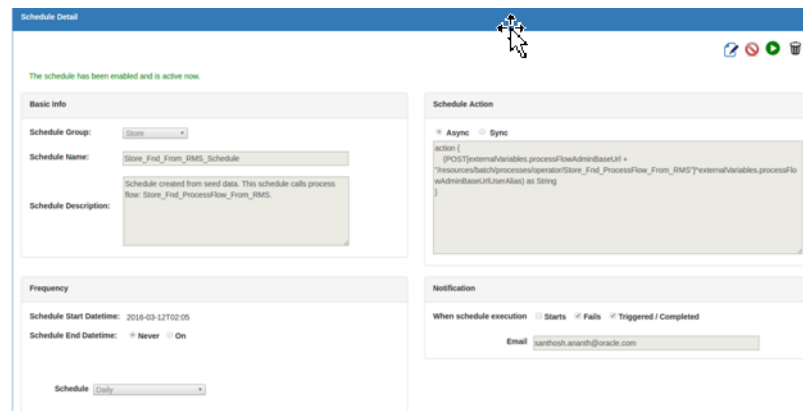


Enabling Schedule

A disabled schedule can be enabled again using the 'Enable schedule' option from the Schedule Detail view. Only admin and operator users can enable a schedule.

Enabling the schedule will change the status of the schedule to 'Active' and the schedule will resume running at the specified frequency.

Figure 6–14 Enabling Schedule



Deleting Schedule

A schedule can be deleted using the 'Delete schedule' option in the Schedule Detail view. Only admin users can delete a schedule.

Note: Deleting a schedule will delete the schedule definition and also its entire execution history. The schedule will no longer exist and will not run after deletion. There is no way to restore a deleted schedule except by creating the schedule again.

Figure 6–15 Deleting Schedule

The screenshot shows the 'Schedule Detail' page with a confirmation message at the top: "Deleting the schedule will also delete its execution history and the schedule will no longer run. Please confirm." Below this are 'Details' and 'Cancel' buttons. The page is divided into four sections: 'Basic Info', 'Schedule Action', 'Frequency', and 'Notification'. 'Basic Info' includes 'Schedule Group' (Store), 'Schedule Name' (Store_Fnd_From_RMS_Schedule), and 'Schedule Description'. 'Schedule Action' shows 'Async' selected and a code block. 'Frequency' shows 'Schedule Start Datetime' (2016-03-12T02:05) and 'Schedule End Datetime' (Never / On). 'Notification' shows 'When schedule execution' options (Starts, Fails, Triggered / Completed) and an 'Email' field (santosh.ananth@oracle.com).

Schedule Manual Run

Any schedule can be manually run using the 'Run Schedule Now' option from the Schedule Detail view. Inactive and disabled schedules can also be manually run.

This option is provided so that the user can run a schedule on demand when required. Only admin and operators can access this function.

When the schedule is run manually, the schedule action is submitted for execution in the backend and the result of the execution can be seen from the Schedule Executions view.

Figure 6–16 Schedule Manual Run

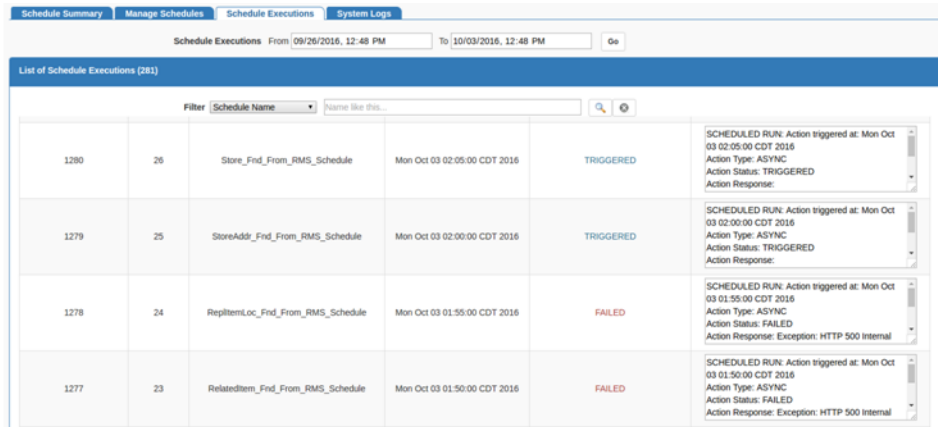
The screenshot shows the 'Schedule Detail' page with a confirmation message at the top: "Schedule Action successfully submitted for execution. See Schedule Executions for further detail on the status/result of execution." Below this are 'Details', 'Cancel', and 'Run' buttons. The page layout is identical to Figure 6-15, showing 'Basic Info', 'Schedule Action', 'Frequency', and 'Notification' sections.

Schedule Executions

From the Schedule Executions page, the user can view all available historical schedule executions. The page will display schedule executions for the last one week by default. The user can use the search option to enter a different date range to fetch the corresponding schedule executions.

Within the list of schedule executions, the records can be filtered based on the Schedule Name, Action Execution Status and any string within the Action Execution Log. The list of scheduled executions are sorted by schedule execution datetime, the latest first.

Figure 6–17 Schedule Executions



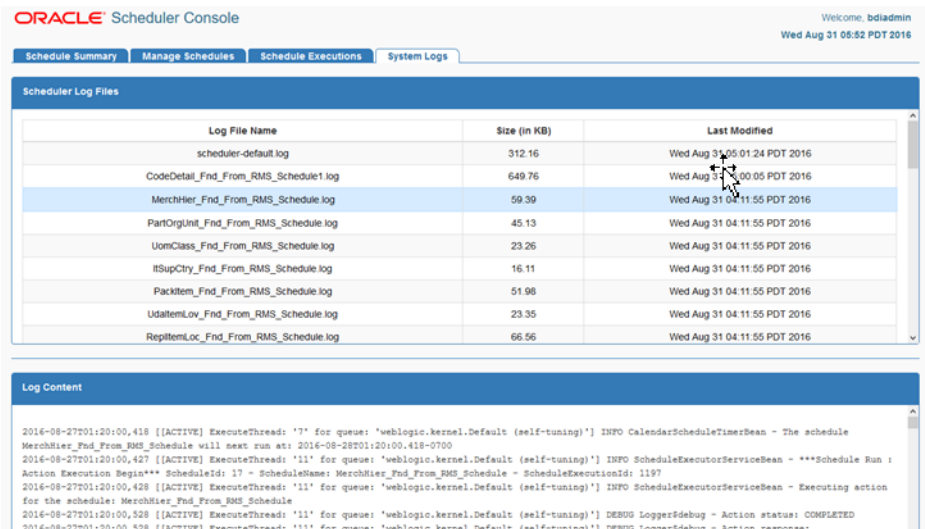
Schedule ID	Job ID	Schedule Name	Next Run Date	Status	Action Details
1280	25	Store_Fnd_From_RMS_Schedule	Mon Oct 03 02:05:00 CDT 2016	TRIGGERED	SCHEDULED RUN: Action triggered at: Mon Oct 03 02:05:00 CDT 2016 Action Type: ASYNC Action Status: TRIGGERED Action Response:
1279	25	StoreAddr_Fnd_From_RMS_Schedule	Mon Oct 03 02:00:00 CDT 2016	TRIGGERED	SCHEDULED RUN: Action triggered at: Mon Oct 03 02:00:00 CDT 2016 Action Type: ASYNC Action Status: TRIGGERED Action Response:
1278	24	RepItemLoc_Fnd_From_RMS_Schedule	Mon Oct 03 01:55:00 CDT 2016	FAILED	SCHEDULED RUN: Action triggered at: Mon Oct 03 01:55:00 CDT 2016 Action Type: ASYNC Action Status: FAILED Action Response: Exception: HTTP 500 Internal
1277	23	RelatedItem_Fnd_From_RMS_Schedule	Mon Oct 03 01:50:00 CDT 2016	FAILED	SCHEDULED RUN: Action triggered at: Mon Oct 03 01:50:00 CDT 2016 Action Type: ASYNC Action Status: FAILED Action Response: Exception: HTTP 500 Internal

System Logs

The System Logs page displays a list of all schedule log files and log contents. Each schedule has its own log file enabling easy access for the user to view the execution logs and other information from the log files for diagnosing and troubleshooting issues.

The list of log files are sorted by the last modified time of file with most recently modified file first.

Figure 6–18 System Logs



Log File Name	Size (in KB)	Last Modified
scheduler-default.log	312.16	Wed Aug 31 05:01:24 PDT 2016
CodeDetail_Fnd_From_RMS_Schedule1.log	649.76	Wed Aug 31 00:00:05 PDT 2016
MerchHier_Fnd_From_RMS_Schedule.log	59.39	Wed Aug 31 04:11:55 PDT 2016
PartOrgUnit_Fnd_From_RMS_Schedule.log	45.13	Wed Aug 31 04:11:55 PDT 2016
UomClass_Fnd_From_RMS_Schedule.log	23.26	Wed Aug 31 04:11:55 PDT 2016
ITSupCtry_Fnd_From_RMS_Schedule.log	16.11	Wed Aug 31 04:11:55 PDT 2016
PackItem_Fnd_From_RMS_Schedule.log	51.98	Wed Aug 31 04:11:55 PDT 2016
UdatemLov_Fnd_From_RMS_Schedule.log	23.35	Wed Aug 31 04:11:55 PDT 2016
RepItemLoc_Fnd_From_RMS_Schedule.log	66.56	Wed Aug 31 04:11:55 PDT 2016

```

2016-08-27T01:20:00.418 [[ACTIVE] ExecuteThread: '1' for queue: 'weblogic.kernel.Default (self-tuning)'] INFO CalendarScheduleTimerBean - The schedule
MerchHier_Fnd_From_RMS_Schedule will next run at: 2016-08-28T01:20:00.418-0700
2016-08-27T01:20:00.427 [[ACTIVE] ExecuteThread: '11' for queue: 'weblogic.kernel.Default (self-tuning)'] INFO ScheduleExecutorServiceBean - ***Schedule Run :
Action Execution Begin*** ScheduleId: 17 - ScheduleName: MerchHier_Fnd_From_RMS_Schedule - ScheduleExecutionId: 1197
2016-08-27T01:20:00.428 [[ACTIVE] ExecuteThread: '11' for queue: 'weblogic.kernel.Default (self-tuning)'] INFO ScheduleExecutorServiceBean - Executing action
for the schedule: MerchHier_Fnd_From_RMS_Schedule
2016-08-27T01:20:00.528 [[ACTIVE] ExecuteThread: '11' for queue: 'weblogic.kernel.Default (self-tuning)'] DEBUG Logger$Debug - Action status: COMPLETED
2016-08-27T01:20:00.528 [[ACTIVE] ExecuteThread: '11' for queue: 'weblogic.kernel.Default (self-tuning)'] DEBUG Logger$Debug - Action response:
    
```

Scheduler Security Considerations

This section describes the scheduler security considerations.

Scheduler Security

The Scheduler application uses basic authentication to authenticate users and allow access to the requested resources based on authorization. Only valid users can access the Scheduler Console and the REST resources. The Scheduler accesses the BDI process flows using basic authentication.

Users need to belong to one of these roles:

- Admin (assigned to BdiSchedulerAdminGroup in WebLogic Server)
- Operator (assigned to BdiSchedulerOperatorGroup in WebLogic Server)
- Monitor (assigned to BdiSchedulerMonitorGroup in WebLogic Server)

Only authorized users of specific role are allowed to access certain functionalities in the Scheduler Console.

Users of the Admin role have access to all the functions in Scheduler, the users of Operator role have limited authorizations to use only certain functions, and users of the Monitor role only have view/read-only access to the Scheduler Console.

Function	Admin Role	Operation Role	Monitor Role
View and search	Yes	Yes	Yes
Create schedule	Yes	No	No
Edit schedule	Yes	No	No
Delete schedule	Yes	No	No
Manual run schedule	Yes	Yes	No
Disable schedule	Yes	Yes	No
Enable schedule	Yes	Yes	No

Scheduler Operational Considerations

This section describes the scheduler operational considerations.

Users Roles for Monitoring and Administration

The Scheduler application is secured with role based security authorization. It is recommended to use separate users for Monitor, Operator and Admin roles.

Monitoring Schedules

Schedules and executions can be effectively monitored using the Scheduler Console. The console provides detailed action execution log and log files for each of the schedules that can be used to verify the runtime executions of schedules and related information.

Schedule Action Execution Log

Each schedule execution contains a 'Schedule Action Execution Log' that provides descriptive information on the scheduled run or manual run of the schedule. The Schedule Action Execution Log provides information as follows.

```
<SCHEDULED or MANUAL> RUN: Action triggered at: <Date and time>
Action Type: <ASYN or SYNC>
Action Status: <TRIGGERED or STARTED or COMPLETED or FAILED>
Action Response: <The response string as returned by the schedule action dsl, or
the error message in case of an exception>
```

For example, for a successful execution of schedule ItemHdr_Fnd_From_RMS_Schedule at the scheduled frequency, and action that triggers the process flow ItemHdr_Fnd_ProcessFlow_From_RMS, the Schedule Action Execution Log will be:

```
SCHEDULED RUN: Action triggered at: Wed Jul 27 12:00:01 EDT 2016
Action Type: ASYNC
Action Status: TRIGGERED
```

```
Action Response: {"executionId":"ItemHdr_Fnd_ProcessFlow_From_
RMS#0d3d656d-041a-4068-8daf-8d17ee1ad899", "processName":"ItemHdr_Fnd_ProcessFlow_
From_RMS"}
```

In case of an exception (say, connection error when invoking process flow), the action execution log will look as follows:

```
SCHEDULED RUN: Action triggered at: Sat Aug 06 00:40:00 EDT 2016
Action Type: ASYNC
Action Status: FAILED
Action Response: Exception: java.net.ConnectException: Tried all: '1' addresses,
but could not connect over HTTP to server: java.net.ConnectException: Connection
refused
Check the logs for more details.
```

The above action execution log examples indicate async actions. For sync actions, the the action execution log also shows when the schedule action started and when it completed, which is particularly useful for a long running action for which the Scheduler waits for the response until completion. For example,

```
SCHEDULED RUN: Action execution started at: Wed Aug 03 12:00:00 EDT 2016
Action Type: SYNC
Action execution ended at: Wed Aug 03 12:22:10 EDT 2016
Action Status: COMPLETED
Action Response: Batch process completed.
```

Note: The Action Response shows the value that the schedule action dsl finally returns after completion.

Scheduler Log Files

Each schedule has its own log file. For example, a schedule named Store_Fnd_From_RMS_Schedule will have its log file named Store_Fnd_From_RMS_Schedule.log.

The log file contains detailed information on schedule executions which can be scheduled runs or manual runs, logs of actions such as disabling and enabling the schedule, action log on schedule updates such as change in schedule frequency, and in case of any exceptions, the exception stack trace.

Users can use the following keywords to search for specific information in the schedule log file.

Keyword	Description
ScheduleId	The primary key Id of the schedule
ScheduleName	The schedule name
ScheduleExecutionId	The execution Id of schedule run instance
Action Execution Begin	Indicates the start of the log when schedule action begins.
Action Execution End	Indicates the end of the log when schedule action ends. The log of the schedule action execution can be found between the two strings: ***Schedule Run : Action Execution Begin*** and ***Schedule Run : Action Execution End*** For manual run, it will be ***Manual Run : Action Execution Begin*** and ***Manual Run : Action Execution End***

Keyword	Description
Action execution exception	The detailed exception message and stacktrace will be shown following this string, when an exception has occurred in schedule action execution.

Maintaining Historical Schedule Executions

As the schedules run, schedule execution records are stored in the BDI_SCHEDULE_EXECUTION table.

This table will grow larger as the number of schedule executions increase. Hence it is recommended to periodically purge historical scheduled executions from the tables that are older and no longer necessary, and only retain recent schedule executions of a particular period, say for the last one month to now. This will help keep the table size within a certain limit and prevent database growth.

Scheduler Customization

This section describes the scheduler customization.

Seed Data Reload

The sql script containing the seed data schedule definitions is located in the bdi-scheduler-home/setup-data/dml folder.

During the initial deployment of the Scheduler application, seed data schedules get loaded to the schedule definition table and the corresponding schedules are created.

If the Scheduler application needs to be redeployed and the seed data schedules need to be reloaded during the redeployment (that is, to reset the schedules to the initial state as per seed data), set the LOADSEEDDATA column in the BDI_SYSTEM_OPTIONS table to TRUE, and undeploy and redeploy the application.

Note: The above redeployment procedure will reset the current schedule definitions (that is, existing schedules and any changes will be deleted) and the schedules will be recreated as per seed data definitions. Use this option with caution and only when absolutely necessary.

Customizing Seed Data Schedules

By default all BDI seed data schedules are scheduled to run daily starting at midnight (each schedule running with a gap of 5 minutes). The User can edit the seed data and add new schedules to be loaded during deployment, by updating the seed data sql script and adding corresponding schedule action scripts in the bdi-scheduler-home install directory, before starting the installation.

Seed data sql file: bdi-scheduler-home/setup-data/dml/seed-data.sql

Schedule Action dsl files: bdi-scheduler-home/setup-data/dsl

An insert statement for a schedule seed data definition will look as follows (SQL for Oracle database):

```
INSERT INTO BDI_SCHEDULE_DEFINITION (schedule_id, schedule_name, schedule_group,
schedule_description, schedule_status, schedule_start_datetime, schedule_type,
schedule_frequency, schedule_notification, schedule_notification_email, schedule_
action_type, schedule_action_definition) VALUES (7, 'InvAvailStore_Tx_From_RMS_
Schedule', 'Inventory', 'Schedule created from seed data. This schedule calls
process flow: InvAvailStore_Tx_ProcessFlow_From_RMS.', 'ACTIVE', TIMESTAMP
```

```
'2016-03-12 00:30:00', 'SIMPLE', 'DAILY', 'ON_SUCCESS,ON_ERROR', 'user@example',
'ASYNC', 'InvAvailStore_Tx_From_RMS_Schedule_Action.sch')
```

Note: When adding or editing schedule definitions in seed data to be loaded at application startup. All these fields (as shown in the sql statement above) are required fields to create a schedule at startup.

- schedule_id should be a unique number for each schedule.
- schedule_name should be unique.
- schedule_status needs to be 'ACTIVE' for schedule to be created and active.
- schedule_type should be 'SIMPLE' with any of the schedule_frequency values mentioned above. Advanced schedule (calendar schedules with complex cron expression) is not supported through seed data during deployment.
- schedule_start_datetime:
 - Need to be in the format yyyy-mm-dd hh:mm:ss
 - For example, 2016-01-01 00:00:00, 2016-01-01 18:30:00
- schedule_frequency:
 - Valid values are: DAILY, HOURLY, WEEKLY, MONTHLY, WEEKDAY, WEEKEND, SATURDAY, SUNDAY, FIRSTDAYOFMONTH, LASTDAYOFMONTH, ONCE
- schedule_notification:
 - Valid values are: ON_START, ON_SUCCESS, ON_ERROR (separate multiple values by comma)
- schedule_email:
 - Valid email-id for notification (separate multiple emails by comma). Email is required if schedule_notification is specified.
- schedule_action_type:
 - Valid values are (based on the action specified): 'ASYNC' or 'SYNC'.
- schedule_action_definition in seed data refers to the name of the corresponding schedule action dsl file (this will get loaded at startup).

Each schedule should have a corresponding schedule action dsl script defined. This will be the action that gets executed when the schedule runs.

To load the schedule action dsl during deployment, add the schedule action dsl file under bdi-scheduler-home/setup-data/dsl with file name convention: <Schedule Name>_Action.sch.

For example for adding a new schedule named Schedule_1, add schedule action dsl script Schedule_1_Action.sch. During deployment, the Scheduler will create Schedule_1 and update the schedule definition with the action script from the corresponding file Schedule_1_Action.sch.

Customizing Schedule Actions

The seed data schedules in the Scheduler are the schedules that call the BDI process flows provided out-of-the-box. The Schedule Actions define the REST calls to the BDI process flows.

In an enterprise implementation, there will be requirements to schedule batch processes, any recurring jobs or activities that are not BDI process flows. There can also be existing batch processes or services that need to be scheduled.

The Scheduler can be used for such scheduling requirements by defining appropriate Schedule Actions to invoke the services.

The Scheduler can be used to schedule RESTful services and as the Schedule Action is a DSL based on Groovy, valid Groovy or Java code can also be used within the action part that will be executed by the Scheduler based on the defined schedule.

The syntax for Schedule Action is as simple as follows.

```
action {
    //your implementation goes here
}
```

The following Schedule Action syntax specifies how a REST service can be called from the Scheduler (assuming the REST resource does not require any authentication). The response from the REST service will be treated as string.

```
action {
    (POST[<your REST service URL here>]) as String
}
```

This is a simple approach for scheduling existing and new services that can be exposed as REST services.

The Schedule Action syntax to call a REST service with authentication and with base URL configured in System Options will look as follows.

```
action {
    POST[externalVariables.myRESTServiceBaseUrl +
        "/resources/myRESTresource"]^externalVariables.myRESTServiceBaseUrlUserAlias) as
    String
}
```

The externalVariables is the name of the variable used internally by the Scheduler to access system options parameters. Any parameters (key-values) configured in System Options can be accessed using the notation externalVariables.<my-system-option-parameter>

Admin users can utilize System Setting RESTful service to add or update system options parameters, and setting up credentials (stored in wallet) for any authentication to be used by the application. Refer Appendix D for details on the System Setting REST resources.

In the above example, the user can add system option parameters named 'myRESTServiceBaseUrl' with the REST resource base url value (for example, http://<myserverhost>:<port>/myapp) and 'myRESTServiceBaseUrlUserAlias' which will be the alias name to be used for authentication and the value of this parameter should be GET_FROM_WALLET:GET_FROM_WALLET to indicate that the corresponding credentials for the alias need to be obtained from the wallet during runtime by the application.

Scheduler Troubleshooting

Any failure in schedule execution can be analysed in the Scheduler application by checking the Scheduler log files for the corresponding schedule.

If a schedule execution is 'FAILED' due to an exception response from the process flow, then the details of corresponding process flow execution instance, the exception details and any stack trace can be viewed in the corresponding process flow logs using the Process Flow Admin console for further troubleshooting.

Note: The schedule execution where the BDI process flow is called is only a trigger for the process flow execution, hence the actual execution of the process flow and the status and logs thereof can only be viewed in the BDI Process Flow Admin console.

Scheduler Known issues

The Scheduler Console provides a calendar widget for datetime fields that are currently supported only by the Chrome browser. Hence it is recommended to use the latest version of the Chrome browser to access the Scheduler Console.

If any other browser is used that does not support the calendar widget for the datetime input, the datetime fields may appear as textbox. Users can enter the datetime input as text, but the value should be in the format of 'yyyy-MM-ddTHH:mm', for example, 2016-01-01T20:00. There is no loss of functionality due to this limitation however.

The BDI suite provides two CLI (Command-Line Interface) tools as part of this release.

- BDI CLI Job Executor BDI
- CLI Batch Transmitter

The following sections describe in detail the above CLI components, their setup and usage.

BDI CLI Job Executor

The BDI CLI Job Executor is a standalone command line utility that helps in basic operation of BDI batch jobs through commands. It uses the REST services that the BDI Batch Job Admin provides to list jobs and executions, get status of a job, and start, stop and restart a batch job.

Tool Setup

To prepare the tool for use, follow these steps.

The `bdi-cli-job-executor` home directory (where the tool software package is extracted) contains a `conf` directory where the tool related configuration file will be present, and `bin` directory where the executable to run the tool will be present.

- Configure BDI Batch Job Admin URL and alias name for the credentials to access Job Admin URL.
 - Edit `conf/bdi-job-admin-info.json` file to add the BDI Batch Job Admin URL value for the `jobAdminUrl` property.
 - * Example:
`"jobAdminUrl":"http://<hostname>:<port>/bdi-rms-batch-job-admin/"`
 - Add alias name in the property `jobAdminUserAlias`.
 - * Example:
`"jobAdminUserAlias":"rmsJobAdminBaseUrlUserAlias"`
- Run: `bdi-cli-job-executor.sh -setup-credentials`

Note: `bdi-cli-job-executor.sh` will be in the `bin` directory.

- This prompts for the credentials for the given alias. Enter the corresponding username and password to be used to access the Job Admin URL. The

credentials will be stored in the wallet and used to invoke the BDI Job Admin REST services.

Tool Usage

The BDI CLI Job Executor tool is run using the shell script: `bdi-cli-job-executor.sh` from the 'bin' directory.

Usage: `bdi-cli-job-executor.sh` -[option]

Option	Description
<code>list</code>	Lists all available job names and details. <code>bdi-cli-job-executor.sh -list</code>
<code>list runningJobs</code>	Lists all currently running jobs and job execution IDs. <code>bdi-cli-job-executor.sh -list runningJobs</code>
<code>start <jobname></code>	Starts a job of given name. Example: <code>bdi-cli-job-executor.sh -start MyBatchJob</code>
<code>restart <jobname> <executionId></code>	Restarts a failed job execution with the corresponding execution Id. Example: <code>bdi-cli-job-executor.sh -restart MyBatchJob 12345</code>
<code>stop</code>	Stops all the running job executions. <code>bdi-cli-job-executor.sh -stop</code>
<code>stop <executionId></code>	Stops the currently running job execution of given execution Id. Example: <code>bdi-cli-job-executor.sh -stop 12345</code>
<code>status <jobname></code>	Gets the status of the job of given job name. Example: <code>bdi-cli-job-executor.sh -status MyBatchJob</code>
<code>status <jobname> <instanceId></code>	Gets the status of the job of given job name and job instance Id. Example: <code>bdi-cli-job-executor.sh -status MyBatchJob 54321</code>

BDI CLI Transmitter

The BDI CLI Transmitter is a standalone command line tool to transmit batch interface data files to a destination BDI receiver system. It is particularly used where the source system is non-BDI (that is, the source system does not have or use BDI Batch Job Admin application) but needs to send interface data files to a receiver system running the BDI Job Admin application.

The tool uses the BDI Job Admin Receiver REST service URL to transmit the data to the destination system. So it is necessary that the destination system runs the BDI Job Admin application to use the tool.

Tool Setup

To prepare the tool for use, follow these steps.

- The bdi-cli-transmitter home directory (where the tool software package is extracted) contains 'conf' directory where the tool related configuration files will be present, and 'bin' directory where the executable to run the tool will be present.
- Configure conf/bdi-file-transmitter.properties. The following describes the properties to be configured. The properties file provides some sample values for reference to start with. The values specified in the properties file can be overridden using the command-line input options if required, when running the tool for file transmission.

Property	Description
source.system.name	The name of the source system or application that provides the source data to be transmitted. For example, source.system.name=RMS
<receiverAppName>.receiver.url	The Receiver REST service URL of the BDI Receiver application indicated by <receiverAppName> (should be in lowercase). For example, if the BDI receiver application is RXM, then specify the property and value as: rxm.receiver.url=http://<bdi-rxm-app-hostname>:<port>/bdi-rxm-batch-job-admin/resources/receiver
<receiverAppName>.receiver.url.useralias	Alias name for the credentials to be used to connect to the corresponding receiver service. The alias name with the credentials are stored in a wallet. <receiverAppName> should be in lowercase. Example: rxm.receiver.url.useralias=rxmReceiverUrlUserAlias
<InterfaceModuleName>.receiver.appname	Name of the BDI receiver application for the interface module <InterfaceModuleName>. Specify the name in lowercase. Example: Diff_Fnd.receiver.appname=rxm Store_Fnd.receiver.appname=sim
<InterfaceModuleName>.dataset.type	The data set type of the data to be transmitted for the interface module identified by <InterfaceModuleName>. Valid value is FULL or PARTIAL. Example: Diff_Fnd.dataset.type=FULL Only FULL data sets are supported in the currently implemented BDI flows.
<InterfaceModuleName>.interfaceShortNames	The interface name(s) for the corresponding interface module <InterfaceModuleName>. Multiple interface names can be specified (each separated by a comma) as multiple interfaces can be part of an interface module. The interface module name and interface names should be the same as expected by the BDI receiver application where the files are transmitted. Example: Diff_Fnd.interfaceShortNames=Diff DiffGrp_Fnd.interfaceShortNames=Diff_Grp,Diff_Grp_Dtl

Property	Description
<InterfaceModuleName>.<InterfaceShortName>.input.filepath	Specify the file location where the corresponding interface data files to be transmitted are present. Each interface in a interface module should have separate file locations. Example: Diff_Fnd.Diff.input.filepath=/home/bdi/diff_fnd/diff/files DiffGrp_Fnd.Diff_Grp.input.filepath=/home/bdi/diffgrp_fnd/diff_grp/files DiffGrp_Fnd.Diff_Grp_Dtl.input.filepath=/home/bdi/diffgrp_fnd/diff_grp_dtl/files

- Run: `bdi-file-transmitter.sh -setup-credentials`.

Note: `bdi-file-transmitter.sh` will be in the 'bin' directory.

Run `-setup-credentials` to configure the BDI Receiver service user credentials. Running this command will prompt for the username and password for each of the `<receiverAppName>.receiver.url.useralias` specified in `bdi-file-transmitter.properties` file.

The credentials entered for each alias will be stored in a secure wallet and used to connect to the corresponding BDI Receiver service for transmission of files.

This is a prerequisite step to use the tool but usually a one-time setup before running `bdi-file-transmitter.sh` for transmission of files.

- Optionally, configure `conf/bdi-file-transmitter-runtime.properties` that contains parameters (described below) for performance tuning of the tool.

Start with default values as present in the properties file, analyze the performance and choose optimal values for the parameters for better performance if required. The tool will use default values for the parameters (mentioned below) when no values are specified in the properties file.

Property	Description
<code>multiple_files_process_limit</code>	The maximum number of files to process in parallel at any given time. Default value is 5.
<code>file_transmission_thread_limit</code>	The number of parallel threads to run to process a single file. Default value is 3.
<code>transmission_record_size</code>	The maximum number of records per block or chunk to transmit to the receiver service per service call. Default value is 20000.
<code>transmission_timeout</code>	The timeout in minutes for file transmission. The process will timeout and end when the file transmission is still not complete after the specified time. Default value is 300 minutes.

Tool Usage

The BDI CLI Transmitter tool is run using the shell script: `bdi-file-transmitter.sh` from the 'bin' directory.

The tool can be run in interactive and noninteractive modes.

Interactive Mode: Run `bdi-file-transmitter.sh`

For user interactive mode where the program prompts for input, just run `bdi-file-transmitter.sh` with no options.

This will prompt for each input with descriptions which will be self-explanatory. The user can enter value as required or skip optional parameters. When no value is specified for optional parameters, the tool will try to use the default values as specified in the `bdi-file-transmitter.properties` file or stop executing when no default value is present.

Non-Interactive Mode: Run `bdi-file-transmitter.sh [input]`

The tool can be run with the following inputs as described below.

Note: The only required input is interface module name, when the other input values are specified in `bdi-file-transmitter.properties` file.

Input	Description
<code>-m</code> or <code>--interfacemodule <interfaceModuleName></code>	(Required) The interface module name. Should be the same as the interface module name expected by the BDI receiver application.
<code>-i</code> or <code>--interfaceshortnames <interfaceShortNames></code>	(Optional) Multiple interface names should be separated by comma. If not specified, the program will use the interface names corresponding to the interface module as specified in <code>bdi-file-transmitter.properties</code> file. The interface names should be the same as expected by the BDI receiver application
<code>-s</code> or <code>--sourcesystem <sourceSystemName></code>	(Optional) The source system name. If not specified, the program will use the <code>source.system.name</code> given in the properties file.
<code>-f</code> or <code>--filelocation <inputFilePaths></code>	(Optional) The location of interface data file(s) that are to be transmitted. This can be single file or a directory path with multiple data files of the interface. Multiple file paths should be separated by comma, for each interface in the corresponding order. If not specified, the program will use the input file paths given for the interfaces as given in the properties file.
<code>-a</code> or <code>--receiverapp <receiverAppName></code>	(Optional) The BDI receiver app name. This is used to get the receiver url and/or useralias from properties file if any of those values are not provided. If not specified, the program will use the receiver app name specified for the interface in the properties file.
<code>-r</code> or <code>--receiverurl <fileReceiverUrl></code>	(Optional) The receiver url. If not specified, the program will use the receiver url of the receiver app specified for the interface in the properties file, for transmission of files.
<code>-u</code> or <code>--useralias <receiverUrlUserAlias></code>	(Optional) The alias name for the credentials to be used to connect to the receiver service url. The credentials corresponding to the alias should exist in the wallet. If not specified, the program will use the receiver url useralias of the receiver app specified for the interface in the properties file.
<code>-d</code> or <code>--datasettype <dataSetType></code>	(Optional) The data set type that specifies the data transmitted is full or delta load. Valid value: 'FULL' or 'PARTIAL'. If not specified, the program will use the interface specific data set type as given in the properties file.

Some examples of running the transmitter tool command-line:

```

bdi-file-transmitter.sh -m Diff_Fnd
bdi-file-transmitter.sh -m Diff_Fnd -i Diff
bdi-file-transmitter.sh -m DiffGrp_Fnd -i Diff_Grp,Diff_Grp_Dtl
bdi-file-transmitter.sh -m Diff_Fnd -a sim
bdi-file-transmitter.sh -m DiffGrp_Fnd -i Diff_Grp,Diff_Grp_Dtl -f
/home/bdi/diffgrp_fnd/diff_grp/files,/home/bdi/diffgrp_fnd/diff_grp_dtl/files
bdi-file-transmitter.sh -m "Diff_Fnd" -i "Diff" -s "RMS" -d "FULL"
bdi-file-transmitter.sh -m Diff_Fnd -i Diff -s "RMS" -f "/home/bdi/diffgrp_
fnd/diff/files" -a "sim" -r
"https://bdisimapphost:9001/bdi-sim-batch-job-admin/resources/receiver" -d "FULL"

```

File Processing

The BDI Transmitter tool supports transmission of flat files, for example, .csv files, in UTF-8 format. The BDI Receiver application supports only csv files. Hence the interface data files to be transmitted need to contain records with comma-separated field values.

The order of the fields in the file should be as expected by the BDI Receiver application, so that each value is inserted in the right columns of the destination interface tables. No header line should be present in the file (each line is treated as data record). Each record should be in a newline.

The interface module name and interface names for the files to be transmitted should be same as expected by the BDI Receiver application.

The transmitter tool can process a single file or a directory containing multiple files. But the tool does not process files recursively in subdirectories.

Files are processed and transmitted per interface module. Each run of processing of files of the interface module will be considered a transaction and a Transaction Id will be generated and associated to the transmission of files (at the interface module level). Files of multiple interfaces in an interface module will be part of the same transaction.

Each file transmission within a transaction will have a Transmission Id associated to it. The same transaction Id and transmission Id are sent to the BDI Receiver application, so the corresponding transmission details can be seen in the Job Admin console of the BDI Receiver application.

After successful transmission, the file will be moved to the **archive** directory:

```
<inputFileDirectory>/archive/<interfaceModuleName>/<transactionId>
```

For example,

if the input file location is '/home/bdi/interface/files' and the interface module of the files is 'Diff_Fnd', and the transaction Id of the file transmission is 'Tx#1475858081837', then after successful transmission the file will be moved to the directory:

```
/home/bdi/interface/files/archive/Diff_Fnd/Tx#1475858081837.
```

Output Logs

The transmitter tool outputs messages and logs to the terminal console where the command is run.

The tool also creates a log file that contains detailed logs about the processing of files. The log will show the Transaction Id and Transmission Id of each file transmission among other details.

The log file is created in the logs directory under the tool home directory (bdi-cli-transmitter/logs).

The name of the log file will be in the format: bdi-file-transmitter_YYYY-MM-DD_HH:MM:SS, for example bdi-file-transmitter_2016-07-04_10:38:59.

Error Reprocessing

In case of any error in file processing, error in transmission of file to the receiver service, timeout of file transmission, or any other failure, the file will be moved to the 'failed' directory:

```
<inputFileDirectory>/failed/<interfaceModuleName>/<transactionId>
```

For example, if the input file location is '/home/bdi/interface/files' and the interface module of the files is 'Diff_Fnd', and the transaction Id of the file transmission is 'Tx#1475858081837', then if the transmission of file fails, the file will be moved to the directory: /home/bdi/interface/files/failed/Diff_Fnd/Tx#1475858081837.

A properties file containing the input details corresponding to the failed file will be created. For example, if the file named 'Item_1.csv' has failed, then a file named 'Item_1.csv.properties' will be created in the 'failed' directory. This acts as the input context that will be used when the file is reprocessed. The user should not delete or modify this properties file, if the data file has to be re-processed with the original input context.

Due to parallel processing of files by the transmitter, there may be a scenario where some records in the file may have been transmitted successfully, but part of the file transmission may have failed. Even in this case, the entire file will be treated as failed and moved to the 'failed' directory.

Reprocessing will be at the file level and not at the block level where the transmission may have failed. In the case of partial transmission of file, the BDI Receiver application also marks the whole transmission as failed and hence the entire file can be retransmitted to be processed again by the receiver application.

To retry failed files (that did not get transmitted successfully in previous transmission) use the below command:

```
bdi-file-transmitter.sh -retry-failed <inputFileDir or inputFilePath>
```

```
For example, bdi-file-transmitter.sh -retry-failed
/home/bdi/interface/files/failed/Diff_Fnd/Tx#1475858081837
```

```
bdi-file-transmitter.sh -retry-failed /home/bdi/interface/files/failed/Diff_
Fnd/Tx#1475858081837/Diff_1.csv
```

Once a file is successfully reprocessed, it will be renamed as <filename>-retransmitted. For example, Diff_1.csv-retransmitted. And, the corresponding properties file will be deleted.

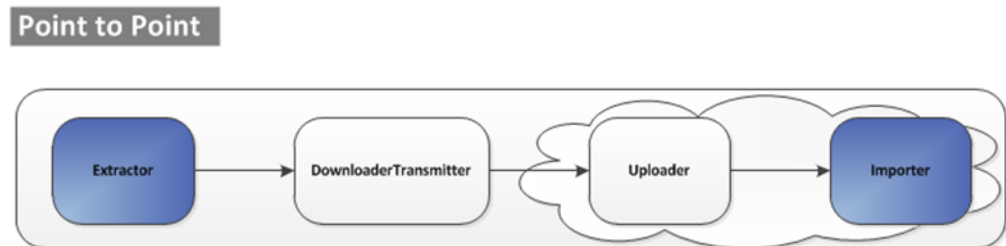
BDI Data Integration Topologies

The BDI infrastructure applications move data from one application to another. So there is data producing applications and data consuming applications. Depending on the customer needs, the data produced by an application may be used by one or more consuming applications. This leads to different deployment architectures for various needs.

In all of the topologies presented, regardless of the examples presented, in practice, the sender and receiver locations can be on-premise, cloud, or hybrid deployments. BDI is designed to be location transparent.

Point to Point Topology

Figure 8-1 Point to Point

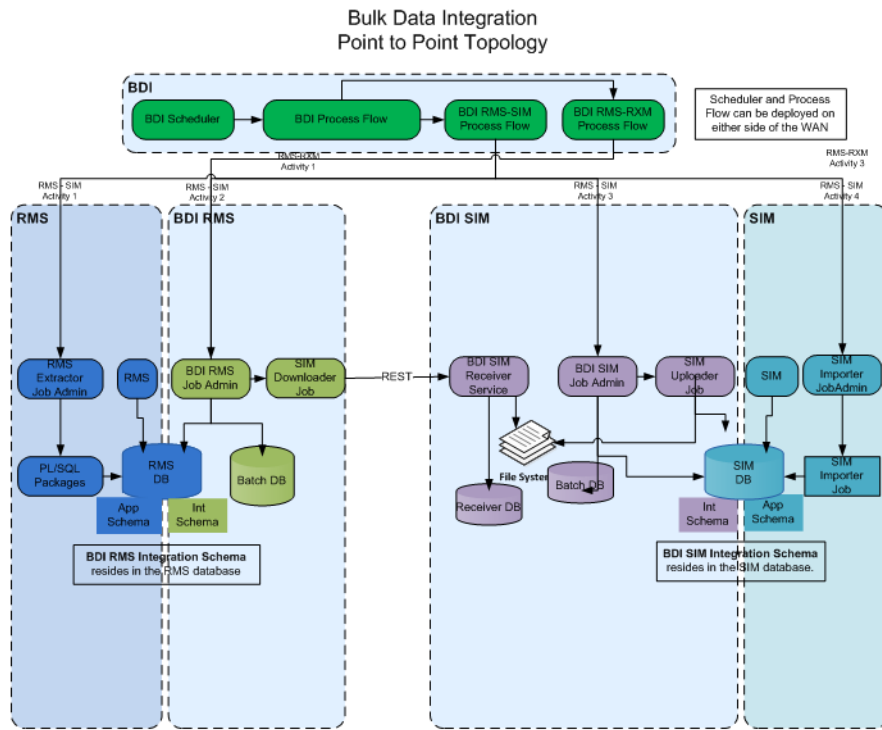


In this deployment topology, there is only one consumer for an interface module (An interface module may have one or more interfaces).

Note: In the case of where there are multiple receivers for the data, this topology is not the most efficient.

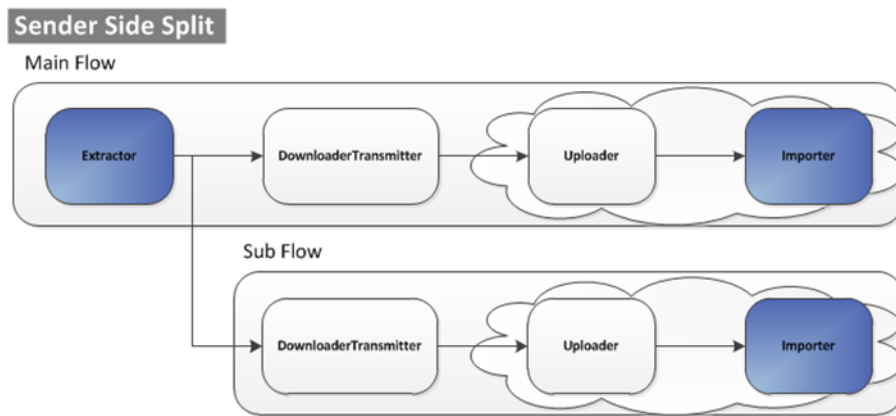
A detailed diagram of a point to point topology usage in Oracle Retail is shown below.

Figure 8–2 Detailed BDI Point to Point Topology



Sender side split

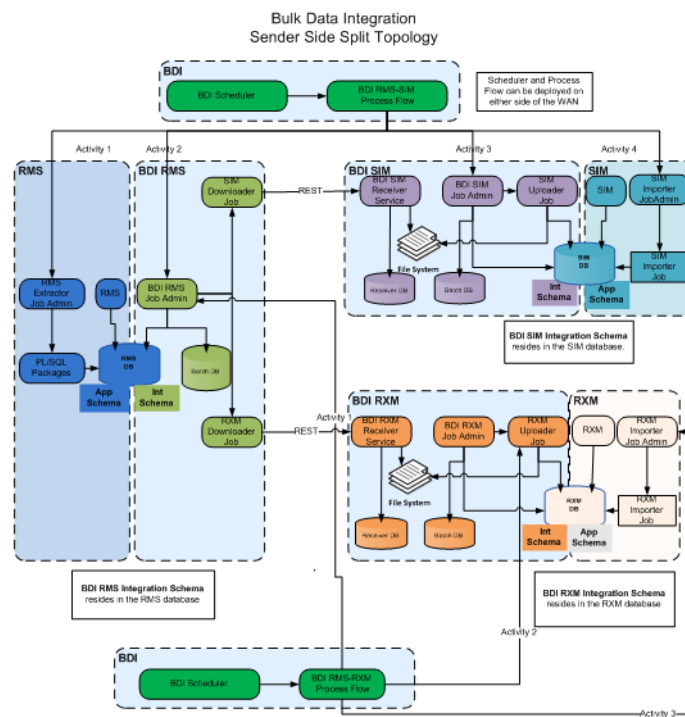
Figure 8–3 Sender Side Split



In the case of Sender Side Split (SSS), the data is extracted once from the source system. The extracted data is transmitted to each destination separately. Since, unlike point to point topology, the extraction done only once regardless of the number of destinations.

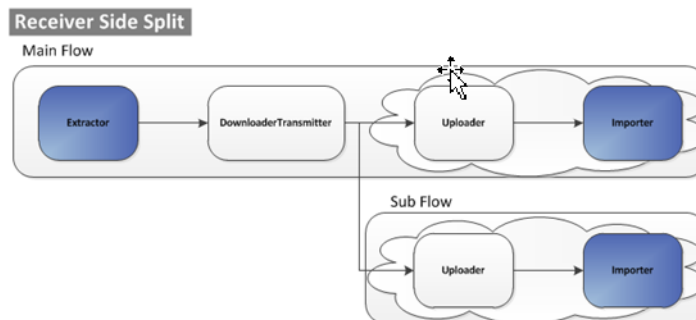
A detailed diagram of sender side split topology usage in Oracle Retail is shown below.

Figure 8–4 Sender Side Split Topology



Receiver Side Split

Figure 8–5 Receiver Side Split

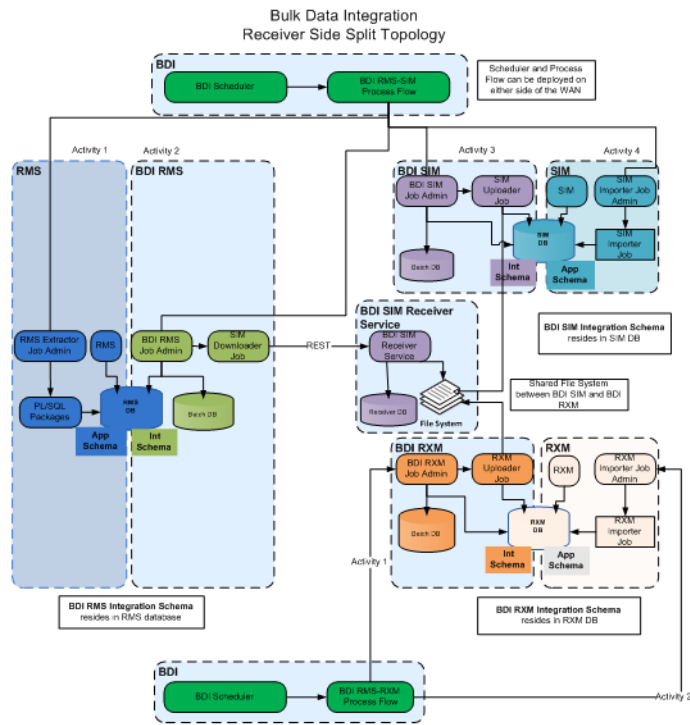


The Receiver Side Split (RSS) topology is used for multi destination data transfer such as Sender Side Split. In this topology data is extracted and transmitted to the destination only once regardless of the number of destinations. This topology differs from the sender side split in the number of times the data is transmitted.

Receiver side split can only be used if all the destinations have a shared network drive access. This is the most optimal multi destination data transfer topology.

A detailed diagram of receiver side split topology usage in Oracle Retail is shown below.

Figure 8-6 Receiver Side Split



Pre-implementation Considerations

Before BDI is installed into an enterprise, there are many factors that need to be considered. Planning and addressing each of the factors will avoid having to reinstall or re-architect because of performance or operational problems.

BDI Software Lifecycle Management

Software applications, after being made generally available (GA), have a well defined lifecycle process. The implementer must manage and perform tasks in these phases:

- Acquire the software components
- Prepare the environment
- Assemble the application
- Deploy and Start the application
- Perform day-to-day monitoring to make sure the application is running properly
- Apply code fixes to the application

Preparation Phase

In this phase, all relevant components are downloaded, extracted, and configured.

Application Assembly Phase

In this phase, site specific configuration changes are made and all relevant BDI wars are generated.

Deployment Phase

In this phase, using the BDI wars created in the previous step, the wars are deployed to application server instances.

Operation Phase

In this phase, day-to-day operations of the BDI applications are performed.

Maintenance Phase

In this phase, code fixes, patching, configuration changes and maintenance of the BDI applications are performed.

Physical Location Considerations

The Oracle Retail Merchandising System (RMS) is the most important core business application from the suite of Oracle Retail Product offerings. RMS provides most of the retail business functionality that offers its customers. In other words RMS is the central hub of Oracle retail applications. Since RMS is the central hub of retail information/data and most information/data flows outward from RMS to other edge retail applications through BDI, the decision on where to physically/logically locate BDI applications is very important and will have direct impact on the functioning of your enterprise.

It is recommended to keep the "bdi-rms" integration schema created in the RMS database server so that the data movement from RMS to outbound tables located in integration schema is fast. Similarly the "bdi-rxm" integration schema is created in the RXM database server so that the data movement from inbound tables located in the integration schema to the RXM transactional tables is fast.

It is also recommended to colocate the "bdi-rms-batch-job-admin" application near RMS application and the "bdi-rxm-batch-job-admin" application near RXM application. The Job Admin application for BDI RMS (bdi-rms-batch-job-admin) need to be deployed in a separate domain. Similarly BDI RXM (bdi-rxm-batch-job-admin) needs to be deployed in a separate domain.

Multiple instances of the BDI RXM application can improve the transfer of bulk data between RMS and RXM.

High Availability Considerations

As businesses are maturing and having to do everything quicker, better, faster, and with less resources and money, they are pushing similar expectation onto their IT infrastructure. Business users are expecting more out of their IT investments, with zero down time. Consistent predictable responding systems, which are highly available, have become a basic requirement of today's business applications.

Modern business application requirements are classified by the abilities that the system must provide. This list of abilities such as availability, scalability, reliability, audit ability, recoverability, portability, manageability, and maintainability determine the success or failure of a business.

With a clustered system many of these business requirement abilities gets addressed without having to do lots of development work within the business application. Clustering directly addresses availability, scalability, recoverability requirements which are very attractive to a business. In reality though it is a tradeoff, a clustered system increases complexity, is normally more difficult to manage and secure, so one should evaluate the pros and cons before deciding to use clustering.

Oracle provides many clustering solutions and options; those relevant to BDI are Oracle database cluster (RAC) and WebLogic Server clusters.

WebLogic Server Cluster Concepts

A WebLogic Server cluster consists of multiple WebLogic Server managed server instances running simultaneously and working together to provide increased scalability and reliability. A cluster appears to clients to be a single WebLogic Server instance. The server instances that constitute a cluster can run on the same machine, or be located on different machines. You can increase a cluster's capacity by adding additional server instances to the cluster on an existing machine, or you can add machines to the cluster to host the incremental server instances. Each server instance in a cluster must run the same version of WebLogic Server.

In an active-passive configuration, the passive components are only used when the active component fails. Active-passive solutions deploy an active instance that handles requests and a passive instance that is on standby. In addition, a heartbeat mechanism is usually set up between these two instances together with a hardware cluster (such as Sun Cluster, Veritas, RedHat Cluster Manager, and Oracle CRS) agent so that when the active instance fails, the agent shuts down the active instance completely, brings up the passive instance, and resumes application services.

In an active-active model all equivalent members are active and none are on standby. All instances handle requests concurrently.

An active-active system generally provides higher transparency to consumers and has a greater scalability than an active-passive system. On the other hand, the operational and licensing costs of an active-passive model are lower than that of an active-active deployment.

Note: See the *Oracle® Fusion Middleware Using Clusters* for Oracle WebLogic Server documentation for more information.

http://download.oracle.com/docs/cd/E15523_01/web.1111/e13709/toc.htm

bdi-<app> application and WebLogic Application Server Cluster

BDI uses the Receiver Service to transfer data from one system to another system. The BDI edge apps such as RXM, SIM can be configured in an active-active cluster mode to achieve better throughput.

In active-active cluster mode, bdi-rms application can send data to multiple instances of the bdi-rxm application simultaneously.

Deployment Architecture and Options

There are no physical location constraints on where bdi-<app> applications can be deployed as long as they are visible from the same network. But the decision on where to physically and logically locate your bdi-<app> applications has a huge impact on the high availability, performance and maintainability of your integration solution, so this decision must be given careful consideration.

Recommended Deployment Options

The BDI applications can be deployed in a variety of physical and logical configurations depending on the retailer's needs. Oracle Retail has the two recommended configuration alternatives.

Distributed

In this deployment, each of the BDI application (bdi-<app>.war) is deployed in a different WebLogic Application Server than the integrating application (<app>.ear) but it is physically close to the integrating application. This is the recommended configuration for production environment.

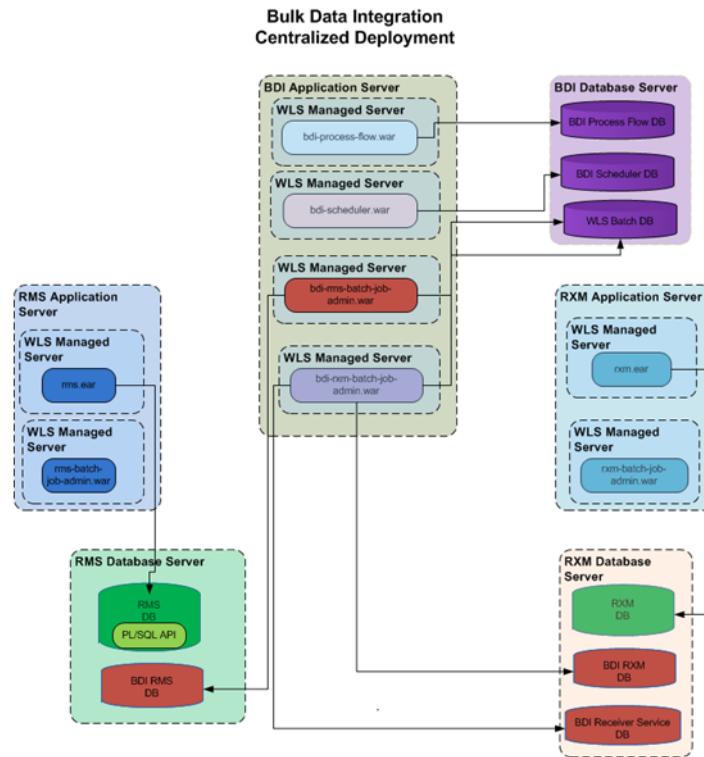
Figure 10–1 Distributed Configuration



Centralized

In this deployment, all bdi applications (bdi-<app>.ear) are deployed in a single WebLogic Application Server (not managed server instance) independent of where the Oracle Retail apps (<app>.ear) WebLogic Application Server is. This is an alternative configuration for non-production environments such as Dev.

Figure 10–2 Centralized Configuration



In all cases, the BDI application (`bdi-<app>.war`) should be deployed in its own managed server instance. It is not recommended to deploy multiple BDI applications into the same WLS managed server instance, or to have the BDI application (`bdi-<app>.war`) deployed into the same WLS managed server instance as the integrating application (`<app>.ear`). The configuration of deploying multiple `bdi-<app>`s in one managed server instance is not recommended or supported by WLS.

Implementation Process

This release of BDI defines the full life cycle of the BDI software product. The BDI life cycle and phases are described in detail in the software lifecycle management section of this document. For every life cycle phase and task that BDI defines, it provides corresponding tools and utilities to manage and operate on those phases.

There are several prerequisite steps that should be followed to have a successful BDI installation and deployment.

- Understand the BDI Core Concepts.
- Understand the deployment options.
- Understand the BDI life cycle.
- Understand the physical and logical requirements and limitations of the BDI Components.
- Understand the BDI Operational Considerations.

The process of implementation should follow these general steps:

- Work with the teams at your organization dedicated to Oracle Retail to coordinate plans for the number and type of environments needed (for example, Dev, Integration, Production).
- Each type of environment needs to be sized, deployed, and managed in conjunction with the implementation of the Oracle Retail applications. It is critical to understand the volume requirements of the production system so that the appropriate decisions can be made about the deployment option and the physical location and sizing.
- All deployments have integration to existing retailer systems. It is critical to understand the position of the BDI as it fits into the overall integration architecture and that the current operations and architecture team understand the BDI and its capabilities.
- Select a deployment option (distributed or centralized). This may be mixed depending on the phases of deployment. Development and test may be centralized and production distributed. Understand the operational complexities of each and plan for the staffing.
- Work with the application server administration teams to determine the physical and logical placement of the BDI components.
- Work with the system administrator and database administrator to appropriately place, size, and configure the file systems and databases.

Work with the system administrators to select the central BDI management location, bdi-home.

-
- The installation of the BDI has many prerequisites and dependencies that require the understanding, support and effort of database administrators, system administrators, application server administrators, and your organization's Oracle Retail application teams. It is a critical role of the BDI system administrator to work with each team, regardless of the site organization structure.
 - Create operational plans for the BDI life cycle.
 - Create plans for environment monitoring and maintenance.
 - Plan to performance test.

Performance Considerations

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

- The application server(s) topology and configuration.
- The BDI deployment approach.
- The hardware sizing and configuration of the BDI hosts.
- The hardware sizing and configuration of the applications that are connected to the BDI.

There are other factors that determine the performance of the overall system.

- Number of partitions and threads used by the batch jobs.
- Item-count and fetchSize used in the downloader-transporter batch job.
- Item-count used in the uploader batch job.
- Size of the data set

Performance Tuning Downloader-Transporter Jobs

Performance of the Downloader-Transporter job can be tuned using the following options.

- Partition
- Thread
- Item-count
- fetchSize

Default values for "Partition" and "Thread" are 10. The Downloader-Transporter job splits the data set rows among 10 partitions. If there are lot of rows in a data set, increasing partitions and threads allow more parallel processing of the data, and can improve the performance.

Keep the partition and thread values the same so that a thread is assigned to each partition by Batch runtime. If there are more partitions than threads, Batch runtime won't start a partition until a thread is available to run.

Partition and Thread values for the Downloader-Transporter job can be changed from the "Manage Configurations" tab of the Job Admin GUI. Partition and Thread values can be changed just for an interface module.

The Default value for "item-count" and "fetchSize" is 1000. Item-count is an attribute of "chunk" element and "fetchSize" is a property in the Downloader-Transporter job xml.

The Downloader-Transporter job reads 1000 records from the database and sends data to Receiver Service for the destination. If performance of a Downloader-Transporter job is not meeting expectations even after changing partitions and threads, increasing the "item-count" and "fetchSize" values may improve the performance as it reduces the number of round trips to the database.

Memory utilization will increase as you increase the "item-count" value.

Performance Tuning Uploader Jobs

Performance of an Uploader job can be tuned using the following options.

- Partition
- Thread
- Item-count

Default values for "Partition" and "Thread" are 10. The Uploader job splits the list of files among 10 partitions.

If a Downloader-Transporter job creates a lot of files, increasing partitions and threads allow parallel processing of more files, and can thus improve the performance.

Partition and thread values are typically the same so that a thread is assigned to each partition by batch runtime. If there are more partitions than threads, the batch runtime won't start a partition until a thread is available to run.

Partition and Thread values for the Uploader job can be changed from the "Manage Configurations" tab of the Job Admin GUI. Partition and Thread values can be changed just for an interface module.

The Default value for "item-count" is 1000. It is an attribute of the "chunk" element in the Uploader job xml. The Uploader job reads 1000 records from a file or list of files and then inserts/updates the data in the inbound table.

If performance of an Uploader job is not meeting expectations even after changing partitions and threads, increasing the "item-count" value may improve the performance as it reduces the number of round trips to the database. Memory utilization will increase as you increase the "item-count" value.

Process Schema

BDI_PROCESS_DEFINITION

Column	Type	Comments
PROCESS_NAME	VARCHAR2(255)	Name of the process
PROCESS_CREATE_TIME	TIMESTAMP	Timestamp when the process was loaded to database
PROCESS_DEF_CONTENT	CLOB	The Process Flow DSL

BDI_PROCESS_EXEC_INSTANCE

Column	Type	Comments
ACTIVITY_EXEC_ID	VARCHAR2(255)	System generated id for activity instance
ACTIVITY_BEGIN_OR_END	VARCHAR2(255)	"B" for Before Image, "A" for after image
ACTIVITY_EVENT_TIME	TIMESTAMP	Time when he activity occurred
ACTIVITY_NAME	VARCHAR2(255)	Name of the activity
ACTIVITY_SEQ_NBR	NUMBER	Sequence number of the activity
ACTIVITY_STATUS	NUMBER	Activity Status
PROCESS_EXECUTION_ID	VARCHAR2(255)	Process Execution Id of the process instance that initiated the activity
PROCESS_VARIABLES	BLOB	Serialized process variable map

BDI_SYSTEM_OPTIONS

Column	Type	Comments
VARIABLE_NAME	VARCHAR2(255)	Name of system variable
APP_TAG	VARCHAR2(255)	The application name
VARIABLE_VALUE	VARCHAR2(255)	Value of the variable



Process Flow REST Endpoints

REST Resource	HTTP Method	Description
/batch/processes/operator/{processName}	POST	Start a new Process Flow execution
/batch/processes/executions/{processName}	GET	List Process Executions for the process name
/batch/processes/executions	GET	List all process execution ids
/batch/processes/executions/status/{status}	GET	List all process execution ids for the specified status
/batch/processes/executions/time/{startTime}/{endTime}	GET	List all process execution ids for the specified time range
/batch/processes/external-variables	GET	List external variables
/batch/processes/external-variables	PUT	Create external variables
/batch/processes/external-variables	POST	Update external variables
/batch/processes/external-variables/{key}	DELETE	Delete external variable
batch/processes/currently-running-processes	GET	List all the currently running process flows
/batch/processes	GET	Get all the available process definitions
/batch/processes/{processName}	GET	Get process DSL for the specified process
/batch/processes/executions/{processName}/{processExecutionId}/activities/{activityExecutionId}	GET	Get Activity execution detail for the activity specified
/batch/processes/executions/{processName}/{processExecutionId}	GET	Get all the activities for the process flow execution
/batch/processes/{processName}/activities	GET	Get all the activities for the process specified
/batch/processes/operator/{processName}/{processExecutionId}	POST	Restart a process execution instance

System Setting Service

The System Setting service is a RESTful service available in all BDI apps (Job Admin, Process Flow and Scheduler) that provides endpoints to manage system option parameters and credentials to be used by the BDI apps. The system options are stored in the BDI_SYSTEM_OPTIONS table.

REST Resource	HTTP Method	Description
/system-setting/system-options	GET	Gets all system options from BDI_SYSTEM_OPTIONS table
/system-setting/system-options	PUT	Creates a system option in BDI_SYSTEM_OPTIONS table. Only the admin user is allowed to perform this operation.
/system-setting/system-options	POST	Updates a system option in the BDI_SYSTEM_OPTIONS table. Only admin user is allowed to perform this operation.
/system-setting/system-options/{key}	DELETE	Deletes a system option from BDI_SYSTEM_OPTIONS table. Only the admin user is allowed to perform this operation.
/system-setting/system-options/{key}	GET	Gets a system option from BDI_SYSTEM_OPTIONS table
/system-setting/system-logs	GET	Gets system logs
/system-setting/system-seed-data	GET	Gets system seed data file
/system-setting/system-seed-data/reset-after-bounce	POST	Resets system seed data after bounce
/system-setting/system-seed-data/reset-now	POST	Resets system seed data now
/system-setting/system-credentials	GET	Gets system credentials. Only the admin user is allowed to perform this operation.
/system-setting/system-credentials	PUT	Creates system credentials. Only the admin user is allowed to perform this operation.
/system-setting/system-credentials	POST	Updates system credentials. Only the admin user is allowed to perform this operation.

REST Resource	HTTP Method	Description
/system-setting/system-credentials/{key}	DELETE	Deletes system credentials. Only the admin user is allowed to perform this operation.

Sample Extractor - PL/SQL application code that calls procedures in PL/SQL package

```
BEGIN
-- First call beginDataSet of the corresponding interface module datactl pkg
before loading data to interface tables.
-- Here interfacemodule is Diff_Fnd and dataload is full set. If partial dataset,
then call beginPartialSet_Diff_Fnd
    IF Diff_Fnd_Out_DataCtl.beginFullSet_Diff_Fnd(O_datacontrol_id,O_error_
message) = 0 THEN
        DBMS_OUTPUT.PUT_LINE('interfaceModuleDataControlId: ' || O_
datacontrol_id);
    ELSE
        DBMS_OUTPUT.PUT_LINE('beginFullSet_Diff_Fnd error: ' || O_error_
message);
    Return;
    END IF;
-- Call application PL/SQL package to populate outbound interface table
-- Then call endDataSet of the corresponding interface module datactl pkg
    IF Diff_Fnd_Out_DataCtl.onSuccEndSet_Diff_Fnd(O_datacontrol_id,O_error_
message) = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Successfully called onSuccEndSet');
        COMMIT;
    ELSE
        DBMS_OUTPUT.PUT_LINE('onSuccEndSet error: ' || O_error_message);
        ROLLBACK;
    END IF;
EXCEPTION
WHEN OTHERS
THEN
    Call onErrDiscardSet in case of an error
    IF Diff_Fnd_Out_DataCtl.onErrDiscardSet_Diff_Fnd(O_datacontrol_id,O_
error_message) = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Successfully called onErrEndSet');
    ELSE
        DBMS_OUTPUT.PUT_LINE('onErrEndSet error: ' || O_error_message);
    END IF;
END;
```



Glossary

Batch	Batch is an industry metaphor for background bulk processing.
Batch Processing	Batch processing is the execution of a series of jobs in a program without manual intervention (non-interactive).
Batch Job	The series of steps in a batch process are often called a "job" or "batch job". A job contains one or more steps that specifies the sequence in which steps must be executed.
Batchlet	In Java Batch a Batchlet is type of batch step that can be used for any type of background processing that does not explicitly call for a chunk oriented approach.
Batch Service	Batch service is a RESTful service that provides endpoints to manage Batch Jobs in BDI. The Batch Service is part of Job Admin.
BDI	The Oracle Retail Bulk Data Integration Infrastructure (BDI) is an Enterprise level infrastructure product for moving bulk data between Sender Applications (for example RMS) and Receiver Applications (for example SIM, RXM).
Bulk Integration Flow	A bulk integration flow moves data for one family from source to destination application(s).
CSV file	Comma separated values file with .csv extension.
Data Service	Data Service is a RESTful service that is used to get data set information using job information.
Data Set	A data set consists of the rows between a begin and end sequence number in the interface table.
Data Set Type	Type of data set - FULL or PARTIAL
Downloader Data Control Table	The Downloader data control tables act as a handshake between the Extractor and Downloader
Downloader-Transporter Job	A Downloader-Transporter Job downloads a data set from Outbound Interface Tables for a family and streams data to a BDI destination application using the Receiver Service.
Extractor Job	An Extractor job extracts data for a family from sender (source) system and moves the data to Outbound Interface Tables.

Family	BDI data flows, identical to the other styles of Oracle Retail integration products, are organized by retail functional areas such as Store, Items, PO, Inventory and so on. These functional areas are called families (for example DiffGrp). Each family can contain one or more tables (for example DiffGrp and DiffGrp_Dtl).
fetchSize	Number of records fetched from the database and cached.
Importer	This is a destination application component that takes data from the inbound interface tables and updates the application tables.
Importer Job	The Importer Job imports data set for an Interface Module from Inbound Interface Tables into application specific transactional tables. Importer jobs are application (for example SIM/RXM) specific jobs.
Inbound Control Tables	Receiving applications use the data set metadata information in the importer control tables to trigger the import process.
Interface Module	Message family (for example DiffGrp_Fnd, InvAvailStore_Tx). An interface module can contain one or more interfaces (DiffGrp and DiffGrp_Dtl).
Interface Module XML File	Source for creating the DDL for the Interface Tables.
Interface Tables (Outbound and Inbound)	Interface tables are created in the integration schema of both on the sender side and receiver side. Sender side interface tables are called Outbound interface tables and receiver side tables are called Inbound interface tables.
item-count	Number of items read by ItemReader before ItemWriter writes.
ItemReader	ItemReader reads one item at a time from the source.
ItemWriter	After "item-count" number of items are read, Item Writer writes the items.
Job Admin	Web application for managing and monitoring batch jobs.
Job Operator	Job Operator provides an interface to manage jobs.
Job Repository	Job Repository holds information about jobs.
Job Specification language (JSL)	
Logical Partitions	A Data Set is divided into logical partitions and the data in each partition is downloaded by a separate thread. A Data Set is divided into logical partitions based on the number of partitions specified in the BDI_DWNLDR_TRNSMITTR_OPTIONS table and the number of rows in the data set.
Outbound Control Tables	Data Set metadata information is saved in database tables called the Outbound Control Tables in the BDI Integration schema of each Sender Application. An entry in BDI Outbound Control Tables indicates the availability of data set to the next component.
Receiver Application	Application that receives data from another application through BDI.

Receiver Service	This is the BDI component that receives the data from the Downloader-Transporter and stores it in a temporary storage
Receiver Side Split	<p>If there are multiple destinations that receive data from a Sender Application, this options is to use the Receiver Service at one destination to receive data from the sender and then multiple destinations use the data from one Receiver Service to upload to Inbound Interface Tables. The requirements for Receiver Side Split are such that:</p> <ul style="list-style-type: none"> ■ The Receiver Service database schema is shared by all the destinations. ■ The File system is shared by all destinations.
Seed Data	Seed data for Downloader-Transporter Jobs or Uploader job is loaded to the database during the deployment of Job Admin
Sender Application	Application that send data to other applications through BDI.
Sender Side Split	In the case of Sender Side Split (SSS), the data is extracted once from the source system. The extracted data is transmitted to each destination separately. Unlike point to point topology, the extraction is done only once regardless of the number of destinations.
Step	A step contains all the necessary logic and data to perform actual processing. A chunk-style step contains ItemReader, ItemProcessor and ItemWriter.
Uploader	The Uploader takes data from the temporary storage and populates the inbound interface tables.
Uploader Interface Module Data Control Table	This table acts as a handshake between Downloader and Uploader jobs.. An entry in this table indicates to Uploader Job that a data set is ready to be uploaded.
Uploader Job	An Uploader Job uploads data from CSV files into Inbound Interface Tables for an Interface Module. It divides files into logical partitions and each partition is processed concurrently.
