Oracle® Retail Bulk Data Integration Cloud Service

Implementation Guide – Concepts

Release 21.0.000

F38944-01

May 2021

ORACLE®

Oracle Retail Bulk Data Integration Cloud Service Implementation Guide – Concepts, Release 21.0.000

F38944-01

Primary Author: Sanal Parameswaran

Contributing Author: Nathan Young

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

4 Integration with External Applications

5 Job Admin UI

6 Process Flow

7 BDI Scheduler

Send Us Your Comments

Oracle Retail Bulk Data Integration Cloud Service Implementation Guide – Concepts, Release 21.0.000

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.

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, 19.0) or a later patch release (for example, 19.0.1). 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.

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

https://docs.oracle.com/en/industries/retail/index.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 Help Center (docs.oracle.com)

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

https://docs.oracle.com/en/industries/retail/index.html

(Data Model documents can be obtained through My Oracle Support.)

Conventions

The following text conventions are used in this document:

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

1
Introduction

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.

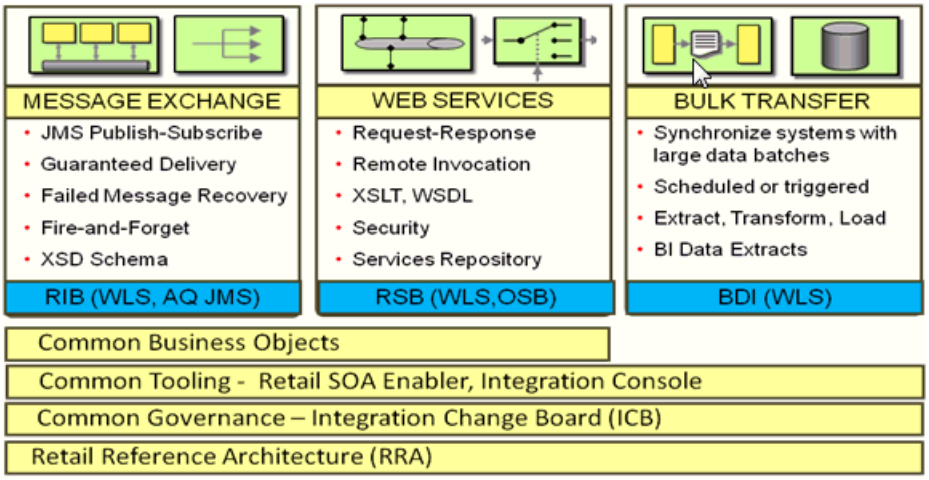
The movement of bulk data remains important because some work is best suited to being performed in bulk. Batch processing was there in the early days; it's still here today; and it will still be here tomorrow. What has changed is the approach to batch processing.

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/12213/wls/index.html

http://www.oracle.com/technetwork/middleware/fusion-middleware/documentation/index.html

---

Java Batch Overview

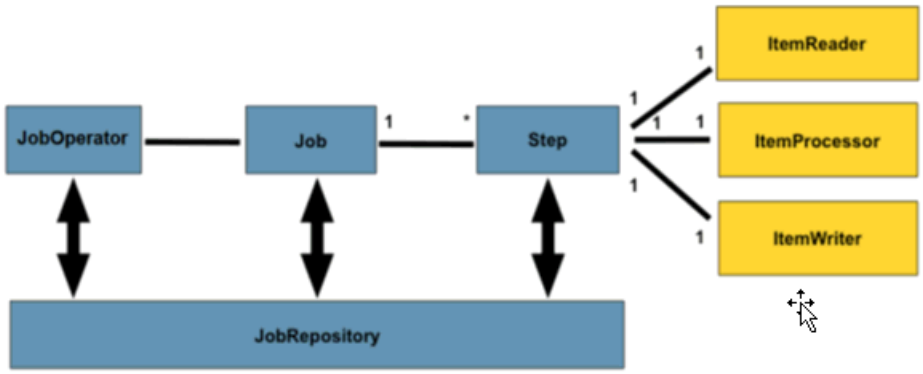
Batch processing for Java platform was introduced in Java EE 7. It provides a programming model for batch applications and a runtime to run and manage batch jobs.

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.

2

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 importer jobs. For example; RMS uses an instance of Job Admin to run extractor jobs whereas RPAS 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, RPAS 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, FileCreator, Uploader, and Importer. These activities are implemented as batch jobs.

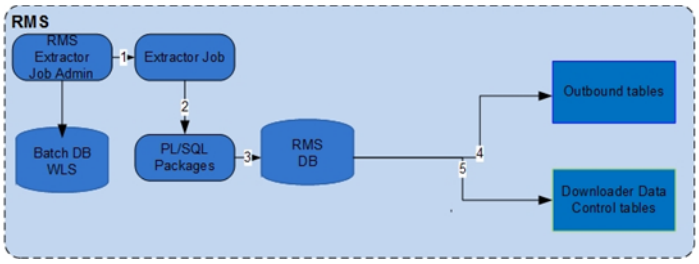
An Extractor Job extracts data for a Family from a source system and moves data to the outbound Interface Tables.

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 E.

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:

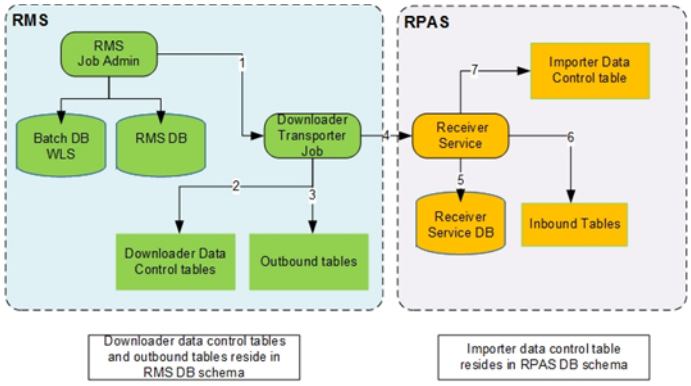
- BDI_DWNLDR_IFACE_MOD_DATA_CTL
- BDI_DWNLDR_IFACE_DATA_CTL

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. By default, the Receiver Service inserts the data directly into receiver and inbound tables.
7. The Receiver Service inserts an entry in the importer 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_TRNSMITTR_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 FALSE. 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
CREATE_TIME	TIMESTAMP	Time it was created
UPDATE_TIME	TIMESTMP	Time it was updated

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

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

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

```
Example:
MERGE INTO BDI_SYSTEM_OPTIONS USING DUAL ON (VARIABLE_NAME='rpasJobAdminBaseUrl'
and APP_TAG='rms-batch-job-admin.war') WHEN MATCHED THEN UPDATE SET VARIABLE_
VALUE='http://rxmhost:7001/rpas-batch-job-admin', UPDATE_TIME=SYSDATE WHEN NOT
MATCHED THEN INSERT (VARIABLE_NAME, APP_TAG, VARIABLE_VALUE, CREATE_TIME)
VALUES('rpasJobAdminBaseUrl', 'rms-batch-job-admin.war',
'http://rpashost:7001/rpas-batch-job-admin', SYSDATE)
```

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

Table 2–2 (Cont.) Interface Control

Column	Type	Comments
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.
COLUMN_FILTER	VARCHAR2(4000)	
ROW_FILTER	VARCHAR2(4000)	

Example:

```
MERGE INTO BDI_DWNLDR_TRNSMITTR_OPTIONS USING DUAL ON (ID=1) WHEN MATCHED THEN
UPDATE SET INTERFACE_MODULE='Diff_Fnd', INTERFACE_SHORT_NAME='Diff', RECVR_END_
POINT_URL='rpasJobAdminBaseUrl', RECVR_END_POINT_URL_
ALIAS='rpasJobAdminBaseUrlUserAlias', PARTITION=10, THREAD=10, QUERY_
TEMPLATE='select * from InterfaceShortName where (bdi_seq_id between ? and ?)
QueryFilter order by bdi_seq_id', AUTO_PURGE_DATA='TRUE'  WHEN NOT MATCHED THEN
INSERT (ID, INTERFACE_MODULE, INTERFACE_SHORT_NAME, RECVR_END_POINT_URL, RECVR_
END_POINT_URL_ALIAS, PARTITION, THREAD, QUERY_TEMPLATE, AUTO_PURGE_DATA) values
(1, 'Diff_Fnd', 'Diff', 'rpasJobAdminBaseUrl', 'rpasJobAdminBaseUrlUserAlias', 10,
10, 'select * from InterfaceShortName where (bdi_seq_id between ? and ?)
QueryFilter order by bdi_seq_id', 'TRUE')

MERGE INTO BDI_DWNLDR_TRNSMITTR_OPTIONS USING DUAL ON (ID=1) WHEN MATCHED THEN
UPDATE SET INTERFACE_MODULE='Diff_Fnd', INTERFACE_SHORT_NAME='Diff', RECVR_END_
POINT_URL='rpasJobAdminBaseUrl', RECVR_END_POINT_URL_
ALIAS='rpasJobAdminBaseUrlUserAlias', PARTITION=10, THREAD=10, QUERY_
TEMPLATE='select * from InterfaceShortName where (bdi_seq_id between ? and ?)
QueryFilter order by bdi_seq_id', AUTO_PURGE_DATA='TRUE'  WHEN NOT MATCHED THEN
INSERT (ID, INTERFACE_MODULE, INTERFACE_SHORT_NAME, RECVR_END_POINT_URL, RECVR_
END_POINT_URL_ALIAS, PARTITION, THREAD, QUERY_TEMPLATE, AUTO_PURGE_DATA) values
(2, 'Diff_Fnd', 'Diff', 'rpasJobAdminBaseUrl', 'rpasJobAdminBaseUrlUserAlias', 10,
10, 'select * from InterfaceShortName where (bdi_seq_id between ? and ?)
QueryFilter order by bdi_seq_id', 'TRUE')
```

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">

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 FALSE.

If sender side split topology is used, this flag needs to be changed to FALSE. Otherwise all destination applications may not get the data.

Auto Purge Delay

New system options have been added to introduce delay in purging data.

Auto Purge System Options for auto purge outbound data

autoPurgeOutboundData.global - Valid values are TRUE or FALSE

autoPurgeOutboundData.<Interface Module> - Interface module level system option that overrides global system option

autoPurgeOutboundDataDelay.global - The value of can be specified in days or hours.Examples are 24h = 24 hours or 30d = 30 days

autoPurgeOutboundDataDelay.<Interface Module> - Interface module level system option that overrides global system option. The value of can be specified in days or hours.Examples are 24h = 24 hours or 30d = 30 days

Auto Purge System Options for auto purge of inbound data

autoPurgeInboundData.global - Valid values are TRUE or FALSE

autoPurgeInboundData.<Interface Module> - Interface module level system option that overrides global system option

autoPurgeInboundDataDelay.global - The value of can be specified in days or hours.Examples are 24h = 24 hours or 30d = 30 days

autoPurgeInboundDataDelay.<Interface Module> - Interface module level system option that over-rides global system option. The value of can be specified in days or hours.Examples are 24h = 24 hours or 30d = 30 days

The value of autoPurgeDelay system option can be specified in days or hours.

Examples

30d - 30 days

24h - 24 hours

If "d" or "h" is not included in the value, then it is considered days.

The default autoPurgeDelay is 30 days. This value applies when autoPurgeDelay system option is not specified.

Extractor Cleanup Job

Extractor cleanup job purges data from outbound tables for data sets that have been successfully processed. It uses the above mentioned system options to decide whether to purge and when to purge data.

Importer Job

Importer job purges data from inbound tables for data sets that have been successfully processed. It uses the above mentioned system options to decide whether to purge and when to purge 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
CREATE_TIME	TIMESTAMP	Time when error occurred
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

Downloader-FileCreator Job

A DownloaderAndFileCreator job downloads data from outbound interface table for an interface module (family) and creates a file. It also creates a zero byte trigger file.

The names of the data and trigger files are specified as properties in the DownloaderAndFileCreator job. It copies the data and trigger files to outbound locations for destinations specified in "destination" property of the job. The outbound locations are specified in system options and the job derives the system option key using destination name. DownloaderAndFileCreator jobs are used by RMS Job Admin to create files for RPAS.

Example properties for Calendar_Fnd_DownloaderAdnFileCreatorToRpasJob

```
<property name="fileName" value="rms_clnd.csv.dat"/>
<property name="fileDataFormat" value="CSV"/>
<property name="triggerFileName" value="rms_clnd.complete"/>
<property name="destination" value="MFP,RDF,AP,IP"/>
```

DownloaderAndFileCreator job allows the order of the columns in the file and columns to be filtered. The columns specified in the columnFilter property are excluded in the file. Data in the file is in the order of the columns specified in columnOrder property.

Example properties for columnFilter and columnOrder. The delimiter for columnFilter is comma and for columnOrder is pipe character.

```
<property name="columnFilter" value="BDI_SEQ_ID,BDI_APP_NAME,BDI_DATASET_TYPE,BDI_
DATASET_ACTION"/>
<property name="columnOrder" value="TO_CHAR(Day,'YYYYMMDD')|TO_
CHAR(Week,'YYYYMMDD')|Month|Quarter|Half|Year|Week_of_Year|Day_of_Week"/>
```

DownloaderAndFileCreator job uses multiple job partitions to download data and create files. It then merges files from all partitions to create a single data file.

The following system options are used by the job for providing flexibility.

mergeFilesFlag

Files created by various partitions are merged by the job by default. Job won't merge files if this flag is set to False.

triggerFileFlag

Trigger file is created by the job by default. Job won't create trigger file if this flag is set to False.

copyFilesFlag

Files are copied to outbound locations by default. Job won't copy files to outbound locations if this flag is False.

overwriteOutboundFilesFlag

Files are not overwritten at outbound location by default. Job will overwrite files if this flag is set to True.

<Destination>_outboundLocation

Job uses this system option to find out the outbound location for a destination. If there are multiple destinations, then multiple system options need to be set.

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. By default, the Receiver Service inserts the data directly into receiver and inbound tables. There is an option to configure whether to transfer data to database or file system using system option property, receiverOutputType. If system option is not provided then receiver service defaults to database output type.

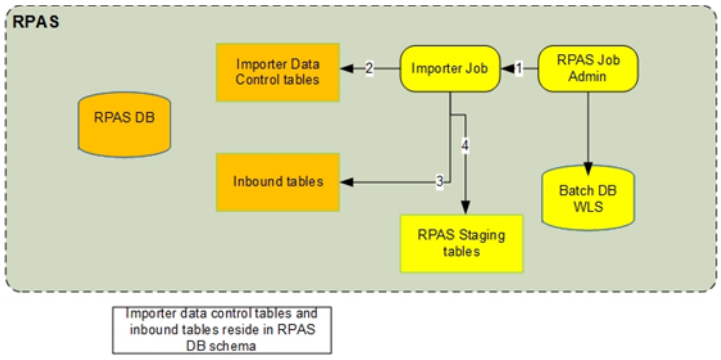
Importer Job

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

the Receiver Service completes processing a data set successfully, it creates an entry in these tables.

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

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



RPAS Importer

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

BDI_IMPRTR_IFACE_MOD_DATA_CTL

Importer File Creator Job

Importer File Creator job imports data from inbound table and creates a file at the outbound location. This job provides the functionality mentioned below.

- Column Filter - Filters data for columns specified in columnFilter property of the job.
- Row Filter - Filters data based on the predicate provided in the rowFilter property of the job
- Column Order - Column order can be specified in the columnOrder property of the job. If it is not specified, job uses order of the columns in the inbound table.
- Merge files - Merges files created for each partition. The system option "importerMergeFilesFlag" can be set to FALSE not to merge the files. By default, job merges the files.
- Copy files - Copies files to outbound location. The system option "importerCopyFilesFlag" can be set to FALSE not to copy the files. By default, job copies the files. Job expects the system option <Destination>_ importerOutboundLocation if importerCopyFilesFlag is set to TRUE.

- Purge Data - Purges data from inbound tables. AUTO_PURGE_DATA flag in BDI_IMPORTER_OPTIONS can be set to FALSE not to purge data after successful completion of the job. Seed data sets the flag to TRUE.
- Import File Location - By default, files are created in "bdi-data" folder. A system option <Job Name>.importFileLocation can be specified so that files are created in configured directory. A global system option "importFileLocation" can be specified for all importer file creator jobs.
- Job won't create any file if dataSetId is not specified
- If dataset has no data, then job will create an empty file.
- Job won't allow processing of a dataset if it is already processed.

Importer File Creator job is currently added only for OMS InvAvailWh_Tx.

Table 2–6 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	VARCHAR2(255)	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
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 2–7 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
IMPORTER_IFACE_MOD_DATACTL_ID	NUMBER	Foreign key to BDI_IMPRTR_IFACE_MOD_DATA_CTL table
SRC_SYS_INTERFACE_DATA_COUNT	NUMBER	
INTERFACE_DATA_COUNTNUMBER	NUMBER	

3

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.
- Telemetry Service - Ability to get job metrics for jobs that ran between fromTime and toTime.

Receiver Service

The Receiver Service is a RESTful service that provides various endpoints to send data transactionally. Receiver Service is part of Job Admin. Receiver Service uploads the data to either database or files. By default, the Receiver Service stores the data in the database. 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

Table 3–1 (Cont.) Receiver Options

Column	Type	Comments
INTERFACE_SHORT_NAME	VARCHAR2(255)	Name of the interface
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>_<Current Time in millis>_<Source System Name> 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

Parameter	Description
InterfaceModule)	Name of the interface module (for example Diff_Fnd
InterfaceShortName)	Name of the interface (for example Diff
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}/{interfaceMod-ule}/{interfaceShortName}/{partitionName}/{partitionBeginSeqNum}/{partitionEndSeqNum}/{beginBlockNumber}

Parameters:

Query Parameter: sourceSystemInterfaceDataCount

Sample Response:

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

Upload 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

Parameter	Description
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 "receiverTransmisionStatus" 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 "receiverTranasctionEndTime" 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_UPLDER_IFACE_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)

Table 3–2 (Cont.) Module Data Control

Column	Type	Comments
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/enable-disable	POST	Enable or disable the 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

Table 3–3 (Cont.) Batch Service

REST Resource	HTTP Method	Description
/batch/jobs/is-job-ready-to-start/{jobName}	GET	Gets job if ready to start
/batch/jobs/group-definitions	GET	Gets all group definitions
/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. If the given job is disabled it throws the exception "Cannot start the disabled Job {jobName}. Enable the Job and start it."

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": ""
}
```

Error Response in case of disable jobs

```
JSON
{
"statusCode": 500,
"status": "Internal Server Error",
"message": "Cannot start the disabled Job {jobName}. Enable the Job
     and start it.",
"stackTrace": "java.lang.RuntimeException:....."
}
```

Restart Job

This end point restarts a job asynchronously using the job execution id and returns the new job execution id. If the given job is disabled it throws the exception "Cannot restart the disabled Job {jobName}. Enable the Job and restart."

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": ""
}
```

Error Response in case of disable jobs

```
JSON
{
"statusCode": 500,
"status": "Internal Server Error",
"message": "Cannot restart the disabled Job {jobName}. Enable the
               Job and restart.",
"stackTrace": "java.lang.RuntimeException:....."
}
```

Enable or Disable the jobs

This endpoint enables or disables the jobs using jobId, jobEnableStatus and returns the jobId, status and message.

Path: /batch/jobs/enable-disable

HTTP Method: POST

Inputs

```
JSON
[
      {
      "jobId": "Diff_Fnd_DownloaderAndTransporterToRxmJob",
      "jobEnableStatus": "false"
   },
      {
      "jobId": "CodeHead_Fnd_DownloaderAndTransporterToSimJob",
      "jobEnableStatus": "true"
   }
]
```

Sample Request

http://localhost:7001/bdi-batch-job-admin/resources/batch/jobs/enable-disable

Successful Response

```
JSON
[
      {
      "jobId": "DiffGrp_Fnd_DownloaderAndTransporterToSimJob",
      "jobEnableStatus": "DISABLED",
      "message": "Job Disabled Successfully"
   },
  {
      "jobId": "CodeHead_Fnd_DownloaderAndTransporterToSimJob",
      "jobEnableStatus": "ENABLED",
      "message": "Job Enabled Successfully"
   }
]
```

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}/instances/{jobInstanceId}	GET	Gets a data set based on job name and job instance id
/data/dataset/{jobName}nextPending	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_DownloaderAndTransporterToRpasJob/nextPending

Sample Response

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

Telemetry Service

Telemetry Service is a RESTful service that provides end points to get job metrics information.

Table 3–5 Data Service

REST Resource	HTTP Method	Description
/telemetry/jobs/	GET	Gets job metrics based on fromTime and toTime
/telemetry/jobs/summary	GET	Gets job metrics summary based on fromTime and toTime

Job telemetry provides an end point to produce metrics for jobs that ran between "fromTime" and "toTime".

Path: /telemetry/jobs HTTP Method: GET Parameters:

fromTime - Query parameter

toTime - Query parameter

Sample Request

http://localhost:7001/<app>-batch-job-admin/resources/telemetry/jobs?fromTime=&toTime=

Sample Response

```
 Returns response in either XML or JSON format.

<job-runtime-monitoring-info data-requested-at="2018-11-08T00:44:57.113-05:00"
data-requested-from-time="2018-11-07T00:44:57.1-05:00"
data-requested-to-time="2018-11-08T00:44:57.1-05:00">
<jobs-server-runtime-info id="external-batch-job-admin.war" app-status="RUNNING"
```

```
up-since="69461" total-jobs-count="2" total-executions-count="2"
successful-executions-count="0" failed-executions-count="2">
<job name="Calendar_Fnd_ImporterJob" slowest-run-duration="0"
fastest-run-duration="0" avg-run-duration="0.0">
<executions execution_count="1" success_count="0" failure_count="1">
<execution execution-id="42" instance_id="42" status="FAILED"
startTime="2018-11-08T00:44:22-05:00" endTime="2018-11-08T00:44:22-05:00">
<step step-execution-id="42" name="batchlet-step" duration="0" status="FAILED"/>
</execution>
</executions>
</job>
</jobs-server-runtime-info>
</job-runtime-monitoring-info>
```

Job telemetry summary provides an end point to produce metrics for jobs summary information that ran between "fromTime" and "toTime".

Path: /telemetry/jobs/summaryHTTP Method: GET Parameters:

fromTime - Query parameter

toTime - Query parameter

Sample Request

http:// localhost:7001/
<app>-batch-job-admin/resources/telemetry/jobs/summary?fromTime= &toTime=

Sample Response

```
   Returns response in either XML or JSON format.

<jobExecutionsSummaryVo data-requested-at="2018-11-08T00:45:50.888-05:00"
data-requested-from-time="2018-11-07T00:45:50.887-05:00"
data-requested-to-time="2018-11-08T00:45:50.887-05:00">
<executions jobName="Calendar_Fnd_ImporterJob" executionId="42" instanceId="42"
status="FAILED" startTime="2018-11-08T00:44:22-05:00"
endTime="2018-11-08T00:44:22-05:00"averageDuration="0.0"/>
<executions jobName="Brand_Fnd_ImporterJob" executionId="41" instanceId="41"
status="FAILED" startTime="2018-11-08T00:43:59-05:00"
endTime="2018-11-08T00:44:02-05:00" averageDuration="0.0"/>
</jobExecutionsSummaryVo>
```

End Points for CRUD operations on Job XML

End points have been added to Batch Service to allow CRUD operations on Job XML.

CREATE Job XML

This end point creates an entry in BDI_JOB_DEFINITION table. It will throw an exception if job already exists.

HTTP Method: PUT

Path: /resources/batch/jobs/job-def-xml/{jobName}

Inputs:

Job Name (e.g. Diff_Fnd_ExtractorJob)

Job XML - It has to be valid XML that conforms to Job XML schema. The value of "id" in the XML should match "jobName" path parameter.

Update Job XML

This end point updates an entry in BDI_JOB_DEFINITION table. It will update if job is not in running state. This end point throws an exception if job doesn't exist in the table.

HTTP Method: POST

Path: /resources/batch/jobs/job-def-xml/{jobName}

Inputs:

Job Name (e.g. Diff_Fnd_ExtractorJob)

Job XML - It has to be valid XML that conforms to Job XML schema. The value of "id" in the XML should match "jobName" path parameter.

Delete Job XML

This end point deletes an entry in BDI_JOB_DEFINITION table. It will delete if job is not in running state and if there is no history in batch database.

HTTP Method: DELETE

Path: /resources/batch/jobs/job-def-xml/{jobName}

Inputs:

Job Name (e.g. Diff_Fnd_ExtractorJob)

Delete Job History

This end point deletes history for a job from batch database. It will delete history if job is not in running state.

HTTP Method: DELETE

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

Inputs:

Job Name (e.g. Diff_Fnd_ExtractorJob)

Bulk API for Batch Job CRUD Operations

ReST Endpoints have been introduced for Bulk create/update and Delete of Jobs in BDI/JOS Job admin application. There were existing end points to create/update/delete the individual jobs but it would be a great effort to use them sequentially for high number of jobs. A single end point service which can be used to take care of bulk create OR update operation for new job set has been added. Another end point for deleting multiple jobs has been added. Both the end points provide Job Level success and failure response for Create/Update/Delete.

End point for create/update job definitions

Http method:

Post

Path:

/batch/jobs/bulk/job-definitions

Consumes:

MediaType.APPLICATION_JSON/MediaType.APPLICATION_XML

Sample request:

http://localhost:7001/bdi-batch-job-admin/resources/batch/jobs/bulk/job-definitions

Input:

JSON

```
{"jobDefinitionsVo":[
{    "jobId": "Job11",
 "jobDefXml":
"PGpvYi-BpZD0iSm9iMTEiIHZlcnNpb249IjEuMCIgeG1sbnM9Imh0dHA6Ly94bWxucy5qY3Aub3JnL3ht
bC9ucy9qYXZhZWUiPgogIAgHyb3BlcnRpZXM==" },
{"jobId": "Job12",
 "jobDefXml":
"PGpvYi-BpZD0iSm9iMTIiIHZlcnNpb249IjEuMCIgeG1sbnM9Imh0dHA6Ly94bWxucy5qY3Aub3JnL3ht
bC9ucy9qYXZhZWUiPgoJPHByb3BlcnRpZXM+ " } ]
}
```

Successful Response:

If the result is complete success from endpoint then provide the list of jobDefinitionVo(List<JobDefinitionVo>) for customer reference.

JSON

```
{
     "jobDefinitionsVo": [
                {"jobId": "Job1",
                 "createTime": "create_timestamp",
                 "updateTime": "update_timestamp",
                 "status": "Success"
                },
                {"jobId": "Job2",
                 "createTime": "create_timestamp",
                 "updateTime": "update_timestamp",
                 "status": "Success"
                }
    ]
}
```

Error Response:

There would be individual calls for each job xml, if any job fails we can still process the next job and can show the list at the end for failed and success jobs.

Possible issues could be:

- Cannot update job XML if that job is running.
- Cannot create/update the Job if job id was not found in input job xml.
- Cannot create/update the Job if input job xml is not readable/parsable.

JSON

```
{
   "jobDefinitionsVo" [
                {"jobId": "Job1",
                "createTime": "create_timestamp",
                "updateTime": "update_timestamp"
                "status": "Success"
                },
               {"jobId": "Job2",
                "status": "Failure"
                "message": "Currently job is running"
```

```
                },
               {"jobId": "Job3",
                "status": "Failure"
                "message": "Job id not found in job xml"
                }
      ]
}
```

Exception Response

End point returns exception response if it is unable to process the request.

```
{
        "statusCode": 500
         "status": "Internal Server Error"
          "message": "Unable to process"
          "stackTrace": ""
}
```

End point for delete job definitions

Http method:

Delete

Path:

/batch/jobs/bulk/job-definitions

Consumes:

MediaType.APPLICATION_JSON/MediaType.APPLICATION_XML

Sample request:

http://localhost:7001/bdi-batch-job-admin/resources/batch/jobs/bulk/job-definitions

Input:

JSON

```
{
 "jobDefinitionsVo":  [
{"jobId":"Job1"},
{"jobId":"Job2"}
  ]
}
```

Successful Response:

If the result is complete success from endpoint then provide the list of jobDefinition-Vo(List<JobDefinitionVo>) for customer reference.

JSON

```
{
    "jobDefinitionsVo":  [
                    {"jobId": "Job1",
                     "deleteTime": "delete_timestamp",
                     "status": "Success"
                    },
                    {"jobID": "Job2",
                     "deleteTime": "delete_timestamp",
                     "status": "Success"
```

```
                    },
   ]
}
```

Error Response:

There would be individual calls for each job xml, if any job fails to delete we can still process the next job and can show the list at the end for failed and success jobs.

Possible issues could be:

- Can't delete job if that job is running.
- Can't delete the Job if job id was not found in existing job list.
- Can't delete the Job if job is in enabled status.
- Can't delete the Job if input job list is not readable/parsable.

JSON

```
{
     "jobDefinitionsVo":  [
                    {"jobID": "Job1",
                     "deleteTime": "delete_timestamp",
                     "status": "Success"
                    },
                    {"JobId": "Job2",
                     "status": "Failure",
                     "message": "Currently job is running"
                    },
                    {"jobID": "Job3",
                     "status": "Failure",
                     "message": "Cant delete job XML as job job3 is not valid. "
                    }
       ]
}
```

Exception Response

End point returns exception response if it's unable to process the main request.

JSON

```
{
        "statusCode": 500
        "status": "Internal Server Error"
        "message": "Unable to process"
        "stackTrace": ""
}
```

Bulk Data Export Service

Bulk Data Export Service is a new Restful Web service packaged under Job Admin, which allows authenticated user to get latest available data sets (either FULL or PARTIAL), fetch data for available data set ids and update status of transaction.

Below are the endpoints provided by Bulk Data Export Service. Ensure to invoke endpoints in below order to export data successfully.

Table 3–6 Bulk Data Export Service

REST Resource	HTTP Method	Description
j9/bulk-data/export/data-set/{interfcace_module}?consumingAppName={consumingAppName}&sourceAppName=rms	GET	Gets latest available FULL or PARTIAL dataset for given interface module
/bulk-data/export/begin?dataSetId={dataset_id}&consumingAppName={consumingAppName }	POST	Generates transaction id for given unprocessed transaction id.
/bulk-data/export/data?transactionId={transaction_id}&page={pageNum}&maxRowsPerPage={maxRowsPerPage}	GET	Gets data from inbound tables for provided transaction id.
/bulk-data/export/begin?dataSetId=15&consumingAppName=external	POST	Updates transaction status as SUCCESS/FAILURE based on input provided.

Endpoints

Ping

This endpoint can be used to ensure that initial handshake with service is successful.

HTTP Method: GET

Path

resources/bulk-data/export/ping

Produces:

MediaType.APPLICATION_JSON/MediaType.APPLICATION_XML

Sample request:

<hostname>:<port>/<app_name>-batch-job-admin/resources/bulk-data/export/ping

Sample json response:

```
{
   "returnCode": "PING_SUCCESSFUL",
   "messageDetail": "Pinged Bulk Data Export Service successfully"
}
```

getAllAvailableFullOrPartialDataSet

This endpoint can be used to get latest available data set. Data sets of type FULL are given priority over datasets of type PARTIAL. If there are no available datasets of type FULL, then this endpoint checks for data sets of type PARTIAL. If there are multiple PARTIAL data sets for given input, then data sets are sorted based on data set ready time before returning to consumer.

Path parameter consumingAppName and query parameter sourceAppName are not case sensitive.

Path parameter interfaceModule is case sensitive. Ensure to maintain appropriate case for interface module in request.

HTTP Method: GET

Path

resources/bulk-data/export/data-set/{interfaceModule is mandatory}?consumingAppName={consuming app name is mandatory}&sourceAppName={sourceAppName is optional}

interfaceModule : This field indicates name of interface module for which latest available data set is to be returned.

consumingAppName : This field indicates requesting system name.

sourceAppName : This field indicates source system name of data set.

Note: interfaceModule and consumingAppName are mandatory inputs.

Produces

MediaType.APPLICATION_JSON/MediaType.APPLICATION_XML

Sample request

<hostname>:<port>/<app_name>-batch-job-admin/resources/bulk-data/export/data-set/Diff_Fnd?consumingAppName=external&sourceAppName=RMS

Sample json response

```
{
   "Message":    {
      "returnCode": "DATA_SET_AVAILABLE",
      "messageDetail": "data set is available for external"
   },
   "interfaceModule": "Diff_Fnd",
   "dataSetType": "FULL",
   "sourceSystemName": "RMS",
   "consumingAppName": "external",
   "AvailableDataSets": {"AvailableDataSet": [   {
      "dataSetId": 10,
      "dataSetReadyTime": "2020-03-06T02:21:25-08:00",
      "InterfaceDetail":       [
                  {
            "interfaceShortName": "DIFF_GRP_DTL",
            "interfaceDataCount": 100
         },
                  {
            "interfaceShortName": "DIFF_GRP",
            "interfaceDataCount": 200
         }
      ]
   }]}
}
```

Error Response

Response when latest data is not available

```
{"Message": {
   "returnCode": "NO_AVAILABLE_DATA_SET",
   "messageDetail": "No data set is available for external"
}}
```

beginExport

This endpoint is used to indicate start of data export. If provided query parameters are valid, then this endpoint generates a transaction id indicating start of transaction, which is required to get data.

If there are multiple datasets (of type PARTIAL) returned by getAllAvailableFullOrPartialDataSet endpoint, ensure to process them in the same order.

HTTP Method: POST

Path

resources/bulk-data/export/begin?dataSetId={dataSetId is mandatory}&consumingAppName={consuming app name is mandatory}

dataSetId: This field indicates the dataset id that is to be exported.

consumingAppName: This field indicates the consuming application name.

> **Note:** dataSetId and consumingAppName is mandatory input for this endpoint

Produces

MediaType.APPLICATION_JSON/MediaType.APPLICATION_XML

Sample request

<hostname>:<port>/<app_name>-batch-job-admin/resources/bulk-data/export/begin?dataSetId=10&consumingAppName=external

Sample json response

```
{
   "Message":    {
      "returnCode": "TRANSACTION_STARTED",
      "messageDetail": "Transaction started for dataset Id 10"
   },
   "transactionid": "external:Diff_Fnd:10:1585633498380"
}
```

getData

This endpoint can be used to get data from inbound tables. It uses transaction id provided in input to identify start and end sequence of available dataset for interface module. This end point uses pagination to fetch records. Query parameters "page" and "maxRowsPerPage" can be used to specify page number and number of records per page respectively.

"hasMorePages" field from response can be used to identify if there are more records to be fetched for dataset. True value indicates that there are more records to be fetched and false value indicates that current page is the last page.

"action" node in response signifies the action to be performed on "data" that is returned.

For example, "action:REPLACE" signifies that replace if existing records with the current one that is returned by getData endpoint.

"next" field from response can be used to get available data from next page.

page and maxRowsPerPage are optional query parameters.

If query parameter page is missing in request, then value of it will be defaulted to 1 by Bulk Data Export Service. If query parameter maxRowsPerPage is missing in request, then value of it will be defaulted to 500 records by Bulk Data Export Service

HTTP Method: GET

Path

resources/bulk-data/export/data?transactionId={transactionId is mandatory}&page={page}&maxRowsPerPage=500

transactionId: Provide transaction id generated by beginExport endpoint as value of this field.

page: This is an optional field. Ensure that value of this field is a numeric value.

Value will be defaulted to 1, if not specified in request.

maxRowsPerPage: This is an optional field. Ensure that value of this field is a numeric value.

Value will be defaulted to 500, if not specified in request

Note: transactionId is mandatory input for this endpoint

Maximum number of records that can be returned by getData end point is controlled through an entry in SYSTEM_OPTIONS table called "ROWS_PER_PAGE_MAX_VALUE".

If value of query parameter maxRowsPerPage is greater than ROWS_PER_PAGE_MAX_VALUE, then maxRowsPerPage is defaulted to value of entry "ROWS_PER_PAGE_MAX_VALUE" in SYSTEM_OPTIONS table.

If SYSTEM_OPTIONS table misses entry for ROWS_PER_PAGE_MAX_VALUE, then ROWS_PER_PAGE_MAX_VALUE is defaulted to 10000 by Bulk Data Export Service.

Refer section "Steps to update values in System Options" to add/update ROWS_PER_PAGE_MAX_VALUE in SYSTEM_OPTIONS table.

Produces

MediaType.APPLICATION_JSON/MediaType.APPLICATION_XML

Sample request

<host>:<port>/<app_name>-batch-job-admin/resources/bulk-data/export/data?transactionId=external:Diff_Fnd:10:1585743226654&page=1&maxRowsPerPage=100

Sample Json Response:

```
{
   "Message":    {
      "returnCode": "GET_DATA_SUCCESSFUL",
      "messageDetail": "Records are available for interface module Diff_Fnd"
   },
   "hasMorePages": true,
   "next":
"http://blr00ccp:55663/rms-batch-job-admin/resources/bulk-data/export/data?transac
tionId=external:Diff_Fnd:10:1585743226654&page=2&maxRowsPerPage=5",
   "interfaceModule": "Diff_Fnd",
   "InterfaceData":    [
```

```
            {
         "interfaceShortName": "DIFF_GRP_DTL",
         "columnNames": "DIFF_GROUP_ID,DIFF_ID,DISPLAY_SEQ",
         "recordCount": 5,
         "Records": {"Record":          [
                        {
               "action": "REPLACE",
               "data": "AAA1,AAAAAAAA,11"
            },
                        {
               "action": "REPLACE",
               "data": "AAA1,AAAAAAAA,12"
            },
                        {
               "action": "REPLACE",
               "data": "AAA1,AAAAAAAA,13"
            },
                        {
               "action": "REPLACE",
               "data": "AAA1,AAAAAAAA,14"
            },
                        {
               "action": "REPLACE",
               "data": "AAA1,AAAAAAAA,15"
            }
         ]},
         "dataSetInterfacesDataVo": {"Record":          [
                        {
               "action": "REPLACE",
               "data": "AAA1,AAAAAAAA,11"
            },
                        {
               "action": "REPLACE",
               "data": "AAA1,AAAAAAAA,12"
            },
                        {
               "action": "REPLACE",
               "data": "AAA1,AAAAAAAA,13"
            },
                        {
               "action": "REPLACE",
               "data": "AAA1,AAAAAAAA,14"
            },
                        {
               "action": "REPLACE",
               "data": "AAA1,AAAAAAAA,15"
            }
         ]}
      },
            {
         "interfaceShortName": "DIFF_GRP",
         "columnNames": "DIFF_GROUP_DESC,DIFF_GROUP_ID,DIFF_TYPE_DESC,DIFF_TYPE_
ID",
         "recordCount": 5,
         "Records": {"Record":          [
                        {
               "action": "REPLACE",
               "data": "Test Value,AAA1,Test Value,seq"
            },
                        {
```

```
                "action": "REPLACE",
                "data": "Test Value,AAA1,Test Value,seq"
            },
                        {
                "action": "REPLACE",
                "data": "Test Value,AAA1,Test Value,seq"
            },
                        {
                "action": "REPLACE",
                "data": "Test Value,AAA1,Test Value,seq"
            },
                        {
                "action": "REPLACE",
                "data": "Test Value,AAA1,Test Value,seq"
            }
        ]},
        "dataSetInterfacesDataVo": {"Record":          [
                        {
                "action": "REPLACE",
                "data": "Test Value,AAA1,Test Value,seq"
            },
                        {
                "action": "REPLACE",
                "data": "Test Value,AAA1,Test Value,seq"
            },
                        {
                "action": "REPLACE",
                "data": "Test Value,AAA1,Test Value,seq"
            },
                        {
                "action": "REPLACE",
                "data": "Test Value,AAA1,Test Value,seq"
            },
                        {
                "action": "REPLACE",
                "data": "Test Value,AAA1,Test Value,seq"
            },
                        {
                "action": "REPLACE",
                "data": "Test Value,AAA1,Test Value,seq"
            }
        ]}
      }
   ],
   "currentPageNumber": 1
}
```


Error Response:


```
{
   "Message": {
      "returnCode": "GET_DATA_FAILED",
      "messageDetail": "Data set is already processed for dataset id .10"
   },
   "hasMorePages": "false",
   "currentPageNumber": "0"
}
```


endExport

This endpoint can be used to mark status of data export for a given transaction. It accepts status of transaction as query parameter.Transaction status will be defaulted to SUCCESS, if not specified in input.

List of acceptable values for transaction status are "SUCCESS" and "FAILURE"

HTTP Method: POST

Path

resources/bulkdata/export/end?transactionId={transaction id mandatory}&transactionStatus={optional, default to success}

transactionId : Provide transaction id generated by beginExport endpoint as input to this field.

transactionStatus: This is an optional input. Value will be defaulted to SUCCESS, if not specified in input.

List of acceptable values for transaction status are "SUCCESS" and "FAILURE"

Any value apart from above two will result in exception.

> **Note:** transactionId is mandatory input to this end point

Produces

MediaType.APPLICATION_JSON/MediaType.APPLICATION_XML

Sample request

<host>:<port>/<app_name>-batch-job-admin/resources/bulk-data/export/end?transactionId=external:Diff_Fnd:10:1585633498380&transactionStatus=SUCCESS

Sample json response

When transactionStatus=SUCCESS

```
{
   "returnCode": "TRANSACTION_COMPLETE",
   "messageDetail": "Transaction completed successfully."
}
```

When transactionStatus= FAILURE

```
{
   "returnCode": "TRANSACTION_FAILED",
   "messageDetail": "Transaction could not be completed.."
}
```

Error Response

Invalid input for transaction Id:

```
{
   "returnCode": "TRANSACTION_ERROR",
   "messageDetail": "Transaction status could not be parsed. Provide one of the
following value {SUCCESS,FAILURE}"
}
```

Steps to test Bulk data export service:

Attached script can be used to test Bulk data export service.

Steps to execute the script:

1. Open the script using any editor or command line.
2. Replace the hostname and port number to point deployed app.
3. Can modify the consumingAppName, by default the script has it as "external"
4. Save the script.
5. Run the script with interface name. Below is the command to run the script.

Note: Need to run using bash only. The script only works for a full data set.

bash bdi_poc.sh <Interface_Name> ---- Pass interface name for Full data set.

Example - bash bdi_poc.sh Clearance_Tx

6. Script will prompt for deployed app username and password.

Auto Purge Inbound Data

A thread runs every hour to identify data sets to be purged based on flags in system options table and purges the data from inbound tables.

Eligible data sets are purged every hour from inbound table. Data set is said to be eligible for purge, if either transaction completion date has crossed predefined time interval or data set ready time has crossed predefined time interval.

Below are steps followed to identify data sets to be purged and purge records from inbound data tables automatically.

- Auto purge delay can be allowed/restricted for any specific interface by adding autoPurgeInboundData.{interface_ModuleName} flag in SYSTEM_OPTIONS table with value TRUE/FALSE respetively.

  Example: autoPurgeInboundData.Diff_Fnd

  If interface level entry is missing in SYSTEM_OPTIONS table, then value of global flag autoPurgeInboundData.global is used to decide whether to purge data for interface module or not.

- For each ACTIVE interface modules, check if auto purge flag is set at global level (autoPurgeInboundData.global) or interface module level (autoPurgeInboundData.{interface_ModuleName}) from SYSTEM_OPTIONS table. If flag is TRUE either at global level or at any interface module level, continue with the next steps.

  If an entry for auto purge flag is missing in SYSTEM_OPTIONS table, then value of flag at global level (autoPurgeInboundData.global) is defaulted to TRUE.

- Prepare a list with interface module names that are eligible to be purged based on value of above flags and get list of datasets that already processed for the interface module list.

- Get delay duration for each eligible transaction from SYSTEM_OPTIONS table. Check for entry "autoPurgeInboundDataDelay.{interface_name}" in SYSTEM_OPTIONS table to get delay duration for each eligible transaction.

  If no entry is available for interface module, then get global delay duration using entry "autoPurgeInboundDataDelay.global" of SYSTEM_OPTIONS table.

  Use default delay duration(30 days) if SYSTEM_OPTIONS table does not contain any entries for auto purge delay. Delay duration can specified either in hours or days. Refer table below for possible values for delay. Check if data is ready to be purged based on delay obtained. Data is ready to be purged if duration between transaction begin time and current time is greater than or equal to delay

- If list from above step is not empty, then use the below query to get list of data sets that are already processed and ready to be purged. If there are data sets eligible to be purged, proceed with next step, exit otherwise.

Query Template: select tr from ImporterExecutionDataSet dataset, ImporterTransaction tr, ImporterInterfaceModuleDataControl dmc

where dataset.importerTransactionId = tr.id AND tr.interfaceModule IN :interfaceNameList

AND (tr.importerTransactionStatus = IMPORTER_COMPLETED)

AND dataset.importerInterfaceModuleDataControlId = dmc.importerInterfaceModuleDataControlId AND dmc.dataSetDeleteTime IS NULL

- Check if there are any data sets that was ready to be consumed before predifined number of days. Data set is considered for purging, if data is set is older than AUTO_PURGE_DELAY_UPPER_LIMIT entry of SYSTEM_OPTIONS table.

  Query Template:

  select modctl

  FROM ImporterInterfaceModuleDataControl modctl, ImporterInterfaceDataControl ctl

  WHERE modctl.dataSetDeleteTime IS NULL

  AND ctl.importerInterfaceModuleDataControl.importerInterfaceModuleDataControlId = modctl.importerInterfaceModuleDataControlId

  AND modctl.dataSetDeleteTime IS NULL

  AND modctl.dataSetReadyTime < :autoDelayLimit

  Note : autoDelayLimit is the date obtained by subtracting number of days specified in AUTO_PURGE_DELAY_UPPER_LIMIT option of System Options table with current date.

- For all the data sets that are ready to be purged, get importer data control details(like begin sequence number, end sequence number, dataset id etc) using transaction ids

  Query Template:

  SELECT modctl FROM ImporterExecutionDataSet dataset, ImporterInterfaceModuleDataControl modctl

  WHERE modctl.importerInterfaceModuleDataControlId = dataset.importerInterfaceModuleDataControlId

  AND dataset.importerTransactionId IN :transactionIdList

- If there are data sets eligible to be purged from previous steps, then prepare a consolidated list of data sets to purge them and proceed with next steps, exit otherwise.

  Purge records from Inbound tables using details fetched from last step.

  Query Template:

  delete from {InterfaceShortName} where (bdi_seq_id between ? and ?)

- Update DATA_SET_DELETE_TIME column of BDI_IMPRTR_IFACE_MOD_DATA_CTL table with current time stamp for the datasets whose inbound data are purged.

Table 3–7 BDI Auto Purge Inbound data

Name	Allowed Values	Comments
autoPurgeInboundDataDelayUpperLimit	45	This flag is used to specify maximum number of days a data set that is unconsumed, can be persisted. After exceeding maximum number of days, data is purged even if it is unconsumed. Ensure that value of this field is a numeric value
autoPurgeInboundData.global	TRUE FALSE	This flag is used to enable/disable purging of records if flag is missing at interface level
autoPurgeInboundData.{interface_ModuleName}	TRUE FALSE	Replace {interface_ModuleName} with name of interface module to restrict/enable purging of records for given interface Example: autoPurgeInboundData.Diff_Fnd
autoPurgeInboundDataDelay.global	30d 20h 20	This flag is used to fetch delay duration if flag is missing at interface level h: Numeric value followed by h indicates duration in hours. d: Numeric value followed by d indicates duration in days. If neither d nor h follows numeric value or numeric value followed by non numeric value other than d or h, delay is considered to be in days. Example: 20 20a Ensure that value of this field is a numeric value followed by d or h

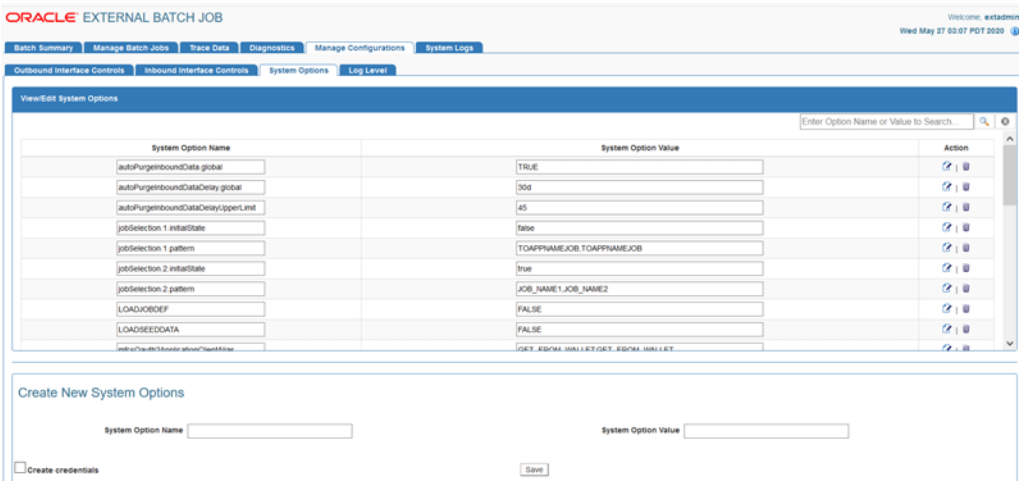
Table 3–7 (Cont.) BDI Auto Purge Inbound data

Name	Allowed Values	Comments
autoPurgeInboundDataDelay.{interface_name}	30d 20h 20	Replace {interface_ModuleName} with name of interface module to specify delay duration specific to interface. h: Numeric value followed by h indicates duration in hours. Example: 20h d: Numeric value followed by d indicates duration in days. Example:30d If neither d nor h follows numeric value or numeric value followed by non numeric value other than d or h, delay is considered to be in days. Example: 20 20a Ensure that value of this field is a numeric value followed by d or h Example: autoPurgeInboundDataDelay.Diff_Fnd

Steps to update values in System Options

Approach 1: Update via UI

1. Login to the job admin web page with valid credentials.
2. Click the Manage Configurations tab.
3. Click the System options tab.



4. Search for the system option to update. Example: autoPurgeInboundData.global, autoPurgeInboundDataDelay.global, autoPurgeInboundDataDelayUpperLimit etc.
5. Click the Edit icon to update the value and click the Save icon to save changes.

Approach 2: Update values via REST service endpoints

Below end point can also be used to update System Options via REST service.

URL:

http://<host:port>/<appName>-batch-job-admin/resources/system-setting/system-options

Request Type :

POST

Input Json Example:

```
{
      "key": "autoPurgeInboundData.global",
      "value": "false"
    }
```

Note: Update key and value in above request with appropriate System Option name and value to be updated.

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.

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 Customization

During the deployment of Job Admin, seed data is loaded to various tables. Seed data files are located in the bdi-edge-<app>-job-home/setup-data/dml folder. If seed data is changed, Job Admin must be reinstalled and redeployed. In order to load seed data again during the redeployment, the LOADSEEDDATA flag in the BDI_SYSTEM_ OPTIONS table must be set to TRUE.

During the deployment, Job XMLs get loaded to BDI_JOB_DEFINITION table. Job XML files are located in the "jos-job-home/setup-data/META-INF/batch-jobs" folder. If job xmls are changed, Job Admin must be reinstalled and redeployed. In order to load job xmls during redeployment, the LOADJOBDEF flag in the BDI_SYSTEM_ OPTIONS table must be set to TRUE.

Note: Restart of job does not load job definition from the BDI_JOB_DEFINITION table. Java Batch loads job xml from JOBSTATUS table during the restart of a job.

If there is an issue with Job XML, job needs to be started after fixing the job XML.

Throttling

Throttling is the capability of regulating the rate of input for a system where output rate is slower than input.

Java Batch runtime will not allow to run multiple instances of a job at same time, it will say job is currently running and fail the job. There can be only one instance of a job at any time (unless the job parameters are different).

Throttling is introduced to address the issue caused when there are many job start requests at the same time. In order to honor the throttle limits "throttleSystemLimit" is introduced to make sure the system never runs more than the throttle limit for the group and the system.

Three new tables are added to job schema to handle throttling, these are BDI_GROUP, BDI_GROUP_LOCK, BDI_GROUP_MEMBER.

Table 3–8 BDI Group

Column	Type	Comments
GROUP_ID	NUMBER	Primary Key
APP_TAG	VARCHAR2(255)	Name of the application
COMMENTS	VARCHAR2(255)	Comments
GROUP_ATTRIB_NAME_1	VARCHAR2(255)	Name of the group attribute ex - THROTTLE_JOBS_IN_SAME_GROUP
GROUP_ATTRIB_NAME_2	VARCHAR2(255)	Name of the group attribute
GROUP_ATTRIB_NAME_3	VARCHAR2(255)	Name of the group attribute
GROUP_ATTRIB_NAME_4	VARCHAR2(255)	Name of the group attribute
GROUP_ATTRIB_NAME_5	VARCHAR2(255)	Name of the group attribute
GROUP_ATTRIB_VAL_1	VARCHAR2(255)	Value of the group attribute
GROUP_ATTRIB_VAL_2	VARCHAR2(255)	Value of the group attribute
GROUP_ATTRIB_VAL_3	VARCHAR2(255)	Value of the group attribute
GROUP_ATTRIB_VAL_4	VARCHAR2(255)	Value of the group attribute
GROUP_ATTRIB_VAL_5	VARCHAR2(255)	Value of the group attribute
GROUP_NAME	VARCHAR2(255)	Name of the group

Table 3–9 BDI Group Member

Column	Type	Comments
GROUP_MEMBER_ID	NUMBER	Primary Key
APP_TAG	VARCHAR2(255)	Name of the application
GROUP_ID	NUMBER	Group id

Table 3–9 (Cont.) BDI Group Member

Column	Type	Comments
MEMBER_NAME	VARCHAR2(255)	Name of the job
MEMBER_TYPE	VARCHAR2(255)	Type of the member ex - job

Table 3–10 BDI Group Lock

Column	Type	Comments
LOCK_ID	NUMBER	Primary Key
APP_TAG	VARCHAR2(255)	Name of the application
GROUP_NAME	VARCHAR2(255)	Name of the group

Prerequisite: In weblogic console, make sure the job admin data source has "Supports Global Transactions" and "Logging Last Resource" checked in the Transaction tab.

Example on how throttling is handled at runtime:

Group1 <--(job1, job2, job3)-Throttle value 3

Group2 <-- (job1, job2) - Throttle value2

Step1:

Start job1, job2, job3 from process1, process2, process3 respectively. All 3 start running.

Step2:

Then start again process1 and process2. Both job1 and job2 get throttled.

There can be only one instance of a job at any time (unless the job parameters are different).

BDI Global Migration Script (BDI_Database_Util_Spec)

To make a re-runnable sql script which globally works for all sqls that makes a sql script re-runnable, there should be a script which creates package with functions/procedures that returns a necessary result value to the calling function to check whether anything exists in the schema before proceeding to execute the query on the schema. Below are the list of functions/procedures used in the package.

Table 3–11 BDI Global Migration Script Function/Procedure Names and Descriptions

Function/Procedure Name	Description
checkTableExists	It checks whether table exists or not in the schema.
checkColumnExists	It checks whether column of a table exists or not in the schema.
checkConstraintExists	It checks whether constraint of table exists or not in the schema.
checkIndexExists	It checks whether index exists for a table or not in the schema.
createTable	It creates table only after checking whether table exists or not in the schema.
AddColumnToTable	It adds column only after checking whether table and column exists or not in the schema.

Table 3–11 (Cont.) BDI Global Migration Script Function/Procedure Names and

Function/Procedure Name	Description
modifyColumnType	It modifies column datatype/any other related to column changes like making column not null or primary key.
updateColumnName	It updates column name of the table.
AddTableConstraint	It adds constraint to table only after checking whether table and constraint exists or not in the schema.
createIndex	It checks whether index exists or not on table in the schema before index is created.
createSequence	It checks whether sequence exists or not on table in the schema before sequence is created.

Note: Before running the global migration script you need to provide three permissions to package.

Permissions are:

- CREATE ANY TABLE TO 'user_schema',
- CREATE ANY SEQUENCE TO 'user-schema'
- CREATE ANY INDEX To 'user_schema'

DB Schema Auto Migration

Schema migration is to make changes to database like alter column, adding new table on running the migration script. Previously, migration scripts are run manually one by one on sqlplus or sqldeveloper to migrate from one to another version. Now, with the db schema auto migration feature with no user intervention in running the migration scripts, it automatically migrates from currently deployed app version to app version which will be deployed to web logic. For example, the current deployed app version in web logic is 16.0.028 and app which will be deployed to web logic is going to be 19.0.000, So db schema auto migration finds scripts between the versions (inclusive of 16.0.028 and 19.0.000), sorts and runs the migration scripts one by one from 16.0.028 to 19.0.000. This feature can be used on passing the -run-db-schema-migration parameter to bdi-app-admin-deployer.sh script with {-setup-credentials | -use-existing-credentials}.

Note: DB Schema auto migration feature should be used from >=16.0.028 version of BDI apps.

Example to run DB Schema auto migration:

```
sh bdi-app-admin-deployer.sh { -setup-credentials | -use-existing-credentials }
-run-db-schema-migration (Optional parameter)
```

4

Integration with External Applications

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

BDI External Job Admin as Sender

BDI integrates with third-party systems utilizing BDI-EXTERNAL to send data. This integration enables the movement of Item/Foundation data from an external system into Oracle Retail applications.

All the required sender artifacts are available as part of the BDI process flows and jobs deliverables. An external extractor job extracts data for a family from the sender (source) system and moves the data to outbound interface tables. A downloader-transporter job downloads the data set from outbound interface tables for a family and streams data to a BDI destination application using the receiver service.

Configure BDI to Enable or Disable the External Process Flows and Jobs

As there are many flows and jobs from and to external application hence GUI is cluttering with many jobs and flows. To avoid confusion disabling the jobs and flows by default related to EXTERNAL application.

{"name":"flowSelection.1.pattern", "value":"_From_EXTERNAL"},

{"name":"flowSelection.1.initialState", "value":"false"}

The following system options in bdi-job-admin-deployment-env-info.json disable the jobs by default at the time of deployment/installation:

```
{"name":"flowSelection.1.pattern", "value":"_From_EXTERNAL"},
{"name":"flowSelection.1.initialState", "value":"false"}
```

If you want to view the disabled jobs, search for "ToExternalJob" and uncheck the Hide Disabled jobs checkbox.

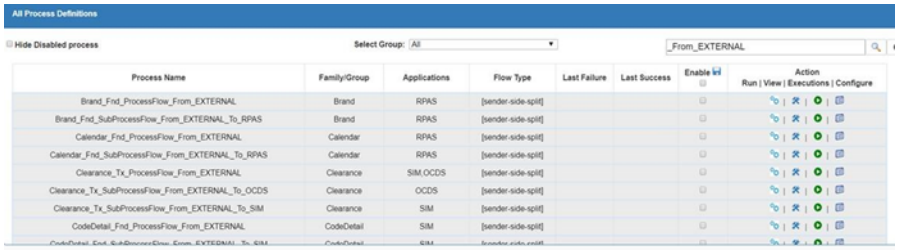
If you want to view the disabled flows, search for "_From_EXTERNAL" and uncheck the Hide Disabled process checkbox as shown below.

Option 1: Enable All _From_External process flows

Perform the following procedure to enable the disabled process flows and jobs:

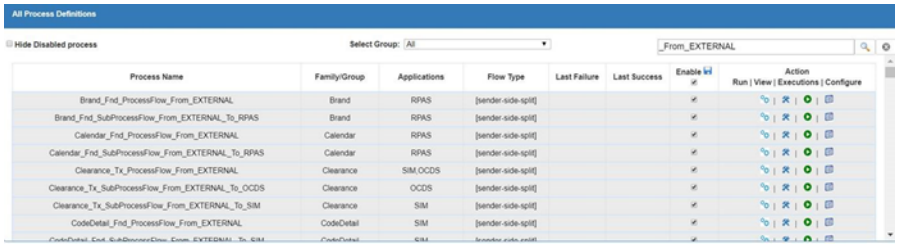
1. Search for "_From_EXTERNAL" and uncheck the Hide Disabled process checkbox.

Figure 4–1 View _From_EXTERNAL Processes



2. Click the header checkbox to enable/disable. The default is disabled.
3. Click a second time on the header checkbox to make it enable/disable. Click the Save icon. The flag is updated in the database to "true" or enabled.

Figure 4–2 Enable _From_EXTERNAL Processes



Follow the similar approach to enable external jobs as well.

Option 2: Update the property flowSelection.1.initialState

1. Update the property flowSelection.1.initialState in the json process flow to "true" to enable all the "from external" flows.

```
{"name":"flowSelection.1.pattern", "value":"_From_EXTERNAL"},
{"name":"flowSelection.1.initialState", "value":"true"}
```

2. Update the property "jobSelection.1.initialState" in job admin json to "true" to enable all the "to external" jobs.

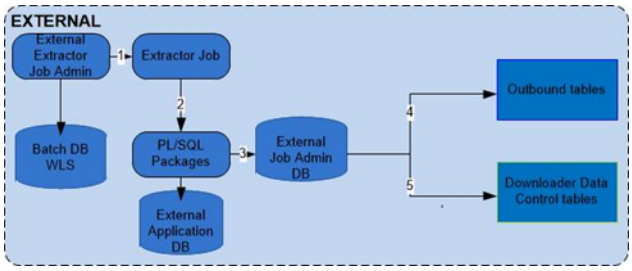
```
{"name":"jobSelection.1.pattern", "value":"ToExternalJob"},
{"name":"jobSelection.1.initialState", "value":"true"}
```

3. Set both the LOADJOBDEF/LOADPROCESSDEF and LOADSEEDDATA flags to TRUE and redeploy the job admin or process flow respectively.

External Extractor Job

The Extractor Job uses a batchlet and PL/SQL stored procedures to move data from transactional tables of source system (for example EXTERNAL, third party system) 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. Whichever application is acting as Sender those teams have to write business logic to move data from application transaction tables to BDI OUT tables.

Figure 4–3 External Extractor Job



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

```
CREATE OR REPLACE PACKAGE BODY Brand_Fnd_Extractor AS

FUNCTION extract(O_error_message  IN OUT  VARCHAR2,
                 O_control_id     IN OUT  NUMBER,
                 I_job_context    IN      VARCHAR2)
   return NUMBER IS

   L_control_id NUMBER;
   returnCode NUMBER := 0;

BEGIN
   -- Call BDI package to get data set id
   if Brand_Fnd_Out_DataCtl.beginFullSet_Brand_Fnd(L_control_id,
                                                   O_error_message) = 1 then
      ROLLBACK;
      return 1;
   end if;

   --dbms_output.put_line('beginFullSet complete.');

   -- Move data from application transactional tables to OUT table
   -- Set returnCode to 0 or 1. Success - 0, Error - 1

   --Call BDI package to track the status
    IF returnCode = 0 THEN
       if Brand_Fnd_Out_DataCtl.OnSuccEndSet_Brand_Fnd(I_job_context,
                                                       L_control_id,
                                                       O_error_message) = 1 then
```

BDI Extractor (PL/SQL Application)

1. The Extractor job is run from the App A (for example EXTERNAL) Extractor Job Admin application through REST or the UI.
2. The Extractor job invokes the PL/SQL stored procedure in the 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.

   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.

BDI External Job Admin as Receiver

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

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

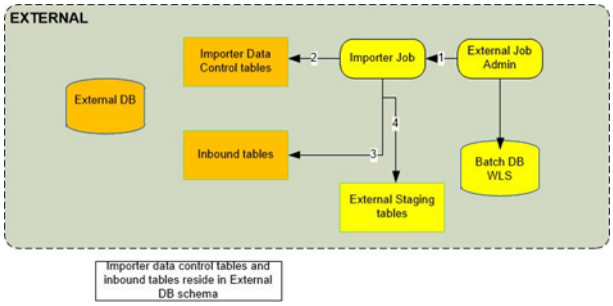
External Importer Job

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

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

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

Figure 4–4 External importer Job



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

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

    --Implement business logic to move data to transaction tables.
    --move_Brand_Fnd(interfaceShortNames(i), beginSequenceNumber, endSequenceNumber, returnCode,
    IF returnCode = 1 THEN
        --dbms_output.put_line('move_Brand_Fnd return code 1');
        EXIT;
    END IF;
```

External Importer

1. Importer job is run from App B EXTERNAL Job Admin application through REST or UI.
2. Importer job checks for data sets in importer data control tables.

3. If data set is available for import, importer job downloads data from inbound table.
4. Importer job loads data to App B EXTERNAL staging tables.

Configure External Job Admin as Receiver in the Process Flow

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

allAvailableDestinationApps property mentions all the applications available as destination.

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

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

External BDI Process Flow

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

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

Figure 4–5 Merchandising to External Process Flow

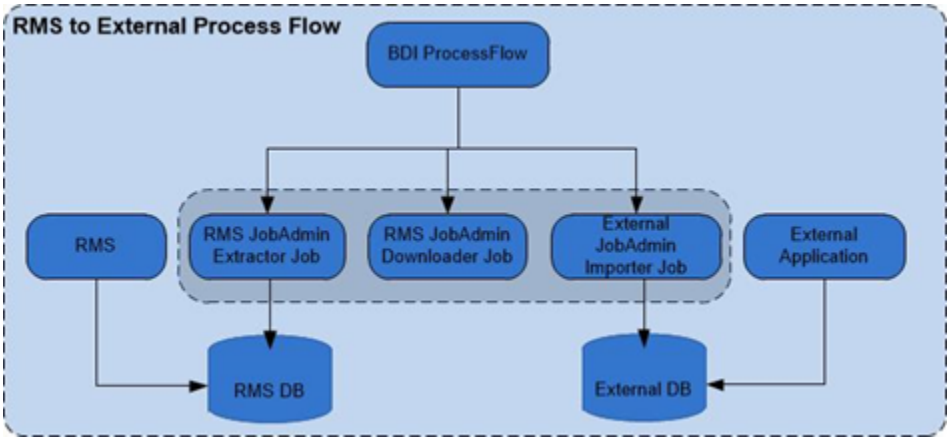
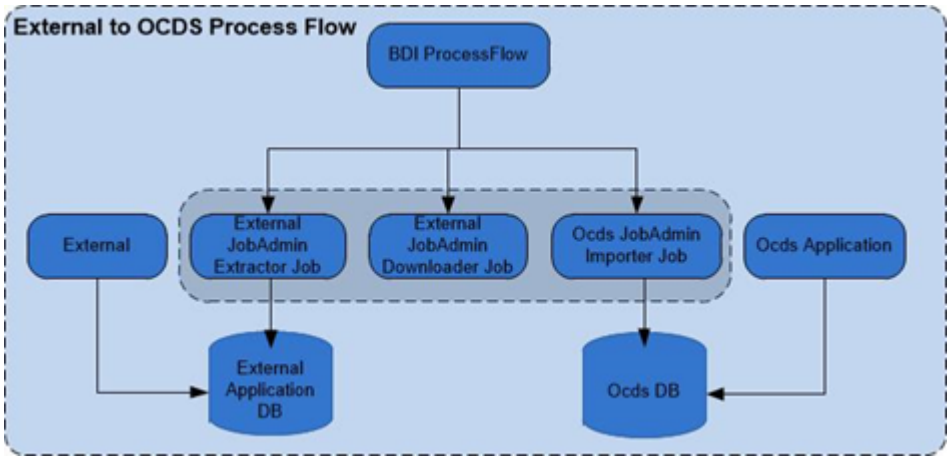


Figure 4–6 External Application to OCDs Process Flow



Installation Details

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

5

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
- Enable or Disable the 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.

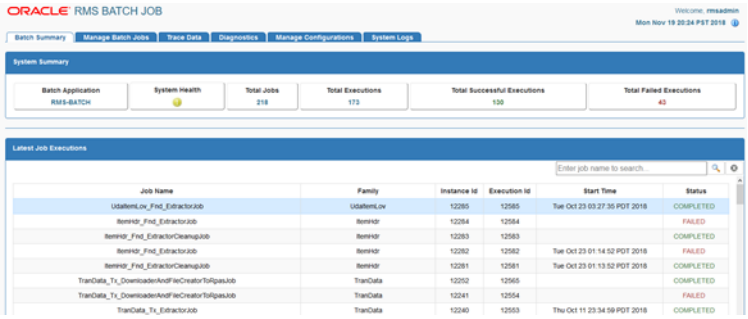
Function	Admin Role	Operator Role	Monitor Role
Edit configuration from UI	Yes	No	No
Create/update/delete system options	Yes	No	No
Create/update/delete system credentials	Yes	No	No
View credentials	Yes	No	No
Run Jobs	Yes	Yes	No
Monitor jobs	Yes	Yes	Yes

Monitoring Batch Jobs Using BDI Job Admin

Batch jobs can be monitored using the Job Admin UI.

Batch Summary Tab

Figure 5–1 Batch Summary Tab



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

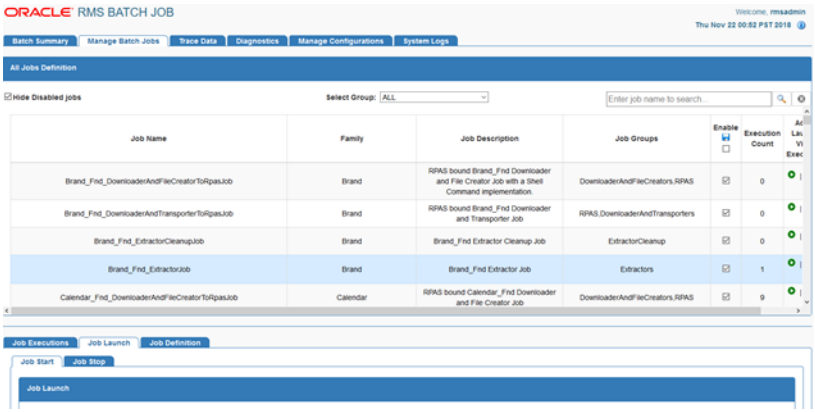
This tab displays the list of available jobs with their details and allows you to Start a job, Restart failed jobs, enable or disable the jobs, list the executions of a job.

By default all the jobs are enabled. Select the job row and check/uncheck the check box of each job and click on save image button in enable column. Only enabled jobs can be launched/restarted. The Launch/Restart button is disabled for the disabled jobs. There is an option to enable or disable all the jobs at a time by clicking on checkbox, present in the enable column, highlighted in red and click on save image button.

Note: Only enabled jobs can be launched or restarted.

The Launch or Restart button is disabled for the disabled jobs.

Figure 5–2 Manage Jobs Tab



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. Only enabled jobs can be restarted otherwise the Restart button is disabled. 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.

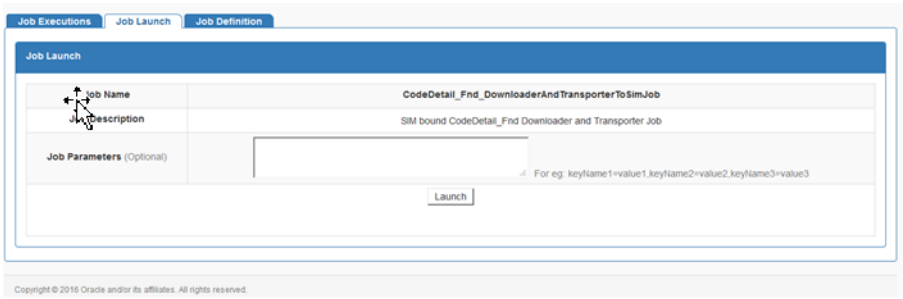
Note: The url is a infrastructure parameter, the user is not allowed to change its value.

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. Only enabled jobs can be launched. The Launch button is disabled for the disabled 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.

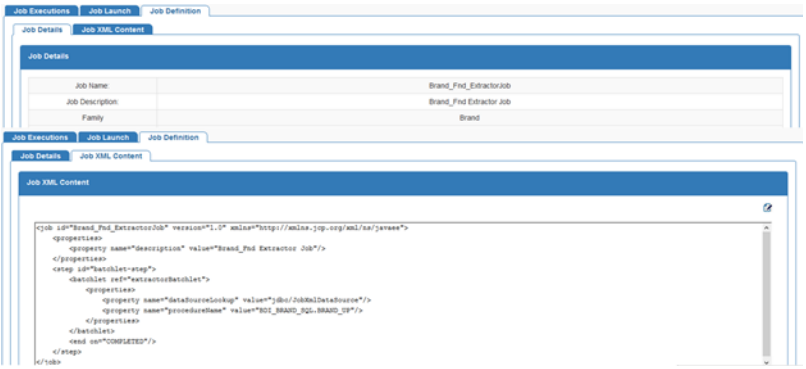
Figure 5–3 Job Launch



Job Details

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

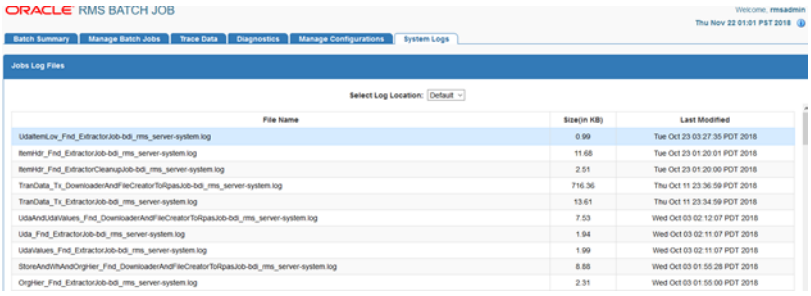
Figure 5–4 Job Details



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 5–5 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>_<Current Time in millis>_<Source System Name>)
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

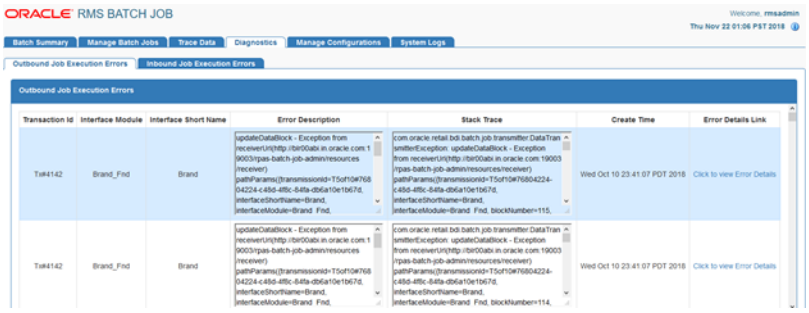
Key Word	Description
DATA_SET_FAILED	Indicates that the job failed to process the data set

Diagnostics 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 5–6 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

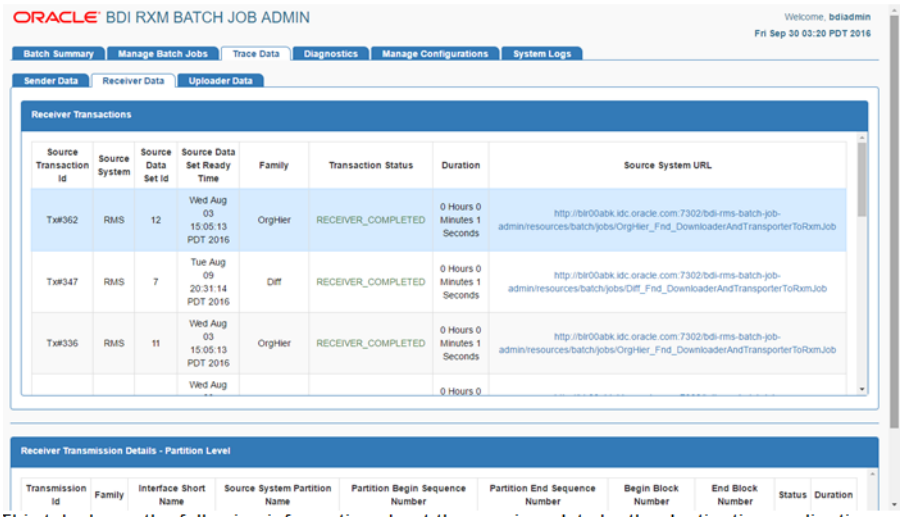
Figure 5–7 Sender Data

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

Field Name	Description
Transaction Id	Transaction Id (Tx#<Job Instance Id>_<Current Time in millis>_<Source System Name>) of the job
Interface Module	Name of the interface module
Job Name	Name of the batch job
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 5–8 Receiver Data



This tab shows the following information about the receiver data by the destination application (for example rpas-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
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

Inbound job Executions

Field Name	Description
Transaction Id	Transaction Id (Tx#<Job Instance Id>_<Current Time in millis>_<Source System Name>) of the uploader job
Remote Transaction Id	Transaction Id of the downloader job
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)

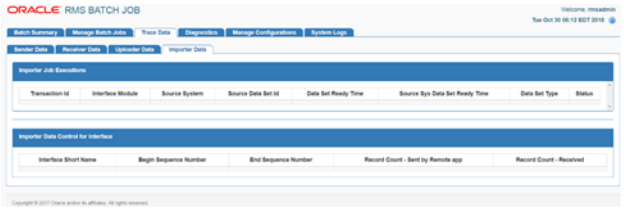
Field Name	Description
Source Sys Data Set Ready Time	Time when source system moved data set to outbound tables
Data Set Ready Time	Time when uploader job uploaded data set to inbound tables
File Merge Level	Merge level of the file (NO_MERGE, MERGE_TO_PARTITION_LEVEL, MERGE_TO_INTERFACE_LEVEL)
Status	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

Importer Data

Figure 5–9 Importer Data



Importer Job Executions

Field Name	Description
Transaction Id	Transaction Id (Tx#<Job Instance Id>_<Current Time in millis>_<Source System Name>) of the uploader job
Interface Module	Name of the interface module (for example Diff_Fnd)
Source System	Name of the source system (for example RMS)
Source Data Set ID	Generated by extractor job for group of data
Data Set Ready Time	Time when importer job uploaded data set to inbound tables

Field Name	Description
Source Sys Data Set Ready Time	Time when source system moved data set to inbound tables
Data Set Type	Type of data set (FULL or PARTIAL)
Status	Status of importer job (COMPLETED or FAILED)

Importer Data Control for Interface

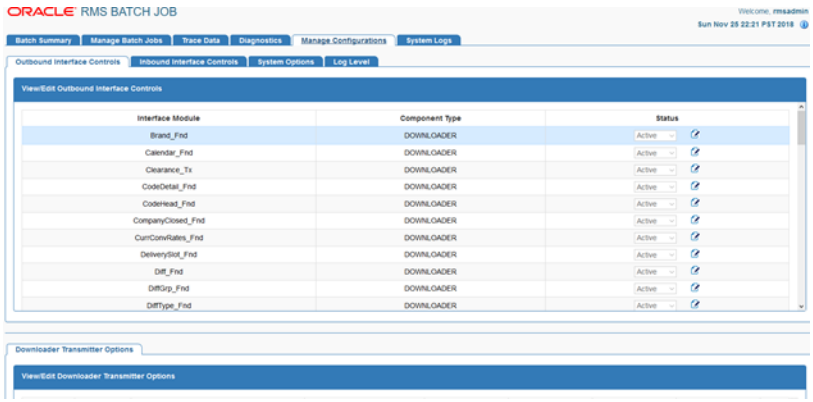
Field Name	Description
Interface Short Name	Name of the interface
Begin Sequence Number	Begin sequence number of data set in the inbound table
End Sequence Number	End sequence number of data set in the inbound table
Record Count - Sent by Remote App	Record count sent by remote application
Record Count - Received	Record count received

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

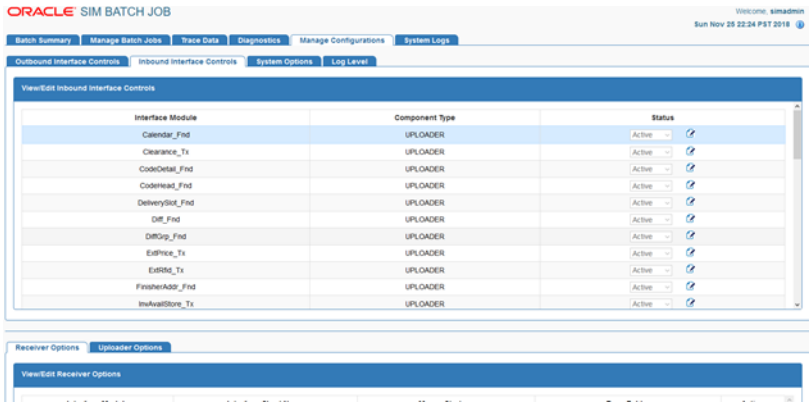
Figure 5–10 Outbound Interface Controls



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

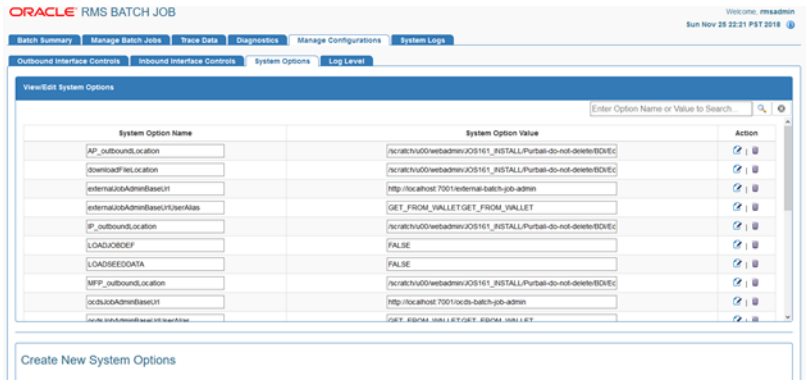
Figure 5–11 Inbound Interface Controls



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 5–12 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 needs 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 WSMException

Issue:

Log files contain this exception:

```
oracle.wsm.common.sdk.WSMException: 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: RPAS) 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 reposiotry 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.

6

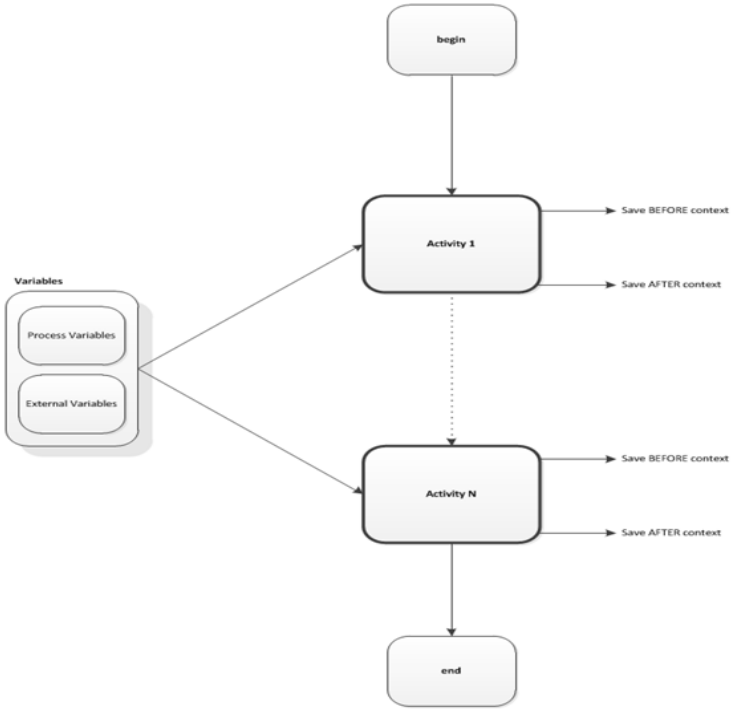
Process Flow

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 6–1 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["rpasJobAdminBaseUrlUserAlias"]
```

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.

A complete list of process flow status are:

- PROCESS_STARTED
- PROCESS_FAILED
- PROCESS_COMPLETED
- PROCESS_STOPPING
- PROCESS_STOPPED

Similar to process statuses, each activity has also a status. There values are :

- ACTIVITY_STARTED
- ACTIVITY_FAILED
- ACTIVITY_COMPLETED
- ACTIVITY_WAITING_DUE_TO_HOLD_STARTED
- ACTIVITY_WAITING_DUE_TO_HOLD_COMPLETED
- ACTIVITY_WAITING_DUE_TO_JOIN_STARTED
- ACTIVITY_WAITING_DUE_TO_JOIN_COMPLETED
- ACTIVITY_SKIPPED
- ACTIVITY_STOPPING
- ACTIVITY_STOPPED

All the runtime status are persisted in the process schema at runtime when the DSL is executed.

Process Flow DSL

This section describes the process flow DSL.

Process Flow DSL characteristics

The following are the characteristics of the Process Flow DSL:

- Every process flow must have a name. The process flow name must match with the filename that the process flow is saved into.
- Process flows are written in a DSL and saved as .flo files.
- Process flow is made up of two special activities called “begin” and “end” and bunch of user defined activity nodes.
- The “begin” and “end” activity will always run.
- User defined activity may or may not run based on “SKIP” or moveTo logic.
- Every user defined activity must have a unique name within a process flow.
- The activity names are used to transfer control from one activity to another. Jumping to an activity is possible using moveTo function.
- Every activity has an “action” block that does the real work. Groovy/Java code written inside the action block.

- Local variables can be defined within the action block.
- Process variables are defined on top and are accessible to all activities within the process.
- There are few implicit variables, like $activityName, $name.
- Errors can be thrown using “error <some message>” function.
- Built-in Conditional branching, looping, error handling.
- Predefined functions for common tasks to reduce boilerplate code.
- Built in REST service DSL to be able to call service with just one line.
- Services available to start/restart/monitor process flows programmatically.
- Can handle chaining of Process Flows.
- Has a built in Service Credential management framework.
- Hybrid Cloud ready.
- Built in activity SKIP functionality.
- Built in activity HOLD and RELEASE functionality
- Built in bulk skip and Hold functionality.
- Built in SPLIT and JOIN functionality between process flows
  - SPLIT - one to many
  - JOIN - many to one

DSL Keywords

This section lists the DSL keywords:

DSL Keywords	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	Begin activity block is the first activity in the DSL. It is mandatory and can be used for initialization.
activity	The executable component of the process flow. A process flow is composed of many activities.
action	Action section is where the executable code should reside. Any Groovy or Java code can be coded in this section.
on "okay" moveTo	Use these keywords inside an activity to move to another activity.
on "error" moveTo	Use these keywords inside an activity to move to error activity.
end	"end" activity in process flow definition appears as the last activity. There should be only one "end"activity.

DSL Blacklisted Keywords – In the process definition, changes can be made in DSL (Domain Specific Language), Groovy, or Java. Since this file is essentially a program, it can modified to cause damages (e.g., delete files from the system). We have introduced a list of keywords that are potentially dangerous to use. If a blacklist word is present in

the DSL, application deployment will fail and an error will be written to the server log (for example, java, groovy, thread etc.).

Process Flow API

This section describes the Process Flow API.

DSL API	USAGE	Description
triggerProcess(def baseUrl, String processDslName, String credentials, String processParameters)	triggerProcess(externalVariables["url"], "ProcessABC", externalVariables["urlUserAlias"], "a=b,c=d")	Method to start a process from DSL. This method sends a POST request to Process Flow to start a process. It returns process
startOrRestartJob(def baseUrl, String jobName, String credentials)	startOrRestartJob(externalV ariable s["url"],"JobAbc", externalVariables["urlUserAlias"])	Method to start or restart a job in Job Admin. This method sends a POST request to a REST end point in Job Admin.
waitForJobCompletedOrF ailed(def targetActivity, def url, String credentials, int waitMinutes=1)	waitForJobCompletedOrFailed("Jo bAbcActivity",externalVariables["u rl"] + "/resources/batch/jobs/JobAbc/" + processVariables["jobExecutionId"] , externalVariables["urlUserAlias"])	Method to wait for job to be completed or to fail. This method checks the status of the job and waits until status is COMPLETED or FAILED.
waitForProcessInstancesTo ReachStatus(def processInstanceList, def targetStatus=[PROCESS_ COMPLETED], def logicalAndOrOr = LOGICAL_AND, int waitSeconds=60)	waitForProcessInstancesTo ReachSt atus(["P~1", "Q~1"], PROCESS_ COMPLETED, LOGICAL_AND)	Method to wait for other process instances to reach a status.
waitForProcessNamesToR eachStatus(Map, processNameToNumberOf ExecutionsAfterStart Mark erTime, LocalDateTime startMarkerTime, def targetStatus = PROCESS_ COMPLETED, def logicalAndOrOr = LOGICAL_AND, def whichExecutionStatus = LAST_ EXECUTION_ STATUS, int waitMinutes	waitForProcessNamesToReachStat us([P:3, Q:3, R:3], now().minusDays(1), PROCESS_ COMPLETED, LOGICAL_AND, LAST_ EXECUTION_STATUS)	Method to wait for processes with names to reach a status.

DSL API	USAGE	Description
persistGlobalUserDat a(Str ing key, String value)	persistGlobalUserData("key ", "value")	Method to persist data to be shared with other processes. Persists key value pairs in BDI_ SYSTEM_ OPTIONS table.
String findGlobalUserData(Strin g key)	findGlobalUserData("key")	Gets value from BDI_ SYSTEM_ OPTIONS table for a given key.
Map findAllGlobalUserDa ta(Str ing key)	findAllGlobalUserData()	Returns a Map with all user data.
removeGlobalUserDa ta(St ring key)	removeGlobalUserData("ke y")	Removes data for given key.
Error	error "report my error"	Generate an error condition and jump to the end activity. Process will be marked as failed.
POST	POST[externalVariables.url] ^exter nalVariables.urlUserAlias def response = (POST[externalVariables.url] + customHttpHeaders & MediaType.APPLICATION _JSON_TYPE ^ BasicAuth.alias1 | MediaType.APPLICATION _JSON_TYPE) << {} as String	Method to make a POST call to a url. externalVariables.url - URL system option key configured in System Options table customHttpHeaders - [a:"b", c:"d"] Use "+" to provide custom http headers Use "&" to provide response media type Use "^" to provide basic authentication alias. User name and password will be Base64 encoded by the API. Use "|" to provide entity media type Use "<<" to post data. The data will be in the format provided in entity media type.
GET	GET[externalVariables.url]^ extern alVariables.urlUserAlias def response = (GET[externalVariables.url] + customHttpHeaders & MediaType.APPLICATION _JSON_TYPE ^ BasicAuth.alias1) as String	Method to make a GET call to a URL. externalVariables.url - URL system option key configured in System Options table customHttpHeaders - [a:"b", c:"d"] Use "+" to provide custom http headers Use "&" to provide response media type Use "^" to provide basic authentication alias. User name and password will be Base64 encoded by the API

DSL API	USAGE	Description
DELETE	DELETE[externalVariables.url]^externalVariables.urlUserAlias def response = (DELETE[externalVariables.url] + customHttpHeaders & MediaType.APPLICATION_JSON_TYPE ^ BasicAuth.alias1) << {} as String	Method to make a DELETE call to a URL. externalVariables.url - URL system option key configured in System Options table customHttpHeaders - [a:"b", c:"d"] Use "+" to provide custom http headers Use "&" to provide response media type Use "^" to provide basic authentication alias. User name and password will be Base64 encoded by the API
log.info log.debug log.error	log.debug "Activity Name: $activityName"	Adds information to log file.

Process Flow Variables

This section describes the Process Flow Variables.

Variables	Implicit or Explicit	Usage Examples	Description
externalVariables	Implicit	def myVar = externalVariables['myKey']	These are global variables that apply to all process flows. It comes from System Options table. Installation specific key values will be here.

processVariables	Implicit	var(["myVar1":"prq", "myVar2":"xyz", "myVar3":"mno"])
		//get value def aVar = processVariables['myVar1']
		//put new value processVariables['myVar2'] = "abc"
			These are process level variables that can be shared by all activities. Process variables are automatically persisted. Restart of a process recovers the process variables to the right value where it left off in the previous run. These are the most common variables you should use. Process variables must be declared using the var key word.

Variables	Implicit or Explicit	Usage Examples	Description
Local variables	Explicit	action{ def a = "xyz" def i = 7	Any variables can be created with the action block and used as local variables. Local variables defined in one activity is not accessible in another
Global external variables	Explicit	persistGlobalUserData(" key1", "value1") def xyz = findGlobalUserData("key 1") removeGlobalUserData(" key1")	For inter process dynamic variable sharing one can persist new variable to DB.
activityName	Implicit	println "My activity is ${activityName}"	Current activity name.
Name	Implicit	Println "My process name is ${processName}"	Current process name.
processExecutionId	Implicit	Println "Current process execution Id is ${processExecutionId}"	Current process execution Id

Process Flow Instrumentation

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

The information is reported through the Process Flow Admin application. Process Flow Admin is a web application that provides a GUI to manage task workflows. This is useful for tracking the process flows as well as troubleshooting. The snapshots also help when 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

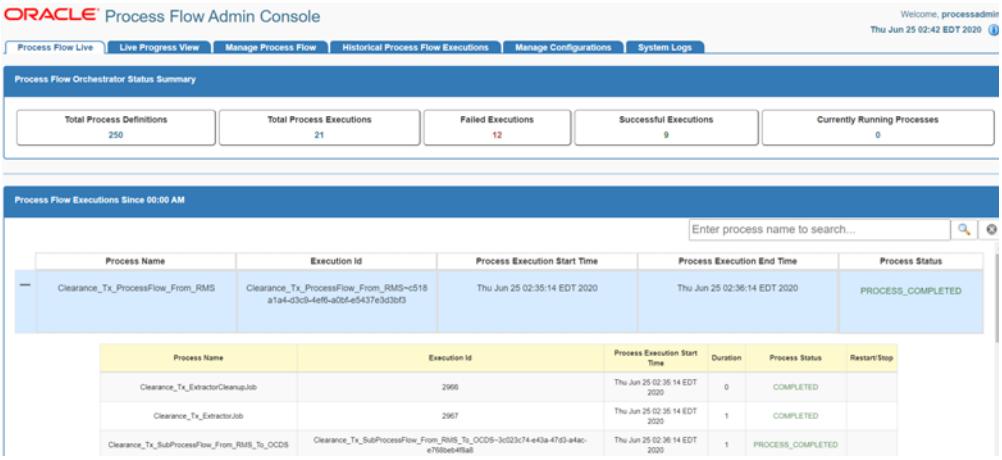
Process Flow (Admin UI) is a web user interface provided by Process Flow where users can view and execute processes, including managing, updating process flow, manually running processes, viewing process executions and process flow logs.

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 6–2 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.

Build version and date is displayed on the info icon when a user selects the same. The icon is on the extreme right top corner of the page.

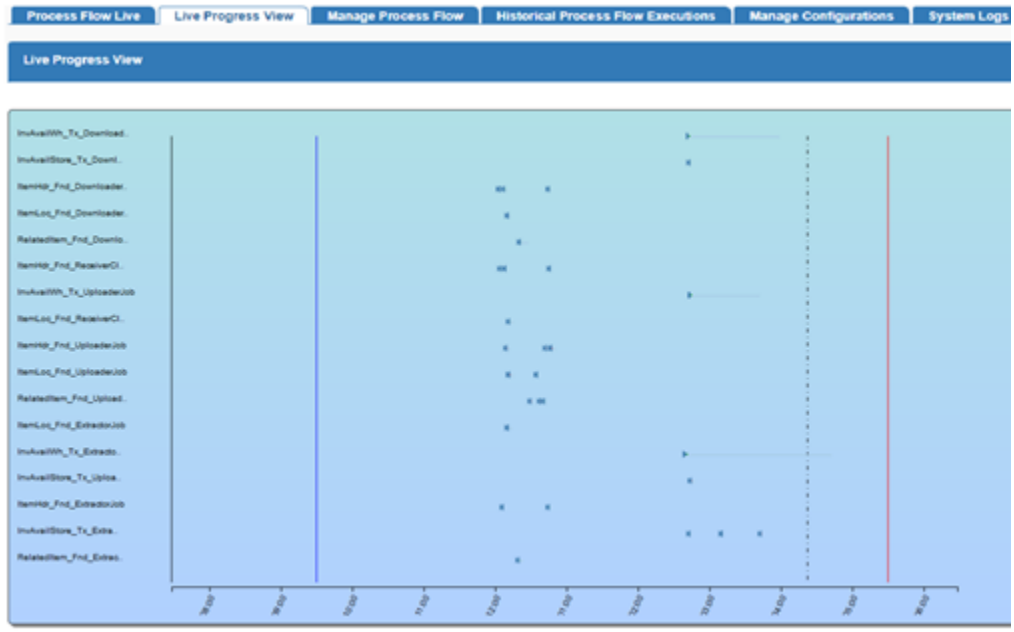
Execution Trace Graph

The Execution Trace Graph is also part of Process Flow Live tab. Execution trace graph shows the sub processes and jobs called from a process. The arrows show the relationship between the caller and the callee. The circular nodes of the graph represent the process or the job that was invoked. On hovering over the node, the details of the execution like name of the process or job, invocation time, status etc. are displayed.

Figure 6–3 Execution Trace Graph

Live Progress View Tab

Figure 6–4 Live Progress View Tab

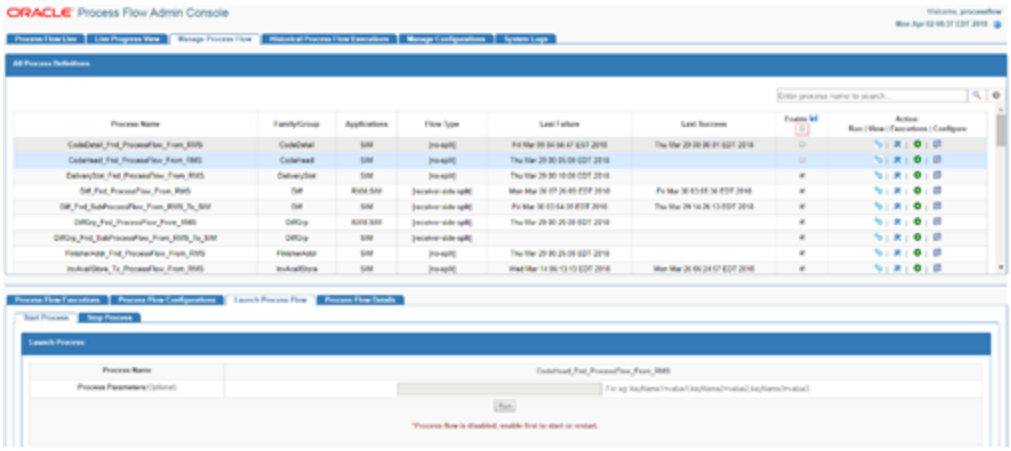


Live progress graph shows the batch status for a time window. Here are the details of the graph.

- It shows all the jobs from all the JobAdmins configured in the process flow.
- Currently the time window is 10 hours before the current time and one hour ahead of the current time.
- Current time is the dotted line axis. The batch start time is the blue axis and batch end time is the red axis
- The graph will refresh itself every second. So the axis, jobs and all related information will update
- The time window will move as the current time axis nears the end of the graph
- Each job is represented by an arrow. The color of the arrow will be red (errored jobs), blue (running jobs) or green (completed jobs). The length of the arrow will be same as the time taken for the job.
- Same jobs will appear in the same line.

Manage Process Flow Tab

Figure 6–5 Manage Process Flow Tab

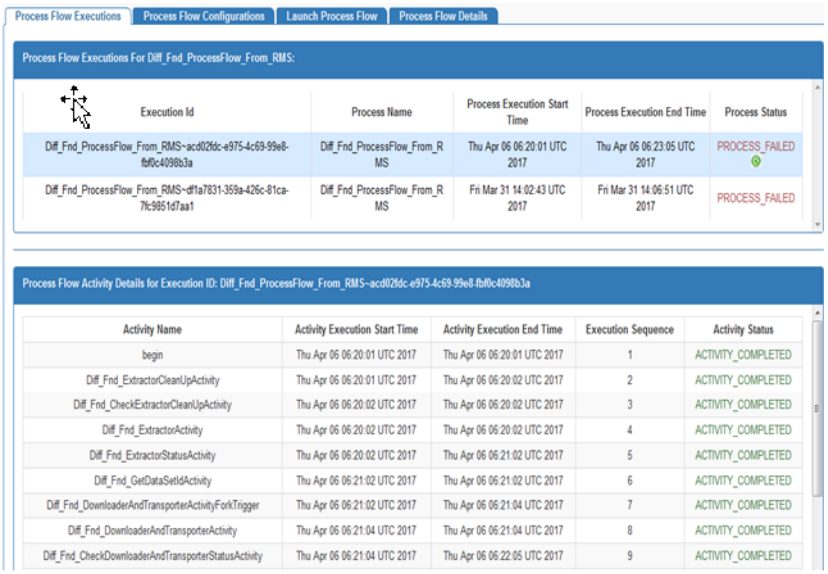


The Manage Process Flow tab allows the user to Start a process flow, Restart a failed process flow, enable or disable the process flow, View/Edit a process flow, Stop a running process flow, List the executions instances of a process flow. User can search process details on this tab. A failed process 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. By default all the process flows are enabled. Select the process flow row and check/uncheck the check box of each process and click on save image button in enable column. Only enabled process flows can be launched/restarted. The Run/Restart button is disabled for the disabled process flows. There is an option to enable or disable all the process flows at a time by clicking on checkbox, present in the enable column, highlighted in red and click on save image button.

Note: Only enabled processes can be launched or restarted. The Run or Restart button is disabled for the disabled processes.

Process Flow Executions

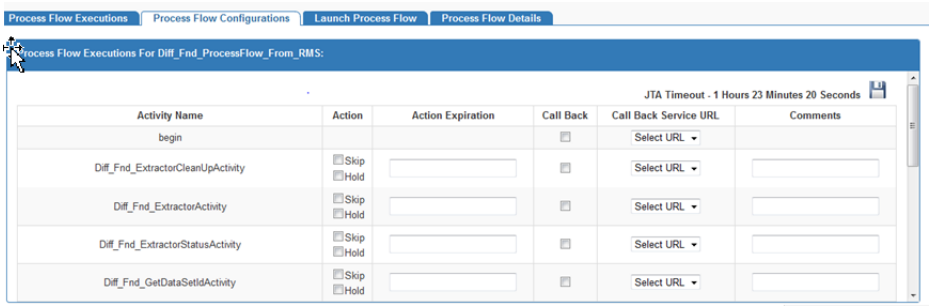
Figure 6–6 Process Flow Executions



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.

Process Flow Configurations

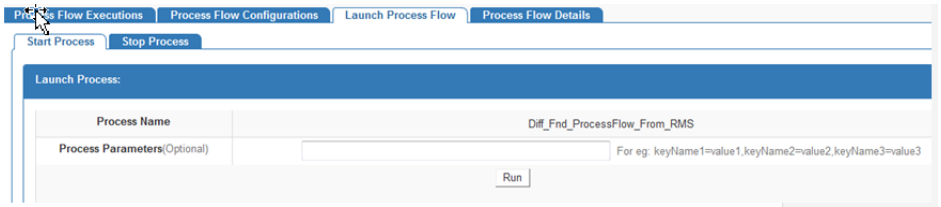
Figure 6–7



This tab provides various features for activity configurations for the selected process like Skip, Hold, Callback. Admin and operator have permissions to update activity configurations.

Launch Process Flow

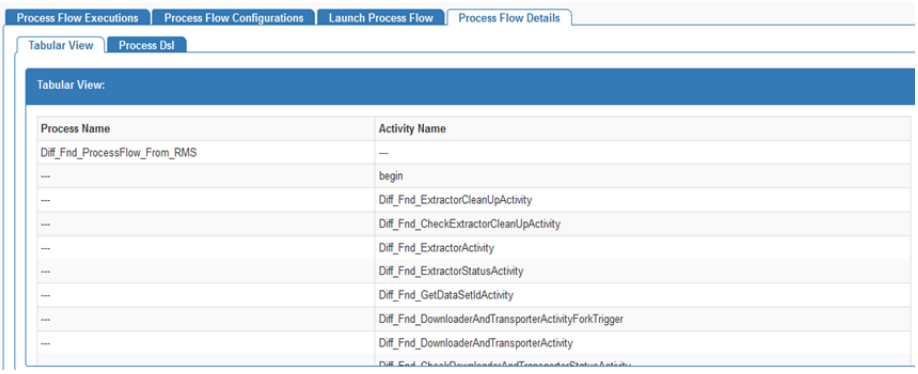
Figure 6–8 Launch Process Flow



This tab can be used to start or stop process the selected process. Start Process subtab used to launch Process. Only enabled processes can be launched. The run button is disabled for the disabled processes. 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. Stop Process subtab is used to Stop a process execution. Stop will be a graceful stop, which means current executing activity will be first completed and then process will be stopped. If activity is not running, Stop will not bring any action.

Process Flow Details

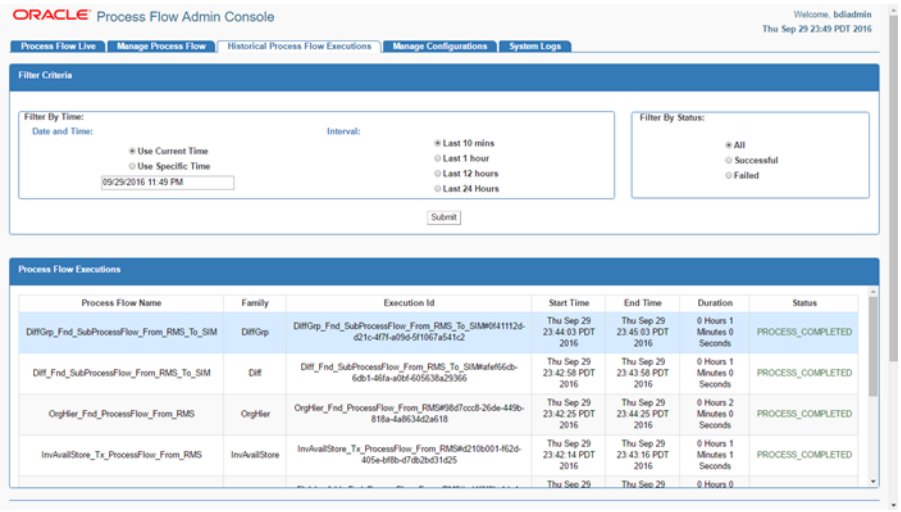
Figure 6–9 Process Flow Details



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

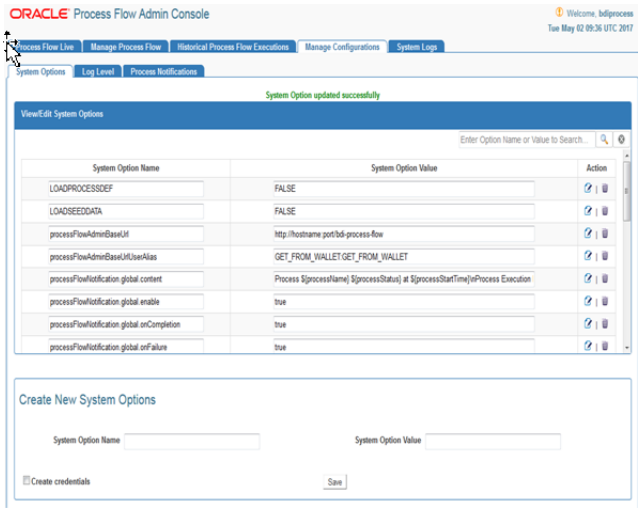
Figure 6–10 Historical Process Flow Executions Tab



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 6–11 Manage Configurations Tab



The Manage Configurations tab allows users to view, edit and create system options, configure process notifications and log levels. 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.

Diagnostics Tab

Ping Feature: The ping utility is to support environment smoke tests and thereby eliminate any bad configuration in System Options.

- The service URL and credentials used for ping test is derived from pattern based keys in the System Options.
- The default Ping Service URL is combination of App specific BaseUrl + Discover service as suffix. POAM ping service URL is combination of BaseUrl + Default ping service as prefix (services/private/ping)
- User can ping individual app URL and see the success/failure message on the TOP. Also, there will be a status column which shows UP Arrow image for success and DOWN arrow image for failure.
- User can also use Ping All feature to ping all the URLs at one time and the responses will be shown on the status column against each URL.
- Reset button is to reset the cache and do a fresh service call.
- All roles are able to ping services i.e. MonitorRole, OperatorRole and AdminRole users.

Figure 6–12 PING with Success Message

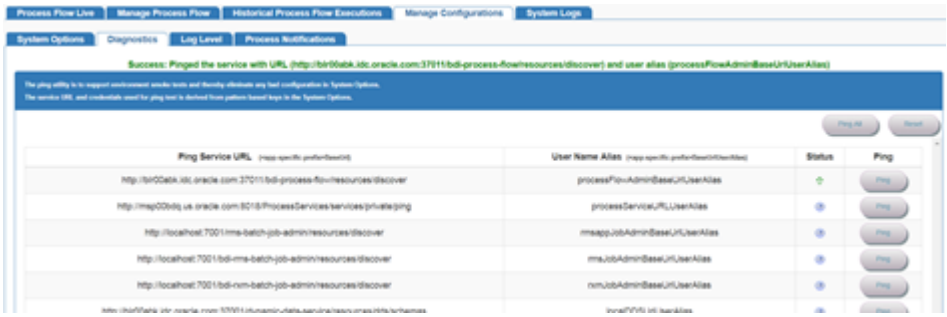


Figure 6–13 PING with Failure Message

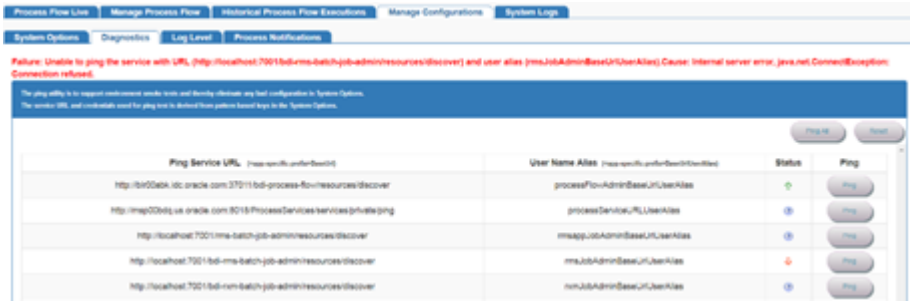
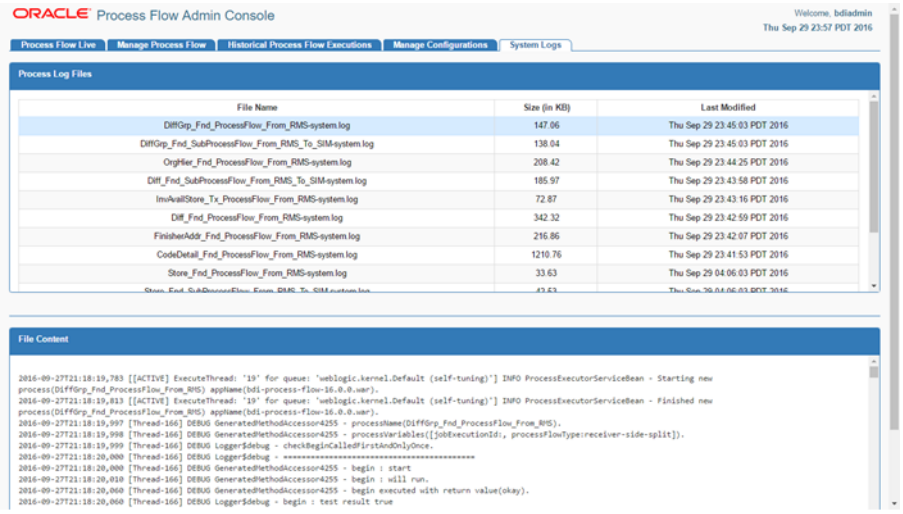


Figure 6–14 PING All with Message

System Logs Tab

Figure 6–15 System Logs Tab



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}

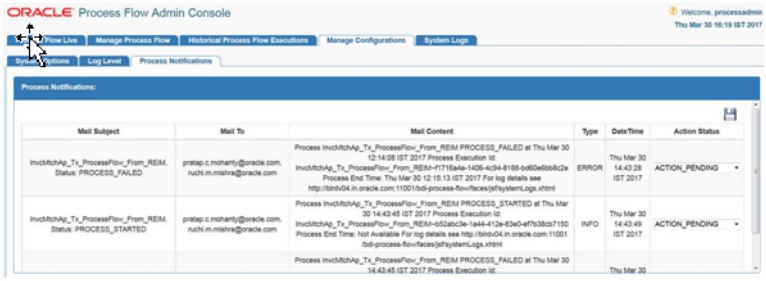
Persisting Process Notifications

All process notifications are persisted to the BDI_EMAIL_NOTIFICATION table. There is a subtab Process Notifications added in Manage Configurations tab which displays all the notifications.

One notification icon appears right top corner of the screen adjacent to the user if there is any notification in PENDING status. User will be navigated to the Process Notifications subtab by clicking on the image.

User can modify the status to COMPLETED after going through the notification and click on save button so that next time it doesn't appear on the screen.

Figure 6–16 Persisting Process Notifications



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.

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. A complete list of process flow statuses includes:

- PROCESS_STARTED
- PROCESS_FAILED
- PROCESS_COMPLETED
- PROCESS_STOPPING
- PROCESS_STOPPED

Similar to process statuses, each activity also has a status. The values include:

- ACTIVITY_STARTED
- ACTIVITY_FAILED
- ACTIVITY_COMPLETED
- ACTIVITY_WAITING_DUE_TO_HOLD_STARTED
- ACTIVITY_WAITING_DUE_TO_HOLD_COMPLETED
- ACTIVITY_WAITING_DUE_TO_JOIN_STARTED
- ACTIVITY_WAITING_DUE_TO_JOIN_COMPLETED
- ACTIVITY_SKIPPED
- ACTIVITY_STOPPING

All the runtime statuses are persisted in the process schema at runtime when the DSL is executed.

Activity Features

This section describes the Activity features.

- Skip Activity
- REST Endpoint to Set the Skip Activity Flag
- Hold/Release Activity
- REST Endpoint to Set the Hold Activity Flag
- Bulk Skip/Hold
- Callback Service
- How to Start Process Flow with Input Parameters
- Call Back from the Process Flow
- How to Invoke the Callback Service Declaratively
- Process Flow Did Not Start
- Deleted Process Flow Still Listed in the UI

Skip Activity

Activities in a process flow can be skipped by setting the skip activity flag through the Process Flow Configurations tab in Process Flow UI or REST endpoint. Skip flag can be set to expire based on date and time. If expiry date is not provided, then that activity will be skipped until skip flag is removed. When an activity is set to skip, process flow engine skips that activity and runs the next activity in the flow.

REST endpoint to set the skip activity flag

/batch/processes/<processName>/activities/<activityName>?skip=true

Hold/Release Activity

Activities in a process flow can be paused by setting the hold activity flag through the Process Flow Configurations tab in Process Flow UI or REST endpoint. Hold flag can be set to expire based on date and time. If expiry date is not provided, then that activity will be paused until hold flag is removed, and process will remain in PROCESS_STARTED state. When an activity is set to hold, process flow engine waits on that activity until hold flag is removed or time expired, and activity state will be moved to ACTIVITY_WAITING_DUE_TO_HOLD_STARTED.

REST endpoint to set the hold activity flag

/batch/processes/<processName>/activities/<activityName>?hold=true

Note: Don't try to Stop a waiting activity, as it can result into deadlock state.

Bulk Skip/Hold

Bulk skip or hold allows to set skip and/or hold flag for a list of activities in multiple process flows.

REST Endpoint: /batch/processes/skip-or-hold POST Data:

```
{"processActivities": [
{"processName" : "…",
 "activityName": "…",
 "skip" : true,   false if not specified
 "hold" : false,  false if not specified
 "actionExpiryDate" : "optional",
 "comments" : "optional"
},
{…}
```

```
      ]
}
```

Curl Command to set bulk skip/Hold

```
curl -i --user processadmin:processadmin1 -X POST -H
"Content-Type:application/json"
http://host:port/bdi-process-flow/resources/batch/processes/skip-or-hold -d
'{"processActivities": [
{"processName" : "OrgHier_Fnd_ProcessFlow_From_RMS", "activityName": "OrgHier_Fnd_
ExtractorActivity", "skip":true},
{"processName" : "DiffGrp_Fnd_ProcessFlow_From_RMS", "activityName": "Activity1",
"skip":true}
,{"processName" : "DiffGrp_Fnd_ProcessFlow_From_RMS", "activityName": "Activity2",
"skip":true}
]
}'
```

Output

```
{"processActivities":[{"actionResult":"OK","activityName":"OrgHier_Fnd_
ExtractorActivity","processName":"OrgHier_Fnd_ProcessFlow_From_
RMS"},{"actionResult":"OK","activityName":"Activity1","processName":"DiffGrp_Fnd_
ProcessFlow_From_
RMS"},{"actionResult":"OK","activityName":"Activity2","processName":"DiffGrp_Fnd_
ProcessFlow_From_RMS"}],"netResponse":"SUCCESS"}
```

Callback Service

Process Flow engine can be configured to call a rest service at each activity. This is useful if the process flow is invoked by an external system (typically a workflow system) and the system wants to be informed of the progress of each activity. This callback can be configured declaratively or programmatically as needed.

The external system will have to implement the CallBack Service that will allow it to receive information from the BDI process flow. The external system can call the process flow passing the context information as process flow parameters. The process flow will pass the information back when it makes the CallBack Service call.

How to start Process Flow with input parameters?

To start a bdi process flow user has to make a REST service call to URL (http://<host>:<port>/bdi-process-flow/resources/batch/processes/operator/<processName>) . The call must be a POST call to the URL.

The process flow start call accepts http query parameters. The format of the query parameters are as follows:

http://localhost:7001/bdi-process-flow/resources/batch/processes/<ProcessName>?processParameters=callerId=<value1>,correlationId=<value2>,callBackServiceDataDetail.<name1>=<value3>,callBackServiceDataDetail.<name2>=<value4>

Spaces are not allowed in query parameters and must be separated by commas.

Example: http://localhost:7001/bdi-process-flow/resources/batch/processes/Abc_Process?processParameters=callerId=123,correlationId=abc,callBackServiceDataDetail.def=xyz,callBackServiceDataDetail.abc=123

Following are the context information that need to be passed to BDI process flow from calling system.

1. **callerId:** CallerId parameter is used to identify the invoker of process flow.
2. **correlationId:** Correlation id is the main identifier used by the calling system to tie the process flow Start call to the eventual CallBack Service call.
3. **callBackServiceDataDetail.<name>=** These are additional key value pairs that may be required in future as required by the caller.

All of the above parameters are optional. However, if the context is not passed the caller may not be able to associate the invocation with the callback.

Call back from Processflow

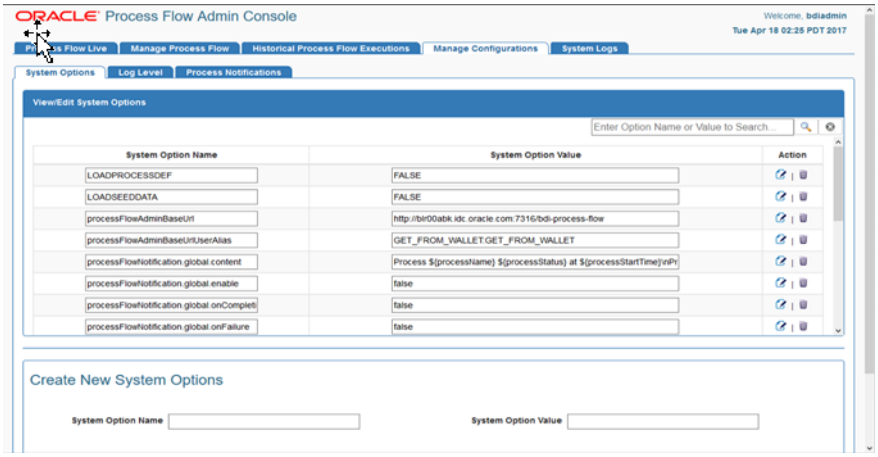
A new method (invokeCallBackService) is available for Process Flow DSL that will allow process flow to call an external service. This service has following features.

- The method internally invokes a REST call to the provided URL
- The method uses Basic Authentication for the rest call. The credentials for the method call must be available in the process flow.
- The payload sent from process flow to the invoking application follows the contract as shown in the example in the next section. All of the values, other than keyValueEntryVo, are populated by the Process Flow engine. The DSL writer can modify the keyValueEntryVo before the callback to pass any custom value from the DSL to invoking application
- The result of the callback REST service must be a String value.
- If the callback service invocation fails for any reason (.e.g., network issue), the process flow activity fails and the process flow is marked as failed.

How to invoke the Callback Service declaratively

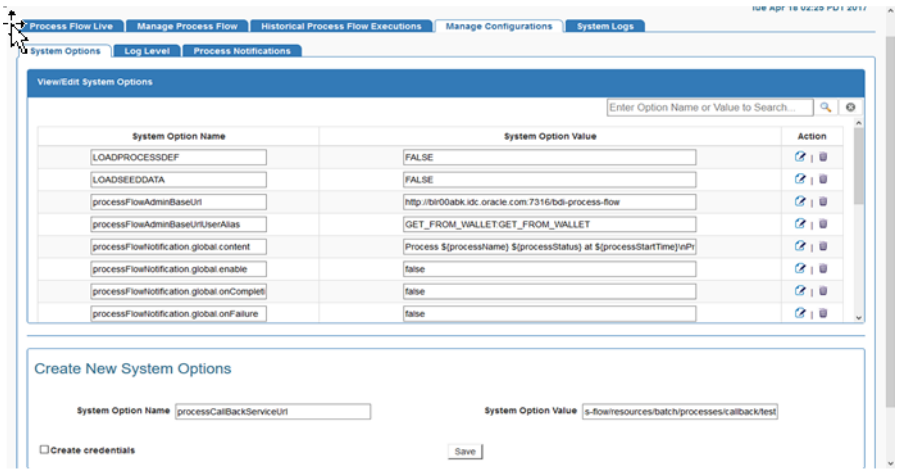
- Setup the callback URL in processflow system options. To configure a callback URL you should add system options like <serviceName>CallbackServiceUrl, for eg., processCallbackServiceUrl.
  - In Process Flow admin console, navigate to Manage Configurations tab and System Options sub-tab.

Figure 6–17 System Options Tab



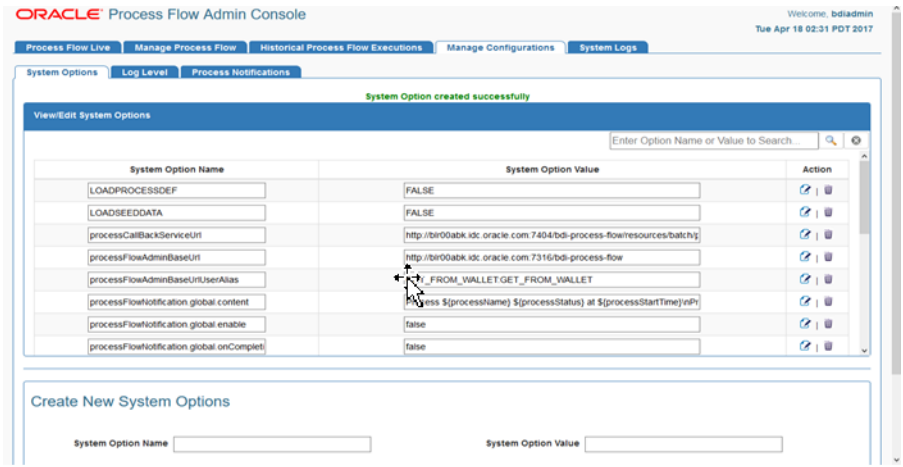
– Scroll down to Create New System Options, enter System Option Name and System Option Value. Url should be a valid ReST Service.

Figure 6–18 Create New System Option Value



– Click Save.

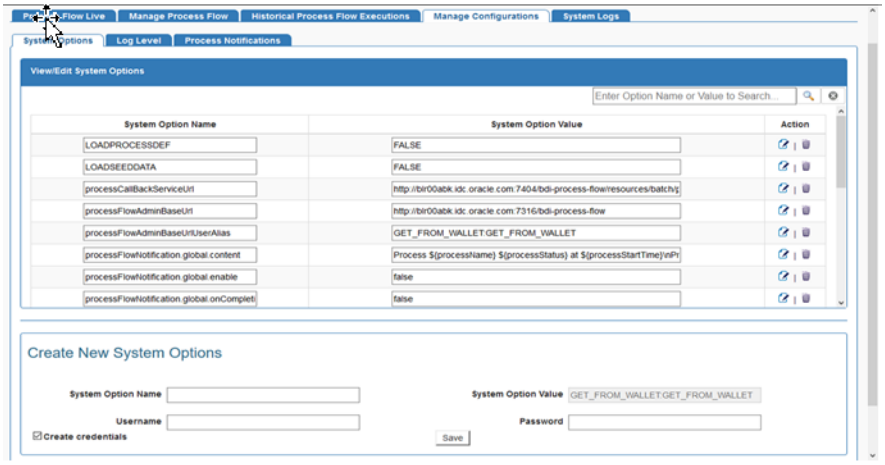
Figure 6–19 View/Edit System Options



- Setup the callback URL credential alias in process flow. To add callback URL credential alias you should add credential alias like <serviceName>CallbackServiceUrlUserAlias, for eg., processCallbackServiceUrlUserAlias.

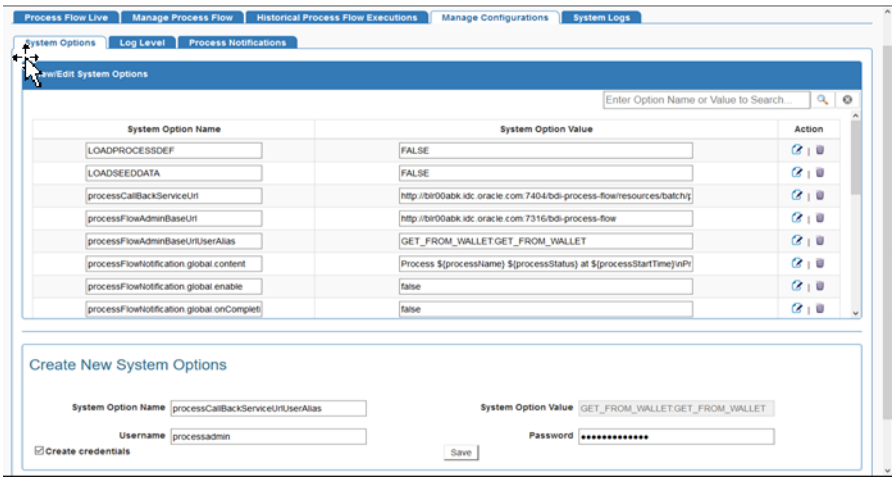
  – In the Create New System Options section, select Create Credentials checkbox.

Figure 6–20 Create Credentials



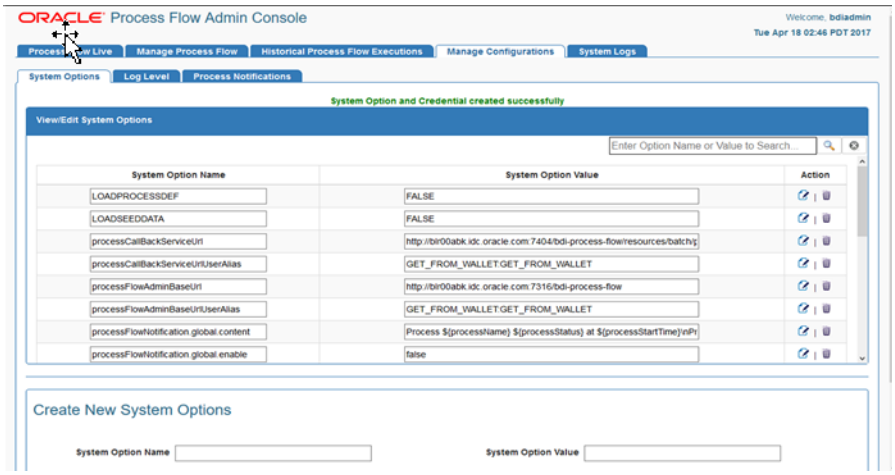
- Enter System Option Name, Username and Password for the URL provided in the previous step. If the System Option Name for the URL is processCallbackServiceUrl then System option name for credential should be processCallbackServiceUrlUserAlias.

Figure 6–21 View/Edit System Options



- Click **Save**.

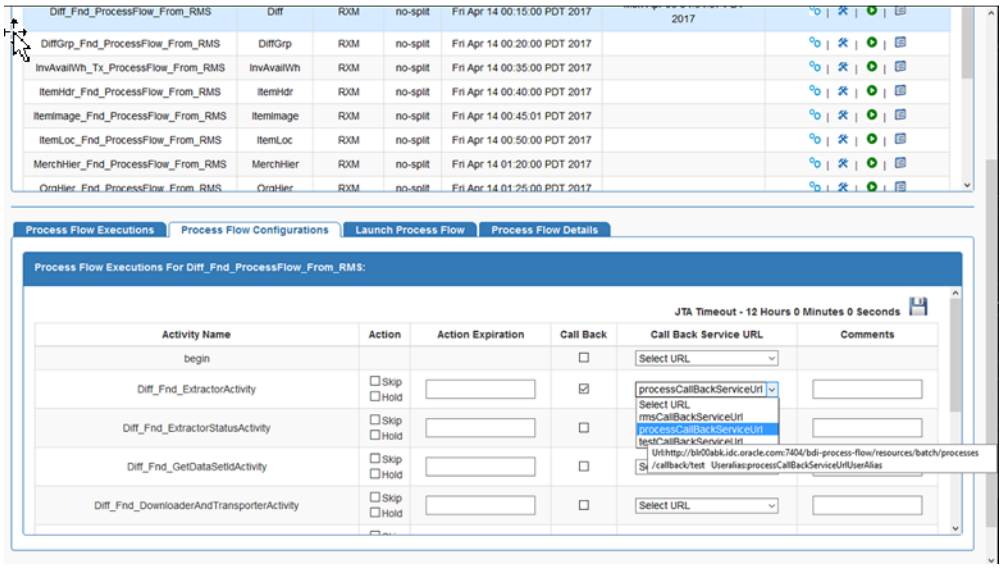
Figure 6–22 Save System Options and Credentials



Note: Credentials created through UI are available after server restart, but after redeployment of the application credentials have to be created again.

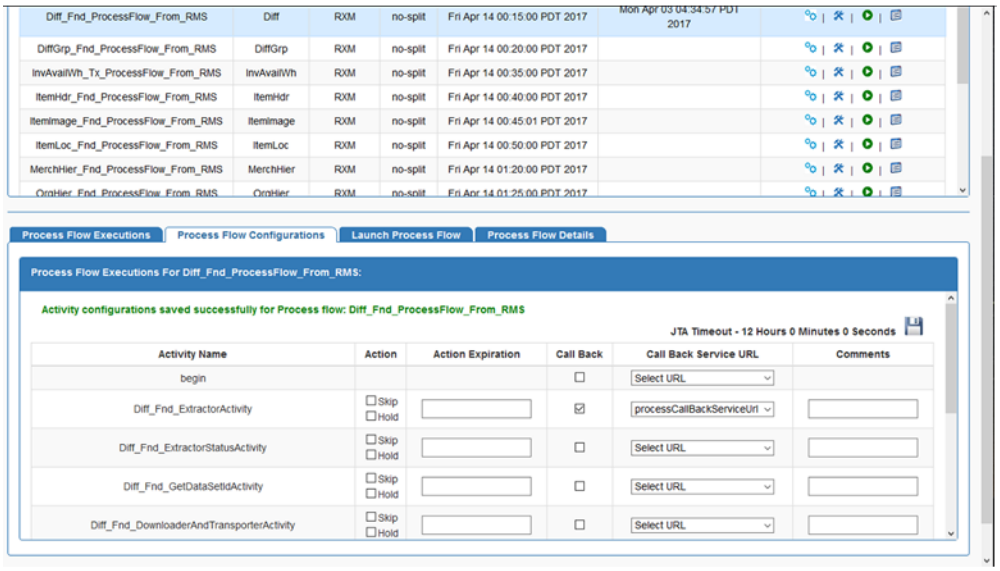
- Navigate to Manage Process Flow tab and select process flow, go to Process Flow Configurations sub-tab.
- Select Callback checkbox for the activities you want callback to be enabled. Select Callback URL from drop down list.

Figure 6–23 Process Flow Configurations



- Click **Save**.

Figure 6–24 Save Process Flow Configuration



How to invoke the Callback Service programmatically

From the Process Flow DSL activity, you can invoke the callback service as shown in the examples below. The callBackServiceUrl and callBackServiceUrlUserAlias property must be setup in the System Options inside process flow.

Example 1: Short Form

Add the following line inside BDI process flow activity.

def retValue = invokeCallBackService(externalVariables.callBackServiceUrl, externalVariables.callBackServiceUrlUserAlias).

Example 2: Long Form

In the long form API the callBackServiceData is an implicit parameter that is automatically defined and user can update it with additional data inside an activity if they want.

Add the following line inside BDI process flow activity.

```
//optionally update some data
callBackServiceData.keyValueEntryVo[0].key = "Some Key"
callBackServiceData.keyValueEntryVo[0].value = "Some Value"
def retValue = invokeCallBackService(externalVariables.callBackServiceUrl, externalVariables.callBackServiceUrlUserAlias, callBackServiceData)
```

Callback request Payload structure

The BDI process flow will make a POST REST call to the callBackServiceUrl passing in the following payload. JSON is the default content type.

JSON Payload Contract

```
{
"processName": "Abcdef_Process",
```

"processExceutionId": "123456",

"activityName": "Def_Activity",

"activityExecutionId": "12345678",

"callerId": "XYZ",

"correlationId": "987654321",

"keyValueEntryVo": [

{

"key": "abc",

"value": "def"

},

{

"key": "pqr",

"value": "123"

}

XML Payload Contract

```
<?xml version="1.0" encoding="UTF-8" ?>
<callBackServiceVo>
<processName>Abcdef_Process</processName>
<processExceutionId>123456</processExceutionId>
<activityName>Def_Activity</activityName>
<activityExecutionId>12345678</activityExecutionId>
<callerId>XYZ</callerId>
<correlationId>987654321</correlationId>
<keyValueEntryVo>
<key>abc</key>
<value>def</value>
 </keyValueEntryVo>
<keyValueEntryVo>
<key>pqr</key>
<value>123</value>
 </keyValueEntryVo>
 </callBackServiceVo>
```

CallBackService Error message contract

Call Back Service Scenarios

Activity Type	Activity Action (Skip or Hold)	Callback behaviour (if callback enabled)	Activity Status sent by Callback	Activity Status if Callback fails
Any	None	Callback will be called after action part is complete	ACTIVITY_ COMPLETE or ACTIVITY_ FAILED according to the action part success or failure.	ACTIVITY_FAILED

Activity Type	Activity Action (Skip or Hold)	Callback behaviour (if callback enabled)	Activity Status sent by Callback	Activity Status if Callback fails
	Skip	Callback will be called after action part is complete	ACTIVITY_ SKIPPED	ACTIVITY_FAILED
	Hold	Callback will be called when hold is released and after the action part of the activity runs	ACTIVITY_ COMPLETE or ACTIVITY_ FAILED according to the action part success or failure.	ACTIVITY_FAILED
Special Cases				
startOrRestartJobActivity	None	Callback will be called as soon as the job start or restart call is complete	ACTIVITY_ COMPLETE if the job was started or restarted successfully. ACTIVITY_ FAILED if the job was not started or restarted successfully.	ACTIVITY_FAILED
waitForJobCompletedOrFailed	None	Callback will be called after the Job status has reached complete or failed	ACTIVITY_ COMPLETE if the job completed successfully. ACTIVITY_ FAILED if the job failed.	ACTIVITY_FAILED
Restart Scenarios				
startOrRestartJobActivity	None	Job will be started or restarted only if the Job was not started earlier or job failed. If the activity failed due to callback failure the job will not be started.	ACTIVITY_ COMPLETE if the job was started or restarted successfully. ACTIVITY_ FAILED if the job was not started or restarted successfully.	ACTIVITY_FAILED

Activity Type	Activity Action (Skip or Hold)	Callback behaviour (if callback enabled)	Activity Status sent by Callback	Activity Status if Callback fails
waitForJobCompletedOrFailed	None	Callback will be called after checking the Job status, if it has reached complete or failed, otherwise process will wait for the job to reach complete or failed status.	ACTIVITY_COMPLETE if the job completed successfully. ACTIVITY_FAILED if the job failed.	ACTIVITY_FAILED

Enable or Disable a Process Flow using REST Service

This endpoint enables or disables the process flows using name, processEnableStatus and returns the name, processEnableStatus and message.

Path: /batch/processes/enable-disable

HTTP Method: POST

Inputs

```
JSON
[
      {
      "name": "MerchHier_Fnd_ProcessFlow_From_RMS",
      "processEnableStatus": "true"
   },
      {
      "name": "ItemLoc_Fnd_ProcessFlow_From_RMS",
      "processEnableStatus": "false"
   }
]
```

Sample Request

http://localhost:7001/bdi-process-flow/resources/batch/processes/enable-disable

Successful Response

```
JSON
[
      {
      "name": "MerchHier_Fnd_ProcessFlow_From_RMS",
      "processEnableStatus": "ENABLED",
      "message": "Process Enabled Successfully"
   },
      {
      "name": "ItemLoc_Fnd_ProcessFlow_From_RMS",
      "processEnableStatus": "DISABLED",
      "message": "Process Disabled Successfully"
   }
]
```

Process Execution Trace

The Process Flow engine keeps track of process execution details in the BDI_PROCESS_CALL_STACK_TRACE table. In order for a sub-process to appear in the trace, the sub-process must be called with the api as shown below.

```
triggerProcess(<Base URL>, <Sub Process Name>, <credentials>, <Process Parameter Map>)
```

Example:

```
triggerProcess("http://host:port/bdi-process-flow", "DiffGrp_Fnd_ProcessFlow_From_RMS", "userid:password", null)
```

REST end point to get process execution trace

http://<host>:<port>/bdi-process-flow/resources/telemetry/processes/execution-trace/{ProcessExectionId}

Sample Output:

```
{
  "executionId": "Diff_Fnd_ProcessFlow_From_
RMS~8e1c7c11-1302-409d-9102-c55fffbdc1ab",
  "executionName": "Diff_Fnd_ProcessFlow_From_RMS",
  "activityExecutionId": "",
  "url": "",
  "status": "PROCESS_COMPLETED",
  "duration": 0,
  "type": "PROCESS",
  "invocationTime": "2017-07-19T12:21:20.061-06:00",
  "children": [
    {
      "executionId": "ItemImage_Fnd_ProcessFlow_From_
RMS~89f46519-50ab-4a51-a6fb-c6c5395afeca",
      "executionName": "ItemImage_Fnd_ProcessFlow_From_RMS",
      "activityExecutionId": "Activity2~a408b407-c4f0-4137-ba32-6ddd148f0838",
      "url":
"http:\/\/msp8917:8001\/bdi-process-flow\/resources\/batch\/processes\/operator\/ItemImage_Fnd_ProcessFlow_From_RMS",
      "type": "PROCESS",
      "invocationTime": "2017-07-19T12:21:20.534-06:00",
      "children": [
      ]
    },
    {
      "executionId": "DiffGrp_Fnd_ProcessFlow_From_
RMS~bb68a1ea-86a5-4108-aa58-b9e791d1fb8c",
      "executionName": "DiffGrp_Fnd_ProcessFlow_From_RMS",
      "activityExecutionId": "Activity1~602ad027-7946-4820-acd8-cf452f5fc937",
      "url":
"http://host:port/bdi-process-flow/resources/batch/processes/operator/DiffGrp_Fnd_ProcessFlow_From_RMS",
      "type": "PROCESS",
      "invocationTime": "2017-07-19T12:21:20.296-06:00",
      "children": [
        {
          "executionId": "ItemHdr_Fnd_ProcessFlow_From_
RMS~3886b39f-6268-4895-8e5e-300ded42665b",
          "executionName": "ItemHdr_Fnd_ProcessFlow_From_RMS",
          "activityExecutionId": "Activity2~8e9f9a6a-440a-41dd-a648-f4322102012b",
```

```
          "url":
"http://host:port/bdi-process-flow/resources/batch/processes/operator/ItemHdr_Fnd_
ProcessFlow_From_RMS",
          "type": "PROCESS",
          "invocationTime": "2017-07-19T12:21:20.705-06:00",
          "children": [

          ]
        },
        {
          "executionId": "InvAvailWh_Tx_ProcessFlow_From_
RMS~6c462406-a991-4754-9d94-73628091114a",
          "executionName": "InvAvailWh_Tx_ProcessFlow_From_RMS",
          "activityExecutionId": "Activity1~e7f8e9fa-7ba6-4a51-81e2-bdcfe752c15e",
          "url":
"http://host:port/bdi-process-flow/resources/batch/processes/operator/InvAvailWh_
Tx_ProcessFlow_From_RMS",
          "type": "PROCESS",
          "invocationTime": "2017-07-19T12:21:20.538-06:00",
          "children": [
          ]
        }
      ]
    }
  ]
}
```

Process Metrics Service

Process Metrics provides an end point to produce metrics for processes that ran between "fromTime" and "toTime".

Path: /telemetry/processes

HTTP Method: GET

Parameters:

fromTime - Query parameter

toTime - Query parameter

Sample Response:

```
<process-runtime-monitoring-info>
    <data-requested-at>2017-10-09T10:24:27.848-06:00</data-requested-at>
    <data-requested-from-time>2017-03-01T00:00:00-06:00</data-requested-from-time>
    <data-requested-to-time>2017-08-01T00:00:00-06:00</data-requested-to-time>
    <process-server-runtime-info>
        <id>bdi-process</id>
        <app-status>RUNNING</app-status>
        <up-since>2017-10-09T10:22:34.498-06:00</up-since>
        <total-executions-count>16</total-executions-count>
        <successful-executions-count>8</successful-executions-count>
        <failed-executions-count>7</failed-executions-count>
        <process>
            <name>DiffGrp_Fnd_ProcessFlow_From_RMS</name>
            <slowest-run-duration>0.0</slowest-run-duration>
            <fastest-run-duration>120.0</fastest-run-duration>
            <avg-run-duration>60.2315</avg-run-duration>
            <executions>
                <exceution-count>1</exceution-count>
```

```
                <success-count>0</success-count>
                <failure-count>1</failure-count>
                <execution>
                    <execution-id>
DiffGrp_Fnd_ProcessFlow_From_RMS~650dba75-b632-42ea-963b-802c560d0c6b
</execution-id>
                    <status>PROCESS_FAILED</status>
                    <start-time>2017-05-17T14:39:32.489-06:00</start-time>
                    <end-time>2017-05-17T14:39:33.535-06:00</end-time>
                    <activity-exe>

<activity-exe-id>begin~2ac2bc4d-6233-41ac-a134-5fb73ebba275</activity-exe-id>
                        <name>begin</name>
                        <duration>0.0</duration>
                        <status>ACTIVITY_COMPLETED</status>
                    </activity-exe>
                    <activity-exe>
                        <activity-exe-id>
DiffGrp_Fnd_ExtractorActivity~035b6e78-411e-4868-b441-f2e79a3dba61
</activity-exe-id>
                        <name>DiffGrp_Fnd_ExtractorActivity</name>
                        <duration>0.0</duration>
                        <status>ACTIVITY_SKIPPED</status>
                    </activity-exe>
                    <activity-exe>
                        <activity-exe-id>
DiffGrp_Fnd_ExtractorStatusActivity~7d92a1c1-721a-416d-86ac-c412f9e49982
</activity-exe-id>
                        <name>DiffGrp_Fnd_ExtractorStatusActivity</name>
                        <duration>0.0</duration>
                        <status>ACTIVITY_SKIPPED</status>
                    </activity-exe>
                    <activity-exe>
                        <activity-exe-id>
DiffGrp_Fnd_GetDataSetIdActivity~423d19e3-8c9d-44b2-93b9-183f41cd0840
</activity-exe-id>
                        <name>DiffGrp_Fnd_GetDataSetIdActivity</name>
                        <duration>0.0</duration>
                        <status>ACTIVITY_SKIPPED</status>
                    </activity-exe>
                    <activity-exe>
                        <activity-exe-id>
DiffGrp_Fnd_DownloaderAndTransporterActivity~70bac2cb-c414-4be8-a5ab-0ef21fd2fc4d
</activity-exe-id>
                        <name>DiffGrp_Fnd_DownloaderAndTransporterActivity</name>
                        <duration>0.0</duration>
                        <status>ACTIVITY_FAILED</status>
                    </activity-exe>
                    <activity-exe>

<activity-exe-id>end~5c07a938-864b-4156-bab7-70b96bcb2d74</activity-exe-id>
                        <name>end</name>
                        <duration>0.0</duration>
                        <status>ACTIVITY_FAILED</status>
                    </activity-exe>
                </execution>
            </executions>
        </process>
    </process-server-runtime-info>
</process-runtime-monitoring-info>
```

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
Skip/Hold/Release	No	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 "okay" moveTo ..	Jump to a specific activity on matching the condition.
on "error" moveTo ..	Use these keywords inside an activity to move to error activity.
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.

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 BDI_PROCESS_DEFINITION.

After deployment, the process flow can be edited by the Admin user through the Process Flow Admin application, tab Manage Process Flow, sub tab Process Flow Details, sub tab Process Dsl. 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. For example keywords System, Thread etc. 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 sub processes 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.
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

Table Name	Description
BDI_ACTIVITY_DYNAMIC_CONFIG	This table stores the user runtime choices like SKIP, HOLD etc at activity level
BDI_SYSTEM_OPTIONS	This table has all the system level information like URLs, credential aliases etc.
BDI_EMAIL_NOTIFICATION	This table persist records for email notifications sent
BDI_EXTERNAL_VARIABLE	This table does temporary storage of variables during process execution.
BDI_PROCESS_CALL_STACK	This table stores call stack for processes
BDI_GROUP	This table stores group names and its attributes.
BDI_GROUP_MEMBER	This table stores all group member details.
BDI_GROUP_LOCK	This table stores group names and lock ids.

Process Customization

Seed Data

During the deployment of Process Flow, seed data gets loaded. Seed data files are located in "bdi-process-home/setup-data/dml" folder. If seed data is changed, Process Flow needs to be reinstalled and redeployed. For loading seed data during redeployment, LOADSEEDDATA flag in BDI_SYSTEM_OPTIONS need to be set to TRUE.

Process DSL Reload

Along with seed data, the process DSL also gets loaded to BDI_PROCESS_DEFINITION table during the deployment time. Process DSLs are located in "bdi-process-home/setup-data/dsl/flows-in-scope" folder. If you want to load DSLs again after DSLs are added or update, Process Flow needs to be redeployed. For loading DSLs during the redeployment, LOADPROCESSDEF flag in BDI_SYSTEM_OPTIONS table need to be set to TRUE.

Deployment of Process Flow first time loads both seed data and process DSLs.

Redeployment loads seed data depending on the LOADSEEDDATA and LOADPROCESSDEF flag values.

Before redeployment make sure for every install/upgrade one needs to look at /available_process_flow_options/rms_enterprise-sender_side_split_flows i.e. /bdi-process-home/setup-data/dsl//available_process_flow_options/rms_enterprise-sender_side_split_flows, to ensure they have the correct set of flows for that installation, each release would bring in functional changes and flows files define the primary functional definition of a BDI integration flow.

Do the following:

1. Download the process flow archive BdiProcessFlow19.1.000ForAll19.x.xApps_eng_ga.zipUnzip the downloaded archive. The Process Home directory will be created under the current directory.

   ```
   unzip BdiProcessFlow19.1.000ForAll19.x.xApps_eng_ga.zip
   ```

2. Modify process flow configuration file (conf/bdi-process-flow-admin-deployment-env-info.json) to match the deployment environment.

> **Note:** The alias names in the configuration files should not be changed.

3. BDI Process flow installer copies all the enterprise flows from bdi-process-home/setup-data/dsl/available_process_flow_options/rms_enterprise-sender_side_split_flows/ to bdi-process-home/setup-data/dsl/flows-in-scope.

   For RFI integration user should copy the flows manually from bdi-process-home/setup-data/dsl/available_process_flow_options/reim_rfi-no_split_flows / to bdi-process-home/setup-data/dsl/flows-in-scope, for example:

   ```
   cp bdi-process-home/setup-data/dsl/available_process_flow_options/reim_rfi-no_split_flows/* bdi-process-home/setup-data/dsl/flows-in-scope/
   ```

4. Configure the appsInScope system options in process flow configuration file.
5. Run the deployer. Make sure that the WebLogic server is running before issuing the following command.

   ```
   cd bin
   bdi-process-flow-admin-deployer.sh -setup-credentials
   -deploy-process-flow-admin-app
   ```

   > **Note:** If you have an existing process flow deployment then, login to Process Flow App, go to Manage Configurations -> System Options and update the following system options before running the above command. LOADPROCESSDEF = TRUE and LOADSEEDDATA = TRUE

   If you have already configured various credentials required for process flow, you can run the deployer with the following syntax. It will not ask the credentials again for the deployment. Make sure you set the LOADPROCESSDEF = true, LOADSEEDDATA = true.

   ```
   bdi-process-flow-admin-deployer.sh -use-existing-credentials
   -deploy-process-flow-admin-app
   ```

6. Make sure the deployment step shows deployment success message at the end.
7. Restrict access to the bdi-process-home folder:

   ```
   cd bdi-process-home chmod -R 700
   ```

8. Bounce the process managed server.

Redeployment scenarios

LOADSEEDDATA	LOADPROCESSDEF	Behavior
TRUE	TRUE	Loads both seed data and process DSLs
TRUE	FALSE	Loads seed data only
FALSE	TRUE	Loads process DSLs only
FALSE	FALSE	Does not load seed data and process DSLs

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 UnmarshallException

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 rebounce the server.

Process Flow Did Not Start

To address this, verify the logs. It could be due to the missing Credentials Access permission, missing system credentials, or a missing system options or DSL parsing error.

Deleted process flow still listed in the UI

Deleting a process flow from bdi-process-home doesn't deletes it from the process flow application, because the process flow application refers the database entries, so in order to delete a process flow from BDI Process Flow app, the script DELETE_ PROCESS_FLOW.sql(bdi-process-home/setup-data/dml/) has to be run in BDI ProcesFlowAdminDataSource Schema.

Best Practices for Process Flow DSL

- Use naming conventions for process flows and activities in process flow so that they are easily identified. It is recommended that name of the process flow includes “Process” and the name of activities ends with “Activity”.
- Use built in “startOrRestartJob” method to start/restart job in Job Admin.

  Use built in “waitForJobCompletedOrFailed” method to wait until job is complete or failed.
- Access system options through “externalVariables”.
- Use “processVariables” to share variables between activities.
- Use built in “waitForProcessInstancesToReachStatus” to wait for other process instances.
- Use built in “waitForProcessNamesToReachStatus” to wait for other processes.
- Use the built-in triggerProcess to start a sub process.
- It is recommended to use “flo” as extension for process flow DSL file.

  Use the built-in REST DSL to make rest calls.
- Organize process flows as hierarchical parent child flows where parent manages the child flows. Avoid using too many waitFor calls as active threads are getting blocked.

7

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 like Thread, System, java etc 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 its successful or failed completion (however long the action may run).

- When sync action starts, the Schedule Action status will 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 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 BDI schedules are 'DISABLED' in seed data user can make the schedules active as per their requirement.

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.

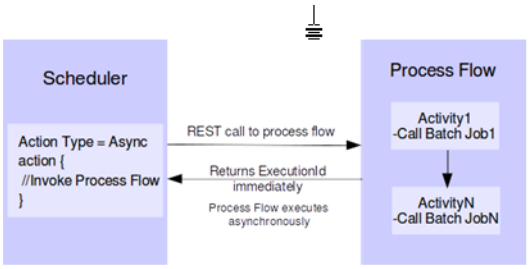
If any of the service calls within the schedule action is a blocking synchronous call and the action is not defined to run in a separate thread, then the action type should be 'sync'.

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 7–1 Schedule Execution - BDI Process Flow

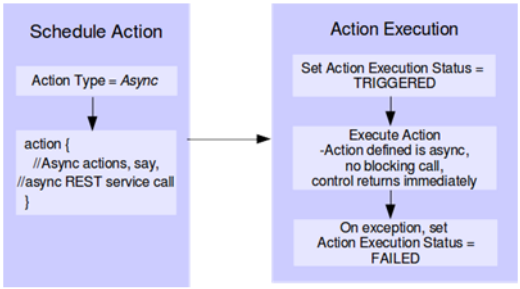


The Process Flow does not block the calling thread (that is, the Scheduler) and returns immediately. The Process flow returns an Execution ID of the process execution instance as a future handle, but will continue to execute the activities defined in the DSL. The activities in BDI process flows may run 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 7–2 Schedule Execution - Async Action

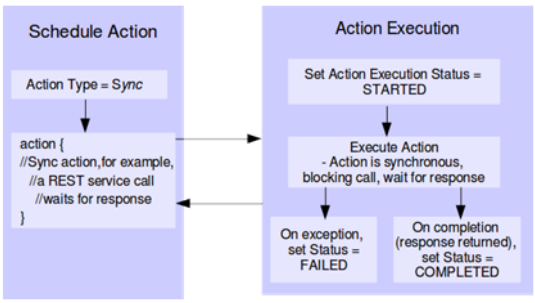


When the schedule action execution starts for async action, the action execution status is set to 'TRIGGERED', and the action is executed. As the action type is specified 'Async', the action should be non blocking, either returning a response immediately or not returning a response and continuing execution, 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 7–3 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'.

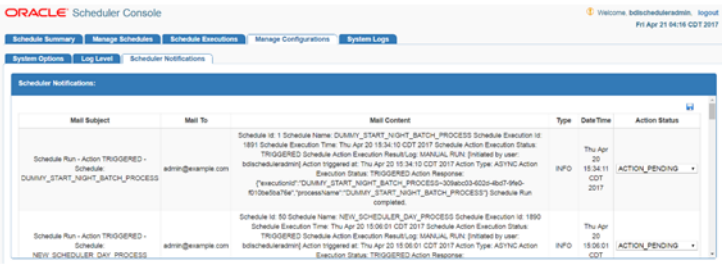
Persisting Schedule Notifications

All schedule notifications are persisted to the BDI_EMAIL_NOTIFICATION table. There is a subtab Schedule Notifications added in Manage Configurations tab which displays all the notifications.

One notification icon appears right top corner of the screen adjacent to the user if there is any notification in PENDING status. User will be navigated to the Schedule Notifications subtab by clicking on the image.

User can modify the status to COMPLETED after going through the notification and click on save button so that next time it doesn't appear on the screen.

Figure 7–4 Persistent Schedule Notifications



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_EMAIL_ NOTIFICATION	All schedule email alerts 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.
BDI_GROUP	This table stores group names and its attributes
BDI_GROUP_MEMBER	This table stores all group member details

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)

REST Resource	Method	Description
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.
/batch/schedules/activateOrDisable-schedules	POST	To update status of one or more schdules to ACTIVE or DISABLED

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.

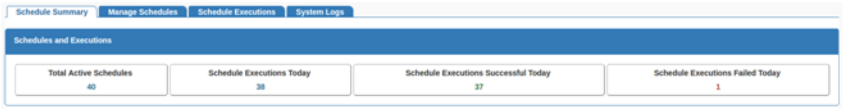
Schedules and Executions

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.

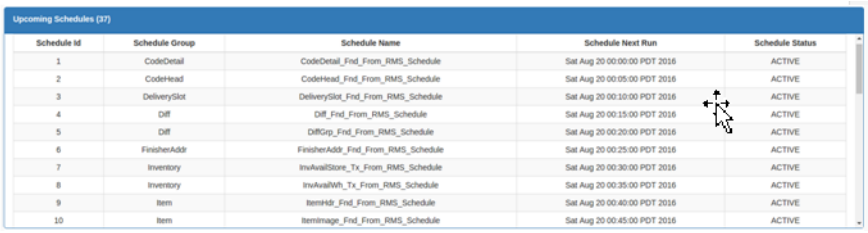
Figure 7–5 Schedules and Executions



Upcoming Schedules

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

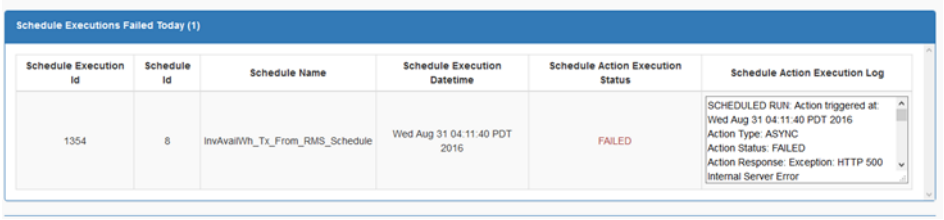
Figure 7–6 Upcoming Schedules



Schedule Executions Failed Today

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

Figure 7–7 Schedule Executions Failed today



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.

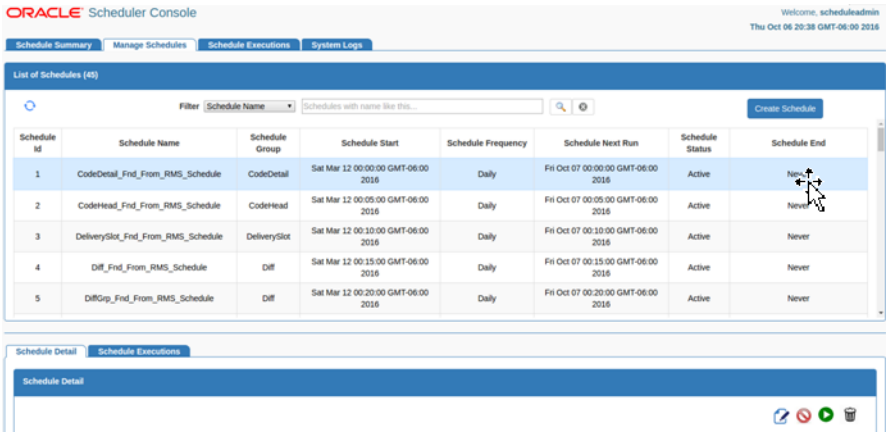
Manage Schedules

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.

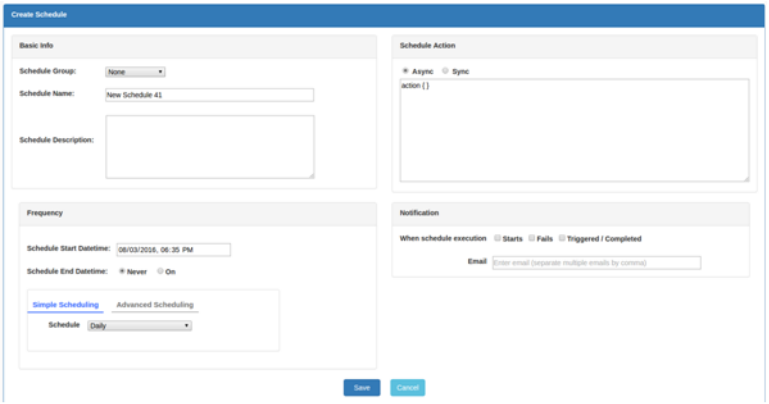
Figure 7–8 Manage Schedules



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 7–9 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

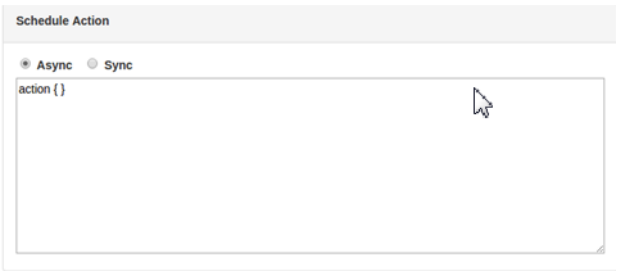
Specify a valid schedule action definition here that will get executed when the schedule runs.

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 7–10 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.

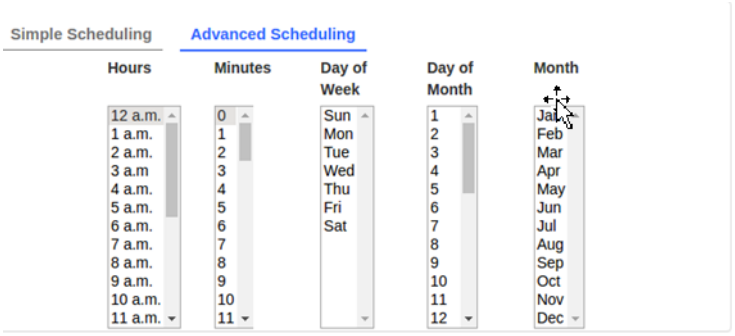
- Hourly
- Daily (selected by default)
- Weekly
- Monthly
- Every Weekday [Mon-Friday]
- On Weekends [Sat-Sunday]
- Every Saturday
- Every Sunday
- First day of every month
- Last day of every month
- One time only
- Specify a different frequency

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 7–11 Schedule Frequency



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.

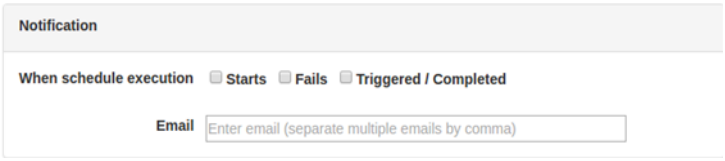
Fails:

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 7–12 Schedule Notification



> **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.

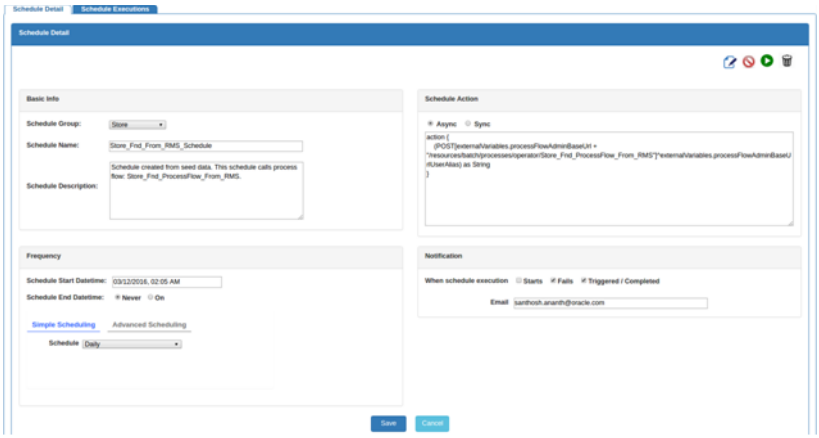
The Edit page is the same as that of the Create Schedule page with the schedule information populated. Update the values as required in the relevant sections as explained previously for creating the schedule. Only admin users can edit a schedule.

> **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 7–13 Updating Schedule



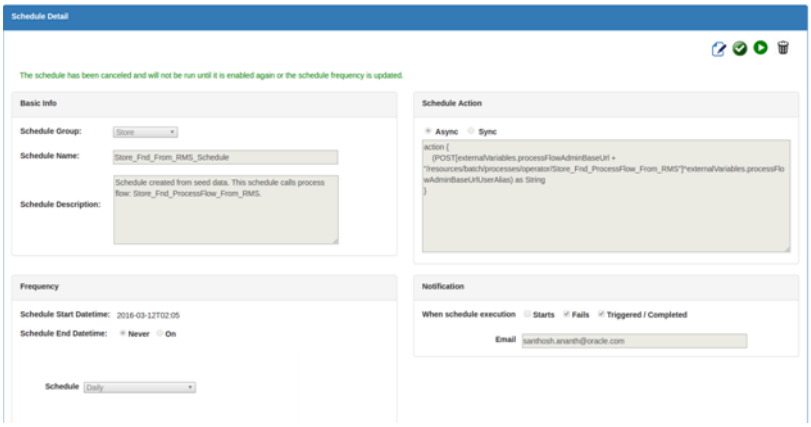
Disabling Schedule

A schedule can be disabled by selecting the schedule from the Mange 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 7–14 Disabling Schedule

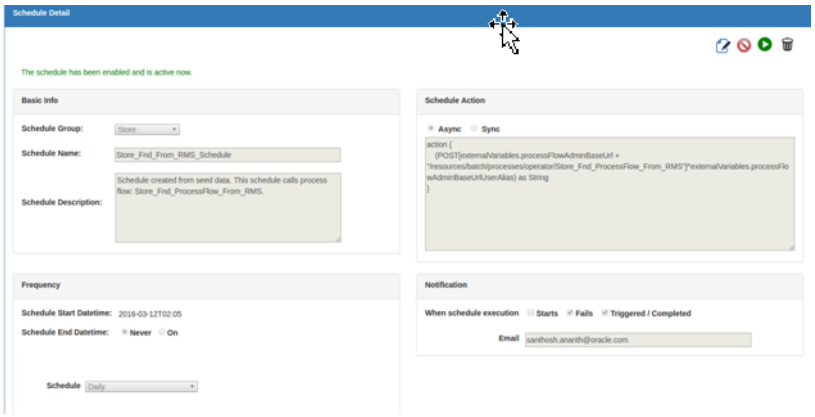


Enabling Schedule

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 7–15 Enabling Schedule

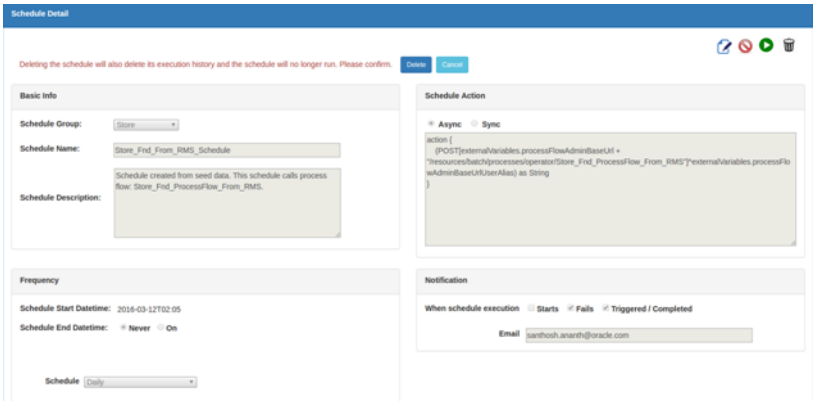


Deleting Schedule

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 7–16 Deleting Schedule



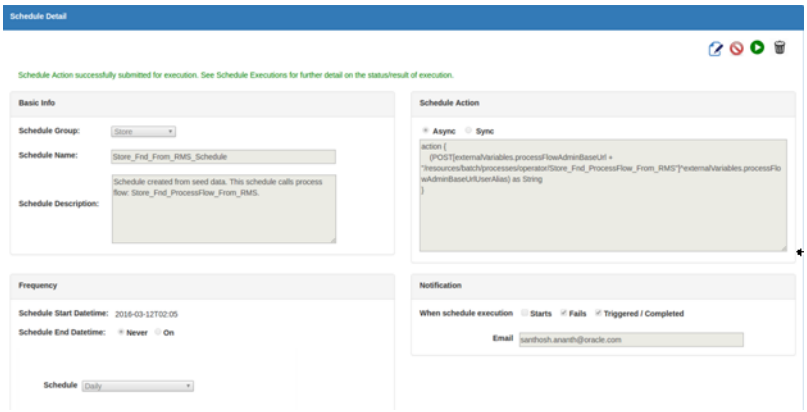
Schedule Manual Run

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 7–17 Schedule Manual Run

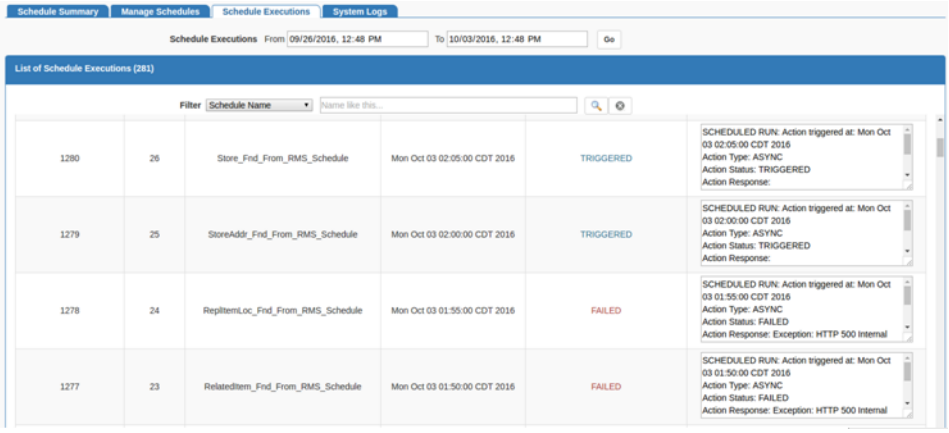


Schedule Executions

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 7–18 Schedule Executions



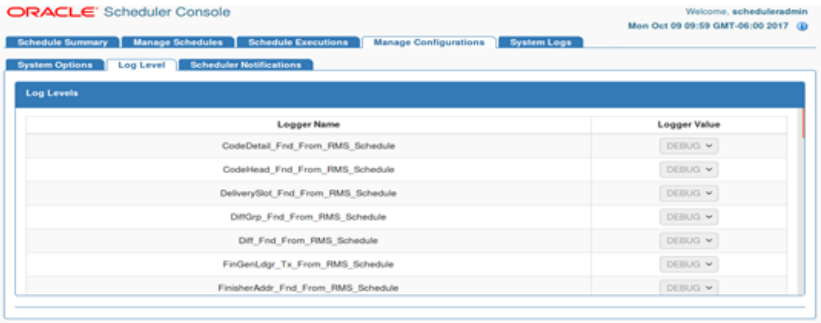
Manage Configurations

From the Manage Configurations page, user can manage log levels, notifications, and system options.

Log Level

The Log Level page displays log levels for all schedules. Users can change log level for one or more schedules.

Figure 7–19 Log Level Page

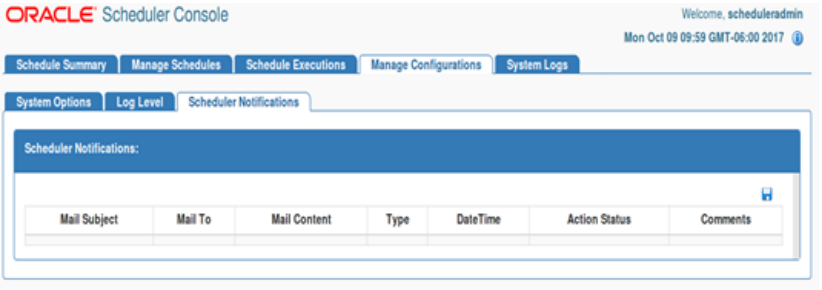


Build version and date is displayed on the info icon when the user selects the same. The icon is on the extreme right top corner of the page.

Notifications

User can view/update notifications details from Scheduler Notifications page.

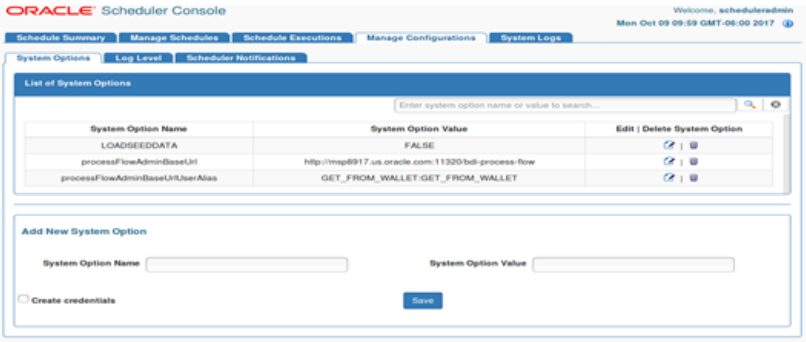
Figure 7–20 Scheduler Notifications Page



System Options

Users can add, update, or delete system options from the System Options page. Credentials can also be created when a system option is created.

Figure 7–21 System Options Page



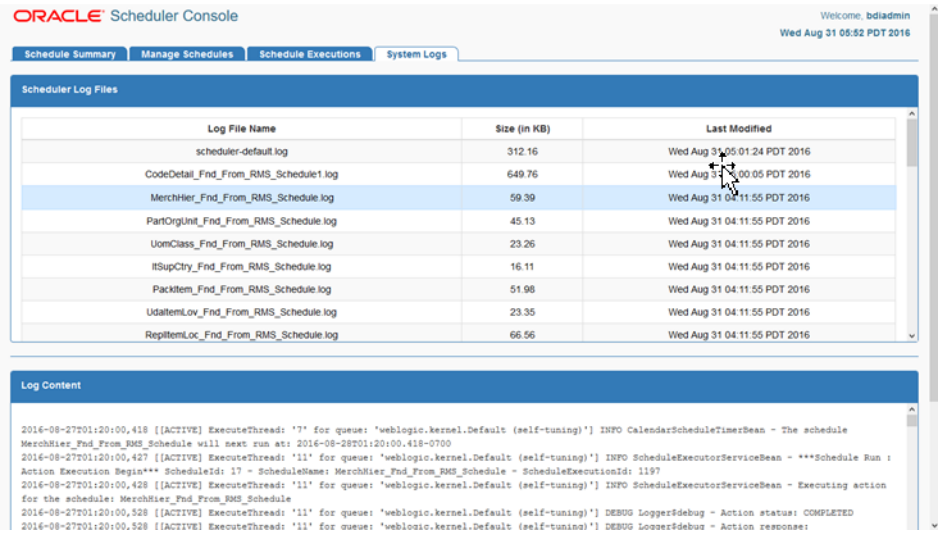
System Logs

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 7–22 System Logs



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

Function	Admin Role	Operation Role	Monitor Role
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: <ASYNC 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***
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 E 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.

8

CLI Tools

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

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 RPAS, then specify the property and value as: rpas.receiver.url=http://<bdi-rpas-app-hostname>:<port>/bdi-rpas-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: rpas.receiver.url.useralias=rpasReceiverUrlUserAlias
<InterfaceModuleName>.receiver.appname	Name of the BDI receiver application for the interface module <InterfaceModuleName>. Specify the name in lowercase. Example: Diff_Fnd.receiver.appname=rpas 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
<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.

- Run: bdi-file-transmitter.sh -get-interface-metadata <receiver-app-name>

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

  For example: bdi-file-transmitter.sh -get-interface-metadata sim

  Run bdi-file-transmitter.sh -get-interface-metadata to configure metadata for a receiver app. On running -get-interface-metadata option the metadata for the receiver app will be saved in the bdi-cli-transmitter/conf/meta-data/<receiver-app-name>.

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

- 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
multiple_files_process_limit	The maximum number of files to process in parallel at any given time. Default value is 5.
file_transmission_thread_limit	The number of parallel threads to run to process a single file. Default value is 3.

Property	Description
transmission_record_size	The maximum number of records per block or chunk to transmit to the receiver service per service call. Default value is 20000.
transmission_timeout	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
-m or --interfacemodule <interfaceModuleName>	(Required) The interface module name. Should be the same as the interface module name expected by the BDI receiver application.
-i or --interfaceshortnames <interfaceShortNames>	(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 bdi-file-transmitter.properties file. The interface names should be the same as expected by the BDI receiver application
-s or --sourcesystem <sourceSystemName>	(Optional) The source system name. If not specified, the program will use the source.system.name given in the properties file.
-f or --filelocation <inputFilePaths>	(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.

Input	Description
-a or --receiverapp <receiverAppName>	(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.
-r or --receiverurl <fileReceiverUrl>	(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.
-u or --useralias <receiverUrlUserAlias>	(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.
-d or --datasettype <dataSetType>	(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#5263_1568696470665_RMS', then after successful transmission the file will be moved to the directory:

/home/bdi/interface/files/archive/Diff_Fnd/Tx#5263_1568696470665_RMS.

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#5263_1568696470665_RMS', then if the transmission of file fails, the file will be moved to the directory: /home/bdi/interface/files/failed/Diff_Fnd/Tx#5263_1568696470665_RMS.

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#5263_1568696470665_RMS

bdi-file-transmitter.sh -retry-failed /home/bdi/interface/files/failed/Diff_Fnd/Tx#5263_1568696470665_RMS/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.

9

BDI Data Integration Topologies

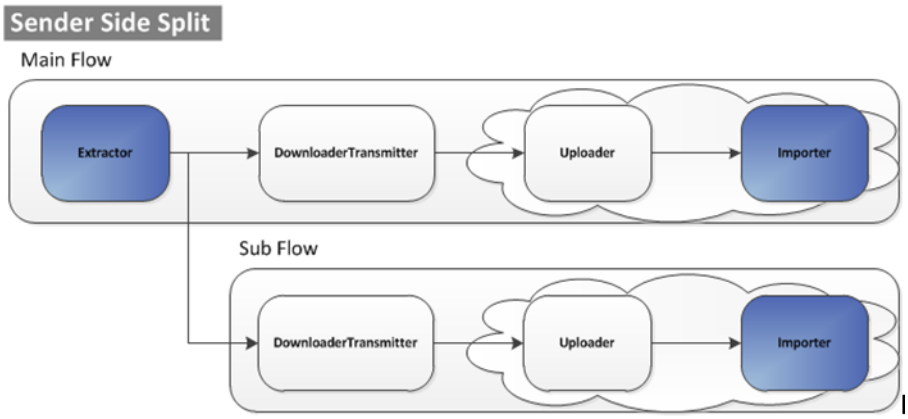
The BDI infrastructure applications move data from one application to another. So there is data producing applications and data consuming applications. Depending on the customer needs, the data produced by an application may be used by one or more consuming applications. This leads to different deployment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

A new change has been introduced to BDI process flows i.e multi destination support. With this change only one instance of BDI process flow will be required at enterprise level. Enterprise process flow by default follows sender side split topology.

Sender side split

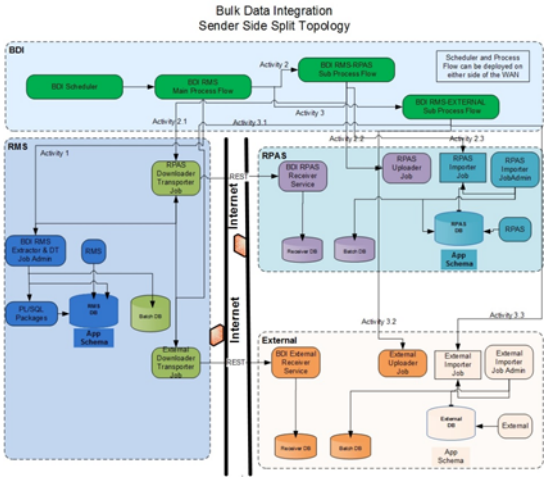
Figure 9–1 Sender Side Split



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.

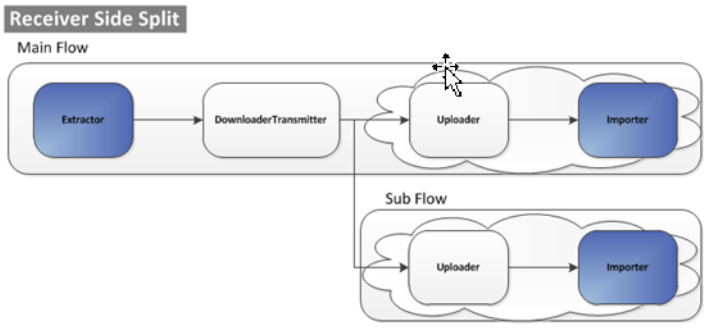
A detailed diagram of sender side split topology usage in Oracle Retail is shown below.

Figure 9–2 Sender Side Split Topology



Receiver Side Split

Figure 9–3 Receiver Side Split

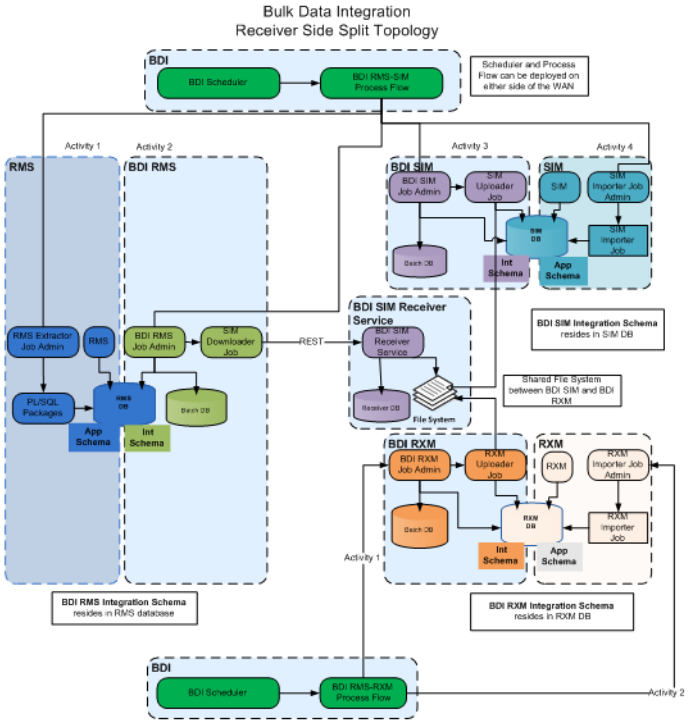


The Receiver Side Split (RSS) topology is used for multi destination data transfer such as Sender Side Split. In this topology data is extracted and transmitted to the destination only once regardless of the number of destinations. This topology differs from the sender side split in the number of times 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 9–4 Receiver Side Split



10

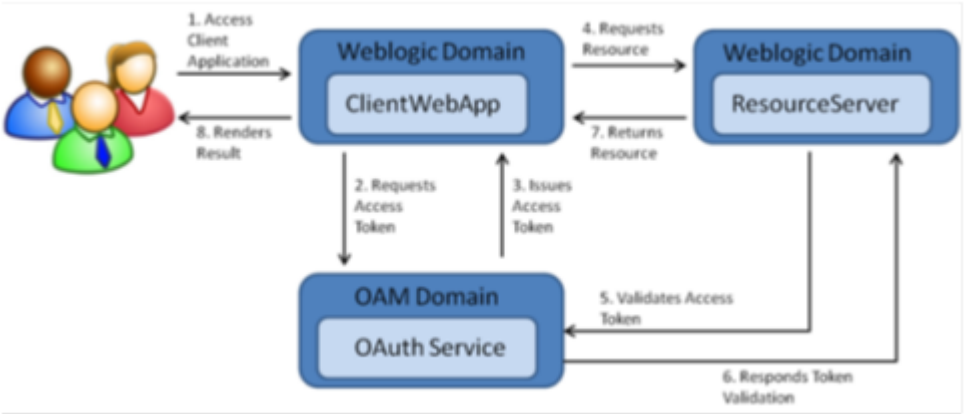
OAuth 2.0

OAuth 2.0 is the industry-standard protocol for authorization. The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

IDCS provides out of the box OAuth Services, which allows a Client Application to access protected resources that belong to an end-user (that is, the Resource Owner).

OAuth 2.0 Architecture Diagram

Figure 10–1 OAuth 2.0 Architecture Diagram



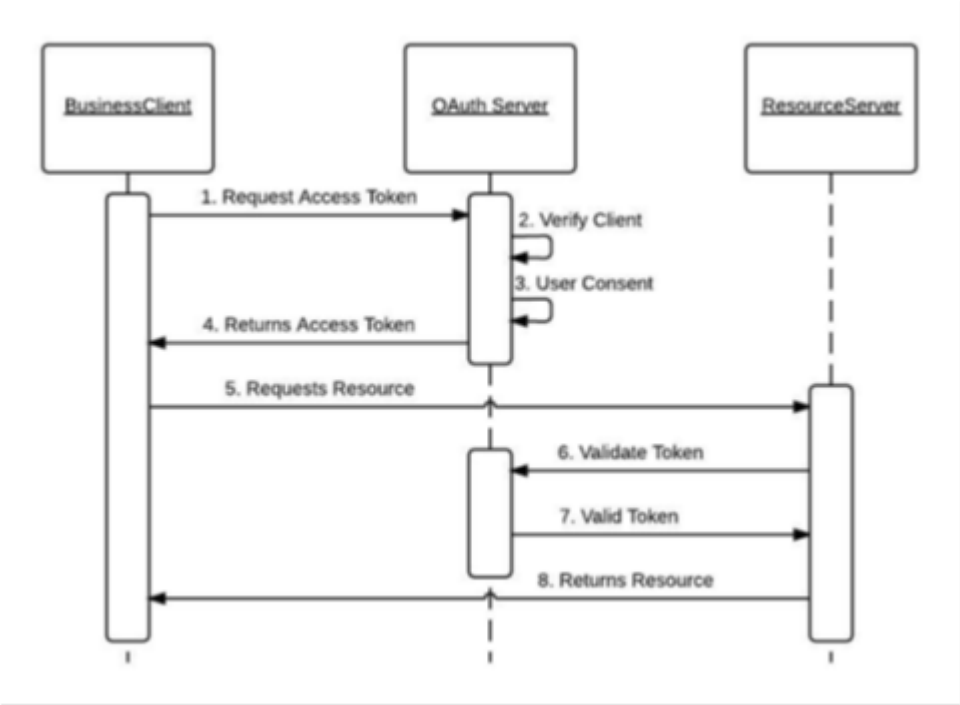
OAuth 2.0 Concepts

Business to Business (2-legged flow)

- It usually represents an application that calls another application or service without end user intervention.
- A client (Business Client application) will make a call to a service, business service (in OAuth spec, a resource server), and request some business information, passing the access token.
- Since there is no end user intervention, the client is pre-authorized to have access to the resource.

OAuth 2.0 Use Case Flow

Figure 10–2 OAuth 2.0 Use Case Flow



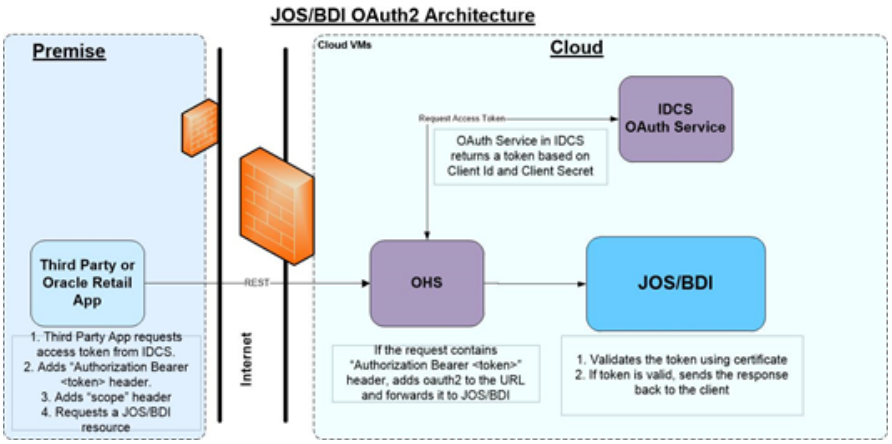
OAuth 2.0 Terms

- **Resource Server** – The server hosting the protected resource.
- **Resource Owner** – An entity capable of granting access to a protected resource.
- **Client** – An application making protected resource requests on behalf of the resource owner. It can be a server-based, mobile or a desktop application.
- **Authorization Server** – The server issuing access tokens to the clients after successfully authenticating the resource owner and obtaining authorization.

BDI OAuth 2.0 Architecture

BDI/JOS uses OAuth 2-legged flow i.e. business to business flow. IDCS provides OAuth services. OHS is Oracle HTTP server that acts as a listener to incoming requests and route them to appropriate service.

Figure 10–3 BDI/JOS OAuth 2.0 Architecture



OAuth 2 Service Provider

BDI/JOS services can be accessed using OAuth 2.0. Use the information provided in Service Consumer section on how to access BDI/JOS services using OAuth 2.0.

Service providers that want to expose services using OAuth 2.0 has to go through the below steps.

Service Provider Configuration

Service provider needs an OAuth identity domain to register resource server information and client profile information so that clients can access the services using OAuth 2.0 protocol.

OAuth 2.0 Service provider distribution includes a configuration file "oauth-configuration-env-info.properties" and install script "oauth-config.sh" to create identity domain, register resource server and client profile information.

Scopes

Scopes allow certain service endpoints to be restricted to clients.

Here are the available scopes.

- AdminAccessScope
- OperatorAccessScope
- MonitorAccessScope

Configuration of scopes for service provider

Service provider needs to configure scope of access for all resource servers in "oauth-configuration-env-info.properties" file.

Here is a sample configuration for scope in service provider. With this configuration, clients can access only BDI Process Flow end points permitted for operator. Multiple scopes can be specified as a comma separated list for a resource server.

```
oauth-configuration-env-info.oauth-resource-server-interface.resourceServerName=bd
i-process-flow,jos-rms-batch-job-admin
oauth-configuration-env-info.oauth-resource-server-interface.bdi-process-flow.scop
eName=OperatorAccessScope
oauth-configuration-env-info.oauth-resource-server-interface.jos-rms-batch-job-adm
```

```
in.scopeName=OperatorAccessScope,MonitorAccessScope
```

OHS Configuration

In cloud environment, all external HTTP requests are routed through OHS (Oracle HTTP Server). OHS needs to be configured to add "oauth2" in the URL after root context and forward the request to appropriate service if the request contains the HTTP header "Authorization: Bearer <token>". This header indicates that the service is protected by OAuth 2.0.

OAuth Server Public Certificate

Service provider uses OAuth server public certificate to validate the token provided in the HTTP request.

Use instructions provided in the *OAuth 2.0 Installation Guide* to import OAuth server public certificate into service provider.

OAuth 2.0 Servlet Filter

Service Provider needs to include "OAuth2ServletFilter" class in "web.xml" to intercept HTTP requests that contain "oauth2" in the path of the URL. The following jars need to be included in the classpath of service provider. The servlet filter validates the token provided in the "Authorization" header and forwards to the service if token is valid.

- oauth2-common-19.1.000.jar
- oauth2-service-provider-api-19.1.000.jar

Add the following in "web.xml" of service provider.

```
<filter>
<filter-name>OAuth2ServletFilter</filter-name>
<filter-class>com.oracle.retail.integration.oauth2.provider.OAuth2ServletFilter</f
ilter-class>
       <init-param>
<param-name>oauth2.serviceProviderConfigClassName</param-name>
<param-value>com.oracle.retail.bdi.common.util.OAuth2ConfigProvider</param-value>
       </init-param>
</filter>

<filter-mapping>
       <filter-name>OAuth2ServletFilter</filter-name>
       <url-pattern>/oauth2/*</url-pattern>
</filter-mapping>

Add the below security-constraint as the last security constraint in "web.xml".

<security-constraint>
       <web-resource-collection>
               <web-resource-name>OAuth2Paths</web-resource-name>
               <url-pattern>/oauth2/*</url-pattern>
       </web-resource-collection>
</security-constraint>
```

OAuth 2.0 Service Consumer

A client can access services protected by OAuth 2.0 using the following methods:

- Use OAuth 2.0 Consumer API
- Use Curl

Access Services using OAuth 2.0 Consumer API

OAuth 2.0 consumer API simplifies access of services protected by OAuth 2.0. The consumer API executes the following steps:

1. Gets the token from IDCS server using client id, client secret, and scope.
2. Adds "Authorization Bearer <token>" HTTP header.
3. Adds "Scope" header with configured scope.
4. Calls the service.

Consumer Configuration

1. Download OAuth2ServiceConsumer19.1.000ForAll19.1.000Apps_eng_ga.zip.
2. Unzip the downloaded archive. The "oauth2-consumer-home" directory will be created under the current directory.

   ```
   Unzip OAuth2ServiceConsumer19.1.000ForAll19.1.000Apps_eng_ga.zip
   ```

   This command extracts the archive. The directories for the installation are shown.

   - /conf/oauth2-service-consumer-config.properties
   - ./lib/oauth2-common-19.1.000.jar
   - ./lib/oauth2-service-consumer-api-19.1.000.jar
   - ./README.txt
3. Edit the oauth-service-consumer-config.properties file to create oauth2 domain environment.

   ```
   vi  oauth-service-consumer-config.properties
   ```
4. Provide the following values in the properties file.

Table 10–1 Configuration Property File Values

Configuration Property	Description
oauth2.default.authorizationServerUrl	URL of OAuth server that issues tokens for default server
oauth2.default.scopeOfAccess.*	Scope of access - *.<scope> for default server (scope - AdminAccessScope, OperatorAccessScope, MonitorAccessScope)
oauth2.default.scopeOfAccess.headers	headers.<scope> for default server (scope - AdminAccessScope, OperatorAccessScope, MonitorAccessScope)
oauth2.default.scopeOfAccess.jos-rms-batch-job-admin	<Root Context>.<scope> for default server
oauth2.srv1.authorizationServerUrl	URL of OAuth server that issues tokens for server "srv1"

OAuth 2.0 Client Sample Code

The following sample code calls discover service of BDI Process Flow application. Make sure that following jars are included in the classpath.

- oauth2-common-19.1.000.jar
- oauth2-service-consumer-api-19.1.000.jar

```
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
Import
com.oracle.retail.integration.oauth2.consumer.OAuth2RestServiceConsumerTokenAppend
er;
import com.oracle.retail.integration.oauth2.consumer.OAuth2ClientBuilder;
import com.oracle.retail.integration.oauth2.consumer.OAuth2Client;

// Code that calls service protected by OAuth 2
void callService() {
       Client client = ClientBuilder.newClient().register(new
OAuth2RestServiceConsumerTokenAppender("JosClientID", "JosClientID1", "srv1"));
       WebTarget target =
client.target("https://host:port/bdi-process-flow/resources/discover");
       String out = target.request().get().readEntity(String.class);
       System.out.println("out=" + out);
}
```

Access Services using Curl

Curl can be used to call a service. There are two steps for calling a service. First issue a curl command to get the token from the authorization server and the second curl command calls the service using the token.

Request Access Token

The following curl command can be used to request access token.

```
Curl -X POST -H "Authorization: <Base64 encoded credentials for Authorization
Server>" -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8" - H
"Accept-Charset: UTF-8" -H "Connection: keep-alive" -H "Content-Length: <length>"
-d "grant_type=client_credentials&scope=<scope>" <url>
```

Sample Scope:

```
bdi-process-flow.OperatorAccessScope
```

Sample Authorization Server URL for JosDomain:

http://host:port/ms_oauth/oauth2/endpoints/JosDomainserviceprofile/tokens

Call Service

```
Curl -X GET -H "Authorization: Bearer <token>" - H "Scope: <scope>" <url>
```

Token: Token obtained using the above curl command

Scope: Scope used to obtain the token

Sample URL: http://host:port/bdi-process-flow/oauth2/resources/discover

IDCS WTSS and WLS Configuration Instructions

IDCS

1. If TAS Service Manager Payload is available, run the script create-property-file-from-service-manager-payload.sh service-manager-payload-file.json
2. If TAS Service Manager Payload is not available, update conf/idcs-tools.properties manually

   e.g.,

   TenantId=tenantId

   ClientID=RGBURICSApp-APPID

   ClientSecret=secret

   IdcsUrl=https://idcs.com
3. Change the email-id in input/rics-users.csv add-users.sh rics.
4. add-groups.sh rics
5. add-app.sh rics prod
6. update-idcs-web-tier-policy-json.sh rics prod
7. Add cloudgate to App Roles in IDCS (Currently a manual step, until a solution is figured out)

   App --> Configuration --> Client Configuration --> +Add (Grant the client access to Identity Cloud Service Admin APIs)

   Select "Cloud Gate" --> Save

WTSS

1. Generate routes.config for WTSS generate-wtss-to-app-routes-info-json.sh service-name environment-label wtssserver-hostname appserver-hostname appserver-port

   For example,

   generate-wtss-to-app-routes-info-json.sh rics prod <wtsserver-hostname> <appserver-hostname> 80
2. Generate IDCS connection info json

   generate-wtss-to-idcs-connection-info-json.sh rics prod
3. Run docker image with the generated files

   For example, docker run --name wtss -v <path>/rics-prod-wtss-to-idcs-connection-info.json:/config/wtss-config.json -v <path>/rics-prod-wtss-to-app-routes-info.json:/config/routes.json -p 80:9999 -d wtss.docker.com/oracle/wtss

WebLogic

1. Add to each managed server startup : -Dweblogic.security.SSL.hostnameVerifier=weblogic.security.utils.SSLWLSWildcardHostnameVerifie
2. Configure IDCS Integrator

Security Realms --> myrealm --> Providers

- Delete OAM and OID providers (if exists)

- Change Control Flag to "SUFFICIENT" or "OPTIONAL" for DefaultAuthenticator

- Add new service provider for IDCS

a. New -->

Name: IDCSIntegrator

Type: OracleIdentityCloudIntegrator

Click OK

b. Click on the created provider

Control Flag: SUFFICIENT

Active Types:

Add "Authorization" to "Chosen:"

Save

c. Click "Provider Specific"

Host: identity.c9dev1.oc9qadev.com

Port: 443

Check SSL Enabled

Tenant: TenantId (from Step #1)

Client Id: (from conf/rics-prod.properties created after Step #4)

Client Secret (from conf/rics-prod.properties created after Step #4)

Confirm Client Secret

Save

Reorder so that IDCS provider is ahead of default provider

OAuth2 (with IDCS) Support in BDI/JOS

- For an application service URL call to work with IDCS OAuth2, need to configure following properties oauth2AuthorizationServerUrl, ClientId, ClientSecret, UserId, UserPassword.
- oauth2AuthorizationServerUrl is configured in BDI System Options table. It needs to point to IDCS URL from where we can get the token. E.g. https://<hostname>/oauth2/v1/token.
- ClientId, ClientSecret are stored in the wallet using the RICS application alias name "ric-sOauth2ApplicationClientAlias". For different oauth applications like MFCS and RPAS we store their ClientId Cli-entSecret under the alias names "mfcsOauth2ApplicationClientAlias", "rpasOauth2ApplicationClientAlias" respec-tively.
-
- ProcessFlow application may call app services that reside in different cloud services(RICS, MFCS, RPAS). Each app service URL that ProcessFlow can call is configured in the BDI System Options table using a "<some name>Url" key naming pattern.

- OAuth2 is enabled or disabled for url "<some name>Url" based on the existence of the OAuth2 alias "<some name>UrlOAuth2ApplicationClientAlias". This "<some name>UrlOAuth2ApplicationClientAlias" must point to the alias name of the ClientId, ClientSecret i.e. "*ricsOauth2ApplicationClientAlias" or "*mfcsOauth2ApplicationClientAlias". The * in the alias name is an indicator that this alias actually points to the shared alias ricsOauth2ApplicationClientAlias or mfcsOauth2ApplicationClientAlias. With this setup we do not have to duplicate the ClientId, ClientSecret for every app service url.
- The BDI install script is modified to ask for OAuth2 specific questions if it detects OAuth2 provider section (CentralAuthenticationSystem/IdcsAuthenticationProvider) is configured.
- The bdi-process-flow-admin-deployment-env-info.json file now has new OAuth2 sections (CentralAuthentica-tionSystem/IdcsAuthenticationProvider). In a typical deployment, only the value of oauth2AuthorizationServerUrl needs to get changed. All the other configuration is required by the system but the default out of the box values are pre-configured correctly so the person doing the install does not have to change anything. Following is a snippet of the json.

Below is the json snippet of OAuth2

```
"CentralAuthenticationSystem":{

            "IdcsAuthenticationProvider":{

                "oauth2AuthorizationServerUrl":"<hostname>/oauth2/v1/token",
                "oauth2Application":[
                    {
                         "oauth2ApplicationName" : "RICS",
                         "oauth2ApplicationScopeOfAccess" :
{"name":"oauth2.default.scopeOfAccess.*", "value":"urn:opc:idm:__myscopes__"},
                         "oauth2ApplicationClientAlias" :
"ricsOauth2ApplicationClientAlias",
                         "oauth2ApplicationClientId" : "GET_FROM_WALLET",
                         "oauth2ApplicationClientSecret" : "GET_FROM_WALLET"

                    },
                    {
                         "oauth2ApplicationName" : "MFCS",
                         "oauth2ApplicationScopeOfAccess" :
{"name":"oauth2.default.scopeOfAccess.*", "value":"urn:opc:idm:__myscopes__"},
                         "oauth2ApplicationClientAlias" :
"mfcsOauth2ApplicationClientAlias",
                         "oauth2ApplicationClientId" : "GET_FROM_WALLET",
                         "oauth2ApplicationClientSecret" : "GET_FROM_WALLET"

                    },
                    {
                         "oauth2ApplicationName" : "RPAS",
                         "oauth2ApplicationScopeOfAccess" :
{"name":"oauth2.default.scopeOfAccess.*", "value":"urn:opc:idm:__myscopes__"},
                         "oauth2ApplicationClientAlias" :
"rpasOauth2ApplicationClientAlias",
                         "oauth2ApplicationClientId" : "GET_FROM_WALLET",
                         "oauth2ApplicationClientSecret" : "GET_FROM_WALLET"

                    }

                ]
```

```
},
"OamAuthenticationProvider":{
}
```

11

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-rpas" integration schema is created in the RPAS database server so that the data movement from inbound tables located in the integration schema to the RPAS transactional tables is fast.

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

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

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 RPAS, 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-rpas application simultaneously.

Logging

Issue

The "System Logs" tab in Scheduler, Process Flow, and Job Admin UIs show only logs from the server that UI is connected to.

Solution

Use a common log directory for each of the BDI components. BDI components use the following directory structure for creating log files.

$DOMAIN_HOME/logs/<server name>/<app name>

Example

$DOMAIN_HOME/logs/server1/rms-job-admin_war

$DOMAIN_HOME/logs/server2/rms-job-admin_war

1. Create a common log directory (e.g. /home/logs/bdijobadmin) for each BDI application.
2. Create symbolic links to the common log directory for each server using the below command from $DOMAIN_HOME/logs directory.

   ln -s /home/logs/bdijobadmin

   server1/rms-job-admin_war

   ln -s /home/logs/bdijobadmin

server2/rms-job-admin_war

3. If the directory $DOMAIN_HOME/logs/<server>/<app> already exists, it needs to be deleted before symbolic link is created.
4. App needs to be restarted after symbolic link is created.

   When weblogic managed servers are in different machines a shared network disk has to be used.

Update Log Level

Issue

When log level is updated through UI or REST end point, it updates the log level only on the server it is connected to.

Solution

Log level needs to be updated through the URL of all the nodes in the cluster using UI or REST endpoint.

Example

http://server1:port1/rms-batch-job-admin/system-setting/system-logs

http://server2:port2/rms-batch-job-admin/system-setting/system-logs

Create/Update/Delete System Options

Issue

When system options are created/updated/deleted using UI or REST end point, the changes are reflected only on the server that client is connected to.

Solution

The reset-cache REST endpoint need to be invoked on the other nodes in the cluster for that application in bdi.

Example

http://server1:port1/rms-batch-job-admin/system-setting/reset-cache

Create/Update/Delete System Credentials

Issue

When system credentials are created/updated/deleted using REST endpoint, the credentials are created/updated/deleted only on the server that client is connected to.

Solution

The REST endpoint that creates/updates/deletes credentials need to be invoked on all the nodes in the cluster for that application in BDI.

Example

http://server1:port1/rms-batch-job-admin/system-setting/system-credentials

http://server2:port2/rms-batch-job-admin/system-setting/system-credentials

Scheduler Configuration Changes for Cluster

1. Two data sources need to be created for scheduler on cluster in the Admin Console.

   - Create a non-XA data source (SchedulerTimerDs) pointing to the schema that contains the WEBLOGIC_TIMERS table. This is the schema with the WLS suffix, created using RCU.

     Specify this schema in the scheduling tab of cluster configuration in WebLogic console.

   - Create a non-XA data source (SchedulerRuntimeDs) pointing to schema that contains ACTIVE table. This is the schema with the WLS_RUNTIME suffix, created using RCU.

     Specify this schema in the Migration tab of cluster configuration in the WebLogic console.

   Perform the following steps to configure the data sources:

   a. Specify the data source for schedule timers in the Admin Console.

   b. Login to Admin Console.

   c. Click Lock & Edit (For Production Mode only).

   d. Click Environment -> Clusters.

   e. Click the cluster name.

   f. Click Scheduling.

   g. Select SchedulerTimerDs for the Data Source For Job Scheduler field.

   h. Click Save.

   i. Click Migration.

   j. Select Migration Basis: DataBase, and Data Source For Automatic Migration: SchedulerRuntimeDs.

   k. Click Save.

   l. Verify Auto Migration Table Name populated with ACTIVE.

   m. Click Activate Changes.

2. Update the weblogic-ejb-jar.xml in WEB-INF folder of the bdi-scheduler-ui-<version>.war in <bdi-home>/dist folder with the contents shown (The entry in red is the change from the existing contents of the file)

   Instructions to update

   a. cd dist

   b. jar xf bdi-scheduler-ui-<version>.war WEB-INF/weblogic-ejb-jar.xml

   c. Update the WEB-INF/weblogic-ejb-jar.xml with the contents below

   d. jar uf bdi-scheduler-ui-<version>.war WEB-INF/weblogic-ejb-jar.xml

   e. Delete dist/WEB-INF folder

   f. Deploy the scheduler application

```
<weblogic-ejb-jar xmlns="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <security-role-assignment>
```

```
            <role-name>AdminRole</role-name>
            <principal-name>BdiSchedulerAdminGroup</principal-name>
        </security-role-assignment>


        <security-role-assignment>
            <role-name>OperatorRole</role-name>
            <principal-name>BdiSchedulerOperatorGroup</principal-name>
        </security-role-assignment>
        <security-role-assignment>
            <role-name>MonitorRole</role-name>
            <principal-name>BdiSchedulerMonitorGroup</principal-name>
        </security-role-assignment>
        <timer-implementation>Clustered</timer-implementation>
    </weblogic-ejb-jar>
```

12

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.

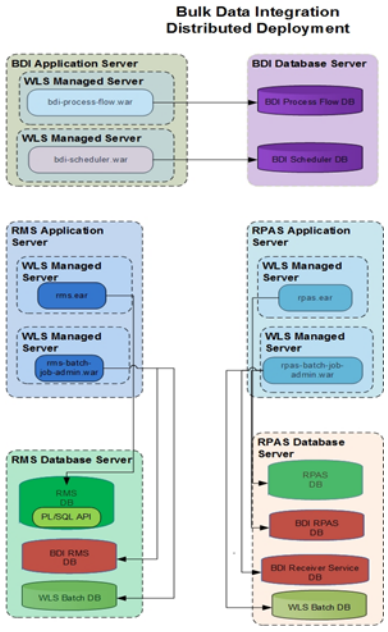
Recommended Deployment Options

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 12–1 Distributed Configuration



BDI-External Application

BDI is an integration infrastructure product which integrates Oracle Retail applications and third party applications. BDI external application is designed to address the complexities for third party integration with Oracle Retail application.

In BDI, bulk data movement happens between sender and receiver application.

External application may be a sender or receiver. But here, we talk about external application as only receiver.

For example, Sender application is RMS and Receiver is a third party application.

There will be two external applications for the integration to happen,

1. External edge application.

Bdi-external application organizes all Downloader Transporter and Uploader jobs. External application organizes all the importer jobs. Both bdi-external and external edge application provides GUI and CLI tool to manage jobs like start/stop/restart jobs.

There are process flow dsl files for each interface from RMS to external application which have all the activities for the particular interface. Scheduler will trigger the process flow to execute the activities within the dsl file.

Installation details

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

13

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.

14

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.

15

Job Admin REST Endpoints

Batch service is a RESTful service that provides various endpoints to manage batch jobs in Job Admin.

The endpoint "discover" can be used to identify all endpoints provided by Job Admin.

REST Resource	HTTP Method	Description
/discover	GET	Lists all available endpoints in Job Admin
/batch/jobs	GET	Gets all available batch jobs
/batch/jobs/enable-disable	POST	Enable or disable 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

REST Resource	HTTP Method	Description
/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
/batch/jobs/is-job-ready-to-start/{jobName}	GET	Is job ready to start for a given job name
/batch/jobs/group-definitions	GET	Gets group definitions
/batch/jobs/job-def-xml/{jobXmlId}	POST	
/telemetry/jobs	GET	Returns runtime job metrics between fromTime and toTime
/manage-group/group	PUT	Update a group
/manage-group/group	POST	Add a group
/manage-group/group/{groupId}	DELETE	Delete a group for a given groupId
/manage-group/group/{groupId}	GET	Gets group info for a given groupId
/manage-group/group/name/{groupName}	DELETE	Delete group info for a given group name
/manage-group/group/name/{groupName}	GET	Gets group info for a given group name
/manage-group/group/group-members	PUT	Update group members info
/manage-group/group/group-member	PUT	Update group member info
/manage-group/group/group-member	POST	Add a group member
/manage-group/group/{groupName}/group-member/{groupMemberName}	DELETE	Delete group member for a given group name and group member name
/manage-group/group/{groupName}/group-members	GET	Gets group members for a given group name
/manage-group/group/group-member/{groupMemberId}	GET	Gets group member info for a given group memberId

REST Resource	HTTP Method	Description
/manage-group/group/group-members/{memberName}/{memberType}	GET	Gets group members for a given member name and memberType
/manage-group/groups	GET	Gets all groups
/manage-group/groups	PUT	Updates all groups
/manage-group/groups	POST	Creates multiple groups specified in request with single request.
/manage-group/group/{groupName}/group-members	POST	Adds multiple members to a given groups at a time.
/manage-group/group/{groupName}/group-members	DELETE	Deletes all members from given group at once.
/manage-group/groups	DELETE	Deletes multiple groups at once
batch/jobs/job-def-xml/{jobName}	PUT	Creates an entry in BDI_JOB_DEFINITION table. It will throw an exception if job already exists.
batch/jobs/job-def-xml/{jobName}	POST	Updates an entry in BDI_JOB_DEFINITION table. It will update if job is not in running state. This end point throws an exception if job doesn't exist in the table
batch/jobs/job-def-xml/{jobName}	DELETE	Deletes an entry in BDI_JOB_DEFINITION table. It will delete if job is not in running state and if there is no history in batch database.
batch/jobs/{jobName}	DELETE	Deletes history for a job from batch database. It will delete history if job is not in running state.
/batch/jobs/bulk/job-definitions	POST	End point for bulk create/update job definitions
/batch/jobs/bulk/job-definitions	DELETE	End point for bulk delete job definitions

A

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.

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.
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_ACTIVITY_DYNAMIC_ CONFIG	This table stores the user runtime choices like SKIP, HOLD etc at activity level
BDI_SYSTEM_OPTIONS	This table has all the system level information like URLs, credential aliases etc.
BDI_EMAIL_NOTIFICATION	This table persists all the process email notifications.
BDI_PROCESS_CALL_STACK	This table stores call stack for processes.
BDI_EXTERNAL_VARIABLE	This table does temporary storage of variables during process execution.
BDI_GROUP	This table stores group names and its attributes
BDI_GROUP_LOCK	This table stores the lock id and group names
BDI_GROUP_MEMBER	This table stores all group member details

BDI_PROCESS_DEFINITION

Column	Type	Comments
PROCESS_NAME	VARCHAR2(255)	Name of the process
PROCESS_ENABLE_ STATUS	VARCHAR2(255)	Enable or disable the process (true or false)
PROCESS_CREATE_TIME	TIMESTAMP	Timestamp when the process was loaded to database
PROCESS_DEF_ CONTENT	CLOB	The Process Flow DSL

BDI_ACTIVITY_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_PROCESS_EXEC_INSTANCE

Column	Type	Comments
PROCESS_EXECUTION_ID	VARCHAR2(255)	Process Execution Id of the process instance that initiated the activity.
PROCESS_NAME	VARCHAR2(255)	Name of the process
PROCESS_EXEC_START_TIME	TIMESTAMP	Time when the process execution started
PROCESS_EXEC_END_TIME	TIMESTAMP	Time when the process execution started
PROCESS_FIRST_RUN_START_TIME	TIMESTAMP	Time when the process execution started, does not change when process is restarted.
PROCESS_STATUS	VARCHAR2(255)	Process status

BDI_ACTIVITY_DYNAMIC_CONFIG

Column	Type	Comments
PROCESS_NAME	VARCHAR2(255)	Name of the process
ACTIVITY_NAME	VARCHAR2(255)	Name of the activity
HOLD_FLAG	VARCHAR2(255)	To hold the activity
SKIP_FLAG	VARCHAR2(255)	To skip the activity
SKIP_OR_HOLD_EXPIRATION	TIMESTAMP	Time when skip or hold activity expires.
COMMENTS	VARCHAR2(255)	Comments

Column	Type	Comments
INVOKE_ CALLBACK_ SERVICE	VARCHAR2(255)	Invoke any callback service
USER_NAME	VARCHAR2(255)	Username
CALLBACK_ SERVICE_URL_ ALIAS	VARCHAR2(255)	Callback Service URL Alias
CALLBACK_ SERVICE_URL	VARCHAR2(255)	Callback Service URL

BDI_EMAIL_NOTIFICATION

Column	Type	Comments
EMAIL_ NOTIFICATION_ID	NUMBER	Process Execution Id of the process instance that initiated the activity
APP_NAME	VARCHAR2(100)	Name of the application
EMAIL_ NOTIFICATION_TO	VARCHAR2(500)	EMail Ids to whom notification will be sent
EMAIL_ NOTIFICATION_ SUBJECT	CLOB	Notification subject
EMAIL_ NOTIFICATION_ CONTENT	CLOB	Notification content
EMAIL_ NOTIFICATION_ DATETIME	TIMESTAMP	At what time notification sent
EMAIL_ NOTIFICATION_ TYPE	VARCHAR2(255)	Type of information
ACTION_STATUS	VARCHAR2(255)	status (PENDING/COMPLETED)
COMMENTS	VARCHAR2(500)	

BDI_SYSTEM_OPTIONS

Column	Type	Comments
CREATE_TIME	TIMESTAMP	Time it was created
UPDATE_TIME	TIMESTAMP	Time it was updated
VARIABLE_NAME	VARCHAR2(255)	Name of system variable
APP_TAG	VARCHAR2(255)	The application name

Column	Type	Comments
VARIABLE_VALUE	VARCHAR2(255)	Value of the variable

B

Process Flow REST Endpoints

The endpoint "discover" can be used to identify all endpoints provided by Process Flow.

REST Resource	HTTP Method	Description
/discover	GET	Lists all available endpoints
/batch/processes/enable-disable	POST	Enable or disable process flows
/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 all the activities for the process flow execution
/batch/processes/{processName}/activities	GET	Get all the activities for the process specified

REST Resource	HTTP Method	Description
/batch/processes/operator/{processName}/{processExecutionId}	POST	Restart a process execution instance
/batch/processes/operator/{processName}/resolve	POST	Resolves stranded process by setting the status of process to PROCESS_FAILED
/batch/processes/{processName}/{processExecutionId}	DELETE	Stops running process
/batch/processes/executions	DELETE	Stops all running processes
/batch/processes/{processName}/activities/{activityName}	POST	Sets skip, hold flags for activity. Query parameters that can be passed with this end point - "skip", "hold", "actionExpiryDate", "comments".
/batch/processes/{processName}/activities/{activityName}	GET	Returns dynamic configuration for activity
/telemetry/processes	GET	Returns process runtime metrics between fromTime and toTime

C

Scheduler REST Endpoints

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

The endpoint "discover" can be used to identify all endpoints provided by Scheduler.

REST Resource	HTTP Method	Description
/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/active-schedules	GET	Returns all active 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

REST Resource	HTTP Method	Description
/batch/schedules/executions/past/days/{days}	GET	Returns schedule executions for last n days
/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.
/batch/schedules/executions/time/{fromDateTime}/{toDateTime}	GET	Returns schedule executions between from and to time
/batch/schedules/activateOrDisable-schedules	POST	To update status of one or more schdules to ACTIVE or DISABLED

D

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 admin user is allowed to perform this operation.
/system-setting/system-options	POST	Updates a system option in 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_SYSETM_OPTIONS table. Only 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 admin user is allowed to perform this operation.
/system-setting/system-credentials	PUT	Creates system credentials. Only admin user is allowed to perform this operation.
/system-setting/system-credentials	POST	Updates system credentials. Only admin user is allowed to perform this operation.

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

Managing System Options using curl

Here are examples of curl commands to list/create/update/delete system options for Process Flow. These commands can be run for Job Admin, and Scheduler as well. Create/update/delete commands can only be run by administrator.

Create system option

This command creates "reimappJobAdminBaseUrlUserAlias" system option in Process Flow.

curl --user userId:password -i -X PUT -H "Content-Type:application/json" http://server:port/bdi-process-flow/resources/system-setting/system-options -d '{"key":"reimappJobAdminBaseUrlUserAlias" , "value":" GET_FROM_WALLET:GET_FROM_WALLET "}'

Update system option

This command updates "reimappJobAdminBaseUrl" system option in Process Flow.

curl --user userId:password -i -X POST -H "Content-Type:application/json" http://server:port/bdi-process-flow/resources/system-setting/system-options -d '{"key":"reimappJobAdminBaseUrl" , "value":"http://server:port/reim-batch-job-admin"}'

Delete system option

This command deletes "reimappJobAdminBaseUrl" system option from Process Flow.

curl --user userId:password -i -X DELETE -H "Content-Type:application/json" http://server:port/bdi-process-flow/resources/system-setting/system-options -d '{"key":"reimappJobAdminBaseUrl"}'

List system options

This command lists all system options from Process Flow.

curl --user userId:password -i -X GET -H "Content-Type:application/json" http://server:port/bdi-process-flow/resources/system-setting/system-options

Managing credentials using curl

Here are examples of curl commands to list/create/update/delete credentials for Process Flow. These commands can be run for Job Admin, and Scheduler as well. Create/update/delete commands can only be run by administrator.

Create credential

This command creates a credential in Process Flow.

curl --user userId:password -i -X PUT -H "Content-Type:application/json" http://server:port/bdi-process-flow/resources/system-setting/system-credentials -d '{"userAlias":" reimappJobAdminBaseUrlUserAlias", "userName":"reimjobadmin" , "userPassword":"xyzxyz"}'

Update credential

This command updates a credential in Process Flow.

curl --user userId:password -i -X POST -H "Content-Type:application/json" http://server:port/bdi-process-flow/resources/system-setting/system-credentials -d '{"userAlias":" reimappJobAdminBaseUrlUserAlias", "userName":"reimjobadmin" , "userPassword":"wwwqqqq"}'

Delete credential

This command deletes a credential from Process Flow.

curl --user userId:password -i -X GET -H "Content-Type:application/json" http://server:port/bdi-process-flow/resources/system-setting/system-credentials -d '{"key":"reimappJobAdminBaseUrl"}'

List Credentials

This command lists credentials from Process Flow.

curl --user userId:password -i -X GET -H "Content-Type:application/json" http://server:port/bdi-process-flow/resources/system-setting/system-credentials.

E

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;
```

F

Purge Strategy

The purge scripts helps in removing the old transactional data.

This script can run in two modes:

- Silent Mode

  Execute the procedure without passing any parameters, it calculates the period and remove the data. By default it keeps 6 months data and remove older than that. For example if some one wants to keep only 2 months of data and remove rest of the data, need to modify the procedure just by updating variable "howManyMonths NUMBER(5) := 6" to the required value.

  howManyMonths = 2;

- Interactive Mode

  Pass the fromDate and toDate parameters to the procedure to purge the specified period of data. Data will be removed for only specified period i.e fromDate to toDate.

  From and To dates format should be in below format:

  FROMDATE := TO_TIMESTAMP('2017-01-01T00:00:00', 'YYYY-MM-DD"T"HH24:MI:SS');

  TODATE := TO_TIMESTAMP('2017-04-30T23:59:59', 'YYYY-MM-DD"T"HH24:MI:SS');

The deleted data will be committed default in both silent and interactive modes.

Execute Purge SQL

Silent Mode

```
SET SERVEROUTPUT ON;
DECLARE
FROMDATE TIMESTAMP;
TODATE TIMESTAMP;
BEGIN
FROMDATE := NULL;
TODATE := NULL;
BDI_PURGE_SQL.PURGE_JOB_INT_REPO (
FROMDATE => FROMDATE,
 TODATE => TODATE  );
END;
```

Interactive Mode

```
SET SERVEROUTPUT ON;
DECLARE
FROMDATE TIMESTAMP;
TODATE TIMESTAMP;
BEGIN
FROMDATE := TO_TIMESTAMP ('2017-01-03T00:00:00', 'YYYY-MM-DD"T"HH24:MI:SS');
TODATE := TO_TIMESTAMP ('2017-01-03T23:59:59', 'YYYY-MM-DD"T"HH24:MI:SS');
BDI_PURGE_SQL.PURGE_JOB_INT_REPO (
  FROMDATE => FROMDATE,
  TODATE => TODATE  );
END;
```

The purge sql removes the data from the following tables for the respective schema.

Schema Name	Sql Name	Table Name
job-int-schema	purge_job_int_repo.sql	BDI_DWNLDR_TRNSMITR_EXE_DSET BDI_DWNLDR_TRNSMITTR_TRANS
job-rcvr-schema	purge_job_rcvr_repo.sql	BDI_RECVR_TRANSMISSION_BLOCK BDI_RECEIVER_TRANSMISSION BDI_RECEIVER_TRANSACTION
batch-db-schema	purge_batch_db_repo.sql	JOBSTATUS STEPSTATUS STEPEXECUTIONINSTANCEDATA EXECUTIONINSTANCEDATA JOBINSTANCEDATA CHECKPOINTDATA
process-schema	purge_process_repo.sql	BDI_ACTIVITY_EXEC_INSTANCE BDI_PROCESS_CALL_STACK BDI_PROCESS_EXEC_INSTANCE
scheduler schema	purge_scheduler_repo.sql	BDI_SCHEDULE_EXECUTION

G

Group and Group Member REST Endpoints

The Group and Group member service is a RESTful service available in BDI Job Admin that provides endpoints to manage group and members.

REST Resource	HTTP Method	Description
/manage-group/groups	GET	Returns list of all Groups
/manage-group/group/{groupId}	GET	Returns details of the group with input group ID
/manage-group/group	PUT	Updates the group details
/manage-group/group	POST	Creates new group
/manage-group/group/{groupName}	DELETE	Deletes the group with input group name
/manage-group/group/{groupName}/group-members	GET	Returns all groups members for the input group name
/manage-group/group/group-member/{groupMemberId}	GET	Returns details of the group member with input group member ID
/manage-group/group/group-member	PUT	Updates the group member details
/manage-group/group/{groupName}/group-member	POST	Create a new member and adds it to the group with input group name
/manage-group/group/{groupName}/group-member/{groupMemberName}	DELETE	Deletes the group member with input group name and group member name
/manage-group/group/{groupName}/group-member/{groupMemberName}	DELETE	Deletes the group member with input group name and group member name
/manage-group/groups	POST	Adds multiple groups provided in input at a time.
/manage-group/group/{groupName}/group-members	POST	Adds multiple members to a given groups at a time.
/manage-group/group/{groupName}/group-members	DELETE	Delete all members from given group.
/manage-group/groups	DELETE	Deletes all groups provided in input.

H

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, RPAS).
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/RPAS) 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	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: ■ 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.